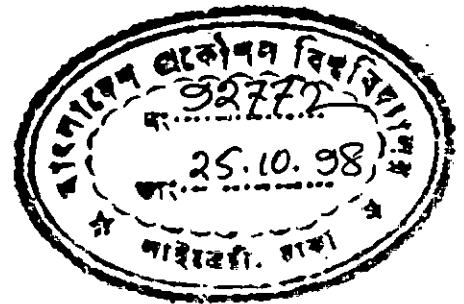
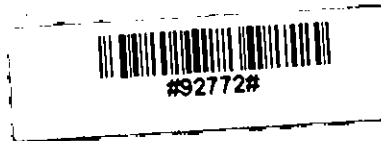


DEVELOPMENT OF A LOW COST 16-BIT MICROPROCESSOR TRAINER

BY
GOLAM MOSTAFA



SUPERVISED BY
DR. CHOWDHURY MOFIZUR RAHMAN



A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE IN ENGINEERING
(COMPUTER SCIENCE AND ENGINEERING)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA, BANGLADESH
OCTOBER, 1998

DEVELOPMENT OF A LOW COST 16-BIT MICROPROCESSOR TRAINER

A thesis submitted by

GOLAM MOSTAFA

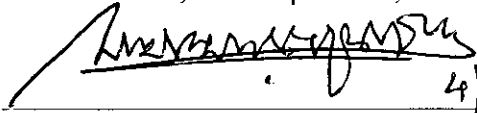
Roll No: 901807P, Session 1988-89, Registration No: 72343, to the
Department of Computer Science and Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Engineering (Computer Science & Engineering).
Examination held on : 4/10/98

Approved as to the style and content by :-

 4/10/98

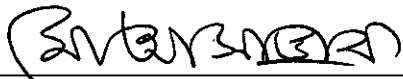
(DR. CHOWDHURY MOFIZUR RAHMAN)
Asst. Professor, CSE Department, BUET

Chairman and Supervisor

 4/10/98

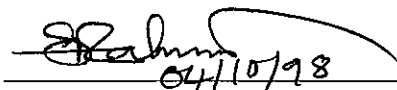
(PROF. DR. M. KAYKOBAD)
Head, CSE Department, BUET

Member



(MD. ABDUS SATTAR)
Asst. Professor, CSE Department, BUET

Member

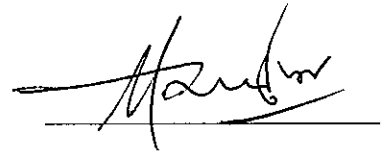
 04/10/98

(PROF. DR. MD. SAIFUR RAHMAN)
EEE Department, BUET

Member (External)

DECLARATION

This is to declare that the work presented in this thesis is the result of my extensive study and experiments on 'The Development of a Low Cost 16-Bit Microprocessor Trainer' under the supervision of Dr Chowdhury Mofizur Rahman of the Department of Computer Science and Engineering of the Bangladesh University of Engineering and Technology, Dhaka. It is further declared that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

A handwritten signature in black ink, appearing to read 'Mofizur Rahman', is written over a horizontal line.

Signature of the Author

ACKNOWLEDGMENT

The author wishes to make due acknowledgments to the following organizations and persons for their generous support, participation and encouragement for the successful realization of a working 8086-based trainer.

To the Department of Computer Science and Engineering of the Bangladesh University of Engineering and Technology for its support in conducting higher studies in the field of computer science and technology. This favorable atmosphere has led to the materialization of an 8086-based trainer.

To Dr. Syed Mahbubur Rahman, ex-Head of the Department of Computer Science and Engineering of BUET, for his keen interest in this project. In fact, the project was initiated during his time.

To Dr. M. Kaykobad, Professor and Head, Department of Computer Science and Engineering, BUET for his active effort in finding a professor to supervise the project works.

To Dr. Mohammed Ali Chowdhury, Professor, Department of Electrical and Electronic Engineering, BUET, for his continuous encouragement.

To Dr. Chowdhury Mofizur Rahman, Assistant Professor, Department of Computer Science and Engineering, BUET - who is the supervisor of the present work. Special thanks and regards are due to him for his pleasant personality and technical expertise that has supplied most of the fuel necessary for successful completion of the present work.

To Mr. Abdus Satter, Assistant Professor, Department of Computer Science and Engineering, BUET, for his dedication in sparing time to discuss many microprocessor related issues.

To the staff of the department of CSE, DAERS and ADMIN. of BUET who helped me in many ways.

To Engr. Akbar-e-Sarwar, Director of Design Group Limited, Dhaka for his untiring efforts for making the double-layer PCB for the trainer. Without this PCB, the trainer would never be transformed from the engineering prototype to the professional version.

To members of the family, friends and others who shared directly or indirectly the sorrow and happiness that the author was experiencing during the development phase of the trainer.

Golam Mostafa
October, 1998

dedicated to.....
Late Moni Singh

Abstract

This thesis contains the technical details of the design, development and construction of an 8086-based 16-bit microprocessor trainer. The trainer has been built using local technology and at lesser cost compared to the foreign made trainers. The hardware and software design are simple and logical to allow others become acquainted with the design rules. The trainer has been built with the features of a 'Learning and Development System.' The features are (a) edge connectors for developing interfacing circuits, (b) integrated peripheral module containing all the common peripheral controllers (c) the IBM-PC to trainer down loading software and (e) many useful routines and subroutines in the EPROMs.

The trainer has been constructed successfully. All the objectives quoted above have also been achieved.

The trainer has the following hardware features : (a) 8086 CPU, (b) 64Kbytes EPROM, (c) 64 Kbytes RAM, (d) Bus Lines at Edge Connectors for Interfacing Experiments, (e) 5.5''x2.5'' Bread Board for Prototyping Circuits, (f) Well-documented User's Manual, (g) 18-Key Hex-key pad for Machine Codes Programming, (h) 9 - Digits 7-Segment Display Window, (i) Memory and Port Decoded Lines Available at Edge Connectors, (j) +5V Power Supply Adapter.

The trainer has also software features like (a) Powerful and Comprehensive Resident Monitor Program, (b) Auto/Manual Data Entry for both Bytes/Word Operations, (c) Program Execution Capability, (d) Forward/Backward/Change/Backspace Facilities, (e) Bytes/Word Examine/Edit Capability, (f) Single Instruction Execution Capability for Program Debugging, (g) Basic Initialization Routines for Many Peripheral ICs like ADC, DAC, 8251, 8259, (h) Register's Contents can be Examined and Changed, (i) Flag Register's Contents can be Examined in Bit-form and Hex-form, (j) Many Stand-alone Useful Routines and Subroutines to facilitate microprocessor based system design.

This thesis contains detailed description of the procedures and techniques employed for the design, development and construction of the trainer. It is a comprehensive reference containing experimented steps that the designers and academicians may consult to solve microprocessor related problems. This thesis has also documented the description of the new ideas conceived to solve varieties of hardware and software problems. The examples are -- the design of composite memory/port decoder and single stepping routine.

The thesis contains 10 chapters, 6 appendices and a reference caption. Attempt has been made to document the work in the form of descriptive language, schematic diagram, flow chart, assembly and C codes.

CONTENTS

<i>List of Figures</i>		x
1	INTRODUCTION	--	--	--	--	--	--	--	1
2	8086 MICROPROCESSOR FUNDAMENTALS								
	2.1 Pin Diagram and Functions			--	--	--	--		5
	2.2 Internal Architecture			--	--	--	--		14
	2.3 Instructions			--	--	--	--		17
3	OPERATING PROCEDURES								
	3.1 Trainer Board Layout			--	--	--	--		21
	3.2 Components Description			--	--	--	--		22
	3.3 Signal Signatures			--	--	--	--		23
	3.4 Keyboard Mnemonics and Meaning			--	--	--	--		25
	3.5 Program Codes/Data Loading			--	--	--	--		27
	3.6 Program Execution			--	--	--	--		28
	3.7 Program Debugging/Register Check			--	--	--	--		28
	3.8 Example Programs			--	--	--	--		30
4	HARDWARE DESIGN								
	4.1 Block Diagram			--	--	--	--		32
	4.2 CPU Subsystem			--	--	--	--		34
	4.3 Memory/Port Decoding Subsystem			--	--	--	--		36
	4.4 Memory Subsystem			--	--	--	--		38
	4.5 Keyboard/Display Subsystem			--	--	--	--		40
5	MONITOR PROGRAM DESIGN								
	5.1 What is a Monitor Program?			--	--	--	--		42
	5.2 Implementing EXA, EXB/EXW, AUT, BKS and DOP Commands								
	5.2.1 Respond/Action for EXA Command			--	--	--	--		52
	5.2.2 Respond/Action to EXB/EXW Commands			--	--	--	--		53
	5.2.3 Respond/Action to AUT Command			--	--	--	--		54
	5.2.4 Backspace Routine			--	--	--	--		55
	5.2.5 Printing at Display Window (Address/Data Field)			--	--	--	--		56
	5.2.6 Respond/Action to DOP Command (Program Execution)			--	--	--	--		57
	5.3 Implementing FRW/BKW/CHG (non S/S) Commands			--	--	--	--		58
	5.3.1 Forward Routine					--	--		59
	5.3.2 Backward Routine					--	--		60
	5.3.3 CHG Command and Byte-data Update for Memory			--	--	--	--		61
	5.3.4 CHG Command and Word-data Update for Memory			--	--	--	--		62

5.4	Theory of Single Stepping (Program Debugging Routine)				
5.4.1	Respond/Action to PC Command	--	--	--	63
5.4.2	Home Key Routine (PC command in S/S Mode)	--	--	--	64
5.4.3	Execution of One Instruction (Single Stepping)	--	--	--	65
5.4.4	Exam/Edit AX,BX,CX,DX Registers	--	--	--	70
5.4.5	Exam/Edit AL,AH,BL,BH,CL,CH,DL,DH Registers	--	--	--	71
5.4.6	Exam/Edit IP,SI,DI,SP,BP Registers	--	--	--	72
5.4.7	Exam/Edit CS,DS,SS,ES Registers	--	--	--	73
5.4.8	Exam/Edit Flag Register	--	--	--	74
5.4.9	Exam/Edit Memory Contents (Memory as a simple Register)	--	--	--	75
5.4.10	Exam/Edit Port Contents	--	--	--	76
5.5	Subroutines	--	--	--	77
5.6	Stand-alone Routines	--	--	--	83
5.7	Data Tables	--	--	--	86
5.8	Interrupt Vector Table	--	--	--	88
5.9	Memory Space Map	--	--	--	90
5.10	Port Space Map	--	--	--	92
5.11	Reserved RAM Space Map	--	--	--	94

6 INTEGRATED PERIPHERAL MODULE

6.1	Introduction and Board Components Layout	--	--	--	96
6.2	Components Description	--	--	--	97
6.3	Signal Signatures	--	--	--	98
6.4	Block Diagram	--	--	--	100
6.5	Local Extended Port Decoder	--	--	--	102
6.6	6 Channel Programmable Interval Timer using two 8253s--	--	--	--	104
6.7	8 Channel 8-Bit ADC using AD0804 + Multiplexer DG508	--	--	--	106
6.8	8-Bit Digital-to-Analog Converter using AD558 --	--	--	--	108
6.9	Multifunction Asynchronous Receiver/Transmitter using 8256				
6.9.1	Parallel I/O using 8256	--	--	--	110
6.9.2	Serial I/O using 8256	--	--	--	112
6.9.3	Timing Functions using 8256	--	--	--	114
6.9.4	Counting Functions using 8256	--	--	--	116
6.9.5	Interrupt Priority Management using 8256	--	--	--	118

7 APPLICATIONS

7.1	As a Microprocessor Learning System				
7.1.1	Developing and Running Program for Hardware Project	--	--	--	120
7.1.2	Developing and Running Program for Software Projects	--	--	--	122
7.2	As a Microprocessor Based System Design Kit				
7.2.1	Generating Timing Functions for Firing the Bedford Inverter	--	--	--	124
7.2.2	Developing an EPROM Programmer	--	--	--	126

8 IBM-PC to TRAINER DOWN LOADING SOFTWARE

8.1	Introduction	--	--	--	141
8.2	Assembly Source Code Listing for IBM-PC's EXE Prog	--	--	--	144
8.3	C Source Code Listing for IBM-PC's EXE Prog	--	--	--	151
8.4	Assembly Source Code Listing for Trainer's EXE Prog	--	--	--	153

9	RESULTS AND DISCUSSION	--	--	--	--	--	157
10	CONCLUSION AND FUTURE SCOPE OF WORKS--	--	--				159
APPENDIX - A MAXIMUM MODE OPERATION OF 8086 with 8087 FPU							
	A.1	Circuit Diagram Description	--	--	--		161
	A.2	Demonstration Example	--	--	--		163
APPENDIX - B 8086 TRAINER WITH BUILT-IN ASSEMBLER							
	B.1	Introduction	--	--	--	--	164
	B.2	Alphanumeric Display /Keyboard Circuit				--	166
APPENDIX - C SELECTED DATA SHEETS (8279, 8256)							
			--	--			170
APPENDIX - D HARDWARE DEVELOPMENT TECHNOLOGY							
	D.1	Introduction to Procedure/Methodology	--	--			194
	D.2	Wire-wrapping Accessories	--	--	--		195
	D.3	Printed Circuit Board Manufacturing	--	--			196
	D.4	Soldering Techniques	--	--	--		197
	D.5	View of the Trainer Board as seen from TOP	--				198
	D.6	Solder side PCB Artwork Looking from Bottom	--				199
	D.7	Component-side PCB Artwork Looking from TOP					200
	D.8	Solder Mask (including PTH) Artwork	--	--			201
APPENDIX - E MONITOR PROGRAM DEVELOPMENT TECHNOLOGY							
							202
APPENDIX - F COLOR PLATES							
	F.1	Component Side of the Prototype 8086 Trainer	--				206
	F.2	Wire-wrap Side of the Prototype 8086 Trainer	--				206
	F.3	Component Side of the PCB of Commercial Trainer					207
	F.4	Solder Side of the PCB of Commercial Trainer	--				207
	F.5	Pictorial View of the Final Commercial Trainer	--				208
REFERENCES							
	--	--	--	--	--	--	209

List of Figures

Figure - 1	: Growth of Maximum Achievable IC Components Density	..	1
Figure - 2.1(a)	: Pin Assignment of the 8086 CPU (Min Mode)	..	5
Figure - 2.1(b)	: Bus Structured Pin Assignment of 8086 CPU (Min Mode)	..	5
Figure - 2.1(c)	: Implementing the DEN/ and DT-R/ Signals	10
Figure - 2.1(d)	: Basic 8086 Timings in Minimum Mode	12
Figure - 2.1(e)	: Schematic to Allocate 1mbyte per Segment of 8086 CPU	13
Figure - 2.2(a)	: Functional Block Diagram of 8086 CPU	14
Figure - 2.2(b)	: Address Computing Chart of 8086 CPU	15
Figure - 2.2(c)	: Internal Register Layout of the 8086	16
Figure - 2.3(a)	: Instructions Mnemonics Summary of 8086 CPU	17
Figure - 2.3(b)	: Coding Template of 8086 Instruction	18
Figure - 2.3(c)	: Coding Template of 8086 Instruction (continued)	19
Figure - 2.3(d)	: Coding Template of 8086 Instruction (continued)	20
Figure - 3.1	: The Component Layout of the 8086 Trainer	21
Figure - 3.4	: Keyboard Template of the 8086 Trainer	25
Figure - 3.8	: Installing Additional 7-Segment Digit with 8086 Trainer	30
Figure - 4.1	: Hardware Block Diagram of 8086 Trainer	33
Figure - 4.2	: CPU Subsystem Schematic Diagram	35
Figure - 4.3	: Memory/Port Decoding Subsystem	37
Figure - 4.4	: Memory Subsystem Schematic Diagram	39
Figure - 4.5	: Keyboard/Display Subsystem Schematic Diagram	41
Figure - 5.1(a)	: Monitor Program Flow Chart Summary - 1	44
Figure - 5.1(b)	: Monitor Program Flow Chart Summary - 2	45
Figure - 5.1(c)	: Monitor Program Flow Chart Summary - 3	46
Figure - 5.1(d)	: Monitor Program Flow Chart Summary - 4	47
Figure - 5.1(e)	: Monitor Program Flow Chart Summary - 5	48
Figure - 5.1(f)	: Monitor Program Flow Chart Summary - 6	49
Figure - 5.1(g)	: Monitor Program Flow Chart Summary - 7	50
Figure - 5.1(h)	: Monitor Program Flow Chart Summary - 8	51
Figure - 5.2.4	: Flow Chart for Implementing Backspace Routine	55
Figure - 5.2.5	: Flow Chart to Implement Digit Printing in the Display Window	56
Figure - 5.4.3	: Flow Chart and Data Structure to Implement Single Step Routine	65
Figure - 5.4.4	: Flow Chart to Implement AX,BX,CX and DX Content Examination	70
Figure - 5.4.5	: Flow Chart to Examine AL,AH,BL,BH,CL,CH,DL,DH Contents	71
Figure - 5.4.6	: Flow Chart to Examine IP,SI,SP,DI,BP Contents	72
Figure - 5.4.7	: Flow Chart to Examine CS,DS,SS,ES Contents	73
Figure - 5.4.8	: Diagram Showing the Mapping of Flag Register's Content to Display	74
Figure - 5.4.9	: Flow Chart to Examine Memory Content in S/S without Exiting the S/S75	76
Figure - 5.4.10	: Flow Chart to Examine Port Contents	76
Figure - 5.6(a)	: Circuit for Prototyping Analog-to-Digital Conversion	83
Figure - 5.6(b)	: Circuit Diagram for Stand-alone Routine No. 4)RUT - 4)	85
Figure - 5.8	: Data Structure for Interrupt Vector Table	89
Figure - 5.9	: Memory Map Table	91
Figure - 5.10	: Port Space Map Table	93
Figure - 5.11	: Memory Map Table for Reserved RAM Space	95
Figure - 6.1	: Broad Layout for the Integrated Peripheral Module	96

Figure - 6.4	: Block Diagram for the Integrated Peripheral Module	101
Figure - 6.5	: Schematic Diagram for the Extended Port Decoder of IPM	103
Figure - 6.6	: Schematic Diagram for 6-Channel Programmable Timer	105
Figure - 6.7	: Schematic Diagram of the 8-Channel AC	107
Figure - 6.8	: Schematic for 8-Bit Digital-to-Analog Converter	109
Figure - 6.9.1	: Schematic for Parallel I/O using 8256	111
Figure - 6.9.2	: Schematic Diagram for Serial I/O Using 8256	113
Figure - 6.9.3	: Schematic Diagram for the Timing Functions Using 8256	115
Figure - 6.9.4	: Schematic Diagram for the Counting functions Using 8256	117
Figure - 6.9.5	: Schematic Diagram for Interrupt Priority Management Using 8256	119
Figure - 7.1.1	: Schematic for Driving LED Arrays Using 8255 and 8086 Trainer	121
Figure - 7.1.2	: Flow Chart to Implement Bubble Sort Algorithm	123
Figure - 7.2.1	: Schematic for Interfacing Bedford Inverter	125
Figure - 7.2.2(a)	: EPROM Programming Chart using 28-PIN ZIF Socket	126
Figure - 7.2.2(b)	: 28-Pin ZIF Socket Pin Diagram	126
Figure - 7.2.2(d)	: Schematic for the EPROM Programmer	128
Figure - 7.2.2(e)	: Details of the Switches of the EPROM programmer	129
Figure - 7.2.2(f)	: Programming Algorithm for 2716	130
Figure - 7.2.2(g)	: Timing Parameters for 2716 EPROM	131
Figure - 7.2.2(h)	: Programming Algorithm for 2732A	132
Figure - 7.2.2(i)	: Timing Parameters for 2732A EPROM	133
Figure - 7.2.2(j)	: Programming Algorithm for 2764A	134
Figure - 7.2.2(k)	: Timing Parameters for 2764A EPROM	135
Figure - 7.2.2(l)	: Programming Algorithm for 27128A EPROM	136
Figure - 7.2.2(m)	: Timing Parameters for 27128A EPROM	137
Figure - 7.2.2(n)	: Programming Algorithm for 27256 EPROM	138
Figure - 7.2.2(o)	: Timing Parameters for 27256 EPROM	139
Figure - 7.2.2(p)	: Timing Parameters for 27256 EPROM (continued)	140
Figure - 8.1(a)	: Hardware Interface Circuit Bet ⁿ Trainer and IBM-PC	142
Figure - 8.1(b)	: Software Flow Chart for IBM-PC's Serial Transmission Program	143
Figure - 8.4	: Software Flow Chart for Trainer's Firmware	156
Figure - A.1	: Schematic Diagram of 8087 with 8086	162
Figure - B.1	: Component Layout of the ROM-BASED Assembler's 8086 Trainer	165
Figure - B.2(a)	: Schematic of the Alphanumeric Display/Keyboard	167
Figure - B.2(b)	: Internal Circuitry of MAN2815 Display Device.. ..	168
Figure - B.2(c)	: Character ROM Data Table	169
Figure - D.2	: Wire Wrap Tools and Accessories	195
Figure - D.4(a)	: Soldering Techniques	197
Figure - D.4(b)	: Good Soldering	197
Figure - D.4(c)	: Poor Soldering	197
Figure - D.5	: View of the Trainer Board as seen from TOP	198
Figure - D.6	: Solder Side PCB Artwork Looking from Bottom	199
Figure - D.7	: Component Side PCB Artwork Looking from TOP	200
Figure - D.8	: Solder mask including PTH of the PCB Artwork	201
Figure - F.1	: Color Plate of Component Side of the Prototype 8086 Trainer	206
Figure - F.2	: Color Plate of Wire Wrap Side of the 8086 Trainer	206
Figure - F.3	: Color Plate of Component Side of the PCB of 8086 Trainer	207
Figure - F.4	: Color plate of Solder Side of the PCB of the 8086 Trainer	207
Figure - F.5	: Color plate of the pictorial View of the final 8086 Trainer	208



1

INTRODUCTION

This chapter gives a brief introduction to the Evolution of Microprocessor, Motivation for the Thesis Work, Detailed Content Outline and the Main Features of the developed Trainer.

Evolution of Microprocessor

A microprocessor is a programmable device. Within it, there are thousands of transistors forming large number of floating type basic logic gates and memory cells. The interconnections between these circuits are established and broken asserting software commands from outside. This way, a single microprocessor chip is used to implement numerous types of work. The credit for the idea of a general purpose microprocessor goes to one of the Intel engineers who was working on the Japanese camera Co. - Busicom around 1970. The microprocessor can take data from different users and can modify the data exactly the way a user wants. The CPU can give back the modified data to the user.

The arrival of microprocessor has brought revolutionary change in the field of 'Information Technology' and 'Industrial Instrumentation and Control.' Without today's high speed microprocessor, it would be simply impossible to build low cost desktop computers that are processing and presenting 'Information' worth million of dollars.

Early microprocessors, during 1974 - 1976, were mainly developed for making business calculators and controlling simple industrial processes. With the passage of time, peoples' mentality evolved and started thinking of 'Time Saving.' They made the personal computers using the early 8-bit microprocessor (8085, 6502,...) to do their word processing and simple spread sheet works at home without going to the mainframe station located somewhere down town.

The first 16-bit microprocessor, called 8086 appeared in 1978. Enhanced version of 8086, called 80186, appeared in the same year with integrated peripherals. In 1983, the 80286 microprocessor appeared with 'Protected Virtual Mode Addressing' capability and 'Multitasking' support. As application begun to demand more speed, the 80386 (32-bit) appeared in 1985. In 1989, Intel released 80486 (32-bit) with built-in math coprocessor and cache. The Pentium (64-bit) appeared in 1993.

Figure - 1 presents a curve showing the growth of component density per chip versus time.

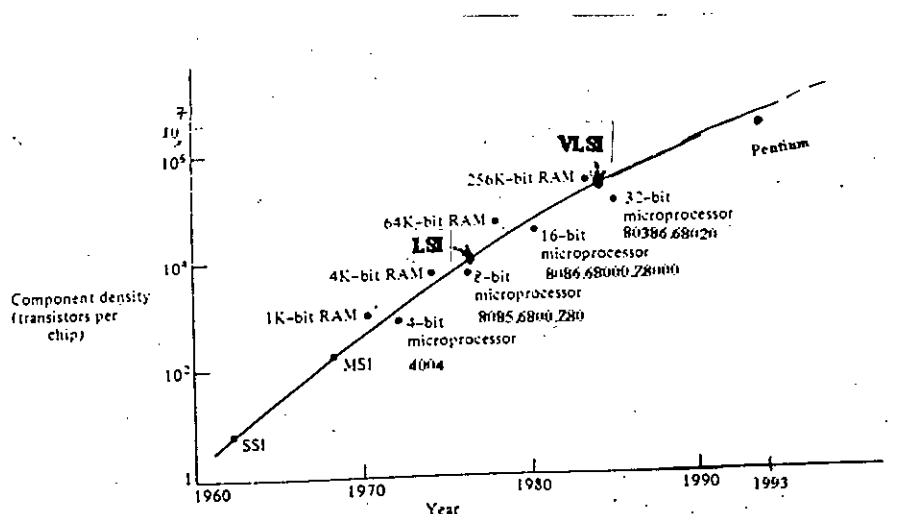


Figure-1 : Growth of Maximum Achievable IC Components Density.

Motivation for the Present Work

“Microprocessor Fundamentals” is a basic subject offered to computer, electrical and electronic engineering and many other applied disciplines. Because of the enormous popularity of Intel microprocessors, it has become a *de facto* standard of including Intel’s 8085 (8-Bit) and 8086 (16-Bit) microprocessors in the curriculum. The study and the understanding of the features of these microprocessors require relevant CPU based trainers. The prescribed text books usually make frequent references to the microprocessor trainers for the clarification of many concepts and the experimentation of new ideas. These trainers are also needed by the professionals engaged in designing and developing microprocessor based industrial control systems and consumer products.

These trainers are non-consumer items and are usually made by the rich companies in the developed countries as a side product in their factories, where the main product is an electronic/electrical consumer product. They have the manpower, machinery, sophisticated development tools etc., worth millions of dollars. Setting up such a million dollar factory to produce these educational trainers, would merely mean to be a dream - at least, in Bangladesh!

The educational institutions of Bangladesh are in a crying need of these trainers. They could not get them in time due to price hulk and import formalities. As a result, the microprocessor courses are being conducted mainly on paper and black board. Some of the institutions might have their trainers but the numbers are certainly very low compared to the number of students.

Outline of this Thesis

Chapter - 1 is the introduction. It makes a brief survey of the evolution of various microprocessors. A curve is shown to indicate the trend of evolution of microprocessor with time. This chapter has included the motivation behind the development of the 8086 trainer presented in this thesis.

Chapter - 2 has documented the operating procedures of the trainer. The meanings for the labels of the Keys of the ‘Keyboard Template’ are also given. A component layout of the trainer is provided. The commercial type numbers of the components are also provided. The names of the signals at various pins of the edge connectors are also given. This chapter also contains some example programs for exercising how to enter a program code into the trainer for execution.

Chapter - 3 describes the fundamental features of the 8086 microprocessor. The functional description of the pins are presented along with diagrams. The use of DT-R/ and DEN/ signals are illustrated showing their connection with 74LS245 data buffer. The definitions of the processor status signals are given. Special attention is given as to the application of the S4,S3 signals in connection with their possible use for accessing 1Mbyte of memory per segment. A circuit diagram is added showing how these status bits could be decoded to allow the 8086 CPU for accessing 4 Mbytes of memory [1].

The chapter also includes the register architecture of the 8086 processor. An address computation chart is provided along with examples. This chapter also contains a summary of the instruction set.

Chapter - 4 includes the complete schematics of the hardware of the trainer. The total circuit is divided into four subsystems viz., (1) CPU Subsystem, (2) Memory/Port Decoding Subsystem, (3) Memory Subsystem, and (5) Keyboard/Display Subsystem. There is a block diagram showing the overall hardware structure of the trainer. Every subsystem is followed by detailed description of the working principles of the circuit of that subsystem.

Chapter - 5 is the full documentation of the monitor program of the 8086 trainer. The logic of the entire monitor program is summarized into eight flow charts. These flow charts will be frequently referred during the study and analyze of the working principles of the monitor program.

There is also description corresponding to each key command. For example, how does the trainer response when EXA key is pressed. Or, how does the CPU response when BKS key is pressed. The description has also been augmented by flow charts and program listing.

The chapter contains the source code listing of all the subroutines and the stand-alone routines which are provided as firmware in the EPROMs of the trainer. The various data and lookup tables are also given.

The memory/port and the reserved RAM space maps are provided. The interrupt vector table is also given.

Chapter - 6 contains full technical documentation of an Integrated Peripheral Module containing 2 - Programmable Interval Timers of the type 8253, 1- Analog-to-Digital Converter of type AD08084, 1- Digital-to-Analog Converter of the type AD558, 1- Eight Channel Analog Multiplexer of the type DG508, RS232-TTL-RS232 Converting Chips of the type 489, 488 and a versatile Controller Chip of the type 8256. The 8256 chip contains all the commonly used five functions viz., Parallel I/O, Serial I/O, Timing, Counting and Interrupt Priority Management. This separate board is provided to help convenient implementation of interfacing experiments. Basic initialization routines of all the peripherals are also given.

Chapter - 7 has indicated the realistic application of the 8086 trainer. The applications are (1) display of the binary data using an 8255 controller (2) coding and running a Bubble Sort Program (3) Complex timing function generation for the Bedford Inverter and (4) the construction of an EPROM programmer. The chapter also contains full data sheet of the commonly used EPROMs of the type 2716, 2732, 2764, 27128 and 27256. The data sheets include the flow charts of numerous programming algorithms. An example program has been included in this chapter to program an EPROM of the type 2716.

Chapter - 8 documents the serial communication software to download program codes from IBM-PC to the 8086 trainer. The schematic of the hardware interfacing between the trainer and the IBM-PC is given. The transmitter software is developed using assembly/C language and the complete source code listing is provided. The resulting exe file is named scom86.EXE and is provided in a 3.5" disk with the trainer. The trainer's firmware listing is also given.

Chapter - 9 shows a study of the results expected from the trainer and the actual results achieved. The reasons for the discrepancies are described in the discussion caption.

Chapter - 10 makes hints as to possible future good technical works that could be done based on this trainer. The possible areas are 80286 trainer to study PVAM features, ROM-Based Assembler, On-board 8087 math coprocessor and improving the existing monitor program.

Appendix - A includes the schematic diagram of the 8086 system in its maximum mode while operated with an 8087 math coprocessor.

Appendix - B documents the preliminary experimental works for the development of a ROM based Assembler 8086 Trainer. The component lay out is given along with an alphanumeric keyboard. The original Hex keyboard is retained to operate the trainer for learning purposes. The appendix contains the full schematic of the 15-segment alphanumeric display system. The ROM character table is also presented. The internal circuitry of the 15-segment display is also provided.

Appendix - C contains data sheets for the 8279 and 8256 programmable controllers. This is provided to make this thesis complete.

Appendix - D describes the manual procedures of making the PCB for the trainer. The PTH (printed through holes) have been made by hand. The procedures of good soldering is also presented in the form of pictorial illustration. This appendix also contains the pictorial view of the PCB artwork.

Appendix - E describes the manual procedures/methodology of fusing the program/data codes of the monitor program into EPROM. This job is usually done by an automatic development system where the program codes are developed in the IBM-PC and then down loaded into the prototype trainer..

Appendix - F includes the color plates of the various board of the trainer. These are the component/wire wrapping sides, the component/solder sides of the final PCB and the solder mask.

Features Summary

HARDWARE and SOFTWARE FEATURES SUMMARY

Hardware Features:

01. 8086 CPU
02. 64Kbytes EPROM
03. 64 Kbytes RAM
04. Bus Lines at Edge Connectors for Interfacing Experiments
05. 5.5''x2.5'' Bread Board for Prototyping Circuits
06. Well Documented User's Manual
07. 18-Key Hex-key pad allows Machine Codes Programming
08. 9 - Digits 7-Segment Display Window
09. Memory and Port Decoded Lines are Available at Edge Connectors
10. +5V Power Supply Adapter.

Software Feature:

01. Powerful and Comprehensive Resident Monitor Program
02. Auto/Manual Data Entry for both Bytes/Word Operations
03. Program Execution Capability
04. Forward/Backward/Change/Backspace Facilities, Bytes/Word Examine/Edit Capability
05. IBM-PC to Trainer Down Loading Software
06. Single Instruction Execution Capability for Program Debugging
07. Basic Initialization Routines for Many Peripheral ICs like ADC, DAC,8251,8259
08. Register's Contents can be Examined and Changed

2

**8086
MICROPROCESSOR
FUNDAMENTALS**

This chapter is an introduction to the fundamental of 8086 microprocessor. Included are, the detailed description of the various pin functions, the basic timing diagram, internal architecture, register layout, address computing chart and the instruction set summary.

2.1 Pin Diagram and Functions

The 16-bit 8086 microprocessor is packaged in a 40-pin CERDIP (CERamic Dual In Package) or plastic DIP package [Figure - 2.1(a), 2.1(b)]. It is 0.3-inch wide and 2-inch long. The pins are separated from each other by 0.1-inch. The CPU design has been implemented using HMOS technology to achieve high performance. The 8086 operates at 5MHz, 8086-2 at 8MHz and the 8086-1 at 10 MHz with maximum power dissipation of 2.5W. There are corresponding CMOS versions (80C86,80C86-2,80C86-1) which operate at the same frequencies as for the HMOS 8086 but with maximum power dissipation of 1W. The 8086 has two modes of operation viz., Minimum Mode which does not permit connecting a 2nd processor/co-processor in the system and Maximum Mode which allows connecting a second processor/co-processor. Throughout this thesis, we will be studying the 8086 CPU in a minimum mode for better understanding the underlying concepts. Appendix-A includes the maximum mode operation of the 8086 CPU.

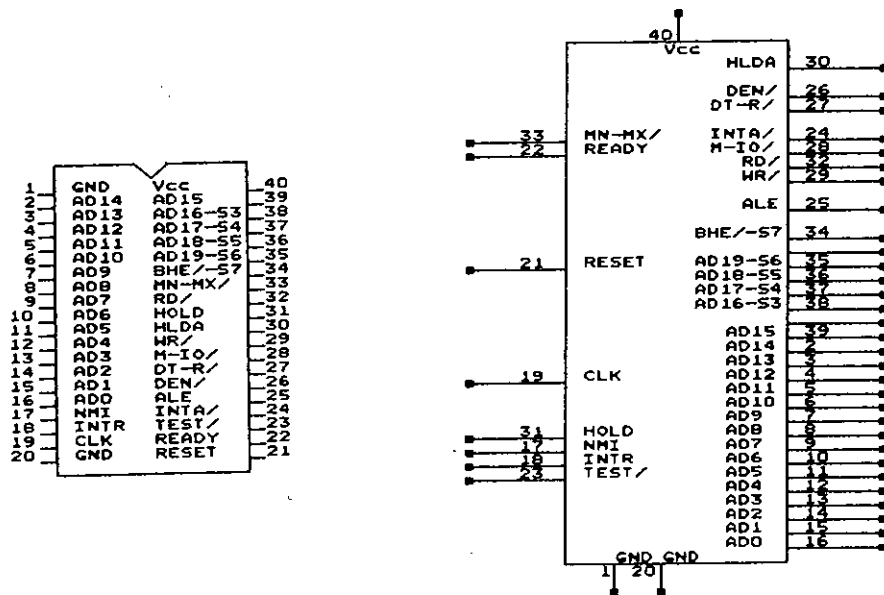


Fig-2.1(a) : Pin Assignment (Min Mode) Fig-2.1(b) : Bus Structured Pin Assignment (Min Mode)

Address Bus - 20 Lines

There are 20 address lines and are designated by A19 - A15 and A15 - A00 (embedded in AD15 - AD00) which are shown in Fig-2.1(a) and Fig-2.1(b). Address lines are used by the processor to select a particular memory location of a standard memory or an interface memory for data read/write operation. Since these lines are always originating from the processor and terminating to the memory devices, they are unidirectional output lines. The possible combinations of these lines range from 0000 0000 0000 0000 0000 (binary) to 1111 1111 1111 1111 1111 (binary).

In hex notation the range is from 00000H to FFFFFH. In decimal, the range is from 0 to 1,048,575 which equals to 1,048,576 combinations. This figure indicates that the 8086 CPU can individually select up to 1,048,576 memory locations. Taking 1024 as 1K and 8-bit size (1 byte) for each memory location as is the normal practice in computer literature, then the total addressable memory becomes 1 mega bytes or 1Mbyte. The address number for the data location of a standard memory or port memory is always treated as an unsigned binary number.

The lower 16 address lines (A15-A00) are used to assert the address of a memory location when that location is considered as a port space and is enabled by the LOW level of the M-I/O/ line. During port read/write operations, the upper four address lines viz., A19-A16 remain zeros. Thus the 8086 CPU can address up to 65,536 locations of port.

In 8086 CPU, the lower 16 address lines viz., AD15 - AD00 share common lines with the 16 data signals D15 - D00. This is possible due to the fact that a memory location is selected first and then the data read/write operation takes place. Stating another way, the address assertion and the data dump events never occur at the same time. Therefore, these two distinct information can be multiplexed over the same physical wires in time axis. And the composite signals being carried by these lines may be shown as AD15 - AD00.

Data Bus - 16 Lines multiplexed with A15-A00 Lines

There are 16 data lines D15-D00 (embedded in AD15-AD00) for the 8086 CPU and are time multiplexed with the lower 16 address lines A16-A00. The default read/write operation of the CPU is byte oriented. That is, while the CPU is reading data from memory on its own (while booting up after power up reset), it reads data byte by byte. Then what is the benefit of having 16 data lines? The answer is given saying that the CPU has got powerful instruction by virtue of which the programmer can instruct the CPU to do a word operation. This increases the speed of the system almost by twice. The lower data byte D7-D0 is called EVEN data while the upper data byte D15-D8 is called ODD data. The Data lines are bi-directional.

Read/Write Control Bus - 2 Lines

Fundamentally, only one line should be enough to complete read and write operation on memory chip. Single line can carry either HIGH or LOW logic value and this feature can be utilized to distinguish between read and write operations. In fact Motorola 6802 and 68000 microprocessors use only one line for both read/write operation. Whereas, Intel 8085,8086 use two lines for read/write operations. The choice of 1 or 2 lines is a matter of design convention and convenience.

To differentiate read/write with standard memory or port memory, use of a separate line makes sense. And it is M-I/O/ line.

Read Control Line: RD/ - 1 line

This is a single line originating from the CPU and going to the memory (both the standard and the port memory) devices. It carries active low signal to indicate the selected memory chip that the address information asserted to its inputs are now stable and it should now dump the desired data byte on the data bus. Figure -2.1(d) is referred for the timing relationship of this signal with other bus signals.

Write Control Line : WR/ - 1 line

This is also a single output line. This line is connected to both the standard memory and port memory. It carries active low signal to trigger the selected memory chip that the address and data asserted to its

inputs are now stable and it should absorb the data byte from the data bus. Figure - 2.1(d) shows the timing relationship of this signal with other bus signals.

Memory/Port Control Line: M-IO/ - 1 line

A microcomputer system can have many memory chips. Some of them are RAM and ROM and the rest are certainly port memories being connected with the users devices. By standard convention and practices, the RAM and ROMs are assigned 20-bit unsigned number for the addresses of their internal memory locations. And the input/output controllers (port memories or simply port) are assigned 16-bit/8-bit numbers for the internal memory locations. A single line designated as M-IO/ is used to distinguish between memory or port selection. A high signal on M-IO/ line will enable the memories and a low signal on M-IO/ line will enable the ports.

Memory Bank Control Lines

BHE/ (Byte High Enable) Signal

This signal is multiplexed with S7 signal on pin-34 of the CPU. This is a memory related signal and hence needs to be sampled by an optional latch (refer to section-4.2 schematic). The CPU asserts low level signal over this line when a user instruction requests a WORD oriented or ODD byte data read/write operation. The BHE/ signal has a close association with A00 address line as shown below:-

<i>BHE/</i>	<i>A00</i>	<i>Meaning</i>
0	0	: one word operation in one machine cycle when aligned with even address
0	1	: D15-D8 (upper byte = ODD byte) operation
1	0	: D7-D0 (lower byte = EVEN byte) operation

Multiplexed Bus Control Line

ALE Line (Address Latch Enable)

The 16-bit address information from the composite AD15 - AD00 signals is to be sampled and kept to the input address lines of the intended memory location until the data read/write operation is finished. To accomplish this, it is necessary to know the 'Time Point' at which the address signal is being asserted on the composite AD15 - AD00 lines. The 8086 CPU does indicate this 'Time Point' by generating a single pulse called ALE over pin-25. This ALE signal could be used to trigger optional D-type flip-flops (for example 74LS373 in section - 4.2) to latch the address information from their inputs to the output. Since ALE signal is generated only once during a read or write operation (refer to page-22), the output of the D-FFs which is in fact address information will remain constant at the input of the memory chip until the data movement operation is completed.

The above reasoning may also be applied for the upper 4 address lines viz., A19 - A16. The physical wires assigned to carry these signals might in fact carry composite signals like A19/S6 - A16/S3. S6 - S3 stand for Status signals. An optional latch (section - 4.2) can be triggered to sample the A19- A16 information from the composite signal and hold them at the output. Now the lines are free and they may be used to carry the processor Status signals S6-S3.

Processor Control Lines

RST (ReSeT) Line

It is an input line and carries a positive pulse to start the processor from the cold state. The minimum requirement for the width of this pulse is at least four clock periods. The reset pulse is usually supplied by an auxiliary clock chip of type 8284 (Sec-4.2) and is synchronized with processor clock. At the rising edge of the reset pulse, the CPU will terminate all operations if it was doing something. The CPU will

remain 'Idle' for the duration of the pulse. During the falling edge of the reset pulse, the CPU will undergo an internal reset sequence and will last for about 10 clock periods. During the reset sequence, the following events occur within the CPU:-

Data Segment register is initialized to 0000H	DS = 0000H
Stack Segment register is initialized to 0000H	SS = 0000H
Extra Segment register is initialized to 0000H	ES = 0000H
Code Segment register is initialized to FFFFH	CS = FFFFH
Instruction Pointer register is initialized to 00000H	IP = 0000H
Flag Register is initialized to 0000H	FR = 0000H

At the end of the reset sequence, the processor starts booting up from the absolute memory location CS:IP = FFFF:0000H

MN-MIX/ (MiNimum mode or MaXimum mode)

The meaning of the pin signals of the 8086 CPU are sufficiently different when it operates in minimum and maximum modes (Appendix B). Maximum mode allows connecting additional processors/coprocessors to realize multiprocessing environment. Whereas, minimum mode does not permit so. An IC having only 40 pins is not capable of furnishing all the functions required by min and max mode operations. To keep the number of pins at minimum while achieving the multiprocessing performance, a jumper pin (pin-33) has been added to the CPU. When MN-MX/ pin is strapped to +5V, the 8086 asserts signals at its various pins as required by the minimum mode operation (Section - 3.1). When the jumper pin is connected to ground potential, the CPU asserts maximum mode signals at its various pins.

RDY (ReaDY)

This is an input logic to the processor. The CPU functions normally as long as the logic level at this pin remains high. When the logic level of this pin goes low, the CPU starts inserting 'WAIT States' in its operating clock (page-22). It means that the clock period begins stretching and all bus activities gets frozen. Activities resume when the pin restores high logic level.

The addition of the RDY pin in the CPU allows the utilization of slow memory in the system design. Slow memories are cheap in price but the access time is greater than 200nS. There are in fact some commercial systems requiring huge on-board memories with moderate importance in speed. The RDY line is connected with the controller of the slow memory module. After the reception of the address, data and read signal from the buses, the controller pulls down the RDY line. After the known delay, the slow memory dumps the data on the bus. The controller also receives this information and it immediately releases the RDY line.

The RDY signal is usually provided by the clock chip generator (8284). The output of the slow memory controller is connected to the input of the clock chip (Section - 4.2). This is to allow synchronous operation of the RDY signal with the processor clock.

CLK

The CLK signal is the prime mover to the CPU for the generation of all the timing functions. The clock signal is usually provided by an auxiliary clock generator chip viz., 8284. The clock has a duty cycle of 33% for optimum operation. For details of the clock circuitry, refer see section - 4.2.

+5V Supply

To supply the power for the operation of the internal electronics. The tolerance of the voltage is $5V \pm 10\%$ for the 8086 and 5% tolerance for the 8086-1 and 8086-2. The maximum current that may be drawn from the power supply is 340 mA at room temperature.

0V Supply

To sink the +5V supply current. Two pins have been used to provide parallel paths for minimizing the noise.

Interrupt Control Lines

NMI (Non-Maskable Interrupt)

It is an asynchronous external input signal for interrupting the CPU. The active signal is rising edge which is sampled by the processor during the last clock cycle of the instruction being executed. The CPU is directed to an interrupt service routine (abbreviated as ISR) for interrupt 'type 2'. The starting address of the ISR is found by consulting a lookup table called Interrupt Vector Table (IVT) located at space 00000H - 003FFH of the main memory. This interrupt can not be disabled by setting the IF-bit of the flag register. This type of interrupt is designated as externally triggered internally vectored hardware interrupt. This input is usually terminated by a pull down resistor of 5Kohm.

INTR (INTeRrupt)

This is also an asynchronous external input to interrupt the CPU. It is level sensitive and is sampled by the CPU during the last clock cycle of the instruction being executed. This interrupt is usually funneled by an Interrupt Priority Controller chip like 8259. The type code for the interrupt is supplied by the interrupting device via 8259. The CPU is directed to an ISR consulting the IVT. This interrupt can be disabled by setting the IF-bit of the flag register. This type of interrupt is designated as externally triggered externally vectored hardware interrupt.

INTA/ (INTerrupt Acknowledge)

This is an active low signal, generated by the CPU in response to INTR signal. It is asserted to inform the interrupting device to dump the 8-bit interrupt type code on the data bus. There are two such pulses that are generated. This output is terminated by a pull up resistor of 5Kohm.

DMA Control Lines

HOLD

An asynchronous input signal asserted by DMA device. The signal must remain high until the DMA service is completed. The CPU samples this input during every machine cycle of an instruction. The input is terminated by a pull down resistor of 5Kohm.

HLDA (HoLD Acknowledge)

The CPU generates an active high signal at this pin in response to the HOLD input. This signal is propagated to the DMA device to indicate that the local bus has been isolated from the system bus (Section - 4.2). Now the system bus mastership may be owned by the DMA device for direct data transfer to the RAM bypassing the CPU. At the end of DMA action the HOLD line goes low and the CPU also pulls down the HLDA line. The local bus becomes associated with system bus and the CPU regains the bus mastership.

Interprocessor Communication Line

TEST Line

The CPU samples this input during every clock cycle and keeps working if logic low is detected. This line is usually connected with a math coprocessor called 'Floating Point Unit = FPU' in maximum mode operation (8087). The FPU asserts this pin high to inform the 8086 to stop executing its instructions and wait for some computational result to be supplied by the FPU. Refer to Figure- A.2 at page-162.

Utility Lines

DEN/ (Data buffer Enable)

The local bus of the CPU has a very limited current driving capability and is just good to drive one TTL load. If a user wishes to connect more loads, suitable data buffer (like 74LS245) are to be installed to protect the local bus from being damaged due to overloading. The asynchronous DEN/ (active low) signal is generated by the CPU may be used to enable such data buffers. For minimum systems, this line usually remains open. Refer Fig-2.1(c) for the application of this signal.

DT-R/

This alternate signal is generated by the CPU asynchronously to change the direction of the data buffer being enabled by DEN/ signal. The signal level becomes high if the instruction being executed refers to a write operation on external memory. The signal becomes active low if the instruction is involved in data read operation from the external memory. Please see Fig-2.1(c) below for the implementation of this line.

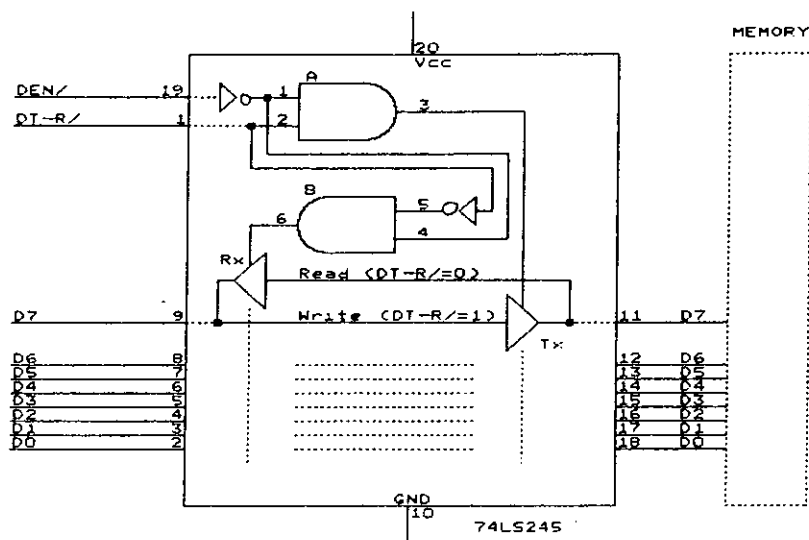


Fig - 2.1(c) : Implementing the DEN/ and DT-R/ Signals

Processor Status Lines

S3, S4, S5, S6 and S7 Lines

Signals S3, S4, S5, S6 and S7 appearing at the indicated pins of the Fig-2.1(b) are termed as Status lines. These signals become available just after the assertion of the A19-A16 address information. To grasp these status information, suitable electronic circuitry are to be employed. The meaning of these signals are as follows:

Status Signals		Meaning
S4	S3	
1	1	: The memory reference instructions are using Data Segment for read/write operations during the execution of the current instruction
1	0	: The memory reference instructions are using Code Segment for read/write operations during the execution of the current instruction. or : The CPU is accessing the Interrupt Vector Table for jumping to Interrupt Service Routine in response to an Interrupt.
0	1	: The memory reference instructions are using Stack Segment for data accessing.
0	0	: The memory reference instructions are using Extra (Alternate) Segment for read/write operations during the execution of the current instruction.
S5 = 1		: The IF-bit of the flag register is set.
S6 = 0		: always
S7 = ?		: Spare

S3 and S4 - these two lines indicate the status of the segment being used by the processor for accessing the data/code at the current instruction. These two lines can be used to access up to 4 Mbytes of memory (1 Mbyte per Segment) [1]. Refert to Fig - 2.1(e) for the hardware circuit to realize this concept.

Pin Loading Considerations

Sourcing

Every pin can source maximum 400µA current at minimum 2.4V. if the load increases, the logic level may decrease and the behavior of the CPU will become unpredictable.

Sinking

Every pin can sink about 2.5mA current at maximum 0.45V.

Figure-2.1(d) shows the activities on the 8086 bus during simple read/write operations. The first line show the clock waveform. One cycle of this clock is referred to as 'State' and is designated by the symbol 'T'. The group of states required for the completion of one read or write operation is called a 'Machine Cycle'. The total time needed by the CPU to complete the activities of one instruction is called an 'Instruction Cycle'.

To read data from a memory location, the CPU asserts the 20-bit address information on the bus. The ALE signal is generated to demultiplex the address information from the composite AD15 - AD0 signals. M-I/O/ signal is made 'High' to select memory device. After some delay, the RD/ signal is asserted to bring the data from the bus into the accumulator of the CPU. DT-R/ and DEN/ signals are generated to enable the data buffers if there is any.

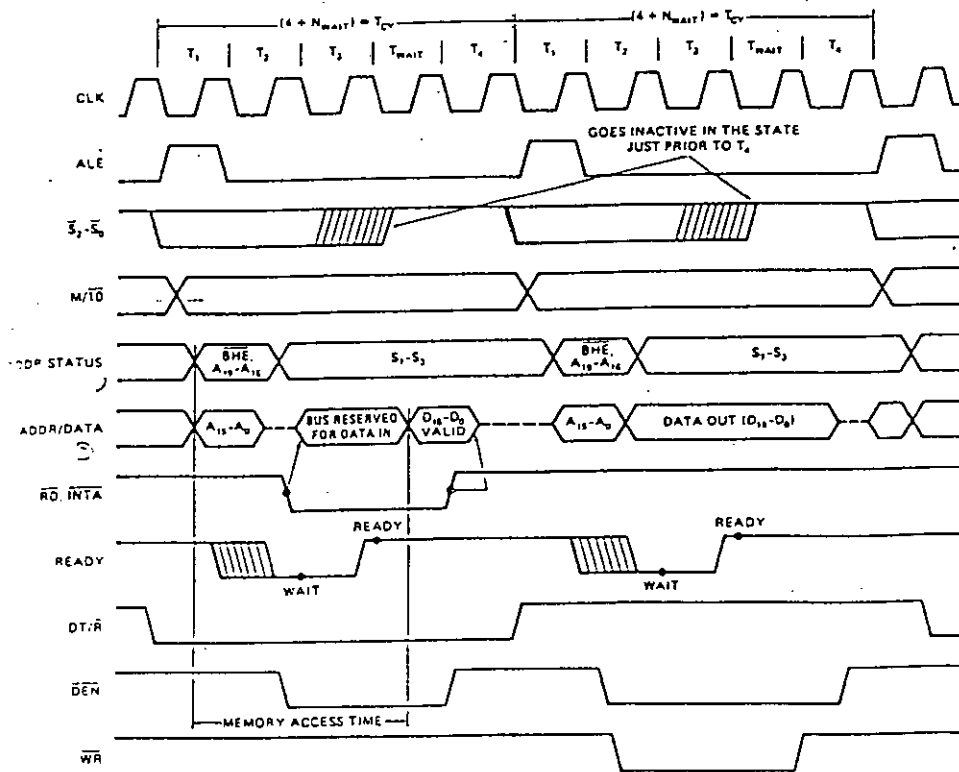


Fig - 2.1(d) : Basic 8086 System Timing in Minimum Mode (Courtesy Intel Corp.)

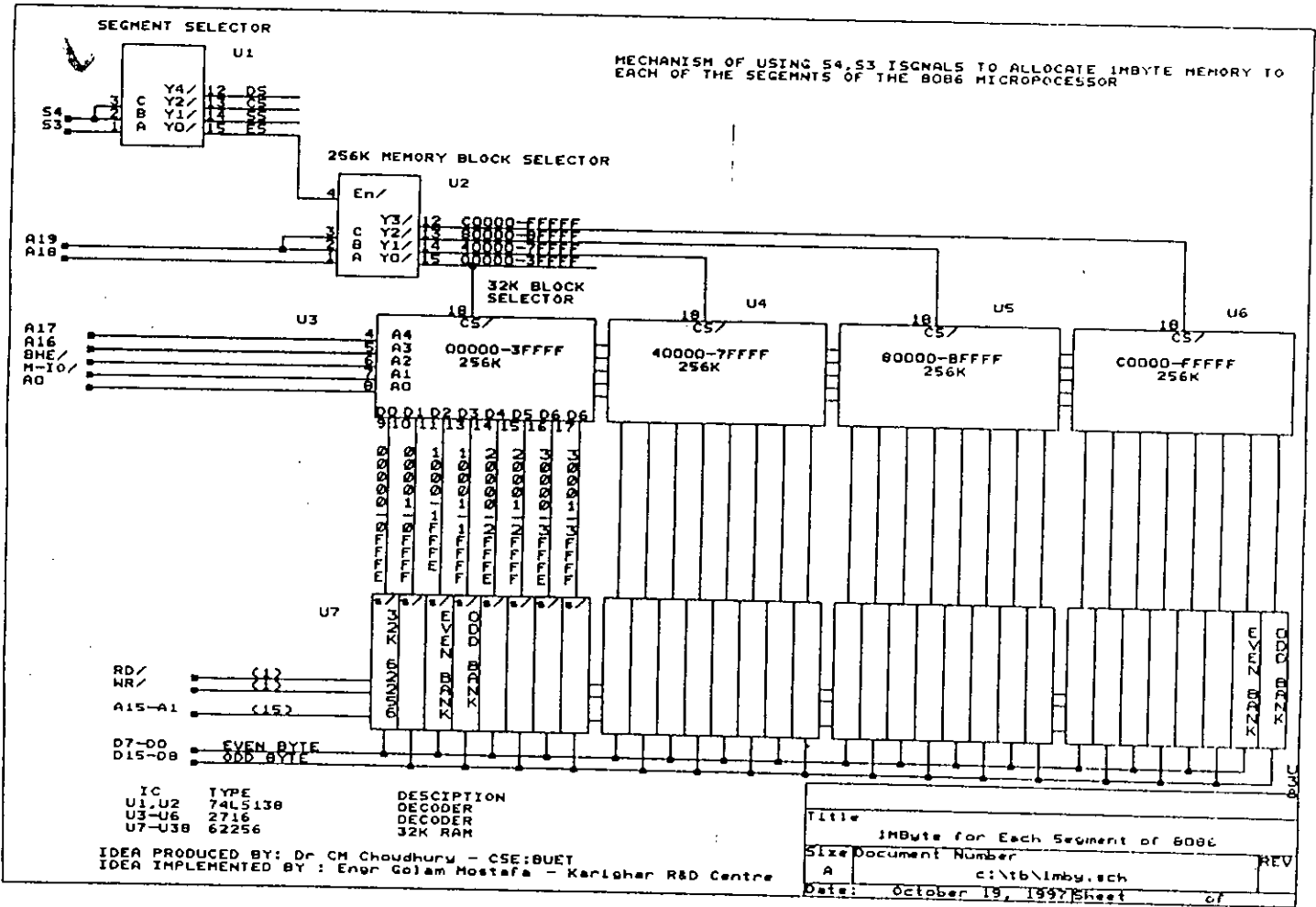


Fig - 2.1(e) : Schematic Diagram to Allocate 1MByte per Segment for 8086 using S3, S4 Signals

2.2 Internal Architecture

Figure - 2.2(a) is the block diagram of the internal architecture of 8086 microprocessor. The machine is primarily composed of two functional units viz., 'Bus Interface Unit (BIU)' and 'Execution Unit (EU)'. Each functional unit is also composed of many subunits.

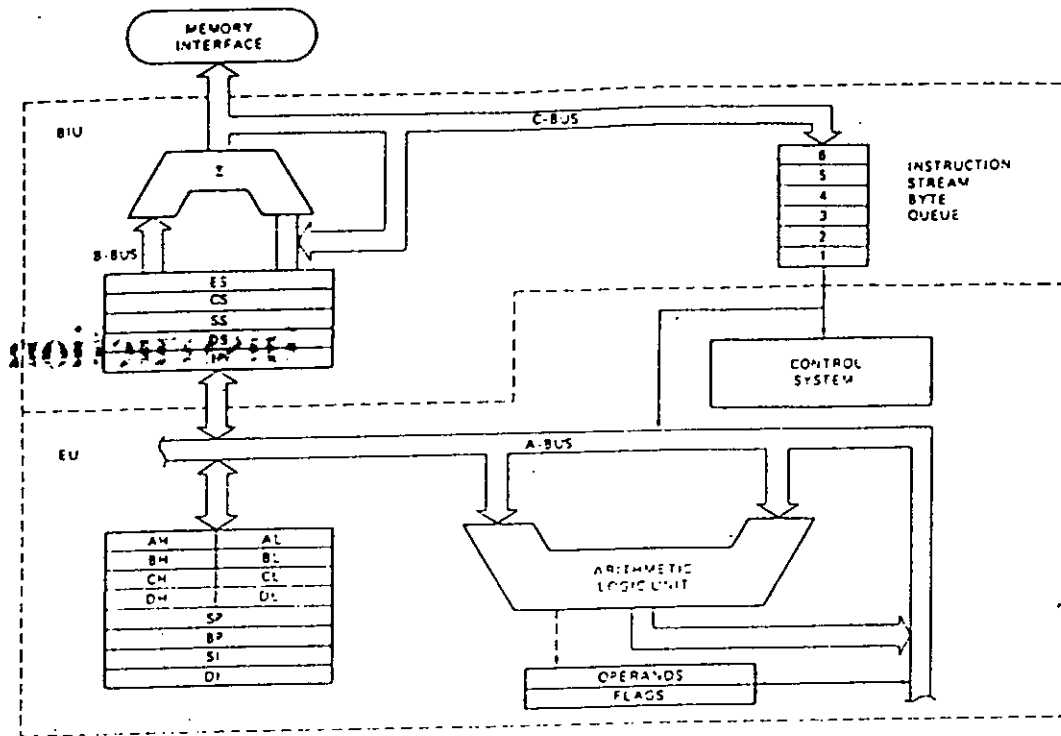


Fig - 2.2(a) : Functional Block Diagram

BIU : Bus Interface Unit

This unit is composed of the following subunits:-

Memory Interface Unit

This unit asserts the 20-bit physical address on the external bus to read/write data/code out of memory.

Instruction Stream Byte Queue

The instruction codes (program codes) goes to the 'Instruction Stream Byte Queue (ISBQ)' over the C-Bus. When the CPU is busy for calculating the address or internal processing, the bus is free and is used to read the instruction codes from the external memory and are stored in the ISBQ. Thus the CPU always

gets the instruction bytes from inside except during branch and jump. This increases the throughput of the system.

Segment and Instruction Pointer Registers

There are five registers. These are:-

ES : Extra Segment Register. It holds the upper 16-bit of the 20-bit base address of the extra segment.

CS : Code Segment Register. It holds the upper 16-bit of the 20-bit base address of the code segment.

SS : Stack Segment Register. It holds the upper 16-bit of the 20-bit base address of the stack segment

DS : Data Segment Register. It holds the upper 16-bit of the 20-bit base address of the data segment.

IP : Instruction Pointer Register. Its 16-bit content is added with the CS-base address to read code byte.

Address Computing Subunit

Figure - 2.2(b) is the expanded view of the address computing unit of the BIU unit. The chart shows all possible modes for calculating the 20-bit physical address of a memory operand. Examples :-

Single Index: `mov ax,[bx]` : default segment is DS
`mov ax,[bx+d8]` : default segment is DS
`mov ax,[bx+d16]` : default segment is DS
`mov ax,cs:[bx+d8]` : default segment is overridden CS

Double Index: `mov ax,[bx+si]` : default segment is DS
`mov ax,[bx+si+d8]` : default segment is DS
`mov ax,[bx+si+d16]` : default segment is DS
`mov ax,cs:[bx+si+d8]` : default segment is overridden by CS

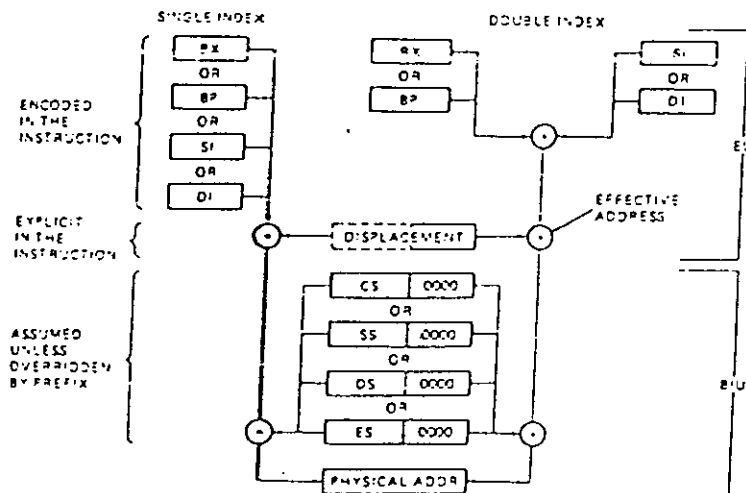


Fig - 2.2(b) : Address Computing Unit

EU = Execution Unit

This is composed of the following subunits:-

Working Register Bank

Figure - 2.2(c) shows the layout of the internal registers of the 8086 microprocessor. The programmer can access these registers except the CS and IP registers.

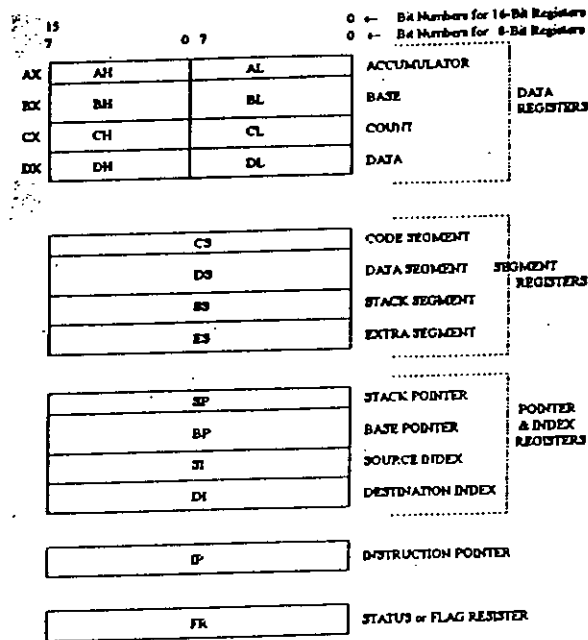


Fig - 2.2(c) : Internal Register Layout

Status/Flag Register

There is a 16-bit status register of which only 9 bits are active. This register reflects the operational status after the execution of each instruction. The flag bits are :-

- B0 - CF : Carry Flag - when a carry is generated after the addition of two operands
- B2 - PF : Parity Flag - number of 1s becomes even after operation
- B4 - AF : Auxiliary Flag - a carry is generated out of 4th bit during BCD operation.
- B6 - ZF : Zero Flag - the result due to operation is zero
- B7 - SF : Sign Flag - the MSB of the data is 1 due to the execution of an instruction.
- B8 - TF : Trap Flag - to allow the CPU to execute one instruction at a time.
- B9 - IF : Interrupt Flag - to enable/disable hardware interrupts viz., INTR.
- B10 - DF : Direction Flag - used to control the direction of string transfer.
- B11 - OF : Overflow Flag - when the result of an operation exceeds the capacity of the destin.

2.3 Instructions

The instructions of 8086 microprocessor are broadly classified into 6 groups. The mnemonics summary is given in Figure-2.3(a). Coding templates are given in Figures-2.3(b), 2.3(c) and 2.3(d).

DATA TRANSFER INSTRUCTIONS	ARITHMETIC INSTR.	PROCESSOR CONTROL
<p>General Purpose mov push pop xchg xlat</p> <p>Input/Output in out</p> <p>Address Object lea lds les</p> <p>Flag Transfer lahf pushf popf</p>	<p>Addition add adc inc aaa daa</p> <p>Subtraction sub sbb dec neg cmp aas das</p> <p>Multiplication mul imul aam</p> <p>Division div idiv aad cbw cbd</p>	<p>Flag Operations clc cmc stc cld std cli sti</p> <p>External Synchronizn. hlt wait esc lock</p> <p>No Operation nop</p>
<p>STRING INSTRUCTIONS movs rep cmps scas lods stos</p>		
<p>SHIFT/ROTATE/LOGICAL INSTRUC</p> <p>Logicals not and or xor test</p> <p>Shift shl/sal shr sar</p> <p>Rotate rol ror rcl rcr</p>	<p>PROGRAM TRANSFER INSTRUCTION</p> <p>Conditional Transfer je/jz jl/jnge jle/jng jb/jnae jbe/jna jp/jpe jo js jne/jnz jnl/jge jnl/jg jnb/jae jnbe/ja jnp/jpo jno jns</p>	<p>Unconditional Transfers call ret jmp</p> <p>Iteration Controls loop loope/loopz loopne/loopnz jcxz</p> <p>Interrupts int into iret</p>

Fig - 2.3 (a) : Instruction Mnemonics Summary

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued):					
JL/JL → Jump on equal	0 1 1 1 0 1 0 0 disp	7+m or 3	7+m or 3		16
JL/JNGE → Jump on least greater or equal	0 1 1 1 1 1 0 0 disp	7+m or 3	7+m or 3		16
JLE/JNG → Jump on less or equal greater	0 1 1 1 1 1 1 0 disp	7+m or 3	7+m or 3		16
JL/JNAE → Jump on less than or equal of least	0 1 1 1 0 0 1 0 disp	7+m or 3	7+m or 3		16
JLE/JNA → Jump on less or equal greater of least	0 1 1 1 0 1 1 0 disp	7+m or 3	7+m or 3		16
JPL/JPE → Jump on parity even	0 1 1 1 1 0 1 0 disp	7+m or 3	7+m or 3		16
JO → Jump on overflow	0 1 1 1 0 0 0 0 disp	7+m or 3	7+m or 3		16
JB → Jump on sign	0 1 1 1 1 0 0 0 disp	7+m or 3	7+m or 3		16
JNB/JNC → Jump on not overflow	0 1 1 1 0 1 0 1 disp	7+m or 3	7+m or 3		16
JNB/JOE → Jump on not overflow or equal	0 1 1 1 1 0 1 1 disp	7+m or 3	7+m or 3		16
JNB/JNO → Jump on not less or equal greater	0 1 1 1 1 1 1 1 disp	7+m or 3	7+m or 3		16
JNB/JAE → Jump on not less than or equal of least	0 1 1 1 0 0 1 1 disp	7+m or 3	7+m or 3		16
JNB/JAE → Jump on not less than or equal greater	0 1 1 1 0 1 1 1 disp	7+m or 3	7+m or 3		16
JNB/JPO → Jump on not parity odd	0 1 1 1 1 0 1 1 disp	7+m or 3	7+m or 3		16
JNO → Jump on not overflow	0 1 1 1 0 0 1 1 disp	7+m or 3	7+m or 3		16
JNS → Jump on not sign	0 1 1 1 1 0 0 1 disp	7+m or 3	7+m or 3		16
LOOP → Loop CX times	1 1 1 0 0 1 0 0 disp	8+m or 4	8+m or 4		16
LOOPE/LOOPNE → Loop while overflow	1 1 1 0 0 0 0 1 disp	8+m or 4	8+m or 4		16
LOOPE/LOOPNE → Loop while not overflow	1 1 1 0 0 0 0 0 disp	8+m or 4	8+m or 4		16
JCC → Jump on CX zero	1 1 1 0 0 0 1 1 disp	8+m or 4	8+m or 4		16
INTERRUPTS:					
INT → Interrupt	1 1 0 0 1 1 0 1 type	23+m		2,7,8	
Type specified	1 1 0 0 1 1 0 0	23+m		2,7,8	
Type 3	1 1 0 0 1 1 0 0	23+m		2,7,8	
INTO → Interrupt on overflow	1 1 0 0 1 1 1 0	23+m or 1	01% error	2,6,8	
Protected Mode Only:					
Via interrupt or trap gate to same privilege level		40+m		7,8,11,12,18	
Via interrupt or trap gate to different privilege level		78+m		7,8,11,12,18	
Via Task Gate		167+m		7,8,11,12,18	
IRRT → Interrupt return	1 1 0 0 1 1 1 1	17+m	31+m	2,4	8,9,11,12,15,18
Protected Mode Only:					
To different privilege level		55+m		8,9,11,12,15,18	
To different task (NT = 1)		167+m		8,9,11,12,18	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC → Clear carry	1 1 1 1 1 0 0 0	2	
CMC → Complement carry	1 1 1 1 0 1 0 1	2	
STC → Set carry	1 1 1 1 1 0 0 1	2	
CLD → Clear direction	1 1 1 1 1 1 0 0	2	
STD → Set direction	1 1 1 1 1 1 0 1	2	
CLI → Clear interrupt	1 1 1 1 1 0 1 0	2	
STI → Set interrupt	1 1 1 1 1 0 1 1	2	
HLT → Halt	1 1 1 1 0 1 0 0	2	
WAIT → Wait	1 0 0 1 1 0 1 1	6	if test = 0
LOCK → Bus lock prefix	1 1 1 1 0 0 0 0	2	
ESC → Processor Extension Escape	1 0 0 1 1 1 1 1 mod LLL r/m	6	(LLL are opcode for processor extensions)

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems

Footnotes

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

REG is assigned according to the following table:

16-BIT (w = 1)	8-BIT (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP*

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

3

OPERATING PROCEDURES

This chapter briefly describes the purpose of the trainer. The board layout is given to identify the physical locations of various components. The part list is also provided in terms of 'Commercial Type Number.'. The names of the signals at various pins of the edge connectors are also described. The detailed operating procedures of the trainer are described along with examples.

3.1 Purpose and the Component Layout

The 8086 trainer can be used to verify the functionality of a control program of any complexity level. The trainer also provides the facility of debugging a faulty program. The trainer has a hex key-pad to enter program/data codes. There are also command keys to instruct the trainer for executing an user program. Figure - 3.1 shows the 'Component Layout' diagram of the trainer.

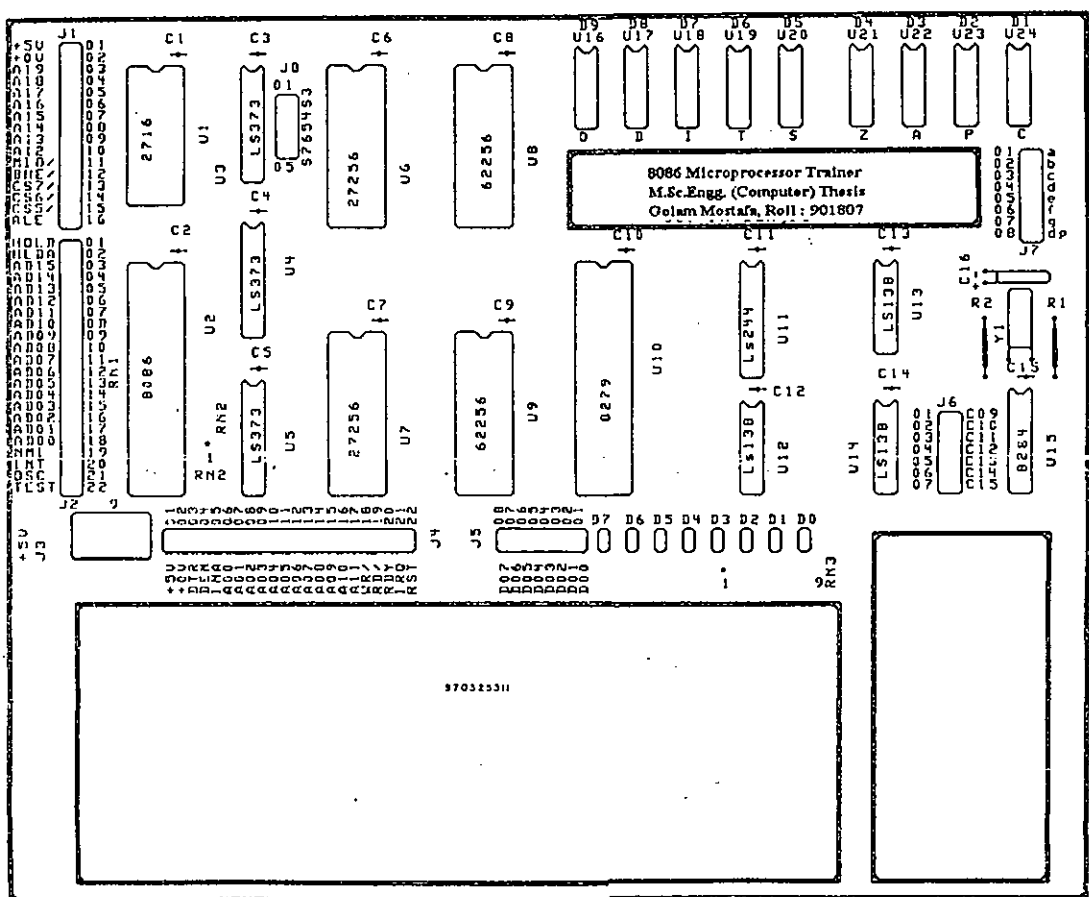


Fig - 3.1 : The Component Layout of the 8086 Trainer

3.2 Components Description

Circuit	Description	Type	Quantity
U1	EPROM	2716	1
U2	Microprocessor	8086	1
U3 - U5	Data Latch	74LS373	3
U6 - U7	EPROM	27256	2
U8 - U9	RAM	62256	2
U10	Controller	8279	1
U11	Data Buffer	74LS244	1
U12 - U14	Decoder	74LS138	3
U15	Clock Chip	8284	1
U16 - U24	7-Seg Display	Common Cath	9
C1 - C15	AC Capacitor	.1uF/63V	15
C16	DC capacitor	100uF/16V	1
R1 - R2	Resistor	1/4w, 5k	2
RN1 - RN2	Resistor Net	8x4k7	2
RN3	Resistor Net	8x560	1
Y1	Crystal	6.144 Mhz	1
LED0 - LED7	Light Emit	Normal	8
J01	Connector	Dual - 16 pin	1
J02	Connector	Dual - 22 pin	1
J03	Connector	+5V Conn.	1
J04	Connector	Dual - 21 pin	1
J05	Connector	Dual - 8 pin	1
J06	Connector	Dual - 7 pin	1
J07	Connector	Dual - 8 pin	1
J08	Connector	Dual - 5 pin	1
-	Bread Board	5''x2.5''	1
-	Keys	ON/OFF	18

3.3 Signal Signatures

Connector	Input/Output	Signal	Remarks/Circuit
J01 - 01	In/Out	+5V	-
J01 - 02	In/Out	+0V	-
J01 - 03	Out	A19	Unbuffered Address Line A19
J01 - 04	Out	A18	" A18
J01 - 05	Out	A17	" A17
J01 - 06	Out	A16	" A16
J01 - 07	Out	A15	" A15
J01 - 08	Out	A14	" A14
J01 - 09	Out	A13	" A13
J01 - 10	Out	A12	" A12
J01 - 11	Out	M-IO/	Unbuffered Control Line
J01 - 12	Out	BHE/	"
J01 - 13	Out	CS7/	Decoded Space (Even Port: 3000-3FFF)
J01 - 14	Out	CS6/	Decoded Space (Even Port: 2000-2FFF)
J01 - 15	Out	CS5/	Decoded Space (Even Port: 1000-1FFF)
J01 - 16	Out	ALE	Unbuffered Control Line
J02 - 01	In	HOLD	CPU Control
J02 - 02	Out	HLDA	Control
J02 - 03	In/Out	AD15	Composite A15 & D15
J02 - 04	In/Out	AD14	Composite A14 & D14
J02 - 05	In/Out	AD13	Composite A13 & D13
J02 - 06	In/Out	AD12	Composite A12 & D12
J02 - 07	In/Out	AD11	Composite A11 & D11
J02 - 08	In/Out	AD10	Composite A10 & D10
J02 - 09	In/Out	AD09	Composite A09 & D09
J02 - 10	In/Out	AD08	Composite A08 & D08
J02 - 11	In/Out	AD07	Composite A07 & D07
J02 - 12	In/Out	AD06	Composite A06 & D06
J02 - 13	In/Out	AD05	Composite A05 & D05
J02 - 14	In/Out	AD04	Composite A04 & D04
J02 - 15	In/Out	AD03	Composite A03 & D03
J02 - 16	In/Out	AD02	Composite A02 & D02
J02 - 17	In/Out	AD01	composite A01 & D01
J02 - 18	In/Out	AD00	Composite A00 & D00
J02 - 19	In	NMI	External Interrupt to MPU
J02 - 20	In	INT	External Interrupt to MPU
J02 - 21	Out	OSC	TTL Clock = 6.144 MHz
J02 - 22	Out	PCLK	TTL Clock, 1024 KHz
J03 - Centre	In	+5V	+5V Supply In
J03 - Outer	In	+0V	+0V Line
J04 - 01	In/Out	+5V	+5V Supply
J04 - 02	In/Out	+0V	+0V Line
J04 - 03	Out	DT - R/	Control Signal from MPU
J04 - 04	Out	DEN/	Control Signal from MPU

J04 - 05Out	INTA/	Control Signal from MPU
J04 - 06Out	A00	Unbuffered Address Line A00
J04 - 07Out	A01	" A01
J04 - 08Out	A02	" A02
J04 - 09Out	A03	" A03
J04 - 10Out	A04	" A04
J04 - 11Out	A05	" A05
J04 - 12Out	A06	" A06
J04 - 13Out	A07	" A07
J04 - 14Out	A08	" A08
J04 - 15Out	A09	" A09
J04 - 16Out	A10	" A10
J04 - 17Out	A11	" A11
J04 - 18Out	WR/	Write Control Signal from MPU
J04 - 19Out	RD/	Read Control Signal from MPU
J04 - 20In	RDY	Control Signal to 8284 = U15
J04 - 21Out	IRQ	Interrupt Signal from 8279 =U10
J05 - 01In	D07	Data Monitoring
J05 - 02In	D06	Data Monitoring
J05 - 03In	D05	Data Monitoring
J05 - 04In	D04	"
J05 - 05In	D03	"
J05 - 06In	D02	"
J05 - 07In	D01	"
J05 - 08In	D00	"
J06 - 01Out	C09	To Enable 8279's Digit - 09
J06 - 02Out	C10	" Digit - 10
J06 - 03Out	C11	" Digit - 11
J06 - 04Out	C12	" Digit - 12
J06 - 05Out	C13	" Digit - 13
J06 - 06Out	C14	" Digit - 14
J06 - 07Out	C15	" Digit - 15
J07 - 01Out	a	8279's multiplexed segment - a
J07 - 02Out	b	" - b
J07 - 03Out	c	" - c
J07 - 04Out	d	" - d
J07 - 05Out	e	" - e
J07 - 06Out	f	" - f
J07 - 07Out	g	" - g
J07 - 08Out	dp	" - dp
J08 - 01Out	S3	Status Signal from MPU
J08 - 02Out	S4	"
J08 - 03Out	S5	"
J08 - 04Out	S6	"
J08 - 05Out	S7	"

3.4 Keyboard Mnemonics and Meaning

The 8086 trainer has a hex key pad consisting of 18 keys. All the keys are of double functions. Figure - 3.4 shows the pictorial view of the keyboard template.

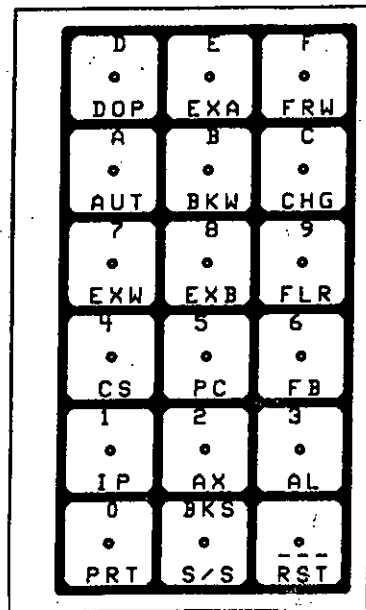


Fig - 3.4 : Keyboard Template

RST/ - (Reset and Start) - a hardware command to the CPU to start from the cold state.

E/EXA : E - data value in hex
EXA (EXAmination) - command to examine/edit memory contents.

A/AUT : A - data value A in hex
A (AUTo increment) - command to enter data into memory on auto incrementing the address field.

D/DOP : D - a data value D in hex.
DOP (DO a Program) - command to the CPU to execute a user's program.

5/PC : 5 - a data value 5 in hex
PC (Program Counter) - command to enter the starting address of the 1st instruction of a user's program to be single stepped. Also works as a home key to bring the display to show the address of the instruction to be single stepped when the display isn't showing so.

8/EXB : 8 - a data value 8 in hex.
EXB (EXAmination Byte data) - command to examine/edit the content of only one memory location - one byte.

- 7/EXW** : 7 - a data value in hex.
EXW (Examination Word-data) - command to examine/edit the contents of two consecutive memory locations.
- F/FRW** : F - a data value in hex.
FRW (FoRWard) - a command to examine the content of the next memory location or of the next Register or Port.
- B/BKW** : B - a data value B in hex.
BKW (BacKWARD) - command to examine the content of the previous memory location or Register or Port.
- C/CHG** : C - a data value in hex.
CHG (CHAnGe) - command to alter the contents of memory location or Register or Port.
- 0/PRT** : 0 - a data value in hex.
PRT (PoRT) - command to examine the content of a Port location (yet to be implemented).
- 9/FLR** : 9 - a data value 9 in hex.
FLR (Flag Register) - command to examine the content of the flag register in hex.
- 6/FB** : 6 - a data value in hex.
FB (Flag Bit) - command to examine the content of the flag register in bit form.
- 4/CS** : 4 - a data value 4 in hex.
CS (Code Segment) - command to examine the content of the code segment register.
- 1/IP** : 1 - a data value 1 in hex.
IP (Instruction Pointer) - command to examine the content of the Instruction Pointer.
Please use FRW key to examine the contents of the registers DI,SI,SP,BP.
- 2/AX** : 2 - a data value 2 in hex.
AX - command to examine the content of register AX. FRW key should be used to examine the contents of BX,CX,DX registers.
- 3/AL** : 3 - a data value 3 in hex.
AL - command to examine the content of register AL. FRW key should be used to examine the contents of registers AH,BL,.BH,CL,CH,DL,DH.
- BKS/S-S** : BKS (BackSpace) - command to correct typing mistakes.
S-S (Single Step) - command to execute one instruction at a time.

3.5 Program Codes/Data Loading into Memory

Byte Data Entry/Change with Manual Increment of the Address Filed

Sample Program:

(When executed, this program will display the message dO)

```

00500 -   B0 D0       : mov  al,D0h
00502 -   88 47 4E   : mov  BYTE PTR [bx+4Eh],al
00505 -   9A 7C F4 00 F0: call SUR#8 (F000:F47C)
0050A -   C7 47 44 00 00 : mov  WORD PTR [bx+44h],0000h
0050F -   C7 47 46 00 00 : mov  WORD PTR [bx+46h],0000h
00514 -   C7 47 48 00 00 : mov  WORD PTR [bx+48h],0000h
00519 -   C7 47 4A 00 00 : mov  WORD PTR [bx+4Ah],0000h
0051E -   9A B6 FF 00 F0: call SUR#3 (F000:FFB6)
00523 -   EA 23 05 00 00 : jmp  0000:0523
00509 -

```

Procedures of Loading the Codes into memory

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press RST/	8086 CPU	the trainer is ready
02.	press EXA	___ _ Ad	the address field is opened
03.	press 0,0,5	0 0 5 _ _ Ad	the data is getting entered
04.	press 4	0 0 5 4 _ Ad	wrong digit entry
05.	press BKS	0 0 5 _ _ Ad	corrects typing mistake at address field
06.	press 0,0	0 0 5 0 0 Ad	20-bit address is entered
07.	press EXB	0 0 5 0 0 X X	X X indicates random value
08.	press CHG	0 0 5 0 0 _ _	data field is opened to enter new value
09.	press 6	0 0 5 0 0 6 _	wrong digit entry
10.	press BKS	0 0 5 0 0 _ _	corrects typing mistake at the data field
11.	press D,0	0 0 5 0 0 d 0	data byte is entered
12.	press FRW	0 0 5 0 1 X X	next memory location
13.	press BKW	0 0 5 0 0 d 0	previous memory location & content
14.	press FRW,CHG and finish entering the remaining data bytes.		

Word Data Entry/Change with manual Increment of the Address Field

Procedures for loading the program/data codes of the sample program of Section-3.5 of this page.

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press RST	8 0 8 6 C P U	the trainer is ready
02.	press EXA	___ _ Ad	address field is opened
03.	press 0,0,5,0,0	0 0 5 0 0 Ad	20-bit address is entered
04.	press EXW	0 0 5 0 0 X X X X	16-bit random value of two locations.
05.	press CHG	0 0 5 0 0 _ _ _ data	is opened to receive new data
06.	press B,0,d,0	0 0 5 0 0 B d 0 0	16-bit data is entered
07.	press FRW	0 0 5 0 2 X X X X	next word location
08.	press CHG,FRW and finish entering the remaining data.		

Byte Data Entry with Auto Increment of the Address Filed

Procedures to load the codes of the sample program of Section - 3.5 of page-27

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press RST/	8 0 8 6 C P U	the trainer is ready
02.	press AUT	_____ Ad	address field is opened
03.	press 0.0.5.0.0	0 0 5 0 0 _ _	data field is opened without CHG
04.	press B	0 0 5 0 0 b _	1 digit is entered
05.	press 0	0 0 5 0 1 _ _	data entry done = updated and the next memory location is automatically opened.
06.	finish entering the remaining data.		

3.6 Program Execution

It is assumed that the user has finished entering the data/code bytes of the sample program of Section-3.5 into the trainer. Now, to execute the program one has to enter the starting address of the program which is 0 0 5 0 0.

Procedures:

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press RST/	8 0 8 6 C P U	the trainer is ready
02.	press DOP	_____ d o	address field is opened
03.	press 0,0,5,0,0	r u n	program is running

Note that the message 'r u n' may not be visible due to the very little execution time of the program. If you want to see the static message 'r u n', please terminate the program into a loop.

3.7 Program Debugging/Single Stepping

The procedures outlined below will show the ways of executing one instruction at a time. After the instruction has executed, the user may examine the registers and the port contents or the memory contents to check the correctness of the instruction.

Say, we wish to single step the sample program of Section- 3.5 at page-27.

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press RST/	8 0 8 6 C P U	the trainer is ready
02.	press PC	_____ P C	enter address of the 1st instruction
03.	press 0,0,5,0 0	0 0 5 0 0 b 0	display shows 1st instruction to be executed.
04.	press S/S	0 0 5 0 2 8 8	2nd instruction ready for execution
05.	and so on.....		

Now, examine/change memory contents if required using the FRW,BKW and CHG commands. It is always recommended to press the PC key to bring the display to home position to show the starting address of the instruction that is to be executed.

Exam/Edit AX,BX,CX,Dx Registers

It is recommended to bring the trainer into single stepping mode using the procedures of Section-3.7 of page-28. CHG,FRW,BKS commands are valid. Now, do as follows:-

<i>Step</i>	<i>Action</i>	<i>Display</i>	<i>Remarks</i>
01.	press AX	AX X X X X	shows AX's content
02.	press CHG	A X _ _ _ _	enter new value
03.	press 1,2,3,4	A X 1 2 3 4	new value for AX register is updated
04.	press FRW	BX X X X X	BX register's content
05.	press CHG	BX X X X X	can't and shouldn't be changed Why? It is because the BX register is used as a pointer by the operating system while implementing the various routines of the Single Stepping mechanism. If the value of bx-register is changed, the Single Step routine will not work at all. And even the CPU might crash. Try to change..!
06.	use FRW key to check and edit the remaining registers.		
07.	at the end of the register exam/edit, please press the PC key.		

Exam/Edit AL,AH,BL,BH,CL,CH,DL,DH Registers

The procedures are similar to examining/editing AX,BX,CX,DX registers. WE will notice that the contents of BL,BH are not changeable. FRW,BKS and CHG commands are active.

Exam/Edit CS,DS,ES and SS Registers

Similar procedures as above. There is no provision to change the contents of these registers. FRW command is active.

Exam/Edit IP,DI,SI,SP and BP Contents

Similar procedures as described for other registers. However, the content of SP register can not be changed. FRW,CHG and BKS commands are active.

Examining Flag Register

To examine flag register contents in hex form, please press FLR key. To examine the content in bit form., please press FB key. The contents can not be changed.

Exam/Edit Port Contents

To be implemented in future. Please see section 5.4.10.

3.8 Example Programs

A: Adding two unsigned 8-bit hex numbers.

Entering and Executing the following program at location 05010h will give the above output. The data values are to be deposited at the indicated memory locations using the EXA command. The result will be displayed at D2D1 positions of the display window of the trainer.

```

05006 - 23                                ; 1st data byte
05007 - 75                                ; 2nd data byte
05008 - 98                                ; expected result after addition
;Program Codes:
05010 - BB 00 50                          ; initialize local pointer
05013 - B8 00 00                          ; data to set DS=0000h
05016 - 8E D8                            ; DS=0000h
05018 - 8B 47 06                          ; getting the data from memory, ax.= 7523h
0501B - 02 C4                            ; adding two numbers, al=result
0501D - 88 47 08                          ; result is stored temporarily at 05008h

```

;outputting result in the display

```

05020 - BB 00 04                          ; getting back the value of bx
05023 - 88 47 4E                          ; putting the result in T2
05026 - 9A 7C F4 00 F0                    ; xferring T2 into T1
0502B - C7 47 44 00 00                    ; blanking D4D3
05030 - C7 47 46 00 00                    ; blanking at D6D5
05035 - C7 47 48 00 00                    ; blanking at D8D7
0503A - C7 47 4A 00 00                    ; blanking at D9
0503F - 9A B6 FF 00 F0                    ; xferring T1 into 8279
05044 - EA 44 50 00 00                    ; loop here

```

B: Expanding the display of the trainer

The 8279 chip of the trainer has been initialized to handle 16 display devices. There are only 11 display devices installed in the trainer. This example (Fig - 3.8) shows the technique of adding extra display devices (maximum 4) without the need of 'additional electronics like, display controller, decoder and etc.) devices

01. Let us build the circuit as per Fig-3.8 on the breadboard using hook up wires.

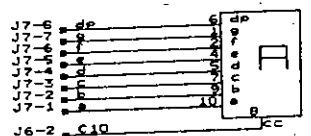


Fig - 3.8 : Installing Additional 7-segment Digit with the Trainer Display

02. Enter and execute the following program.

03. The character 'A' should be seen on the display of the breadboard.

Program Codes:

```

07000 - BA 02 00                          ; pointing at the control register of the 8279
07003 - B0 8A                            ; data for the position of the new display
07005 - EE                                ; position is set
07006 - BA 00 00                          ; point at the data register of the 8279
07009 - B0 77                            ; data for character 'A'
0700B - EE                                ; character is sent
0700C - EA 0C 70 00 00                    ; jmp 0000:700C ; loop

```

C: Demonstration of a Recursive Procedure by calculating the factorial of a number (upto 5 decimal).

The result of the factorial will be in hex. The value will be displayed at positions D2D1 of the trainer. The remaining digit positions of the display window will remain blank.

;Data Value

(00475)(00474) = xxxx (upto 0005) is to be entered first by the user

;Execution starts

```

01600 - BC FE FF      : mov  sp,0FFFEh          ;known TOP of STACK, good programming
01603 - BB 00 04      : mov  bx,0400h          ;bx will work as a pointer
01606 - 90 90 90      : nops                    ;
01609 - 81 EC 04 00    : sub  sp,0004h          ;keeping four RAM locations to hold X!
060F - 8B 47 74      : mov  ax,[bx+74h]       ;number is loaded to ax register
0612 - 50             : push ax                ; 1st factor X is saved onto stack
0613 - 9A 66 16 00 00 : call #FACTO (0000:1660) ; get next factor if there is any
0618 - the factorial of the contents of memory locations (00475)(00474) is at memory locations (0FFFD
      0FFFA). Get the results from these locations and display it at D4-D1 positions of the MicroTalk-8086
      trainer. D9-D5 positions will remain blank.

```

```

01618 - BB 00 04      : mov  bx,0400h          ;
0161B - BD FA FF      : mov  bp,0FFFAh        ;
0161E - 90 90         : nops                    ;
01620 - 8B 46 00      : mov  ax,[bp+00h]       ; get X! from (0FFFB)(0FFFA) into ax
01623 - 89 47 4E      : mov  WORD PTR [bx+4E],ax ; put into T3 of the reserved RAM
01626 - 8B 46 02      : mov  ax,[bp+02h]       ; get X! from (0FFFD)(0FFFC)
01629 - 89 47 4F      : mov  WORD PTR [bx+4Fh],ax
0162C - 9A 7C F4 00 F0 : call SUR#8(F000:F47C)  ;to convert packed hex to cc-codes
01631 - C7 47 44 00 00 : mov  WORD PTR [bx+44h],0000h ; blanking D4D3
01636 - C7 47 46 00 00 : mov  WORD PTR [bx+46h],0000h ; blanking D6D5
0163B - C7 47 48 00 00 : mov  WORD PTR [bx+48h],0000h ; blanking D8D7
01640 - C7 47 4A 00 00 : mov  WORD PTR [bx+4Ah],0000h ; blanking D9
01645 - 9A B6 FF 00 F0 : call SUR #3(F000:FFB6) ;to transfer cc-code to 8279 of the trainer
0164A - EA 4A 16 00 00 : jmp  #HERE (0000:164A) ;loop

```

#FACTO Subroutine

```

01660 - 8B EC         : mov  bp,sp              ;programmer can not use sp as a pointer
01662 - 8B 46 04      : mov  ax,[bp+04h]        ; get the number X
01665 - 3D 01 00      : cmp  ax,0001h           ; check if X=1
01668 - 75 0F         : jne  #DETER (0000:1679) ; X>1, determine (X-1)
0166A - C7 46 06 01 00 : mov  [bp+06h],0001h     ; X=1, X! = 1
0166F - C7 46 08 00 00 : mov  [bp+08h],0000h     ;
01674 - EA 96 16 00 F0 : jmp  #EXIT (0000:1696)  ; factorial process is end.

```

#DETER

```

01679 - 81 EC 04 00    : sub  sp,0004h
0167D - 48             : dec  ax
0167E - 50             : push ax
0167F - 9A 60 16 00 00 : call #FACTO (0000:1660)
01684 - 8B Ec         : mov  bp,sp
01686 - 8B 46 02      : mov  ax,[bp+02h]
01689 - F7 66 0A      : mul  WORD PTR [bp+0Ah]
0168C - 89 46 0C      : mov  [bp+0Ch],ax
0168F - 89 46 0E      : mov  [bp+0Eh],ax
01692 - 81 C4 06 00    : add  sp,0006h
01696 - CB           : ret

```

4

**HARDWARE
DESIGN**

This section briefly describes the art of drawing the block diagram of a microcomputer system and also the procedure of reading it to retrieve the design information.

4.1 Hardware Block Diagram

A hardware block diagram, when properly drawn contains almost all information regarding the system design. The block diagram for the 8086 trainer is given in Figure-4.1.

The study of the block diagram starts from the microprocessor. The microprocessor unit (or the module) can be detected as the one from which the address bus has originated. According to this formula, the module M7 is the microprocessor (MPU or CPU). This is Intel's high performance 8086 microprocessor.

The next step is to find the memory devices laying around the CPU and classify them as memory or ports. The devices which are connected with the CPU by the Address, Data and Control busses are the memory/port devices and these are M0, M1, M2, M3 and M4. Obviously, M0-M3 are memory devices. M4 is a port memory because it is connected with users devices like keyboard and display. However, a port memory like M4 can also be called a standard memory if it is accessed by 20-bit physical address.

The memory/port decoder is the the module M5 because the chip select lines (CS0/ - CS4/) of all the memory devices are connected to the outputs of this module. The diagram also indicates that the decoder module has been implemented using a 2716 EPROM (for details refer to Section-4.3).

Modules M2 and M3 are the ROMs because they have accepted only the RD/ signals from the control bus. M2 is connected with lower data bytes (D7-D0) and M3 is connected with the upper data byte (D15-D8). M0 and M1 are RAMs due to their connectivity with the RD/ and WR/ signals. M0 is communicating with the D7-D0 lines and M1 is doing with D15-D8 lines.

What is the function of the module M6?

The memory devices are connected to the CPU address bus via the output lines of module M6. The CPU shares common wires for the lower 16 address lines (A15-A00) with the 16-bit data lines (D15-D00). The module M6 separates the address information from the composite AD15-AD00 signals. It also separates the A19-A16 bits from the status signals S3-S6. This signal separation is done at the active high level of the ALE signal.. The BHE/ signal also gets separated from the status signal S7.

Function of M12

The heart-bit of the CPU is the clock signal that is generated by the module M12, utilizing an IC of type 8284. The frequency of the clock signal is equal to the 1/3rd of the crystal frequency connected to the 8284. The reset signal is also conditioned by the 8284.

Keyboard and Display:

A 18-key hexadecimal keypad is interfaced to the CPU via M4, which is designed by using an 8279. Three of the scan lines of the 8279 are decoded by the module M9 to generate walking 1's signals over the column lines of the keyboard. The rows of the key pad are connected to the 8279. The scan code generated by each pressed-down key is a function of its position in the matrix.

The display unit is of multiplexed type, 9-digit common cathode type. Scan lines are provided by M8 after decoding S3-S0 lines. The users' data bits are available at B7-B0 lines of the 8279.

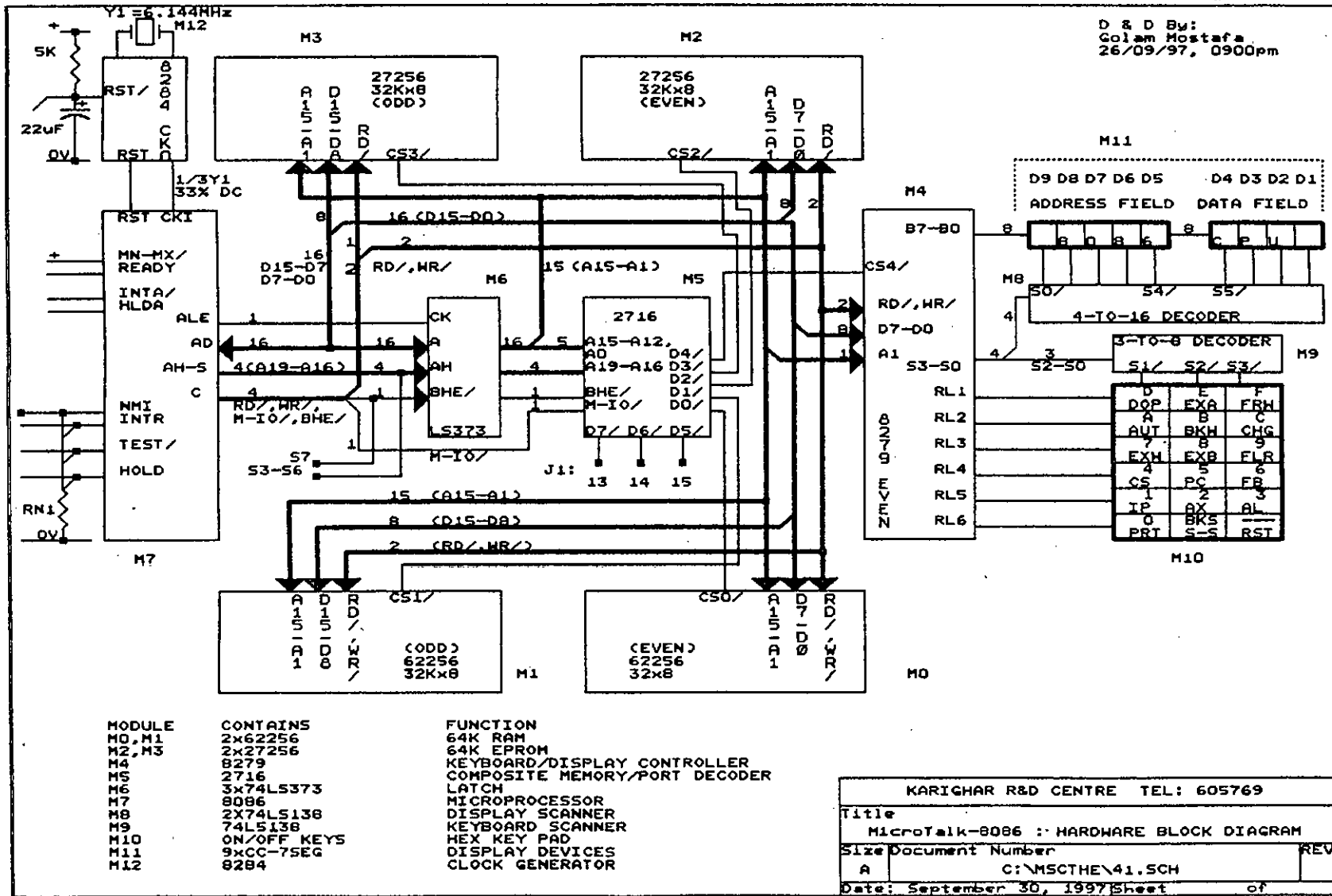


Fig-4.1 : Hardware Block Diagram

4.2 CPU Subsystem

Let us now refer to diagram of the CPU subsystem in Figure-4.2.

U2 is the Microprocessor Unit and is configured to work in its minimum mode having a jumper between the MN-MX/ pin and +5V. This mode does not allow a 2nd co-processor such as 8087 (please see Appendix-B) math co-processor to work in parallel with the main processor.

U15 is the clock generator. It generates a clock frequency of 1/3rd of the crystal frequency. The duty cycle is 33%. The reason for choosing a 6.144mhz crystal is to derive an auxiliary clock suitable for serial interface (refer to Section-6.9.2).

U3,U4 and U5 are the demultiplexers. U3 latches at its outputs the signals A16-A19 and BHE/ from the composite signals A16/S3 - A19/S6 and BHE/-S7. U4 and U5 are for A0-A7 and A8-A15 signals respectively. The signals are latched at the respective outputs by ALE signal asserted by the CPU at the beginning of the machine cycle [please see Figure-2.1(d)] for the timing diagram).

Read/write lines are terminated by pull up resistors to meet the timing specifications prescribed by Intel. INTA/ signal being an open collector signal must also have a pull up termination.

Pull down resistor network RN1 ensures that the corresponding signal pin remains very close to the ground potential when there is no active signal at the input.

NMI, INTR are the two interrupt input lines of the 8086 CPU. HOLD and HLDA are the DMA control lines. TEST/ input is used when there is a co-processor in the system and the 8086 is working in its maximum mode.

There are two pins viz., pin-20 and pin-1 which are ground.. This is to distribute the ground path in order to reduce noise.

The address, data and control lines are available at the edge connectors for interfacing experiments. However, it is to be remembered that the bus lines of the 8086 can drive one TTL logic. Therefore, if there is a need to drive more loads, suitable buffers have to be used for the data, address and control lines. The DEN/ and DT-R/ lines may be used to activate the bi-directional data buffers like 74LS245 (refer to Section-2.1).

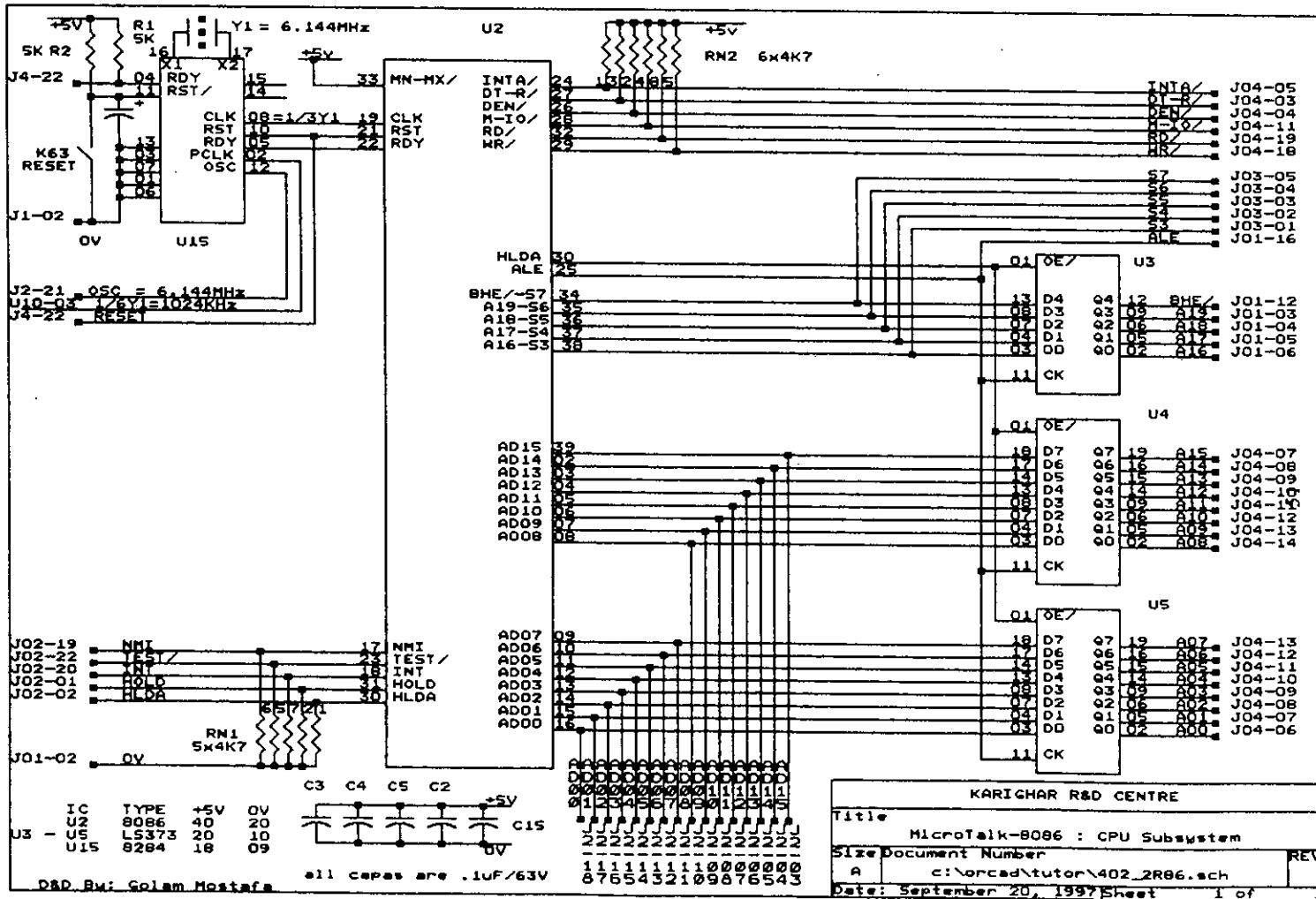


Fig - 4.2 : CPU Subsystem Schematic

4.3 Memory/Port Decoding Subsystem

Reference is made to circuit diagram in Figure-4.3 to study the following:-

The design requirement demands that:-

01. For all the EVEN addresses ranging from 00000,00002,...,0FFFC,0FFFE; the pin-9 output of the decoder should go low. The other 7-outputs of the decoder must remain high. This is the way how a decoder circuit helps the CPU to accomplish a conflict free sequential read/write operations with all the available memory/port chips in the system.
02. The same reasoning is applicable for the other memory/port devices while conducting byte oriented operations.
03. To accomplish word oriented operation starting from an EVEN address (00000,00002,...,0FFFE), the outputs of pin-9 and pin-10 of the decoder must remain low simultaneously so that both the U9 and U8 memory chips get selected at the same time. The other outputs of the decoder should remain high.
04. The same reasoning is applicable for the other memory/port devices while conducting word oriented operations.

Design Parameters Tabulation:

Memory Devices

IC	Type	Capacity	RAM/EPROM Bank	Space Allocated	Selected By
U9	62256	32 KBytes RAM	EVEN	00000,00002,...,0FFFC,0FFFE	U1-9
U8	62256	32 KBytes RAM	ODD	00001,00003,...,0FFFD,0FFFF	U1-10
U7	27256	32 KBytes EPROM	EVEN	F0000,F0002,...,FFFEC,FFFFE	U1-11
U6	27256	32 KBytes EPROM	ODD	F0001,F0003,...,FFFED,FFFFF	U1-13

Port Devices:

IC	Type	Capacity	RAM/EPROM Bank	Space Allocated	Selected By
U10	8279	2 Bytes	-----	EVEN 0000,0002,...,0FFC,0FFE	U1-14
----	----	32 KBytes	-----	EVEN 1000,1002,...,1FFC,1FFE	U1-15
----	----	32 KBytes	-----	EVEN 2000,2002,...,2FFC,2FFE	U1-16
----	----	32 KBytes	-----	EVEN 3000,3002,...,3FFC,3FFE	U1-17

The decoder is designed using an EPROM of the type 2716 to implement the above requirements. The explanation of the decoder Truth Table as indicated in Fig-4.3 is given below.

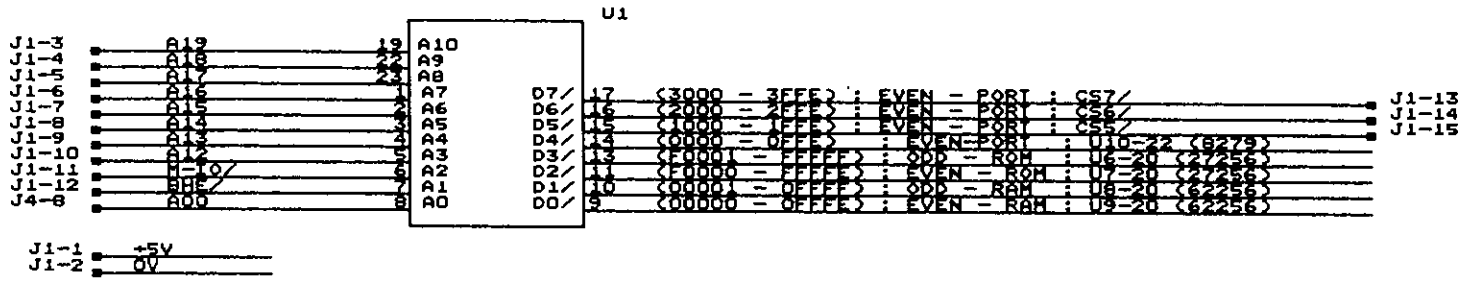
Asserted Memory Address	Locations of the 2716 EPROM	Data Fused	Active Low Output
00000,00002,...,0FFFC,0FFFE	006,00E,016,01E,...,076,07E	1111 1110 = FE	D0 (Pin-9)
00001,00003,...,0FFFD,0FFFF	005,00D,015,01D,...,075,07D	1111 1101 = FD	D1 (Pin-10)
00000-00001,...,0FFFE-0FFFF	004,00C,014,01C,...,074,07C	1111 1100 = FC	D0,D1 (Pin-9,10)
F0000,F0002,...,FFFEC,FFFFE	786,78E,796,79E,...,7F6,7FE	1111 1011 = FB	D2 (Pin-11)
F0001,F0003,...,FFFED,FFFFF	785,78D,795,79D,...,7F5,7FD	1111 0111 = F7D3	(Pin-13)
F0000-F0001,...,FFFEE-FFFFF	784,78C,794,79C,...,7F4,7FC	1111 0011 = F3D2,D3	(Pin-11,13)
Asserted Port Address	Locations of the 2716 EPROM	Data Fused	Active Low Output
0000,0002,...,0FFC,0FFE	002	1110 1111 = EF	D4 (Pin-14)
1000,1002,...,1FFC,1FFE	00A	1101 1111 = DF	D5 (Pin-15)
2000,2002,...,2FFC,2FFE	012	1011 1111 = BF	D6 (Pin-16)
3000,3002,...,3FFC,3FFE	01A	0111 1111 = 7FD7	(Pin-17)

Advantage of ROM Based Decoder:

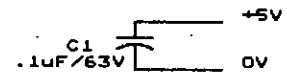
01. Saves many discrete ICs and their interconnection. Hence, a reliable circuit.
02. It is dynamic in the sense that new decoded output lines can be derived without changing the component. Just fusing new data in the ROM locations will yield new decoded lines.

MEMORY/PORT COMPOSITE (EPROM BASED COMPLEX) DECODER

2716 (U1) SIGNALS										MEMORY/PORT		U1'S PARAMETER			SELECT DATA		X86 M/C
8086 SIGNALS										RANGE		CONT	ACT	IC	DATA		
U1- M0	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00	B BZ	ENT	LOW			
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	00	U5	BYTE
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	01	U5	UB
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	02	U7	U6
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	03	U10	UB
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	04	U5	UB
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	05	U5	UB
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	06	U5	UB
0	0	0	0	0	0	0	0	0	0	0	0	0000	0000	00	07	U5	UB



IC U1 TYPE 2716 +5V 24 0V 12 LOGIC-L 18,20 LOGIC-H 21



KARIGHAR R&D CENTRE	
Title MicroTalk-8086: MEMORY/PORT DECODER	
Size Document Number A	REV C:\nm86II\403_in86.sch
Date: October 23, 1997	Sheet of

Fig - 4.3 : Memory/Port Decoding Subsystem Schematic

4.4 Memory Subsystem

Let us refer to circuit diagram of Figure-4.4. In 8086 system, the memories are arranged as EVEN and ODD banks. Sometimes the EVEN bank is termed as Lower bank and the ODD bank is termed as Upper bank. In this arrangement, the EVEN numbered locations are assigned to one memory chip and the ODD numbered locations are assigned to another memory chip. Given below a short table showing the RAMs and EPROMs used in the 8086 trainer.

<i>Circuit</i>	<i>Type</i>	<i>Capacity</i>	<i>Space Allocated</i>	<i>Decoding</i>
U6	27256 - EPROM 32 Kbytes	32 Kbytes	F0001 - FFFFF : ODD	Fully
U7	27256 - EPROM 32 Kbytes	32 Kbytes	F0000 - FFFFE : EVEN	Fully
U8	62256 - RAM	32 Kbytes	00001 - FFFFF : ODD	Fully
U9	62256 - RAM	32 Kbytes	00000 - 0FFFF : EVEN	Fully

The location no. 0h of the U9 will be seen by the 8086 at system address 00000h, location no. 1h will be seen at system address 00002h. It has been made possible by adopting the following strategy. As opposed to the traditional technique, the A0 line is not connected to the memory instead, the A0 line has been used by the decoder (refer to Section-4.3) to realize such memory address allocation strategy. Similar argument holds good for the ODD numbered chip viz, U8 where the location 0h is seen at system address 00001h, and the location 1h is seen at system address 00003h.

Bank oriented arrangement allows reading/writing two bytes data in one machine cycle provided the data operation starts at EVEN address boundary. Thus, the memory reference instructions get executed in half of the time. For example:-

`mov BYTE PTR [bx+45h], 77h : C6 47 45 77, mov BYTE PTR [bx+44h], 5Eh : C6 47 44 5E`
 instructions require two machine cycles to move data value 77h and 5Eh into two memory locations.

The above operations can be carried out by executing only one instruction like -
`'mov [bx+44h], 775Eh : C7 47 44 5E 77.'` Now, the CPU is taking only one machine cycle to move data value 77h and 5Eh into two memory locations. Now, the time taken by the CPU is half of the previous.

However, the word oriented operation starting at ODD address boundary will take two machine cycles but the total execution time will be less due to lesser number of instruction bytes. For example:-

`mov [bx+45h], 1234h : C7 47 45 34 12.` In this case, the data from [bx+45h] location will be read first and then from the [bx+46h] location.

Data read/write can be done on the EVEN or ODD bank only. The following examples may clarify some of the underlying concept.

- `mov al, BYTE PTR [bx+24h] : data from EVEN bank over D7-D0 lines → al`
- `mov al, BYTE PTR [bx+25h] : data from ODD bank over D15-D8 lines → al`
- `mov ah, BYTE PTR [bx+34h] : data from EVEN bank over D7-D0 lines → ah`
- `mov ax, [bx+57h] : data from ODD bank of location [bx+57h] over D15-D8 → al
 data from EVEN bank of location [bx+58h] over D7-D0 → ah`

By default, the 8086 microprocessor does byte oriented operations as occurs while booting up. However, the CPU possesses instruction for doing word oriented read/write operations.

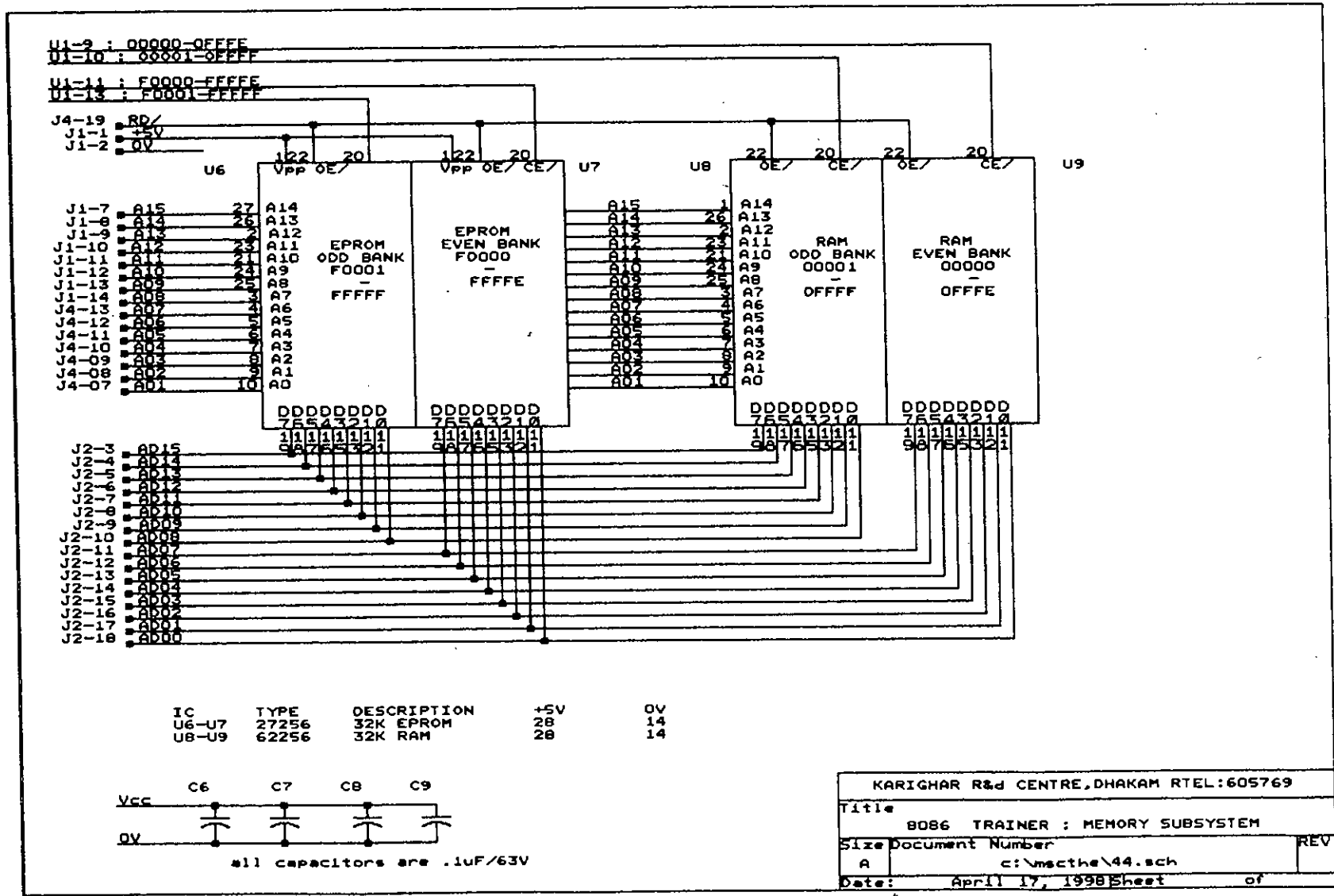


Fig - 4.4: Memory Subsystem Schematic

4.5 Keyboard/Display Subsystem

In a microcomputer application system, data entry and display are the two most important functions that a user desires. The keyboard and the display serve these purpose. Figure - 4.5 shows the schematic diagram of the keyboard/display subsystem of the 8086 trainer.

In the trainer presented in this thesis, the keyboard consists of 18 keys and is good enough to program the trainer in machine language. The display unit consists of nine 7-segment common cathode display devices. Five of them are for the 20-bit address and the remaining four for the 16-bit data.

The keyboard/display units have been interfaced with the 8086 CPU using 8279 controller. The internal ports of the U10-IC has been configured to work as a variable ports. The port addresses of the registers are:-

<i>Register Name</i>	<i>Port Address</i>	<i>Mode of Operation</i>
Data Register	0000h	read/write
Control Register	0002h	write only
Status Register	0002h	read only

Display Unit:

There are 16 display RAM locations inside the 8279. They are internally numbered as 0 to F i.e., 0000, 0001,.....,1111. The digits of the display window i.e., D9,,D1 correspond to RAM locations 0000,.....,1111, respectively. If one wishes to send 3 at D9 position of the display, then the code 4F (cc-code for 3) has to be written at RAM location 0000 of U10 and so on.

The contents of RAM locations 0 to F are sequentially dumped at B0-A3 of U10 and is in synchronism with the scan lines S0-S3. If the present data is, say 4F, and is coming from location 2(0010), then cc-terminal of D7 goes low. Other cc-terminals remain high. Thus, character 3 will appear at position D7 of the display window. The data multiplexing signals which determine where to display the present data, are generated by the U10 automatically and appear as S0-S3 scan lines. U13 is a 4-to-16 lines decoder.

Keyboard:

There are 18 keys in the key pad. The are labeled with some mnemonics whose meanings have been described at section-3.4.

The six row lines of the keyboard are connected to the six return lines of the 8279. These lines are internally terminated by pull-up resistors. Three column lines viz., Y0/-Y2/ have been derived from S0-S2 lines of the 8279 using U12 decoder. The bit pattern 01111111 rotates around the Y0/-Y7/ lines at 100khz rate and thus at a particular time, only one column line becomes 0. This type of keyboard is called a walking 0's keyboard.

When a key is closed, the corresponding return line at some time switches from logic-H to logic-L. This causes a unique 8-bit code depending on the position of the key. This is called SCAN CODE and gets stored in keyboard FIFO inside the 8279. At the same time, the 1st bit of the status register of the 8279 changes from 0 to 1 and IRQ line of the 8279 also goes high. When data is read from the FIFO, the status register gets cleared and the IRQ line drops to low.

With the help of the IRQ line, the key SCAN CODE may be read by the processor on interrupt basis. Or, the CPU can keep polling the status register of the 8279 and check for LSB=1.

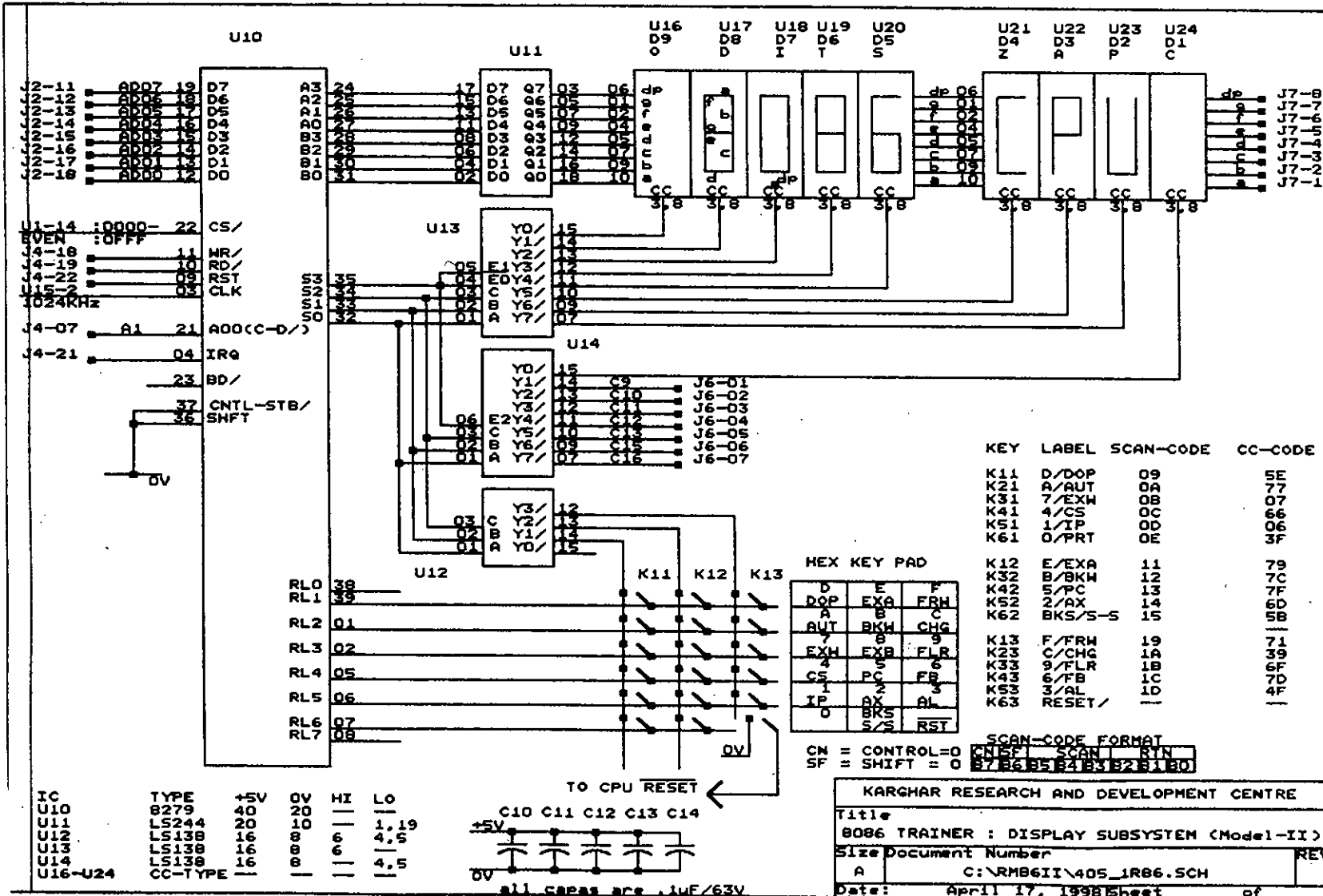


Fig - 4.5 : Keyboard/Display Subsystem Schematic

5

**MONITOR
PROGRAM
DESIGN**

5.1 What is a Monitor Program?

After powering up the 8086 trainer, we see the prompt message 8086 CPU in the display window. Now, we press the E/EXA key, the message _____ A d appears in the display. How does the CPU know the meaning of the symbol 'E/EXA'? The answer to this question will lead us to define the meaning of a monitor program, its essential features and the design aspects.

Looking at the trainer board, we see that there are two EPROM chips. If these two chips are replaced by another two EPROMs of the same type but blank, the events cited in the top para will never occur. This indicates that the original two EPROMs contain 'Something' which has guided the CPU to output binary data into the display buffer corresponding to the message '_____ A d'. This 'Something' is a collection of complex data/code base consisting of many routines and subroutines used to interpret the meaning of the command EXA and takes action accordingly. The other name of this 'Something' is Monitor Program.

Now, we press the same key, the symbol E is outputted and the display looks like E _____ Ad. This time, the meaning of the key has been changed. Again - how does the CPU correctly determine the meaning of a Key depending on the current context?

In fact, the CPU knows nothing. It is one of the most foolish semiconductor devices the human being has ever created. It has no sense of 'Good' or 'Bad'. It does exactly what it is instructed to do. It is the user who employs a microprocessor to do the job he is supposed to do.

The user knows the meaning of the key E/EXA. It is a command key if pressed after power up. The same key will work as a data key for the data value E if the display shows the message _____ Ad. The user has coded all these definitions into binary data and has fused in the said two EPROMs. Depending on the requirements of the user, the CPU uses one or more of these definitions to get the meaning of the external symbolic command like EXA or DOP.

The purpose of a microprocessor trainer is to allow understanding the working principles of the Instructions and the Addressing Modes of the 8086 CPU. It needs entering the binary codes of an instruction into RAM locations and execute them. So, the trainer's Monitor Program should allow a user to accomplish the following basic tasks:-

01. A request to the CPU to open the address field so that the user can input the 20-bit address of a RAM location for depositing the instruction codes.
02. A request to the CPU to move to the next RAM locations (i.e., Forwarding).
03. And finally a request to execute the instruction (i.e., Single Stepping).

The actual monitor program allows a user to accomplish many varieties of tasks . These may be :-

01. RAM location backwarding
02. Editing of the entered data
03. Correcting of the typing mistake (Backspace)
04. Execution of a large block of instructions (i.e., one complete program).
05. Examining and changing the registers contents.
06. Data/code entry with auto incrementing the address field.
07. Execution of one instruction at a time (Single Stepping).

The monitor program has to take various decisions while implementing a user request. This is done by maintaining a table of flags. These flags are the reserved RAM locations. Some RAM locations are also used as counters in order to keep track of the number of digits already or to be printed in the display window. These are reserved RAM locations and are shown in section 5.11.

The monitor program spends a considerable amount of time in the data conversion from one form to another. For example, the CPU gets the scan code 11 from E/EXA key closure. But to print E, the code 11 has to be converted to the 7-segment code corresponding to E which is 79. This conversion is being done using various look up tables and are shown in section 5.7. For carrying out EXA command, the scan code does not under go any conversion.

One of the desired characteristics of a monitor program is its ability to protect the reserved RAM space by insulating it from the users application codes. This is usually done by creating a software fence. The fence compares the users asserted address with the boundary addresses of the reserved RAM. If violation is detected, the users is forced to revert his initial condition. Some microprocessors like 80286, 80386 have built-in electronics to work as hard ware fence called fence registers. The data structure of the reserved RAM is a vital parameters for the monitor program to work. If this data table or its part goes corrupted, the computer system is bound to crash! Please see section 5.11 for reserved RAM space map.

In the case of the monitor program for the trainer introduced in this thesis, no such protection is employed. This has been done intentionally so as to allow a learner to manipulate the reserved RAM data and see that the system does crash. For example, after powering up the 8086 trainer, one can change the value of the memory location 00401 to 01h by executing the following codes and observe that the trainer is not working! A user may take the self assignment to develop the protection software routine or to design an electronics fence register to implement the monitor program codes isolation from that of the users codes.

```
06000 - mov    BYTE PTR [bx+01h],01h : C6 47 01 01
06004 - jmp    8086 CPU              : EA 21 00 00 F0
```

The design and documentation of the monitor program of a trainer should be as simple as possible. This is to allow the users to follow easily the working logic of various routines. Since, in the case of a trainer, the speed is not a concern, the monitor program has been developed using ladder structure. Compact structure would impose great difficulties on the readers to read and understand the instructions.

In the following pages, attempts have been made to document the whole logic of the monitor program in the form of flow charts into eight summary pages. These are shown in Figures-5.1(a) to 5.1(h). Then the assembly and binary codes are provided task wise. Brief comments are also provided at the end of the most instructions.

The program discussed above is just the kernel part of the monitor program. A monitor program to be of users friendly or useful to the users must contain some utility/ready made routines and subroutines. These are in fact not the essential part of the monitor program. The monitor program of this trainer contain a good amount of routines and subroutines which can easily be linked with the application program of a user. For the listing of these routines and subroutines, please see section 5.5 and 5.6.

The above discussion reveals that the design and the implementation of a monitor program requires a good level of understanding of the total system as well patience. A learner needs much more patience to study this program to get something useful.

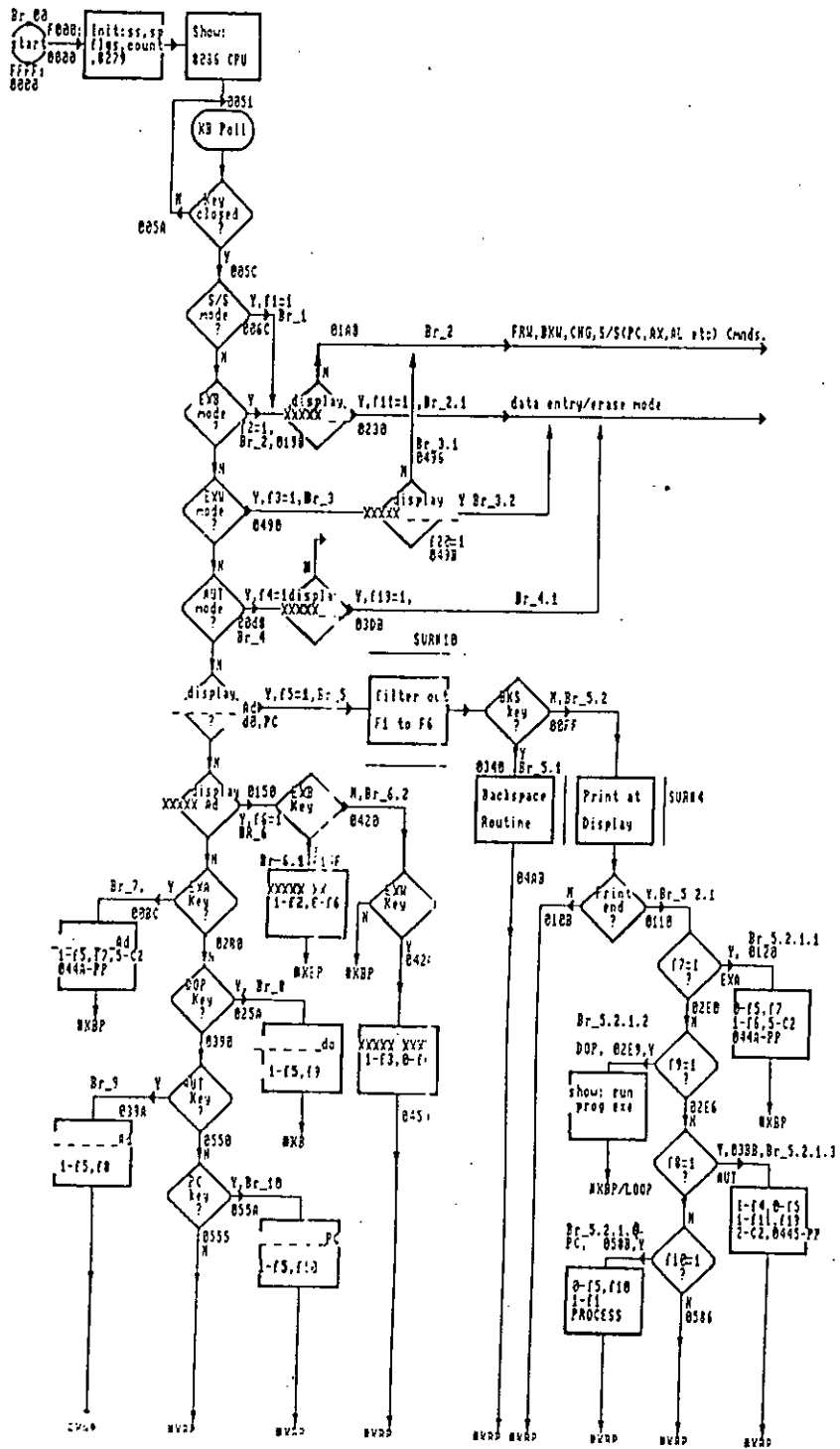


Fig - 5.1 (a) : Monitor Program Flow Chart Summary - 1

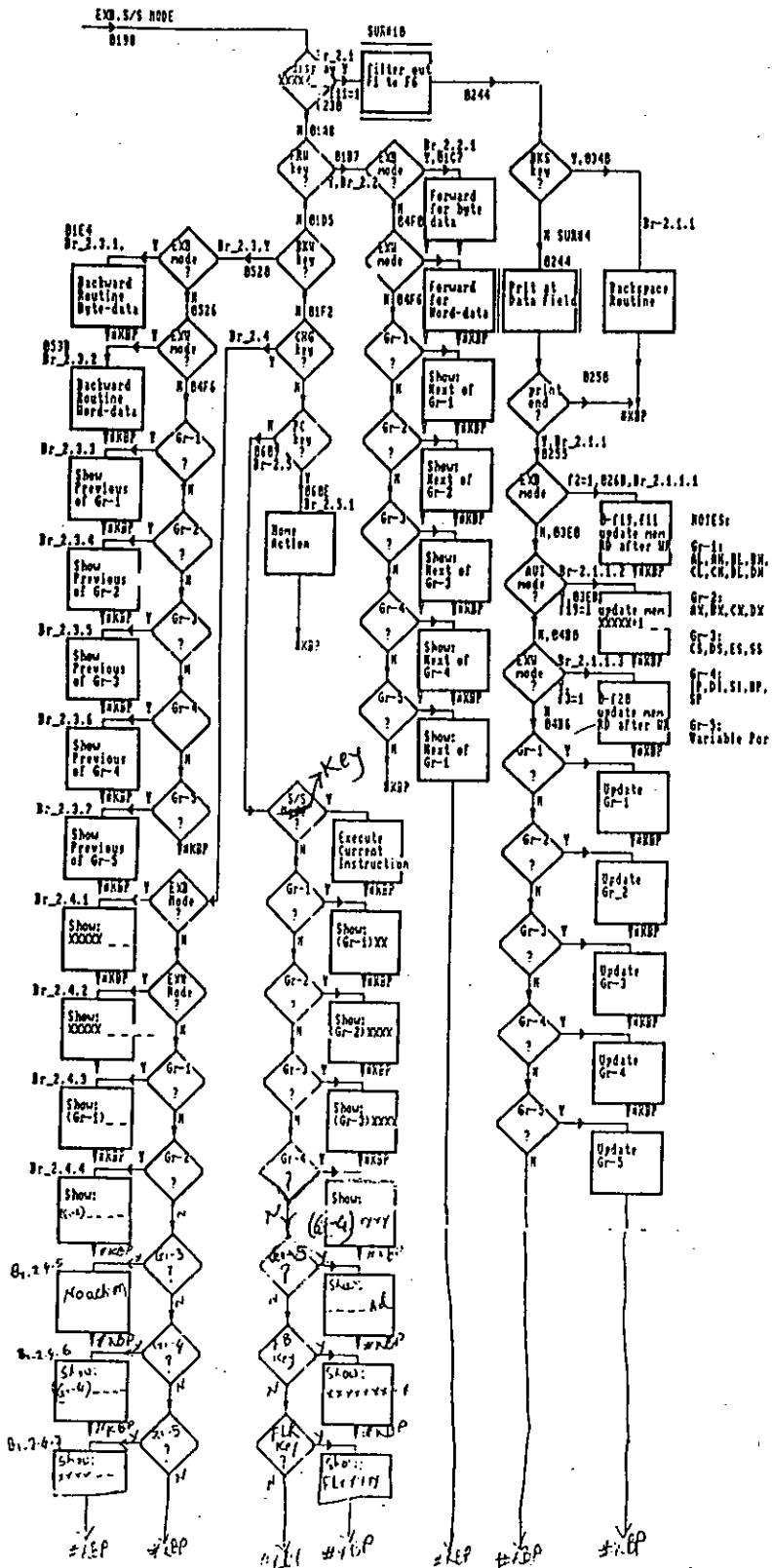


Fig - 5.1 (b) : Monitor Program Flow Chart Summary - 2

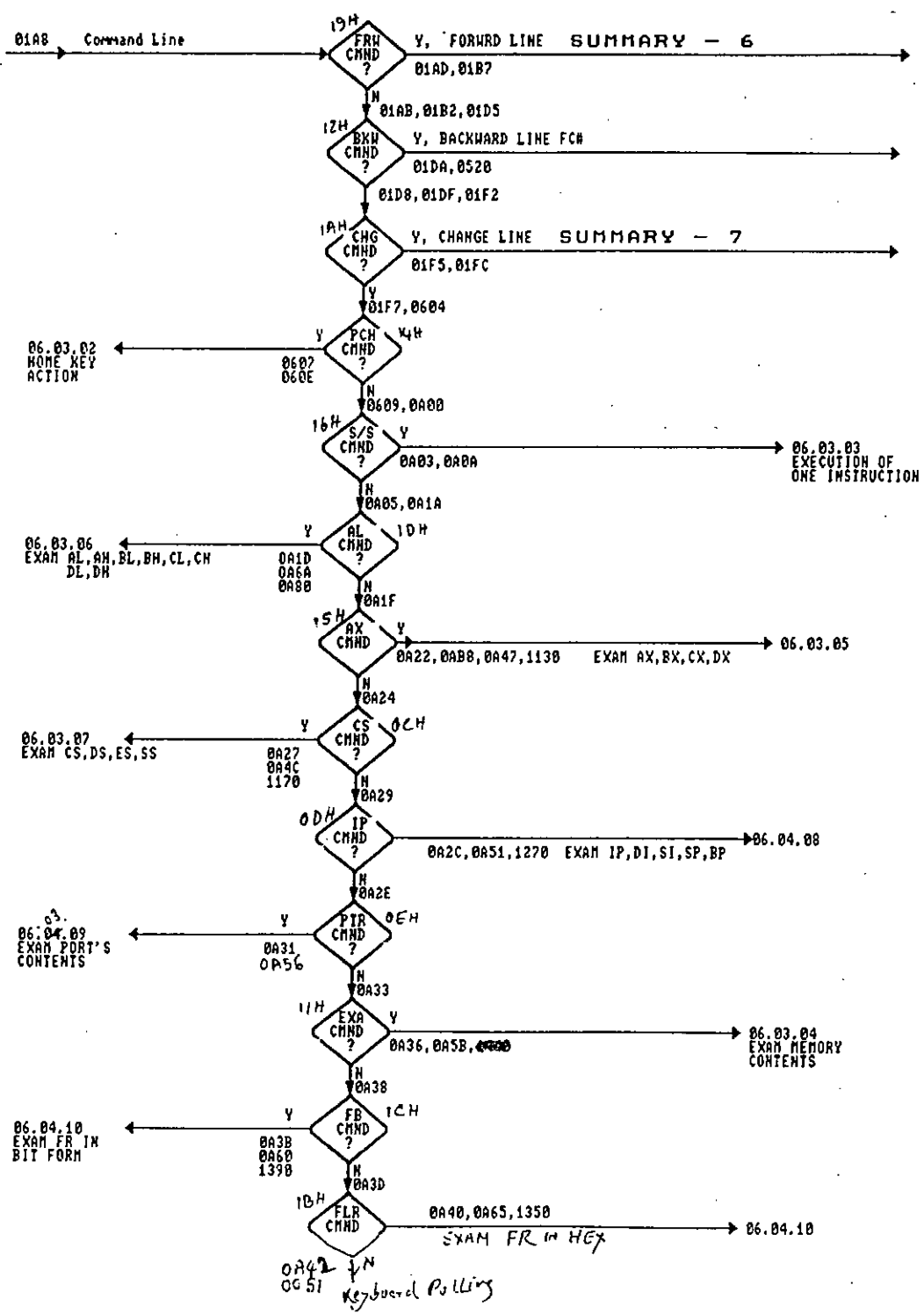


Fig - 5.1 (d) : Monitor Program Flow Chart Summary - 4

SUMMARY - 5

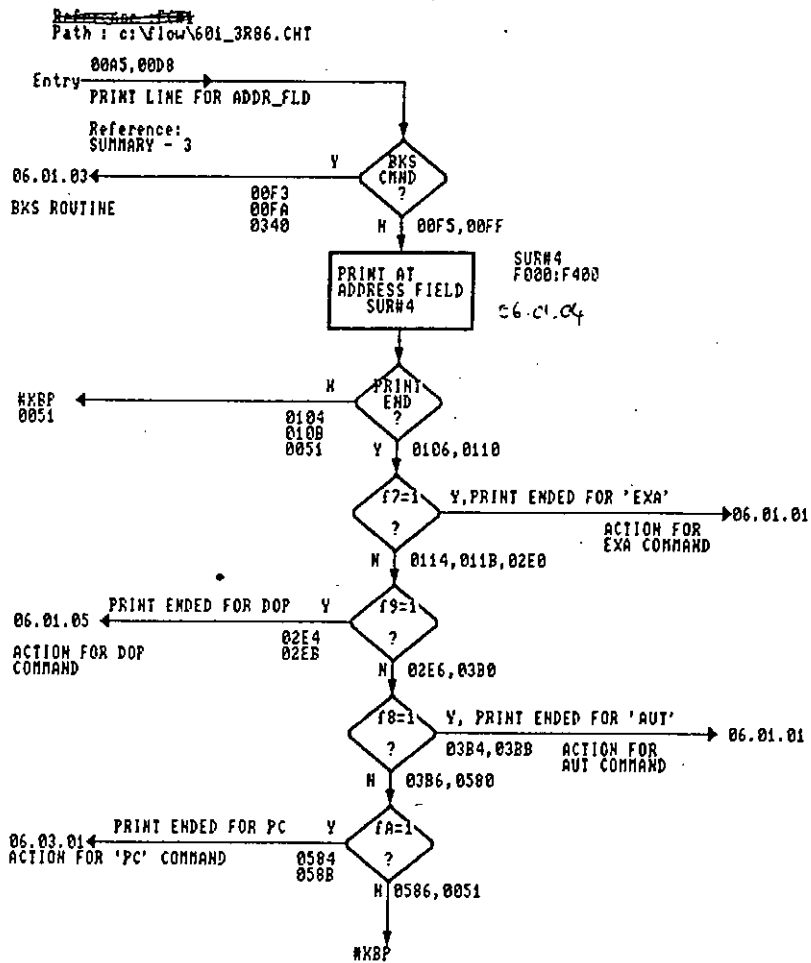


Fig - 5.1 (e) : Monitor Program Flow Chart Summary - 5

SUMMARY - 6
 FRW COMMAND BREAKDOWN
 Ref: SUMMARY - 4
 path c:\flow\601_4r86.cht

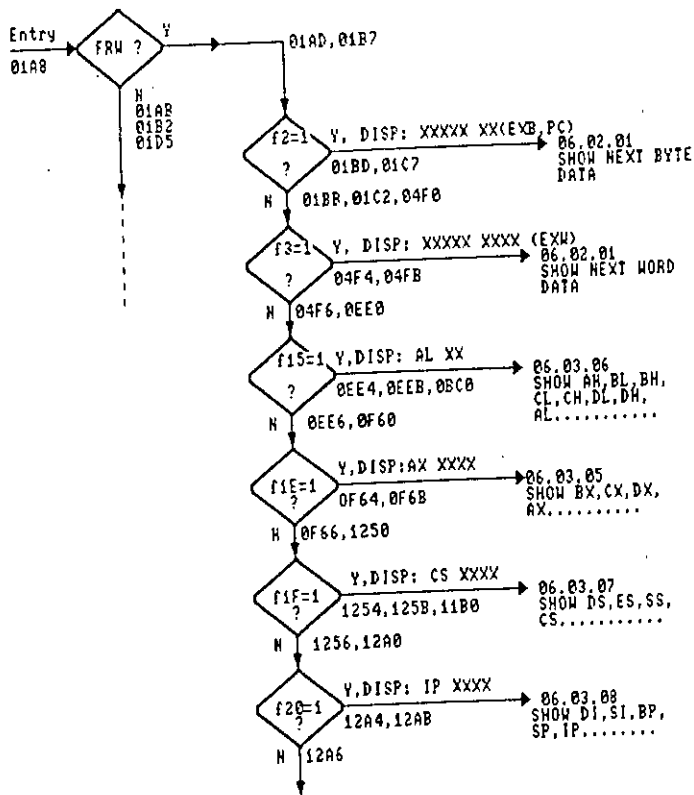
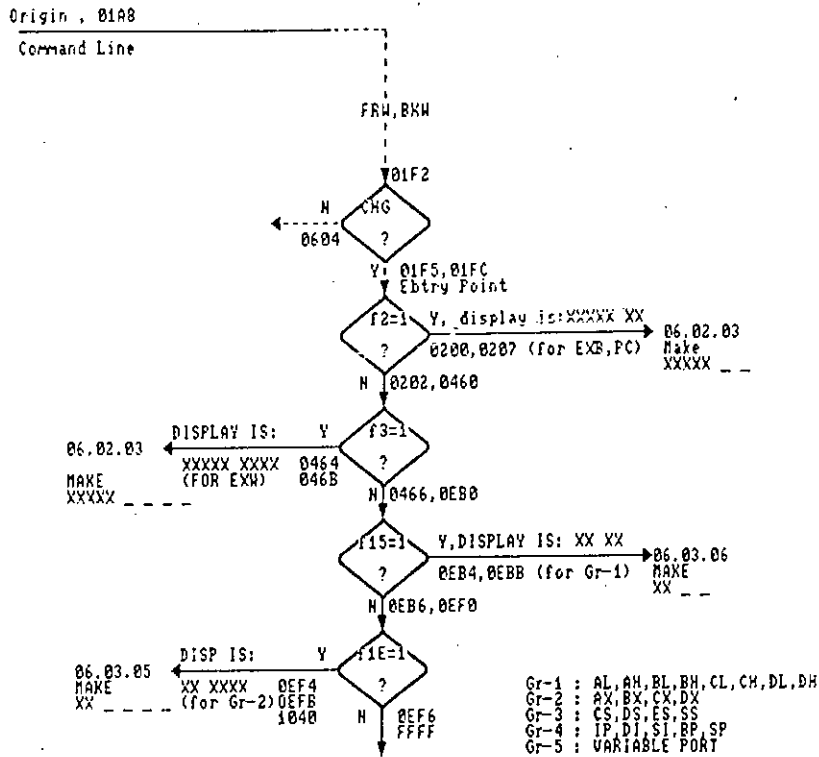


Fig - 5.1 (f) : Monitor Program Flow Chart Summary - 6

SUMMARY - 7
 CHG COMMAND BREAKDOWN
 Ref: Summary - 3

Path c:\flow\601_5r06.cht



'CHG' command is active if the display shows memory/register/port contents. If this is the case, the programme (monitor programme) branches at 01A8H and looks for the 'CHG' command.

If the 'CHG' command is detected, the programme looks at the display to see the type of information being displayed there (memory/register/port content).

And accordingly, the CPU branches to various paths and implement the 'CHG' command to open the data field for digits entry.

Instruction Codes:

```

01FC - 80 7F 02 01 : cmp BYTE PTR [bx+02h], 01h ;check if display is XXXXXX XX
0200 - 74 05 : jz 0F00:0207 ;yes display is XXXXXX XX
0202 - EA 68 04 00 F0 : jmp 0F00:0460 ;memory allocation
0207 - display is XXXXXX XX and goto 06.02.03 for implementation

0460 - 80 7F 03 01 : cmp BYTE PTR [bx+03h], 01h ;check if display XXXXX XXXX
0464 - 74 05 : jz 0F00:046B ;yes display is XXXXX XXXX
0466 - EA 80 0E F0 : jmp 0F00:0EB0 ;memory allocation
046B - goto section 06.02.03

0EB0 - 80 7F 15 01 : cmp BYTE PTR [bx+15h], 01h ;check if display fpor GR-1
0EB4 - 74 05 : jz 0F00:0EBB ;yes display is due to Gr-1
0EB6 - EA F0 0E 00 F0 : jmp 0F00:0EF0 ;memory allocation
0EBB - goto section 06.03.06

0EF0 -
  
```

Fig - 5.1 (g) : Monitor Program Flow Chart Summary - 7

SUMMARY - 8
 ACTION TO BE TAKEN ONCE DIGIT ENTRY
 FINISHED IN THE DATA FIELD.

Ref: Summary - 3

path c:\flow\601_6r86.cht

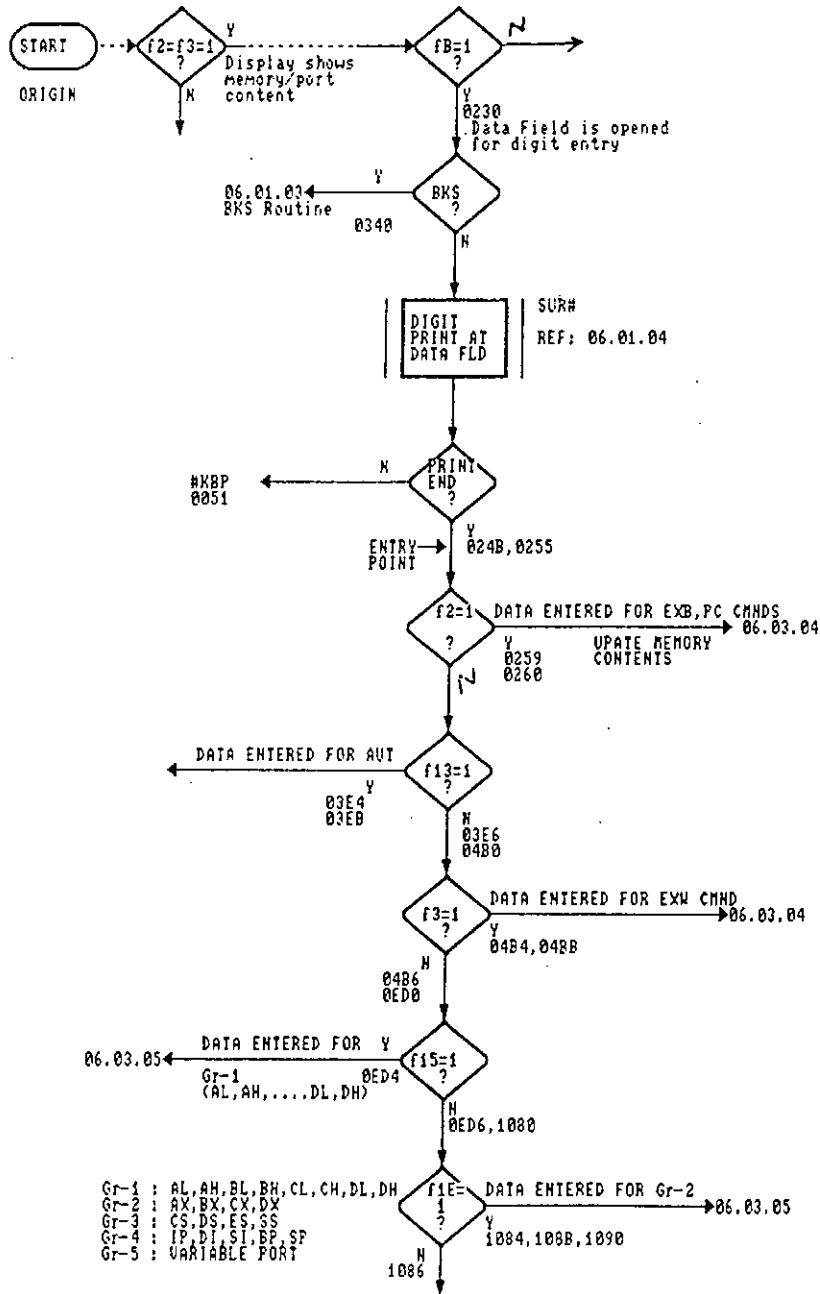


Fig - 5.1 (h) : Monitor Program Flow Chart Summary - 8

5.2 Implementing EXA, EXB/EXW, AUT, BKS and DOP Commands

5.2.1 Respond/Action for EXA Command.

EXA command allows an user to make a request to the CPU for entering progra/data codes into the trainer's RAM space. Upon detecting the EXA command, the CPU sets up some internal data structure so that the user can safely type 20-bit address of the desired RAM location.

Responding to EXA Command :Reference Figure - 5.1(a)

The logical 1st step of the user of the 8086 trainer is to enter program data/codes into memory locations. This is done by pressing the EXA key which is a request to the CPU to open the 20-bit address field. Now the user can enter the desired starting address of the memory locations. The CPU will show the current content. The user will enter the new data if needed by changing it with the CHG (Change) command. Data entry may be byte or word.

The user has pressed the EXA key. The CPU has got the scan code 11 from the keyboard FIFO of the 8279 controller. The code will be interpreted as a command key. And accordingly, the CPU has opened the address field showing the message '_____ Ad' in the display. The monitor program set f5=1 (refer flow chart: Summary-1 and Level; Br_7) so that 20-bit address printing is possible following Br_5. f7 is also set to 01h. Printing position (PP) and counter-2(C2) are also initialized. Refer to reserved RAM space map and its description in section 5.11 for better understanding.

Instruction Codes: F000:00BC

```
00BC - 9A 8E FF 00 F0 : call SUR#2 (F000:FF8E) ; to display _____ Ad at addr field
00C1 - C6 47 05 01 : mov BYTE PTR [bx+05h],01h ; 1 → f5 (00405)
00C5 - C6 47 07 01 : mov BYTE PTR [bx+07h],01h ; 1 → f7 (00407)
00C9 - C6 47 40 05 : mov BYTE PTR [bx+40h],05h ; 5 → C2 (digits to print at Addr fld)
00CD - C7 47 4C 4A 04 : mov [bx+4Ch],044Ah ; 1st digit printing at D9 of display
00D2 - EA 51 00 00 F0 : jmp KBP (Keyboard Polling) ; now take digits from the keyboard.
```

The above codes have set that 5 hex-digits corresponding to 20-bit physical address of a memory location will be allowed to have printing at the address field of the display. The 1st digit will appear at the D9 position of the display. This D9 position is corresponds to the 0000:044A location of the reserved RAM space.

Action for EXA Command

Let us assume that the user has finished entering the 20-bit physical address of RAM location. Say, for example it is 06666. Now, what should happen? The logical answer is this that the CPU should show the content of this memory location. But, there is a question - the 8086 has 16-bit data bus, so it should show one byte data for the current entered memory location or two bytes data for the two consecutive memory locations? The user has to supply the answer to this question. So, at the end of 20-bit address entry, the monitor program should wait for the users command requesting Byte or Word entry (section 5.2.2).

Display became like _____ Ad in response to EXA command. 20-bit address printing is done. The display is XXXXX Ad. Now the next step - EXB or EXW command. The CPU knows that the user wants to examine/edit a memory content. But, is it byte or word? Hence, the processor goes back to the keyboard to get the specific request from the user. The CPU sets the flag f6=1 to allow branching at Br_6 where EXB (Examination of byte data) or EXW (Examination of word data) request is sensed. f7,f5 are reset to 00s as the requirement for branching to the corresponding paths are not there.

Instruction Codes: Br_5.2.1.1

```
0120 - C6 47 07 00 : mov BYTE PTR [bx+07h],00h ; f7 is reset
0124 - C6 47 05 00 : mov BYTE PTR [bx+05h],00h ; f5 is reset
0128 - C6 47 06 01 : mov BYTE PTR [bx+06h],01h ; f6 is set to 1
012C - C6 47 40 05 : mov BYTE PTR [bx+40h],05h ; address digit counter is back to 5
0130 - C7 47 4C 4A 04 : mov [bx+4Ch],044Ah ; 1st digit printing back to D9
0135 - EA 51 00 00 F0 : jmp KBP
```

5.2.2 Respond/Action for EXB/EXW Commands

The display is like XXXXX Ad. The CPU will respond only to EXB or EXW commands. If EXB command is encountered, the CPU will display the content of the memory location printed at the display following Br_6.1.

If EXW command is encountered, the CPU will follow Br_6.2 to display the contents of two consecutive memory locations in the data field of the display window. The content of the entered memory location will be displayed at D4D3 positions and the content of the next higher memory location will be shown at D2D1 positions. In either case, the monitor program will enable the software logic for FRW, BKW, BKS and commands.

Br_6 : Instruction Codes

Sensing EXB Command

```
0150 - 80 3F 13      : cmp   BYTE PTR [bx],13h      ;checking if EXB (scan code 13h) command
0153 - 75 05         : jnz   F000:015A           ; not EXW command
0155 - EA 5F 01 00 F0 : jmp   F000:015F           ;EXB command sensed, goto Br_6.1
015A - EA 20 04 00 F0 : jmp   F000:0420           ; memory allocation
```

Sensing EXW Command

```
0420 - 80 3F 0B      : cmp   BYTE PTR [bx],0Bh     ; checking for EXW (scan code 0Bh) command
0423 - 74 05         : jz    F000:042A           ; EXW command found, goto Br_6.2
0425 - EA 51 00 00 F0 : jmp   KBP                 ; if not EXB/EXW, poll KBP for them.
```

Br_6.1 : Responding/Action to EXB Command

Till EXB command, digits being shown in the display window are laying as common cathode format at Table-1(T1) of the reserved RAM space. Upon EXB command, the CPU converts T1 to T3 and then to T2. The cc-codes of T1 are first converted to unpacked hex and then to packed hex. The CPU determines the segment base by manipulating the content of RAM location D9 of the T3. Offset part is calculated from (D8D7)(D6D5) of T2. Now, the CPU makes a read operation from the appropriate memory location as requested by the user via key-pad. The data byte is stored at D2D1 of T2. T2 is converted to T1. 00s are written at D2,D1 of T1. And finally, T1 is transferred to 8279's display RAM for outputting at the display window. Flag f2 is set to 01h, so that the CPU can branch at Br_2 for responding to FRW, BKW, BKS and CHG commands.

Instruction Codes:

```
015F - 9A B0 F0 00 F0 : call  SUR#5                ; to convert T1 to T3, cc-codes to unpacked hex
0164 - 9A D0 F4 00 F0 : call  SUR#6                ; to convert T3 to T2, unpacked hex to packed hex
0169 - 9A 5A F4 00 F0 : call  SUR#7                ; to determine segment/offset and passed to es/di
016E - 26 8A 05       : mov   al, BYTE PTR es:[di]  ;data byte is read from requested memory location
0171 - 88 47 4F       : mov   BYTE PTR [bx+4Fh],al, ;data → D4D3 position of T3
0174 - 9A 7C F4 00 F0 : call  SUR#8                ; convert packed hex to cc-codes (T2→ T1)
0179 - C6 47 42 00    : mov   BYTE PTR [bx+42h],00h ;blank → D1 of T1
017D - C6 47 43 00    : mov   BYTE PTR [bx+43h],00h ;blank → D2 of T1
0181 - 9A B6 FF 00 F0 : call  SUR#3                ; cc-codes of T1 goto 8279's display RAM
0186 - C6 47 02 01    : mov   BYTE PTR [bx+02h],01h ; 1 → f2 (00402)
018A - C6 47 06 00    : mov   BYTE PTR [bx+06h],00h ; 0 → f6 (00406)
018E - EA 51 00 00 F0 : jmp   KBP                 ; sense FRW,BKW,CHG commands
```

Br_6.2 : responding/Action to EXW Command

The logic is same as EXB command except that, now a word-oriented read operation will be done. The contents of two consecutive memory locations will be displayed at D4D3 and D2D1 respectively. f3=1 to branch at Br_3.

Instruction Codes:

```
042A - 9A B0 F4 00 F0 : call  SUR#5                ; to convert T1 to T3
.....
0454 - EA 51 00 00 F0 : jmp   KBP                 ; poll keyboard for FRW,BKW and CHG commands
```

5.2.3 Respond/Action to AUT Command

Br_9 : Responding to AUT Command

With AUT command, a user can save a lot of time while entering data into memory. The address field automatically steps to the next memory location as the data bytes are entered. Also, the data field becomes ready to accept the next data byte. There is no need to use the CHG command to open the data field to enter new data for the next memory location. The data byte gets stored in the desired memory location as they are entered. f5 is set to 01h to allow printing at the address field for entering the 20-bit memory address.

Instruction Code:

```
039A - 9A 8E FF 00 F0 : call SUR#2 ;to display _____ Ad at the address field
039F - C6 47 05 01 : mov BYTE PTR [bx+05h],01h ; 1→f5 (00405)
03A3 - C6 47 08 01 : mov BYTE PTR [bx+08h],01h ; 1 → f8 (00408)
03A7 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for data entry at the address field.
```

Br_5.2.1.3 : Actions for AUT Command

Printing at the address field was due to AUT command. Since, the user wants to enter data into memory location in auto-increment mode that is after each data byte entry, address field will be automatically incremented and the data field will also be ready to accept the next byte of data and the display will become XXXXX __.

f19 (00419) is set to 01h to allow printing at the data field following Br_2.1. In this mode, all commands are inactive except two digits data entry and BKS (Backspace). The printing position and the number of digits to be printed are passed by PP and C2 memory locations of the reserved RAM (section 5.11).

Instruction Codes:

```
03BB - C6 47 04 01 : mov BYTE PTR [bx+04h],01h ;
03BF - C6 47 05 00 : mov BYTE PTR [bx+05h],00h ;
03C3 - EA 07 02 00 F0 : jmp F000:0207 ; memory allocation

0207 - C6 47 0B 01 : mov BYTE PTR [bx+0Bh],01 ; 1 → f11
020B - C6 47 13 01 : mov BYTE PTR [bx+13h],01 ; 1 → f19
020F - C6 47 44 08 : mov BYTE PTR [bx+44h],08h ; writing '_' symbol at D3 of T1
0213 - C6 47 45 08 : mov BYTE PTR [bx+45h],08h ; writing '_' symbol at D4 of T1
0217 - 9A B6 FF 00 F0 : call SUR#3 ; T1 → 8279's display RAM
021C - C7 47 4C 45 04 : mov WORD PTR [bx+4Ch],0445h ; 1st digit printing position is at D4
0221 - C6 47 40 02 : mov BYTE PTR [bx+40h],02h ; 2 digits will be printed
0225 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for data for next location
```

5.2.4 Backspace Routine

BKS (Backspace Routine) : Br_5.1 of Fig-5.1(a).

This routine allows erasing typing mistake with the help of BKS command. Since, printing could be at address or data field, the logic works accordingly. For address field, if C2=05h, that means that no digit has yet been printed and BKS has no meaning. Similar reason applies for data field both for byte and word data. The flow chart of this routine is shown in Figure - 5.2.4.

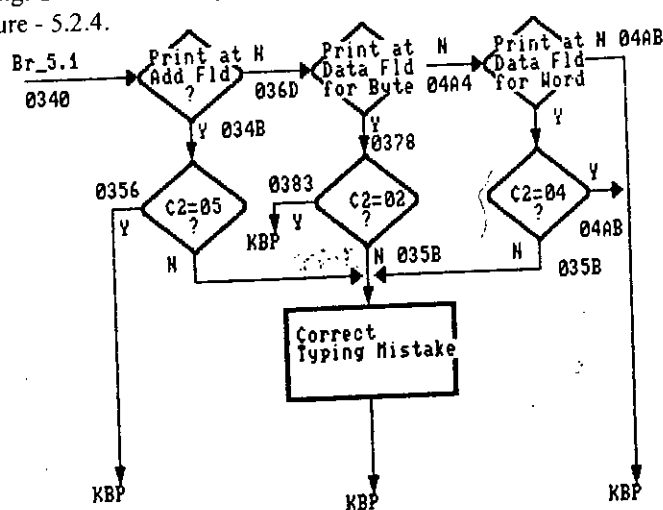


Fig-5.2.4 : Flow Chart for Implementing Backspace Routine

The BKS command replaces the digit just printed by the symbol ‘_’ (underscore). This data manipulation occurs only in the display image (T1 of the reserved RAM) and consequently at 8279’s display RAM. Nothing happens in the users RAM until 20-bit address printing or data entry is complete.

Instruction Codes:

0340 - 80 7F 05 01	: cmp	BYTE PTR [bx+05h],01h	;
0344 - 74 05	: jz	F000:036D	; printing has done in the address field
0346 - EA 6D 03 00 F0	: jmp	F000:036D	; memory allocation
034B - 80 7F 40 05	: cmp	BYTE PTR [bx+40h],05h	;
034F - 74 05	: jz	F000:0356	; C2=05h, no action for BKS command
0351 - EA 5B 03 00 F0	: jmp	F000:035B	; memory allocation
0356 - EA 51 00 00 F0	: jmp	KBP	; polling keyboard for valid command
035B - 8B 7F 4C	: mov	di,WORD PTR [bx+4Ch]	; collect PP to get next position of printing
035E - 47	: inc	di	; calculate current PP
035F - 89 7F 4C	: mov	WORD PTR [bx+4Ch],di	;
0362 - C6 05 08	: mov	BYTE PTR [di],08h	; erase the current digit with the symbol ‘_’.
0365 - FE 47 40	: inc	BYTE PTR [bx+40h]	;
0368 - EA 9B 02 00 F0	: jmp	F000:029B	; share common codes
036D - 80 7F 0B 01	: cmp	BYTE PTR [bx+0Bh],01h	;
0371 - 74 05	: jz	F000:0378	;
0373 - EA A0 04 00 F0	: jmp	F000:040A	; memory allocation
0378 - 80 7F 40 02	: cmp	BYTE PTR [bx+40h],02h	;
037C - 74 05	: jz	F000:0383	; printing has occurred at data field
037E - EA 5B 03 00 F0	: jmp	F000:035B	;
0383 - EA 51 00 00 F0	: jmp	KBP	; polling keyboard for valid command
04A0 - 80 7F 40 04	: cmp	BYTE PTR [bx+40h],04h	;
04A4 - 74 05	: jz	F000:04AB	; print at data field (word)
04A6 - EA 5B 03 00 F0	: jmp	F000:035B	;
04AB - EA 51 00 00 F0	: jmp	KBP	; polling keyboard for valid command

5.2.5 Printing at Display Window (Address/Data Field)

Say, the display is like _____Ad. The first digit entered from the keyboard will be printed at D9 position of the display, the next one at D8 position and so on. The nine 7-segment display devices are mapped over nine RAM locations of T1 of the reserved RAM space (refer section 5.11). To demonstrate how the printing takes place, we take an specific example. Flow chart of the routine is shown in Figure - 5.2.5.

Say, we wish to print digit 5 at D9 position of the display. In order to do it, the CPU will have to write the corresponding cc-code of 5 (cc-code = 7F) at memory location 0044AH (for D9 position) of T1 of the reserved RAM space. The cc-codes for digits 0-F are pre-stored at Lookup Table-1 (LUT-1: section 5.7) of the EPROM. This LUT-1 has been built on the basis of the scan codes of the keys of the hex-key pad. The CPU uses the scan code of the pressed key to form a 20-bit physical address to address the LUT-1 in order to collect the right cc-code. The CPU, then knows the printing position from the memory location of PP of the reserved RAM space map. The total number of digits to be printed is also known to the CPU from the content of the C2 of the reserved RAM space map as well.

Since, the number of digits and their printing places are variable, a generalized subroutine (SUR#4; section 5.5) has been designed which is good enough to carry out printing both at the address and data field.

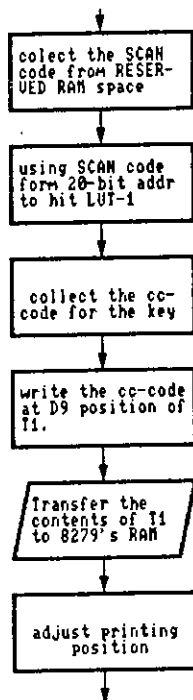


Fig-5.2.5 : Flow Chart to Implement Digit Printing in the Display Window

Instruction Codes: Br_5.2

00FF - 9A 00 F4 00 F0	: call	SUR#4	; for printing digit in the display window
0104 - 75 05	: jnz	F000:010B	; printing not finished
0106 - EA 10 01 00 F0	: jmp	F000:0110	; printing finished. Now for next action
010B - EA 51 00 00 F0	: jmp	KBP	; poll keyboard to receive valid command

5.2.6 Respond/Action to DOP Command

Responding to DOP Command

DOP (DO a Program) is detected. The CPU has opened the address field for getting the 20-bit address of the program to be executed. Flag f5 is set to 01h to allow printing following Br_5. f9 is set to 01h so that the CPU, at the end of the 20-bit address printing can remember that the user requested a program execution.

Instruction Codes: Br_8

```
02BA - C6 47 05 01 : mov BYTE PTR [bx+05h],01h ;l → f5
02BE - C6 47 09 01 : mov BYTE PTR [bx+09h],01h ;l → f9
02C2 - 9A 8E FF 00 F0 : call SUR#2 ; to display message _____ Ad
02C7 - C6 47 45 5E : mov BYTE PTR [bx+45h],5Eh ; writing d at D4 position of T1
02CB - C6 47 44 5C : mov BYTE PTR [bx+44h],5Ch ; writing 'o' at position D3 of T1
02CF - 9A B6 FF 00 F0 : call SUR#3 ; T1 → 8279, display is _____ do
02D4 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

Actions for DOP Command

The display became like _____ do in response to DOP command. The command informed the CPU that the user wanted to execute a program either written into RAM or already stored into EPROM. Now, the starting address of the program to be executed is to be typed at the address filed. say, it is F05B0H.

Once the 20-bit physical address typing is complete, the CPU separates the segment base and the offset by using many routines and subroutines. These four bytes information viz., OFF_L(B0), OFF_H(05), SEG_L(00), SEG_H(F0) are passed to RAM locations 0045F,00460,00461 and 00462 respectively (Table: T5) of the reserved RAM space map. The CPU also passes code EA(opcode for far jump) at location 0045E in T5 of the reserved RAM space map.

The CPU then makes a far jump at location 0045EH. The PC (CS:IP) is replaced by 0000:045E. At location 0045E....., the CPU finds another jump instruction viz., EA B0 05 00 F0. The PC's contents again changed to F000:05B0 which is the starting address of the program to be executed. This is how the CPU is managed to arrive at the starting address of a user program to be executed.

There is one point to mention is that the CPU outputs the 'run' message at the display before jumping to the user program. This 'run' message may not be visible if the program execution time is relatively small. The message can be viewed if the program is terminated in a loop. The program does not regard the flags as they are not meaningful any more.

Instruction Codes: Br_5.2.1.2

```
02EB - 9A B0 F4 00 F0 : call SUR#5 ; to convert T1 to T3 (cc-codes to unpacked hex)
02F0 - 9A D0 F4 00 F0 : call SUR#6 ; T3 to T2 conversion (unpacked hex to packed hex)
02F5 - 9A 5A F4 00 F0 : call SUR#7 ; to determine segment/offset, passed to es/di
02FA - C6 47 5E EA : mov BYTE PTR [bx+5Eh],EAh ; passing code EA for long jump at T5
02FE - 89 7F 5F : mov WORD PTR [bx+5Fh],di ; passing OFF_L and OFF_H
0301 - 89 47 61 : mov WORD PTR [bx+5Fh],ax ; passing SEG_L and SEG_H
0304 - C6 47 4A 00 : mov BYTE PTR [bx+4Ah],00h ; blank → D9 of T1
0308 - C6 47 49 00 : mov BYTE PTR [bx+49h],00h ; blank → D8 of T1
030C - C6 47 48 50 : mov BYTE {PTR [bx+48h],50h ; writing 'r' at D7 of T1
0310 - C6 47 47 1C : mov BYTE PTR [bx+47h],1Ch ; writing 'u' at D6 of T1
0314 - C6 47 46 54 : mov BYTE PTR [bx+46h],54h ; writing 'n' at D5 of T1
0318 - C6 47 45 00 : mov BYTE PTR [bx+45h],00h ; blank → D4 of T1
031C - C6 47 44 00 : mov BYTE PTR [bx+44h],00h ; blank → D3 of T1
0320 - C6 47 43 00 : mov BYTE PTR [bx+43h],00h ; blank → D2 of T1
0324 - C6 47 42 00 : mov BYTE PTR [bx+42h],00h ; blank → D1 of T1
0328 - 9A B6 FF 00 F0 : call SUR#3 ; the display shows run message
032D - EA 5E 04 00 00 : jmp 0000:045E ; the CPU jumps at T5 and then at user program.
```

5.3 Implementing FRW/BKW/CHG (in non S/S mode) Commands

The display is showing XXXX XX, for example. This means that the address field shows a 20-bit address and the data byte indicates its contents. This display could have been resulted either by EXB or PC command.

The user can now change the data using CHG command. He can also check the contents of the previous or next memory location with BKW or FRW commands respectively.

The PC command is checked to see if the display was in S/S mode and the user wanted a home action. Home action means that the display will be brought back to the initial position where it was showing the address of the instruction to be executed by S/S command. Details will be found at Br_2.5.1

Instruction Codes: Br_2

```

0198 - 80 7F 0B 01      : cmp   BYTE PTR [bx+0Bh],01h      ;
019C - 74 05           : jz    F000:01A3                ; display is XXXX __ goto Br_2.1
019E - EA A8 01 00 F0  : jmp   F000:01A8                ; memory allocation
01A3 - EA 30 02 00 F0  : jmp   F000:0230                ; goto Br_2.1 for printing at data field

01A8 - 80 3F 19       : cmp   BYTE PTR [bx],19h        ;
01AB - 75 05         : jnz   F000:01B2                ;
01AD - EA B7 01 00 F0  : jmp   F000:01B7                ;FRW desired , goto Br_2.2
01B2 - EA D5 01 00 F0  : jmp   F000:01D5                ; memory allocation

01D5 - 80 3F 12       : cmp   BYTE PTR [bx],12h        ;
01D8 - 75 05         : jnz   F000:01F2                ;
01DA - EA 20 05 00 F0  : jmp   F000:0520                ; BKW desired, goto Br-2.3
01DF - EA D5 01 00 F0  : jmp   F000:01F2                ; memory allocation

01F2 - 80 3F 12       : cmp   BYTE PTR [bx],1Ah        ;
01F5 - 74 05         : jz    F000:01FC                ; CHG desired, goto Br-2.3
01F7 - EA 04 06 00 F0  : jmp   F000:0604                ; memory allocation

0604 - 80 3F 14       : cmp   BYTE PTR [bx],14h        ;
0607 - 74 05         : jz    F000:060E                ; PC (Home Action) desired goto Br_2.5.1
0609 - EA 51 00 00 F0  : jmp   KBP                      ; poll keyboard for valid command

```

5.3.1 Forward Routine

Forward (FRW) command can be used to:-

- a. examine the contents of the next memory location
- b. examine the contents of the next two consecutive memory locations
- c. examine the contents of next 16-bit general purpose registers (AX, BX, CX and DX)
- d. examine the contents of next 8-bit general purpose register (AL, AH, BL, BH, CL, CH, DL, DH)
- e. examine the contents of next segment register (CS, DS, ES, SS)
- f. examine the contents of next 16-bit index registers (IP, DI, SI, BP, SP)
- g. examine the content of the next variable port.

Instruction Codes: Br_2.2

```
01B7 - 80 7F 02 01 : cmp BYTE PTR [bx+02h],01h ;
01BB - 75 05 : jnz F000:01C2 ;
01BD - EA C7 01 00 F0 : jmp F000:01C7 ; FRW for byte goto Br_2.2.1
01C2 - EA F0 04 00 F0 : jmp F000:04F0 ; memory allocation

04F0 - 80 7F 03 01 : cmp BYTE PTR [bx+03h],01h ;
04F4 - 74 05 : jnz F000:04FB ; FRW for word data goto Br_2.2.2
04F6 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

Forward for Byte Data : Br_2.2.1

The routine reads the current OFFSET from the location OF(OFFset) of T2 of the reserved RAM. It is incremented and is written back to T2. A read operation is done using the new OF and the data byte is placed at D4D3 position of T2. And finally, T2 is transferred to the 8279 for displaying.

Instruction Codes:

```
01C7 - 9A 5A F4 00 F0 : call SUR#7 ; to deter segment/offset, passed via es/di
01CC - 47 : inc di ;
01CD - 89 7F 50 : mov [bx+50h],di ;
01D0 - EA 6E 01 00 F0 : jmp F000:016E ; memory allocation to share codes

016E - 26 8A 05 : mov al, BYTE PTR es:[di] ; data byte is read from memory
0171 - 88 47 4F : mov BYTE PTR [bx+4Fh],al ;
0174 - share common codes at Br_6.1
```

Forward for Word Data : Br_2.2.2

The routine extracts the segment base and the offset from T3 and T2 by using SUR#7. The CPU then increments the offset by twice and does a word-oriented read operation from the memory. The word data is saved at 'OF' position of T2. Finally, the content of T2 is displayed.

Instruction Codes: Br_2.2.2

```
04FB - 9A 5A F4 00 F0 : call SUR#7 ; to deter segment/offset, passed via es/di
0500 - 47 : inc di ;
0501 - 47 : inc di ;
0502 - 89 7F 50 : mov [bx+50h],di ;
0505 - 26 8B 05 : mov ax,es:[di] ; word data is read from the meory
0508 - 88 47 4F : mov BYTE PTR [bx+4Fh],al ;
050B - 88 67 4E : mov BYTE PTR [bx+4Eh],ah ;
050E - 9A 7C F4 00 F0 : call SUR#8 ; convert T2 into T1 (packed to cc-codes)
0513 - EA 9B 02 00 F0 : jmp F000:029B ; memory allocation

029B - 9A B6 FF 00 F0 : call SUR#3 ; to transfer T1 into 8279
02A0 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

5.3.2 Backward Routine

The function of the BKW command is similar to FRW command except that now it is for the previous memory location or the register.

Instruction Codes: Br_2.3

```
0520 - 80 7F 02 01 : cmp BYTE PTR [bx+02h],01h ;
0524 - 74 05 : jz F000:052B ; BKW desired for byte data
0526 - EA 30 05 00 F0 : jmp F000:0530 ; memory allocation
052B - EA E4 01 00 F0 : jmp F000:01E4 ; goto Br_2.3.1 for byte data

0530 - 80 7F 03 01 : cmp BYTE PTR [bx+03h],01h ;
0534 - 74 05 : jz F000:053B ; BKW for word data
0536 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

BKW for Byte Data : Br_2.3.1

The logic is same as for Frw except now for the previous memory location

Instruction Codes:

```
01E4 - 9A 5A F4 00 F0 : call SUR#7 ; deter seg/off, passed via es/di
01E9 - 4F : dec di ;
01EA - 89 7F 50 : mov [bx+50h],di ;
01ED - EA 6E 01 00 F0 : jmp F000:016E ; share common codes under Br_2.2.1
```

BKW for Word Data : Br_2.3.2

The logic is same for FRW except that now., it is for the previous memory contents.

Instruction Codes:

```
053B - 9A 5A F4 00 F0 : call SUR#7 ; deter segment/offset, passed via es/di
0540 - 4F : dec di ;
0541 - 4F : dec di ;
0542 - EA 02 05 00 F0 : jmp F000:0502 ; share common codes under Br_2.2.2
```

5.3.3 CHG Command and Byte Data Update for Memory - Br_2.4

Byte-Data Edit for Memory

Let us see how the monitor program helps to modify the current content of a RAM location. Take an specific example of location 0C000 XX. We wish to deposit 32H in this location.

We give the CHG command with the key - K23. The display becomes like 0C000 __ and prompts us to enter one byte data. we type 3,2. The display becomes 0C000 32. The following three inter-related software routines have worked together here.

- a. responding to CHG command : Br_2.4.1
- b. printing at the data field : Br_2.1
- c. data update in the memory : Br_2.1.1.1

Responding to CHG Command : Br_2.4.1

The routine sets f11=1 to allow digit printing at the data field following Br_2.1. It makes data field ready to type digits. It passes values relating to the number of digits to be printed and also the position of the 1st digit to be printed. The routine also sets f19=1, to tell the CPU that the current printing position at the data field is due to byte-wide memory content modification.

Instruction Codes: Br_2.4.1

```
0207 - C6 47 0B 01 : mov BYTE PTR [bx+0Bh],01h ; 1 → f11
020B - C6 47 13 01 : mov BYTE PTR [bx+13h],01h ; 1 → f19
020F - C6 47 44 08 : mov BYTE PTR [bx+44h],08h ; writing '_' at D3 of T1
0213 - C6 47 45 08 : mov BYTE PTR [bx+45h],08h ; writing '_' at D4 of T1
0217 - 9A B6 FF 00 F0 : call SUR#3 ; to transfer T1 to 8279
021C - C7 47 4C 45 04 : move [bx+4Ch],0445h ; printing position is passed
0221 - C6 47 40 02 : mov BYTE PTR [bx+40h],02h ; no of digits to be printed are also passed
0225 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

Printing at the Data Field : Br_2.1

The logic is the same as printing at the address field. In fact, the same subroutine (SUR#4) is used. The non-digit key BKS is filtered out. If a need arises to correct the typing mistake, then 'BKS' routine is called upon. The information relating to the number of digits to be printed and the printing position are taken from C2 and PP of the reserved RAM space map.

Data Update in Memory : Br_2.1.1.1

Now, two digits entry in the data field is complete. The next step is to write this value in the actual memory location. The digits entered in the T1 are in cc-code format. These have to be converted to hex and then to be written at the desired RAM location. A read after write is done in order to demonstrate the the particular RAM location is good and the content of a ROM location can not be changed.

Instruction Codes: Br_2.1.1.1.1

```
0255 - 80 7F 02 01 : cmp BYTE PTR [bx+02h],01h ;
0259 - 74 05 : jz F000:0260 ; it is data-byte modify
025B - EA E0 03 00 F0 : jmp F000:03E0 ; not data-byte modify
0260 - 80 7F 13 01 : cmp BYTE PTR [bx+13h],01h ;
0264 - 74 05 : jz F000:026B ; content modify in EXB command
0266 - EA xx xx xx xx : jmp ??????? ;
026B - C6 47 13 00 : mov BYTE PTR [bx+13h],00h ;
026F - C6 47 0B 00 : mov BYTE PTR [bx+0Bh],00h ;
0273 - 9A B0 F4 00 F0 : call SUR#5 ; to convert T1 to T3
.....
02A0 - EA 51 00 00 F0 : jmp KBP ; poll keyboard for valid command
```

5.3.4 CHG Command and Word-Data Update for Memory - Br_2.4

Word-Data Modify for Memory

We are examining word-data of memory. That means that if the display is showing 0C000 ABCD; that means (0C000) = AB and (0C001) = CD. Now, we wish to deposit 12 and 34 at these RAM locations respectively.

We press CHG key. The display becomes like 0C000 _ _ _ _ . and prompt us to enter the digits 1,2,3,4. When the entry is finished, the display becomes 0C000 1234.

The following three inter-related routines have worked together:-

- responding to CHG command : Br_2.4.2
- printing at the data field : Br_2.1
- data update into memory : Br_2.1.1.3

Responding to CHG Command : Br_2.4.2

The CHG command brings the display XXXXX _ _ _ _ and prompts us to enter 16-bit data. At the end of entry, memory locations are updated. A read operation is done to check the goodness of the RAM locations and also to demonstrate that the ROMs contents can not be modified. The PP and C2 are also initialized at 0000:0445 and 04 respectively.

Instruction Codes: Br_2.4.2

```
046B - C6 47 14 01 : mov BYTE PTR [bx+14h],01h ;
046F - C6 47 42 08 : mov BYTE PTR [bx+42h],08h ; writing '_' at D1 of T1
0473 - C6 47 43 08 : mov BYTE PTR [bx+43h],08h ; writing '_' at D2 of T1
0477 - C6 47 44 08 : mov BYTE PTR [bx+44h],08h ; writing '_' at D3 of T1
047B - C6 47 45 08 : mov BYTE PTR [bx+45h],08h ; writing '_' at D4 of T1
047F - C6 47 40 04 : mov BYTE PTR [bx+40h],04h ; passing no of digits (4) to C2
0483 - C7 47 4C 45 04 : mov [bx+4Ch],0445h ; PP=0445 , 1st digit to be printed at at D4
0488 - EA 9B 02 00 F0 : jmp F000:029B ; share common codes under Br_2.2.2
```

Printing at Data Field: Br_2.1

Same as Br_2.1

Update Word-data into memory : Br_2.1.13

```
04BB - 9A B0 F4 00 F0 : call SUR#5 ; to convert T1 into T3
04C0 - 9A D0 F4 00 F0 : call SUR#6 ; to convert T3 into T2
04C5 - 9A 5A F4 00 F0 : call SUR#7 ; to deter segment/offset, passed via es/di
04CA - 8A 47 4F : mov al,BYTE PTR [bx+4Fh] ;
04CD - 8A 67 4E : mov ah,BYTE PTR [bx+4Eh] ;
04D0 - 26 89 05 : mov es:[di],ax ; word data is written into two locations
04D3 - 26 8B 05 : mov ax,es:[di] ; read after write
04D6 - 88 47 4F : mov BYTE PTR [bx+4Fh],al ;
04D9 - 88 67 4E : mov BYTE PTR [bx+4Eh],ah ;
04DC - 9A 7C F4 00 F0 : call SUR#8 ; to convert T2 to cc-code
04E1 - C6 47 14 00 : mov BYTE PTR [bx+14h],00h ; 0 → f20
04E5 - EA 9B 02 00 F0 : jmp F000:029B ; share common codes under Br_2.2.2
```

5.4 Theory of Single Stepping

5.4.1 Respond/Action to PC Command

Respond

After power up RESET, we usually enter program data/codes into RAM and then execute the program. If the program works, fine!. Otherwise, we proceed to debug the program by executing one instruction at a time - called Single Stepping.

To single step the processor, we use the PC command to enter the 20-bit physical address of the 1st instruction of the program. At the end of the address entry, the CPU shows the address-opcode of the 1st instruction.

f5=1, it will allow us to print the 20-bit physical address in the address field of the display. f10=1 allows the operating system to determine that the 20-bit address entry was done following the 'PC' command. So, actions can be taken accordingly.

Instruction Codes:

92792

0550 - 803F 14	: cmp	BYTE PTR [bx],14h	;check for PC command
0553 - 74 05	: jz	F000:055A	; yes PC command
0555 - EA 51 00 00 F0	: jmp	KBP	; no PC command detected,goto monitor
055A - 9A 8E FF 00 F0	: call	SUR#2	; _____ Ad at T1 of the reserved RAM
055F - C^47 45 73	: mov	BYTE PTR [bx+45h],73h	; P at D4 of T1
0563 - C6 47 44 39	: mov	BYTE PTR [bx+44h],39h	; C at D3 of T1
0567 - C6 47 05 01	: mov	BYTE PTR [bx+05h],01h	; 1 → f5
056B - C6 47 0A 01	: mov	BYTE PTR [bx+0Ah],01h	; 1 → f10
056F - 9A B6 FF 00 F0	: call	SUR#3	; transfer T1 to 8279, display _____PC
0574 - EA 51 00 00 F0	: jmp	KBP	; poll keyboard for valid command

Action

20-bit address entry has been due to 'PC' command. Therefore, the monitor program should display the address-opcode of the 1st instruction to be executed.

Instruction Codes:

0580 - 80 7F 0A 01	: cmp	BYTE PTR [bx+0Ah],01h	;entry for PC?
0584 - 74 05	: jz	F000:058B	; yes PC command
0586 - EA 51 00 00 F0	: jmp	KBP	; not PC command
058B - C6 47 05 00	: mov	BYTE PTR [bx+05h],00h	; 0 → f5
058F - C6 47 0A 00	: mov	BYTE PTR [bx+0Ah],00h	; 0 → f10
0593 - C6 47 01 01	: mov	BYTE PTR [bx+01h],01h	; 1 → f1
0597 - EA 30 06 00 F0	: jmp	F000:0630	; memory allocation
0630 - 9A B0 F4 00 F0	: call	SUR#5	; xlate T1 to T3

.....

064A - EA 74 01 00 F0	: jmp	F000:0174	; memory allocation
-----------------------	-------	-----------	---------------------

.....

0174 - 9A 7C F4 00 F0	: call	SUR#8	; xlate T2 to T1
-----------------------	--------	-------	------------------

.....

0186 - EA C0 0A 00 F0	: jmp	F000:0AC0	; memory allocation
-----------------------	-------	-----------	---------------------

; initializing memory locations 0046A - 00473 as flags and are zeroing them all. These will be used by single step.

0AC0 - C7 47 6A 00 00	: mov	WORD PTR [bx+6A],0000h	; 0 → f6A - f6B
-----------------------	-------	------------------------	-----------------

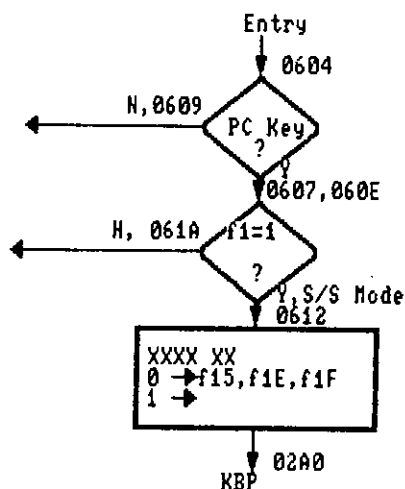
.....

1434 - EA 51 00 00 F0	: jmp	KBP	; poll keyboard for valid command
-----------------------	-------	-----	-----------------------------------

5.4.2 Home Key Routine (PC command in S/S mode)

Pressing 'PC' key after power up, allows entering instruction address for the purpose of single stepping the processor through each of the instructions of a program segment. Once the trainer has entered into single step mode following 1st PC command. 2nd PC command works as a 'Home Key'. Home Key means that the pressing of PC key will bring the display to show the address-opcode of the instruction to be single stepped should the display not showing so. This is necessary as because the display may show something else due to a 'Register Check' command.

We may clarify it by an example. say, the display is in S/S mode and is showing F0000 B1 - the Home Position. Press 'FB' key and the display shows 'XXXXX XXXX'. Now press PC key and the display shows F0000 B1 again. This is what we stated as Home Action.



Flag conditions clearly tell the CPU that the software status is in 'Home Position' and there would be no conflict between the key commands.

Instruction Codes:

0604 - 80 3F 14	: cmp	BYTE PTR [bx],14h	;checking PC command
0607 - 74 05	: jz	F000:060E	; yes PC command
0609 - EA 00 0A 00 F0	: jmp	F000:0A00	; other command
060E - 80 7F 01 01	: cmp	BYTE PTR [bx+01h],01h	; checking if in S/S mode
0612 - 74 05	: jz	F000:0619	;yes S/S mode
0614 - EA 51 00 00 F0	: jmp	KBP	; not S/S mode
0619 - EA 50 06 00 F0	: jmp	F000:0650	; memory allocation

;say the display is not showing the address-opcode of the instruction to be single stepped. T3/T4 of the reserved RAM space contains this information. Therefore, get it from there and display it in the 7-segment output.

0650 - 9A 5A F4 00 F0	: call	SUR#7	;get Segment/Offset, passed via es/di
0655 - 8B 7F 66	: mov	di,[bx+66h]	;get offset from T4
0658 - 26 8A 05	: mov	al,BYTE PTR es:[di]	; grt opcode
065B - 88 47 65	: mov	BYTE PTR [bx+65h],al	; xfer opcode to T4
065E - 9A 34 F5 00 F0	: call	SUR#12	; xferring T4 to T2
0663 - EA E0 0A 00 F0	: jmp	F000:0AE0	; memory allocation

0AE0 - C6 47 15 00	: mov	BYTE PTR [bx+15h],00h	; 0 -> f15h
0AE4 - C7 47 1E 00 00	: mov	WORD PTR [bx+1Eh],0000h	; 0 -> f1E, f1F

02A0 - EA 51 00 00 F0	: jmp	KBP	; poll keyboard valid command
-----------------------	-------	-----	-------------------------------

5.4.3 Execution of One Instruction (Single Stepping)

Single Stepping Routine allows the user to study and understand various features of a microprocessor particularly the 'Instructions' and the 'Addressing Modes' by examing/changing the contents of the registers.

Implementation of the single step routine in the monitor program of the 8086 trainer was a challenging as well as a thrilling task. The author consulted many literature [2,3,4] but found no 'Engineering Hints' as to how to utilize the 8086's 'IRET instruction' and 'Trap Bit' (these two features supports single stepping according to Intel's data sheet) to implement the single stepping mechanism. The author discovered the idea of 'artificial IRET' and then got the routine implemented. Given below the complete documentation of the single step routine of the 8086 trainer.

Implementation of the Routine

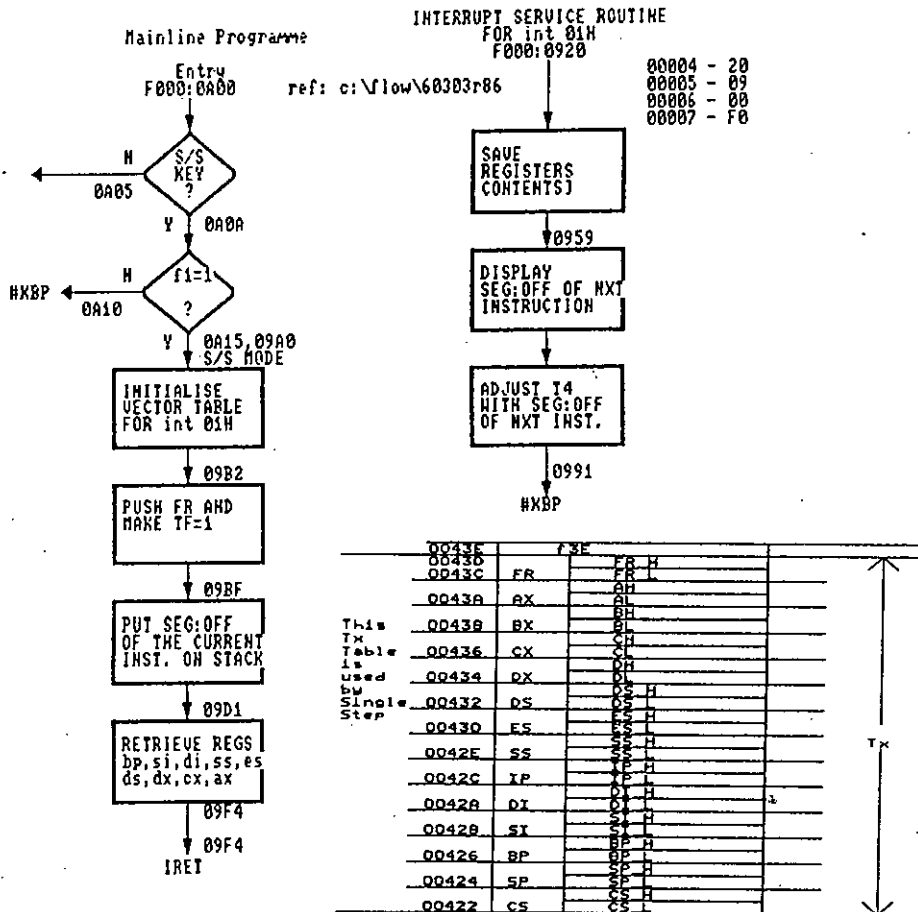


Fig-5.4.3 : Flow Chart and Data Structure to Implement Single Step Routine

In the ISR for 'int 01h', we may update the display with the address and opcode of the next instruction to be single stepped which is here '02000 - BA '. We can also update the main line program with the address of the next instruction so that the CPU can execute it on receipt of the S/S command from the key board of the trainer.

Instruction Codes:

Mainline Program:

F000:0A00

;checking if S/S command in Single-Stepping Mode

```
0A00 - 80 3F 16      : cmp   BYTE PTR [bx],16h      ;check if S/S command from the keyboard
0A03 - 74 03        : jz    F000:0A0A            ;yes S/S command
0A05 - EA 1A 0A 00 F0 : jmp   F000:0A1A            ;not S/S command
0A0A - 80 7F 01 01  : cmp   BYTE PTR [bx+01h],01h ;check if S/S mode
0A0E - 74 05        : jz    F000:0A15            ;yes S/S mode
0A10 - EA 51 00 00 F0 : jmp   F000:0051            ;keyboard polling for command
0A15 - EA A0 09 00 F0 : jmp   F000:090A            ;memory allocation
0A1A -
```

F000:090A

;initializing the Vector Table for 'int 01h'

```
09A0 - BB 00 00      : mov   bx,0000h             ;bx as a local pointer
09A3 - C7 47 04 20 09 : mov   [bx+04h],2009h       ;initializing the OFFSET
09A8 - C7 47 06 00 F0 : mov   [bx+06h],00F0h       ;initializing the SEGMENT Base
09AD - BB 00 04      : mov   bx,0004h             ;getting bx back
```

;put FR onto Stack and make TF = 1

```
09B8 - 4C 4C        : dec   SP,SP                ;pointing at vacates
09B2 - 8B 47 3C      : mov   ax,[bx+3Ch]          ;get FR from the table Tx
09B5 - 8B EC        : mov   bp,sp                ;
09B7 - 89 46 00      : mov   [bp+00h],ax          ;FR onto Stack
09BA - 81 4E 00 00 01 : or    WORD PTR [bp+00h],0001h ; now TF = 1.
```

F000:09BF

;putting CS:IP of the current instruction onto Stack

;section 05.03.01 of the Reference manual says that PC command and subsequent 20-bit address entry of ;the 1st instruction to be single stepped manipulated various data in the following way:-

;MS-byte of Segment Base has been saved at D9 position of T3 ofd the reserved RAM. Offset has been ;saved at (D8D7)(D6D5) positions of T4 as packed hex.

```
09BF - 9A 5A F4 00 F0 : call  F000:F45A (SUR#7)     ; determine Segment/Offset
09C4 - 8C 46 FE      : mov   [bp - 02h],es         ;put Seg onto Stack
09C7 - 8B 7F 66      : mov   di,[bx+66h]          ;get Offset from T4
09CA - 89 7E FC      : mov   [bp-04h],di          ;put Offset onto Stack
09CD - 4C 4C        : dec   sp,sp                ;adjusting the point of sp
09CF - 4C 4C        : dec   sp,sp                ;
09D1 -
```

F000:09D1

;Retrieving the Registers contents from Tx before executing the current instruction

```
09D1 - 8B 6F 26      : mov   bp,[bx+26h]          ;updating bp
```

```

09D4 - 8B 77 28      : mov  si,[bx+28h]      ;updating si
09D7 - 8B 7F 2A      : mov  di,[bx+2Ah]     ;updating di
09DA - 8E 57 2E:      : mov  ss,[bx+2Eh]     ;updating ss
09DD - 8E 47 30      : mov  es,[bx+30h]     ;updating es
09E0 - 8E 5F 32      : mov  ds,[bx+32h]     ;updating ds
09E3 - 8B 57 34      : mov  dx,[bx+34h]     ;updating dx
09E6 - 8B 4F 36      : mov  cx,[bx+36h]     ;updating cx
09E9 - 8B 47 38      : mov  ax,[bx+38h]     ;updating ax
09EC - 8B D8          : mov  bx,ax           ;
09EE - BB 00 04      : mov  bx,0004h        ;don't disturb bx
09F1 - 8B 47 3A      : mov  ax,[bx+3Ah]     ;updating ax
09F4 - CF            : iret                ;artificial IRET
09F5 -

```

Interrupt Service Routine for 'int 01h' (Single Step)

;Vector table for int 01h has been initialized by mainline program as follows:-

```

;00004 - 20      : IP_L
;00005 - 09      : IP_H
;00006 - 00      : CS_L
;00007 - F0      : CS_H

```

;The CPU has executed one instruction and now it has come to the ISR. Do the house keeping job to allow the user examine/changing various register contents.

F000:0920

;save the registers contents at Tx of the reserved RAM

```

0920 - 89 47 3A      : mov  [bx+3Ah],ax     ;saving ax
0923 - 8B C3          : mov  ax,bx           ;
0925 - 89 47 38      : mov  [bx+38h],cx     ;saving bx
0928 - 89 47 36      : mov  [bx+36h],dx     ;saving cx
092B - 89 57 34      : mov  [bx+34h],ds     ;saving ds
092E - 8C 5F 32      : mov  [bx+32h],ds     ;saving ds
0931 - 8C 47 30      : mov  [bx+30h],es     ;saving es
0934 - 8C 57 2E      : mov  [bx+2Eh],ss     ;saving ss
0937 - 8B 47 66      : mov  ax,[bx+66h]     ; offset of the executed instruction being
093A - 89 47 2C      : mov  [bx+2Ch],ax     ;saved in Tx
093D - 89 7F 2A      : mov  [bx+2Ah],di     ; saving di
0940 - 89 77 28      : mov  [bx+28h],si     ;saving si
0943 - 89 6F 26      : mov  [bx+26h],bp     ;saving bp
0946 - 89 67 24      : mov  [bx+24h],sp     ;saving sp
0949 - 8B 47 68      : mov  ax,[bx+68h]     ;Seg of the executed instruction being
094C - 89 47 22      : mov  [bx+22h],ax     ;saved at Tx
094F - 9C            : push ax              ;to save at Tx
0950 - 8B EC          : mov  bp,sp           ;
0952 - 8B 46 00      : mov  ax,[bp+00h]     ;now FR in ax
0955 - 89 47 3C      : mov  [bx+3Ch],ax     ;saving FR at Tx
0958 - 9D            : popf                 ;good sp
0959 -

```

;updating the display and T4 with the Segment/Offset of the next instruction to be single stepped.

```

0959 - 8B EC          : mov  bp,sp           ;
095B - 8B 46 00      : mov  ax,[bp+00h]     ;getting Offset
095E - 89 47 66      : mov  [bx+66h],ax     ;xferring at T4
0961 - 8B F8          : mov  di,ax           ;

```

```

0963 - 8B 46 02      : mov  ax,[bp+02h]          ;getting Segment
0966 - 89 47 68      : mov  [bx+68h],ax         ;xferring at T4
0969 - 8E C0         : mov  es,ax               ;
096B - 26 8A 05      : mov  al,BYTE PTR es:[di] ;getting the opcode
096E - 88 47 65      : mov  BYTE PTR [bx+65h],al ;xfrer at T4
0971 - 8B 46 04      : mov  ax,[bp+04h]         ;getting FR
0974 - 89 47 3C      : mov  [bx+3Ch],ax         ;saving FR
0977 - 9A 34 F5 00 F0 : call F000:F534 (SUR#12)  ; xfer T4 to T2
097C - 9A 7C F4 00 F0 : call F000:F47C (SUR#8)  ; convert T2 to T1
0981 - C7 47 42 00 00 : mov  WORD PTR [bx+42h],0000h ; blank at D2,D1
0986 - 9A B6 FF 00 F0 : call F000:FFB6 (SUR#3)  ;xfer T1 to 8279
098B - 44           : inc  sp                   ;
098C - 44           : inc  sp                   ;
098D - 44 44 44 44   : inc  sp,sp,sp,sp         ; good sp
0991 - EA 51 00 00 F0 : jmp  F000:0051 (KBP)     ; wait for S/S command from the keypad
09996 -

```

5.4.4 Exam/Edit AX, BX, CX, DX Registers

Program Logic

There is only AX key on the key pad. The contents of the remaining registers can be viewed by successively pressing the FRW key. The flow chart is shown in Figure-5.4.4.

The user has to examine the AX value first. This enables FRW command and a flag to check BX register. Next FRW command checks f1E=1 to be sure that the system is S/S mode. The program allows checking the BX register. While checking the BX register, the flag for CX register is enabled. Next FRW command allows checking the CX register. The exam routine of the CX register enables the flag for DX register. The next FRW command allows checking the DX register. While checking the DX register, the flag for AX register is enabled. Next FRW command allows checking the AX register. And the process cycles.

While checking the Register contents, the CHG key active. This way the current content of the registers except the BX register can be changed. The next section shows the flow chart for updating the registers contents after the data have been entered following the CHG command.

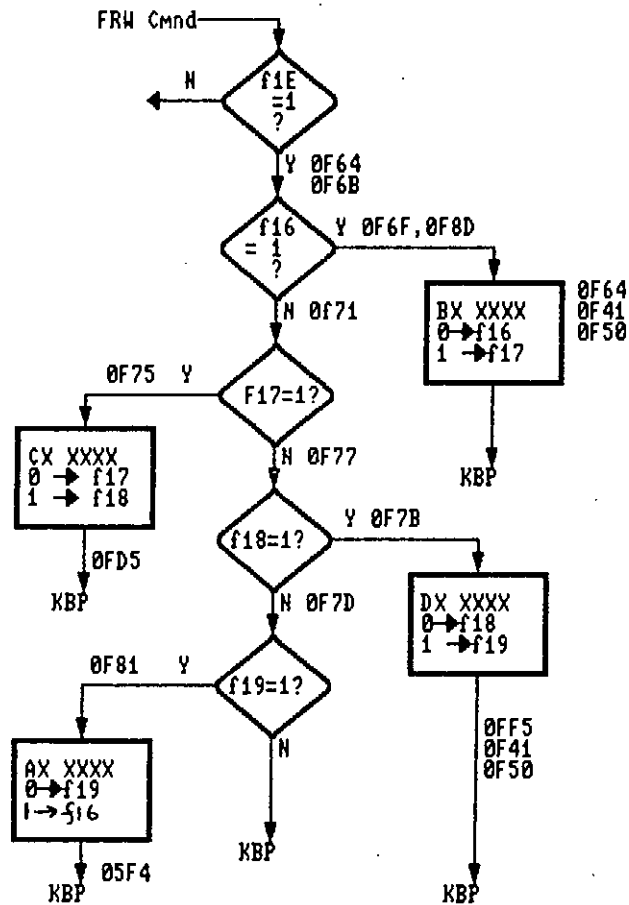


Fig - 5.4.4 : Flow Chart to Implement AX, BX, CX and DX Examination

5.4.5 Exam/Edit AL,AH,BL,BH,CL,CH,DL,DH Registers

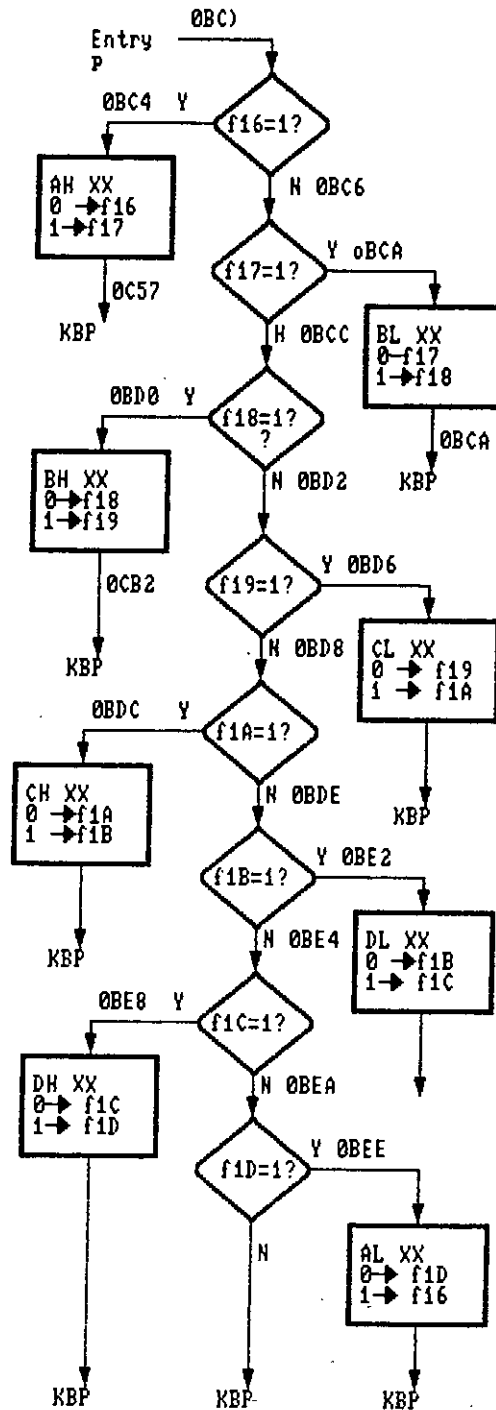


Fig - 5.4.5 : Flow Chart to Implement Contents Examination.

5.4.6 Exam/Edit IP,SI,DI,SP,BP Registers

The four registers DI,SP,SI,IP and BP are grouped together and is tagged by the flag f20. Individual flags are also identified by separate flags and are shown in the above flow chart. There is only one key viz., IP in the keyboard. The user has to check the content of this register first and then use the FRW command to examine the contents of the remaining registers. The BP register check is not implemented and is left as an exercise to the readers. It can be easily implemented by assigning a new flag to the BP register. The contents of this register can only be viewed in the S/S mode. At the moment there is no routine to edit their contents.

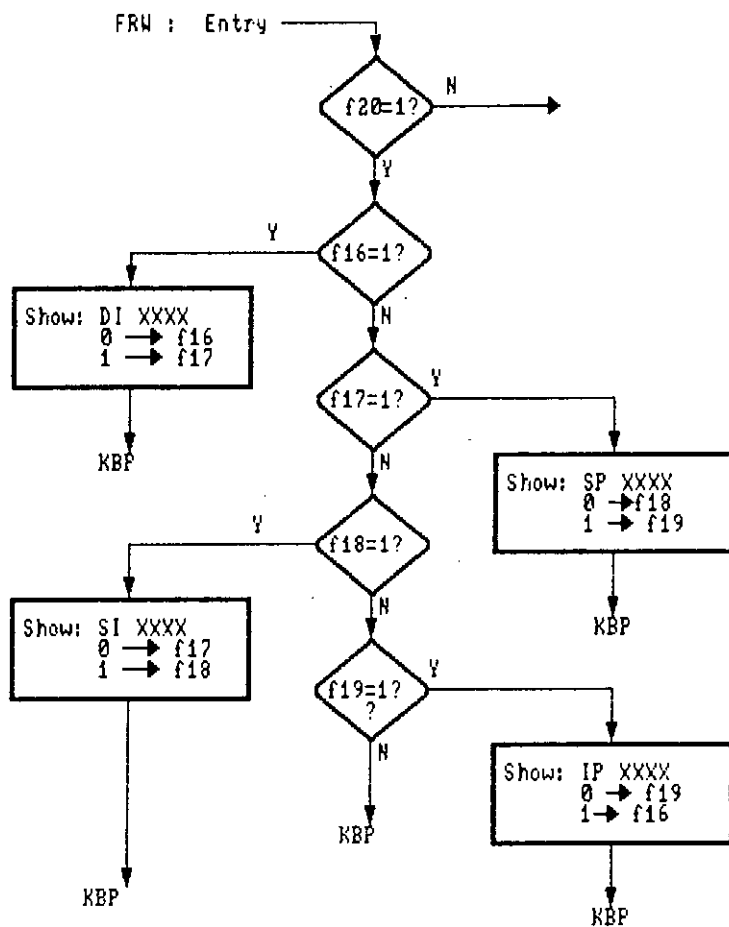


Fig - 5.4.6 : Flow Chart to Implement Contents Examination

5.4.7 Exam/Edit CS, DS, SS, ES Registers

To examine/edit these registers, the system has to be in the single step mode. Pressing CS key, will show the content of the present CS register. The contents of the remaining registers can be viewed by pressing the FRW, FRW commands. At the moment there is no routine to edit the contents of these registers.

These four segment registers are grouped together and is tagged with the flag f1F. The individual segment registers also has the identification flags as shown in the flow chart.

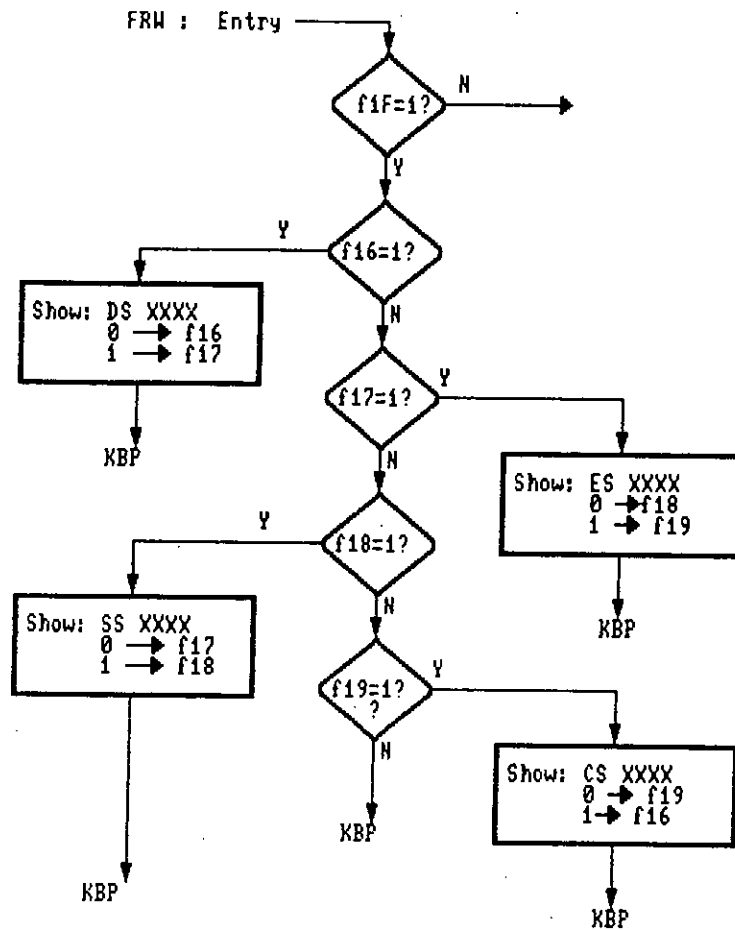


Fig - 5.4.7 : Contents Examination Flow Chart

5.4.8 Exam/Edit Flag Registers

There are two command keys in the trainer to check the contents of the flag register. Both of them work in the single step mode. The FLR key allows checking the content in hex form. The FB key allows examining the content in bit-form which are 9-active flags. The software logic involved in displaying the contents are briefly described below.

The FLR command works this way. The execution of the last instruction saved the flag register into RAM location (0043D)(0043C) [refer page-65]. Upon detection of the FLR command, the CPU just reads the word-data from these two locations and display in the 7-segment output device.

The FB command works in a bit complex way. This time the contents of the flag register is read and then broken to extract the bit values of the nine active flags.

Given below the mapping mechanism between the 16-bit contents of the flag register and the nine display digits of the trainer.

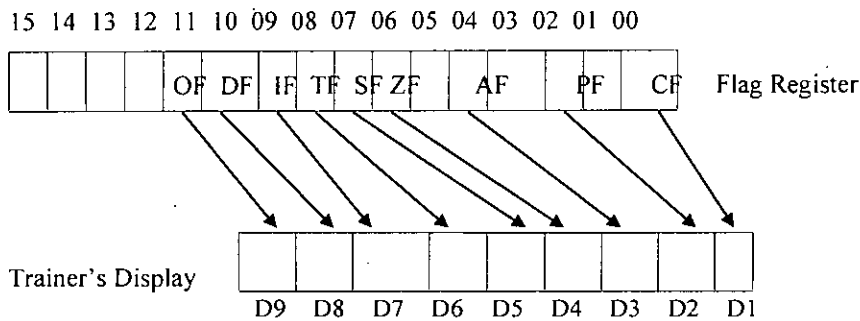


Fig - 5.4.8 : Diagram Showing the Mapping of the Status Register Content to Display

Example : Say, the execution of the last instruction has resulted with ZF = 1, TF = 1, and others are 00s.

FLR command will display

Fr 01 40 H

note : Fr = Flag Register

FB command will display

0 0 0 1 0 1 0 0 0

5.4.9 Exam/Edit Memory Contents (Memory as a simple Register)

In single stepping, the contents of various registers are examined and changed if necessary. This is the usual requirement of the users. But, sometimes, it may be required to examine and change the contents of memory locations after the execution of a memory reference instruction like "mov BYTE PTR [bx+23h], 45h". This is to sure that a data value 45h has really got deposited at memory location ds:[bx+23h]. Implementing such a routine needs a dedicated software interface between the 'Single Stepping Mechanism' and the normal part of the monitor program where the memory locations are edited using EXA and CHG commands.

In single step, EXA key will be treated just like a Register check key. This is to sure that the control will not exit the single step mode. After exam/edit of the memory content, the environment will revert to single step mode following the 'Home Key Action'.

Implementation of this routine will share many instruction codes of the non S/S part of the monitor program relating to entering 20-bit address, memory check, change, update and etc. The implementation demands a close study of the monitor program that deals with address field opening, digit entry. Given below, a flow chart indicating some hints towards creating a routine for examining/editing the memory content without exiting the single step.

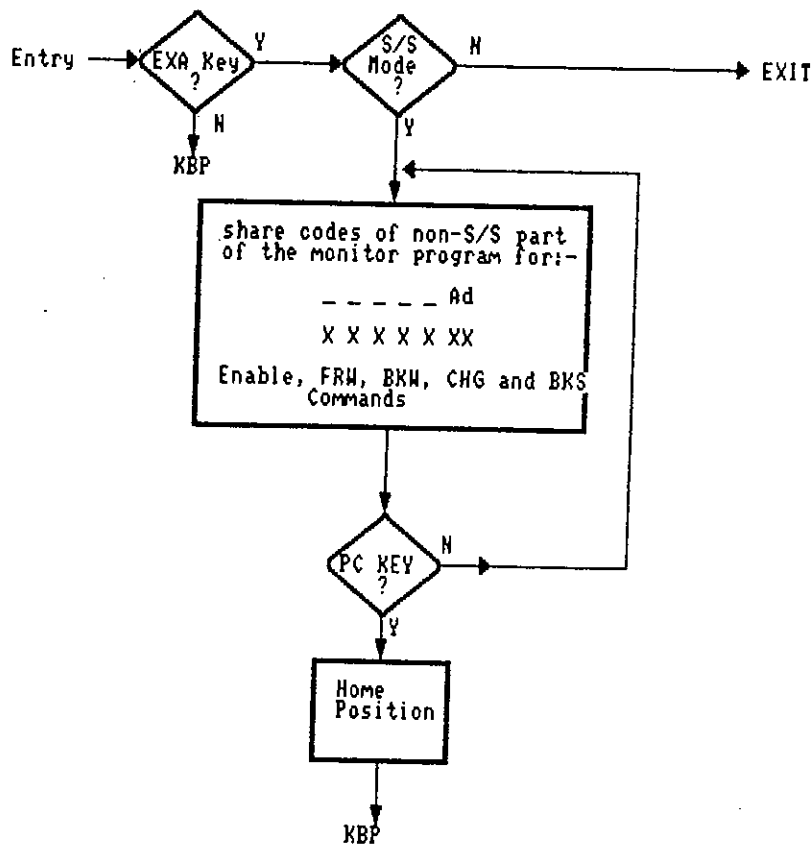


Fig - 5.4.9 : Flow Chart to Implement Mapping Memory as a Register

5.4.10 Exam/Edit Port Contents

The hex key pad of the 8086 trainer introduced in this thesis contains a key labeled as PRT. This key may be used to examine/edit the contents of variable ports. The routine may be working only in the single step mode. The design and the implementation of this routine will be very much similar to that of the routine proposed for exam/edit memory content in S/S mode (section 5.4.9). The port will just be treated as a register. FRW, BKW, CHG and BKS commands may be enabled. In order to implement these functions, most of the codes has to be borrowed from the non-S/S part of the monitor program. The routine will allow reading a port content as well as writing into a port. Given below a rough idea of the routine in the form of a flow chart.

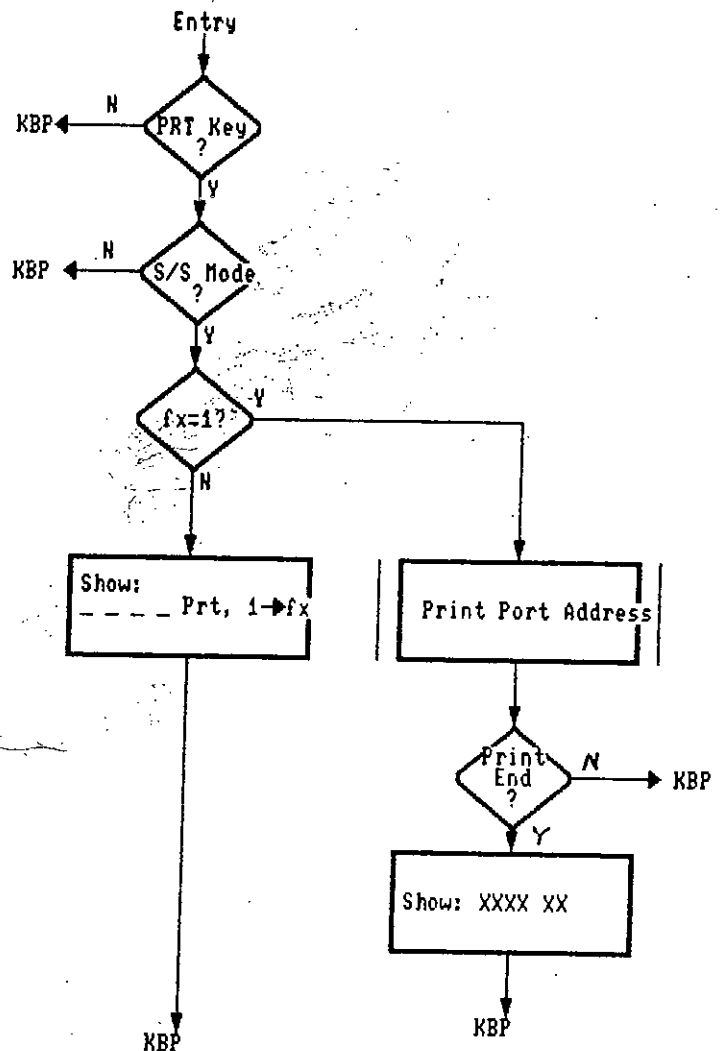


Fig - 5.4.10 : Flow Chart to Exam/Edit Port Content

5.5 Subroutines

Subroutine - 1 (SUR#1)

Purpose : This subroutine reads the scan code from the keyboard FIFO of the 8279. The SUR is called after a key has been pressed.

Entry Parameters : after a key closure

Calling at : F000:FF80

Instruction Codes:

```
FF80 - B0 40      : mov  al,40h          ;
FF82 - BA 02 00   : mov  dx,0002h         ;
FF85 - EE         ; out   dx,al          ;
FF86 - BA 00 00   : mov  dx,00000h       ;
FF89 - EC         : in   al,dx           ;
FF8A - 88 07     : mov  BYTE PTR [bx],al ; key code is save at 00400h
FF8C - CB        : ret                    ;
```

Subroutine -2 (SUR#2)

Purpose: The purpose of this subroutine is to output the message _____ Ad in the display window. The cc-codes of this message is written into T1 of the reserved RAM. Then the user has to call SUR#3 to transfer T1 onto 8279 for displaying the said message.

Entry Parameters : none

Calling at : F000:FF8E

Instruction Codes:

```
FF8E - BF 00 00   : mov  di,0000h        ;
FF91 - B1 05     : mov  cl,05h          ;
FF93 - 47       : inc  di              ;
FF94 - C6 41 45 08 : mov  BYTE PTR [bx+45h],08h ; start writing '_' at T1
FF98 - FE C9     : dec  cl              ;
FF9A - 75 F7     : jnz  F000:FF93       ;
FF9C - 47       : inc  di              ;
FF9D - C6 47 45 77 : mov  BYTE PTR [bx+45h],77h ; writinf 'A' at D4 of T1
FFA1 - C6 47 44 5E : mov  BYTE PTR [bx+44h],5Eh ; writing 'd' at D3 of T1
FFA5 - C6 47 43 00 : mov  BYTE PTR [bx+43h],00h ; balkn at D2 of T1
FFA9 - C6 47 42 00 : mov  BYTE PTR [bx+42h],00h ; blank at D1 of T1
FFAD - 9A B6 FF 00 F0 : call SUR#3          ; transfer T1 to 8279's display RAM
FFB2 - CB        : ret                    ;
```

Subroutine - 3(SUR#3)

Purpose: Thsi subroutine transfers the contents of T1 of the reserved RAM to the display RAM of the 8279.

Entry parameters : write cc-codes at T1 of the reserved RAM

Calling at : F000:FFB6

Instruction Codes:

```
FFB6 - BA 02 00   : mov  dx,0002h         ;
FFB9 - B1 90     : mov  al,90h          ;
```

```

FFBB - EE      : out    dx,al      ;
FFBC - BA 00 00 : mov    dx,0000h    ;
FFBF - B1 09    : mov    cl,09h      ;
FFC1 - BB 4B 04 : mov    bx,044Bh    ;
FFC4 - 4B       : dec    bx          ;
FFC5 - 8A 07    : mov    al,BYTE PTR [bx] ;
FFC7 - EE       : out    dx,al      ;
FFC8 - FE C9    : dec    cl          ;
FFCA - 75 F8    : dec    cl          ;
FFCC - BB 00 04 : mov    bx,0400h    ;
FFCF - CB       : ret                    ;

```

Subroutine - 4 (SUR#4)

Purpose:

This subroutine allows to print hex digit at any position of the 7-segment display unit

Entry Parameters : Printing position via PP of the reserved RAM.
Number of digits to print via C2 of the reserved RAM.

Calling at : F000:F400

Instruction Codes:

```

F400 - 8A 07    : mov    al,BYTE PTR [bx]      ;
F402 - BB D0 FF : mov    bx,BYTE PTR [bx]      ;
F405 - 02 D8    : add    bl,al                 ;
F407 - 2E 8A 07 : mov    al,cs:[di]           ;
F40A - BE 4C 04 : mov    si,044Ch             ;
F40D - 8B 3C    : mov    di,[si]              ;
F40F - 88 05    : mov    [di],al              ;
F411 - 4F       : dec    di                    ;
F412 - 89 3C    : mov    [si],di              ;
F414 - BB 00 04 : mov    bx,0400h             ;
F417 - 9A B6 FF 00 F0 : call  SUR#3                ; to transfer T1 to 8279
F41C - FE 4F 40 : dec    BYTE PTR [bx+40h]     ; one more less digit to be printed
F41F - CB       : ret                    ;

```

Subroutine - 5(SUR#5)

Purpose:

To convert cc-codes of T1 of the reserved RAM into unpacked hex of the form 00,10,20,.....,F0. The result is saved at T3. For example, if D1 of T1 contains 4F (cc-code of 3), then this SUR will deposit 30 at D1 of T3.

Entry Parameters : none

Calling at : F000:F4B0

Instruction Codes:

```

F4B0 - BF 53 04 : mov    di,0453h            ;
F4B3 - B1 0A    : mov    cl,0Ah              ;
F4B5 - B4 FF    : mov    ah,0FFh             ;
F4B7 - BB 41 04 : mov    bx,0441h            ;
F4BA - 43       : inc    bx                   ;
F4BB - 47       : inc    di                    ;
F4BC - 8A 07    : mov    al,BYTE PTR [bx]     ;
F4BE - 8B F0    : mov    si,ax                ;
F4C0 - 2E 8A 04 : mov    al,BYTE PTR cs:[si]  ;

```

```

F4C3 - 88 05      : mov  BYTE PTR ds:[di],al      ;
F4C5 - FE C9      : dec  cl                          ;
F4C7 - 75 F1      : jnz  F000:F4BA                    ;
F4C9 - BB 00 04   : mov  bx,0400h                      ;
F4CC - CB         : ret

```

Subroutine - 6 (SUR#6)

Purpose:

To convert unpacked hex of T3 to packed hex. The contents of T3 is converted to packed hex and is saved at T2. For example, if D2D1 of T3 have contents 20,30; then D2D1 of T2 will have 23. And so on.

```

Entry Parameters  : none
Calling at        : F000:F4D0

```

Instruction Codes:

```

F4D0 - EA 50 F5 00 F0 : jmp  F000:F550                    ; memory allocation

F550 - C6 47 5D 00   : mov  BYTE PTR [bx+5Dh],00h        ;
F554 - B1 04         : mov  cl,04h                        ;
F556 - BF 4D 04      : mov  di,044Dh                      ;
F559 - EA D5 F4 00 F0 : jmp  F000:F4D5                    ; memory allocation

F4D5 - B5 05         : mov  ch,05h                          ;
F4D7 - BB 53 04      : mov  bx,0453h                       ;
F4DA - 43           : inc  bx                              ;
F4DB - 8B 07        : mov  ax,WORD PTR [bx]              ;
F4DD - D2 C8        : ror  al,cl                          ;
F4DF - 24 0F        : and  al,00001111b                  ;
F4E1 - 02 E0        : add  ah,al                          ;
F4E3 - 47           : inc  di                              ;
F4E5 - 88 25        : mov  BYTE PTR [di],ah              ;
F4E7 - 43           : inc  bx                              ;
F4E8 - FE          : dec  ch                              ;
F4EA - 75 EF        : jnz  F000:F4DA                    ;
F4EC - BB 00 04     : mov  bx,0400h                      ;
F4EF - CB          : ret

```

Subroutine - 7 (SUR#7)

Purpose:

To determine the segment base from the contentn of D9 of T3 of the reserved RAM. That is, if D9 of T3 contains F0, then the segment base will be determined as F000 and so on.

```

Entry Parameters  : none
Calling at        : F000:F45A

```

Instruction Codes:

```

F45A - B1 04        : mov  cl,04h                          ;
F45C - B0 FF        : mov  al,0FFh                          ;
F45E - FE C0        : inc  al                              ;
F460 - 8A E0        : mov  ah,al                          ;
F462 - D2 C4        : ror  ah,cl                          ;
F464 - 80 E4 F0     : and  ah,11110000b                    ;
F467 - 38 67 5C     : cmp  BYTE PTR [bx+5Ch],ah            ;

```

```

F46A - 74 02      : jz      F000:F46E      ;
F46C - EB F0      : jmp      F000:F45E      ;
F46E - B0 00      : mov      al,00h          ;
F470 - 8E C0      : mov      es,ax           ;
F472 - 8B 7F 50   : mov      di,WORD PTR [bx+50h] ;
F475 - BB 00 04   : mov      bx,0400h        ;
F478 - CB         : ret                     ;

```

Subroutine - 8 (SUR#8)

Purpose:

To convert packed hex into cc-codes. i.e., from T2 to T1 of the reserved RAM. This SUR uses Lookup Table - 1(LUT-1) for conversion.

```

Entry parameters : none
Calling at       : F000:F47C

```

Instruction Codes:

```

F47C - B1 04      : mov      cl,04h          ;
F47E - B5 05      : mov      ch,0h           ;
F480 - Bf 42 04   : mov      di,0442h        ;
F483 - BB 4E 04   : mov      bx,044Eh        ;
F486 - 8A 07      : mov      al;BYTE PTR [bx] ;
F488 - 8A F0      : mov      dh,al           ;
F48A - D2 C0      : ror      al,cl           ;
F48C - 24 F0      : and      al,11110000b    ;
F48E - B4 FE      : mov      ah,11111111b    ;
F490 - 8B F0      : mov      si,ax           ;
F492 - 2E 8A 04   : mov      al,BYTE PTR cs:[si] ;
F495 - 8A D0      : mov      dl,al           ;
F497 - 8A C6      : mov      al,dh           ;
F499 - 24 F0      : and      al,11110000b    ;
F49B - B4 FE      : mov      ah,11111110b    ;
F49D - 8B F0      : mov      si,ax           ;
F49F - 2E 8A 34   : mov      dh,BYTE PTR cs:[si] ;
F4A2 - 89 15      : mov      WORD PTR ds:[di],dx ;
F4A4 - 47         : inc      di              ;
F4A5 - 43         : inc      bx              ;
F4A6 - FE CD      : dec      ch              ;
F4A8 - 75 DB      : jnz      F000:F486        ;
F4AA - BB 00 04: mov      bx,0400h        ;
F4AD - CB         : ret                     ;

```

Subroutine - 9(SUR#9)

Does not exist

Subroutine - 10 (SUR#10)

Purpose:

This subroutine filters out the key BKS to prevent the monitor program from printing any ghost characters.

```

Entry Parameters : none
Calling at       : F000:F4F8

```

Instruction Codes:


```

F4F8 - B0 01      : mov  al,01h      ;
F4FA - B1 06      : mov  cl,06h      ;
F4FC - 38 07      : cmp  BYTE PTR [bx],al ;
F4FE - 74 05      : jz   F000:F505   ;
F500 - EA 0A F5 00 F0 : jmp  F000:F50A   ;
F505 - EA 51 00 00 F0 : jmp  KBP         ;
F50A - FE C9      : dec  cl          ;
F50C - 74 11      : jz   F000:F51F   ;
F50E - EA 18 F5 00 F0 : jmp  F000:F518   ;
F513 - EA 51 00 00 F0 : jmp  KBP         ;
F518 - FE C0      : inc  al          ;
F51A - EA FC F4 00 F0 : jmp  F000:F4FC   ;
F51F - CB        : ret              ;

```

Subroutine - 11 (SUR#11)

Purpose:

This subroutine is used by the single stepping mechanism. it's purpose is to transfer the contents of T2 into T4 table of the reserved RAM space.

```

Entry Parameters      : none
Calling at           : F000:F520

```

Instruction Codes;

```

F520 - 8B 47 4E      : mov  ax, [bx+4Eh] ;
F523 - 89 47 64      : mov  [bx+64h],ax  ;
F526 - 8B 47 50      : mov  ax,[bx+50h]  ;
F529 - 89 47 66      : mov  [bx+66h],ax  ;
F52C - 8A 47 52      : mov  al,BYTE PTR [bx+52h] ;
F52F - 88 47 68      : mov  BYTE PTR [bx+68h],al ;
F532 - CB           : ret              ;

```

Subroutine - 12 (SUR#12)

Purpose:

This subroutine is used by the single stepping mechanism. It's purpose is to transfer T4 into T2 of the reserved RAM. In S/S mode, while editing the RAM locations, the first byte of the instruction going to be executed might had been changed by the user. In that case, the changed value is to be updated and displayed in the 7-segment window.

```

Entry Parameters      : none
Calling at           : F000:F534

```

Instruction Codes:

```

F534 - 8B 46 64      : mov  ax,[bx+64h]  ;
F537 - 89 47 4E      : mov  [bx+4Eh],ax  ;
F53A - 8B 47 66      : mov  ax,[bx+66h]  ;
F53D - 89 47 50      : mov  [bx+50h],ax  ;
F540 - 8A 47 68      : mov  BYTE PTR [bx+68h] ;
F546 - CB           : ret              ;

```

Subroutine - 13 (SUR#13)

Purpose:

Used by monitor program while displaying the contents of AL,AH,BL,BH,.....,DL,DH. This subroutine formats the data for displaying in the desired way.

Entry Parameters : none
Calling at : F000:0C60

Instruction Codes:

```
0C60 - 88 47 4F : mov BYTE PTR [bx+4Fh],al ; xfer AH to Tx
0C63 - 9A 7C F4 00 F0 : call SUR#8 ; xlate T2 into T1
0C68 - C7 47 42 00 00 : mov WORD PTR [bx+42h],0000h ; blank D2D1 of T1
0C6D - C7 47 48 00 00 : mov WORD PTR [bx+48h],0000h ; blank D8D7 of T1
0C72 - C7 47 4A 00 00 : mov WORD PTR [bx+4Ah],0000h ; blank D10C9 of T1
0C77 - CB : ret ;
```

5.6 Stand-alone Routines

RUT - 1 : (Routine - 1)

Purpose:

To convert analog signal inputted to the ADC. The circuit diagram is given below, The data will be displayed at D2D1 position of the output device. The remaining positions of the display will remain blank.

Circuit Diagram:

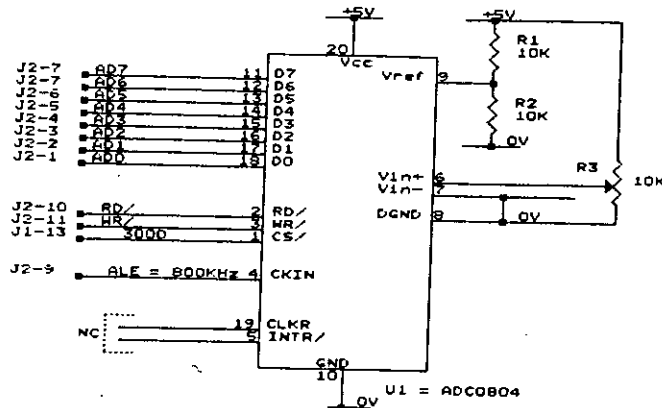


Fig - 5.6(a) : Circuit for Prototyping Analog-to-Digital conversion

Test Procedure:

1. Connect a 10K variable resistor in the breadboard of the trainer. Connect +5 and 0V across its terminals.
2. Connect the wiper of the potentiometer at the Analog-in terminal (pin-6) of the ADC .
3. Reset the trainer pressing the RST key.
4. Execute at F000:0800
5. After executing the following program. If the potentiometer is varied, the Display should vary also..

Instruction Codes:

0800 - BA 00 30	: mov dx,3000h	; point at the Control Register of ADC
0803 - EE	: out dx,al	; write function generates low going pulse.
0804 - B9 02 00	: mov cx,0002h	; to start ADC, delay parameter for convert.
0807 - E2 EE	: loop F000:0807	; wait until conversion is done
0809 - EC	: in al,dx	; reading data from the ADC
080A - 88 47 4E	: mov BYTE PTR [bx+4Eh],al	; data placed at T2 of the reserved RAM
080D - 9A 7C F4 00 F0	: call SUR#8	; convert T2 into T1
0812 - C7 47 44 00 00	: mov WORD PTR [bx+44h],0000h	; blanking positions D4D3
0817 - C7 47 46 00 00	: mov WORD PTR [bx+46h],0000h	; blanking positions D6D5 of the display
081C - C7 47 48 00 00	: mov WORD PTR [bx+48h],0000h	; blanking positions D8D7 of the display
0821 - C7 47 4A 00 00	: mov WORD PTR [bx+4Ah],0000h	; blanking positions D9 of the display
0826 - 9A B6 FF 00 F0	: call SUR#3	; transfer T1 into display RAMs of 8279
082B - B9 FF FF	: mov cx,0FFFFh	; delay para for sampling ADC
082F - E2 FE	: loop F000:082F	; wait until delay is done
0830 - EA 00 08 00 F0	: jmp F000:0800	; acquire ADC's data again and display

RUT - 2 : (Routine - 2)

Does not exist.

RUT - 3 : (Routine - 3)

Purpose: When executed, the scan code of the keys (after pressing down the keys), appear at D4D3 position of the display window. D6D5 positions show the key matrix. For example, S_43 14 means that S= switch, 43 = key K43, 14 =

scan code of the key. This routine uses LUT-4 for conversion processes. This is to note that the present routine assumes that there is missing keys in the 1st row. Therefore the physical 1st row of the key pad appears as 2nd row while displaying the key's scan code.

Execution at : F000:0700

Instruction codes:

```
F0700 - BA 00 20      : mov   dx,2000h      ;
.....
F0830 - EA 00 08 00 F0 : jmp   F000:0800      ; loop
```

RUT - 4 : (Routine - 4)

Purpose:

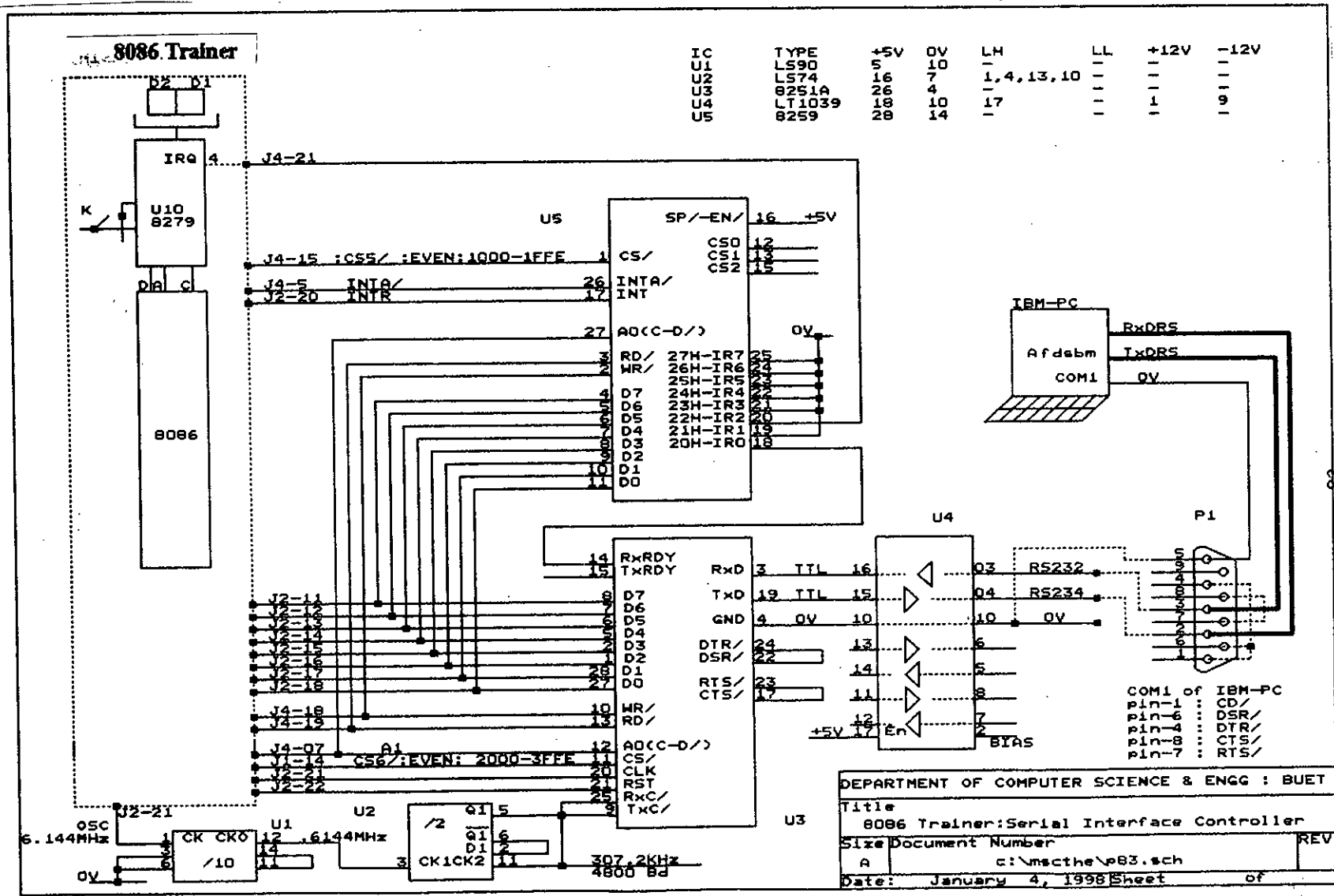
To establish communication between the 8086 Trainer and the IBM-PC. The communication will be implemented using COM1 port of the IBM-PC. Data transmitted from the trainer keyboard will be received by the IBM-PC and will be displayed in the CRT monitor. Data transmitted by the keyboard will also be received by the 8086 trainer and will be displayed at D2D1 positions of the trainer. The interface hardware between the 8086 trainer and the IBM-PC is shown at page-83. To test the functionality of this interfacing experiment, follow all the steps accurately as described below:-

01. Use hookup wires and built the circuit of page-83 in the trainer's breadboard.
02. Manually initialize vector table for 'int 20h' in order to receive the data sent by the IBM-PC. The ISR is at ROM location F07A0.
03. Manually initialize vector table for 'int 22h' in order to transmit data entered from the trainer's keyboard. An ISR is already in EPROM at location F07D0h.
04. Using Macro-Assembler, assemble the following terminal emulation program adopted from [5]. Run it.

<pre>MYSTACK SEGMENT stack DW 200 dup(0) STKTOP LABEL word MYSTACK ENDS MYCODE SEGMENT ASSUME cs:MYCODE, ss:MYSTACK START: mov ax, MYSTACK mov ss, ax mov sp, OFFSET STKTOP mov al, 00h mov dx, 0000h mov al, 11000111b int 14h sti CHKAGN: mov dx, 0000h mov ah, 03h int 14h test ah, 01h jnz RDCHAR jmp KYBD</pre>	<pre>RDCHAR: mov ah, 02h int 14h mov dl, al mov ah, 02h int 21h KYBD: mov ah, 01h int 16h jnz RDKY jmp CHKAGN RDKY: mov ah, 00h int 16h mov dx, 0000h mov ah, 01h int 14h jmp CHKAGN MYCODE ENDS END</pre>
--	---

05. Initialize the 8259 for base type code 20h for IR0. 8251 for 4800 Baud rate, no parity, 2 stop bits. Execute at F0740h
06. Press any key in the trainer. The character 'A' will appear in the CRT of the IBM-PC.
07. Press any key in the keyboard of the IBM-PC. The corresponding ASCII code will appear at D2D1 positions of the 8086 trainer.
08. Now open the EPROM locations for the initialization routines of 8259 and 8251. Disassemble them and read carefully.
09. Also open the codes of the ISRs for 'int 20h' and 'int 22h'. Read them carefully.

IC	TYPE	+5V	0V	LH	LL	+12V	-12V
U1	LS90	5	10	-	-	-	-
U2	LS74	16	7	1, 4, 13, 10	-	-	-
U3	8251A	26	4	-	-	-	-
U4	LT1039	18	10	17	-	1	9
U5	8259	28	14	-	-	-	-



DEPARTMENT OF COMPUTER SCIENCE & ENGG : BUET

Title
8086 Trainer:Serial Interface Controller

Size Document Number
A c:\msc\the\p83.sch

Date: January 4, 1998 Sheet of

Fig-5.6(b) : Circuit Diagram for RUT - 4

5.7 Data Tables

A: Lookup Table - 1 (LUT - 1)

The following ROM locations are fused with the data shown against them. This lookup table is used by SUR#4 to derive the cc-codes corresponding to a hex-digit-key on the key pad of the 8086 trainer.

<i>Location</i>	<i>= CS : OFF +</i>	<i>Key Scan Code</i>	<i>= Data Fused (cc-codes)</i>	<i>Digit</i>
FFFDE	= F000 : FFD0 +	0E	= 3F -- -- --	0
FFFDD	= F000 : FFD0 +	0D	= 06 -- -- --	1
FFF E5	= F000 : FFD0 +	15	= 5B -- -- --	2
FFFED	= F000 : FFD0 +	1D	= 4F -- -- --	3
FFFDC	= F000 : FFD0 +	0C	= 66 -- -- --	4
FFFE4	= F000 : FFD0 +	14	= 6D -- -- --	5
FFFE C	= F000 : FFD0 +	1C	= 7D -- -- --	6
FFFDB	= F000 : FFD0 +	0B	= 07 -- -- --	7
FFFE3	= F000 : FFD0 +	13	= 7F -- -- --	8
FFFE B	= F000 : FFD0 +	1B	= 6F -- -- --	9
FFFDA	= F000 : FFD0 +	0A	= 77 -- -- --	A
FFFE2	= F000 : FFD0 +	12	= 7C -- -- --	B
FFFE A	= F000 : FFD0 +	1A	= 39 -- -- --	C
FFFD9	= F000 : FFD0 +	09	= 5E -- -- --	D
FFFE1	= F000 : FFD0 +	11	= 79 -- -- --	E
FFFE9	= F000 : FFD0 +	19	= 71 -- -- --	F

B: Lookup Table - 2 (LUT - 2)

The following ROM memory locations are fused with the data shown against them. This lookup table is used by SUR#5 to convert cc-codes of T1 of the reserved RAM space to unpacked hex.

<i>Location</i>	<i>= CS : OFF +</i>	<i>CC-Codes</i>	<i>= Data Fused (cc-codes)</i>
FFF3F	= F000 : FF00 +	3F	= 00
FFF06	= F000 : FF00 +	06	= 10
FFF5B	= F000 : FF00 +	5B	= 20
FFF4F	= F000 : FF00 +	4F	= 30
FFF66	= F000 : FF00 +	66	= 40
FFF6D	= F000 : FF00 +	6D	= 50
FFF7D	= F000 : FF00 +	7D	= 60
FFF07	= F000 : FF00 +	07	= 70
FFF7F	= F000 : FF00 +	7F	= 80
FFF6F	= F000 : FF00 +	6F	= 90
FFF77	= F000 : FF00 +	77	= A0
FFF7C	= F000 : FF00 +	7C	= B0
FFF39	= F000 : FF00 +	39	= C0
FFF5E	= F000 : FF00 +	5E	= D0
FFF79	= F000 : FF00 +	79	= E0
FFF71	= F000 : FF00 +	71	= F0

C: Lookup Table - 3 (LUT - 3)

The following ROM locations are fused with the data shown against them. This lookup table is used by Sur#8 to convert packed hex of T2 of the reserved Ram into cc-code.

<i>Location</i>	<i>= CS : OFF +</i>	<i>X0</i>	<i>= Data Fused (cc-codes)</i>
FFE00	= F000 : FE00 +	00	= 3F
FFE10	= F000 : FE00 +	10	= 06
FFE20	= F000 : FE00 +	20	= 5B
FFE30	= F000 : FE00 +	30	= 4F
FFE40	= F000 : FE00 +	40	= 66
FFE50	= F000 : FE00 +	50	= 6D
FFE60	= F000 : FE00 +	60	= 7D
FFE70	= F000 : FE00 +	70	= 07
FFE80	= F000 : FE00 +	80	= 7F
FFE90	= F000 : FE00 +	90	= 6F
FFEA0	= F000 : FE00 +	A0	= 77
FFEB0	= F000 : FE00 +	B0	= 7C
FFEC0	= F000 : FE00 +	C0	= 39
FFED0	= F000 : FE00 +	D0	= 5E
FFEE0	= F000 : FE00 +	E0	= 79
FFEF0	= F000 : FE00 +	F0	= 71

D: Lookup Table - 4 (LUT-4)

The following ROM locations are fused with the data shgown against them. This lookup table is used by the stanad-alone routine RUT-3 of section-5.6.

<i>Location</i>	<i>= CS : OFF +</i>	<i>Key Scan Code</i>	<i>= Data Fused (Key Position)</i>	<i>Key</i>
FFE01	= F000 : FE00 +	01	= 11 -- -- --	K11
FFE09	= F000 : FE00 +	09	= 12 -- -- --	K12
FFE11	= F000 : FE00 +	11	= 13 -- -- --	K13
FFE19	= F000 : FE00 +	19	= 14 -- -- --	K14
FFE02	= F000 : FE00 +	02	= 21 -- -- --	K21
FFE0A	= F000 : FE00 +	0A	= 22 -- -- --	K22
FFE12	= F000 : FE00 +	12	= 23 -- -- --	K23
FFE1A	= F000 : FE00 +	1A	= 24 -- -- --	K24
FFE03	= F000 : FE00 +	03	= 31 -- -- --	K31
FFE0B	= F000 : FE00 +	0B	= 32 -- -- --	K32
FFE13	= F000 : FE00 +	13	= 33 -- -- --	K33
FFE1B	= F000 : FE00 +	1B	= 34 -- -- --	K34
FFE04	= F000 : FE00 +	04	= 41 -- -- --	K41
FFE0C	= F000 : FE00 +	0C	= 42 -- -- --	K42
FFE14	= F000 : FE00 +	14	= 43 -- -- --	K43
FFE1C	= F000 : FE00 +	1C	= 44 -- -- --	K44
FFE05	= F000 : FE00 +	05	= 51 -- -- --	K51
FFE0D	= F000 : FE00 +	0D	= 52 -- -- --	K52
FFE15	= F000 : FE00 +	15	= 53 -- -- --	K53
FFE1D	= F000 : FE00 +	1D	= 54 -- -- --	K54
FFE06	= F000 : FE00 +	06	= 61 -- -- --	K61
FFE0E	= F000 : FE00 +	0E	= 62 -- -- --	K62
FFE16	= F000 : FE00 +	16	= 63 -- -- --	K63

5.8 Interrupt Vector Table

Say, the 8086 CPU is executing some instructions of a program. This is its normal course of action. The CPU can be asked to stop the current task momentarily and spare some times to do a side job and then return back to the interrupted program. This is called 'interrupting' the CPU and this interruption can be done in the following ways:-

1. A low-to-high signal at the NMI input of the CPU. This is called externally triggered hardware interrupt.
2. A high level signal at the INTR input of the CPU. This is also called externally triggered hardware interrupt.
3. Inserting a special instruction like 'int 05h' in the program. This is called internally triggered software interrupt.
4. Some conditions are generated as a result of the execution of an instruction. This condition e.g., 'dividing an operand by zero' forces the CPU to suspend the current program execution and divert to an interrupt service routine. Interrupt 'int 00h - int 04h' are of these type.

The CPU after being interrupted, wants to know the starting address of the 'Interrupt Service Routine (ISR)'. The starting address is a 20-bit physical address and is provided in the form of CS:IP called 'Interrupt Pointer' or 'Interrupt Vector'. The values of the CS and IP have to be initialized before hand by the users before interrupting the processor.

Intel has said that its 8086 CPU does support to have ISRs starting addresses at 256 different 'Places' in the memory. Here 'Place' means a 20-bit physical address and is definable by the users. This also means that the CPU can respond to 256 different interrupts, each interrupt has its unique 20-bit starting address somewhere in the memory. Intel has named these 256 interrupts as 'int 00h - int FFh'. The starting address of the ISR for each of the interrupts is of 4-bytes wide and is composed of CS(CS_H, CS_L): IP (IP_H, IP_L).

Since, these 4-bytes data are definable by the user and could be anywhere in the memory, a suitable way of implementing this concept is to declare 4-RAM locations for each of the possible interrupt types. Hence, a total of 1024 bytes of RAM locations are required to accommodate 256 interrupts. And accordingly, Intel has allocated space 00000h - 003Fh to contain the starting addresses of the interrupts. This RAM space is called 'Interrupt vector Table'.

Q-1: How does the Interrupt Pointer do correspond to an Interrupt Type?

A-1: Intel says that if there is a low-to-high transition signal at the NMI input of the 8086 CPU. the CPU will make a jump to an interrupt service routine whose starting address could be formed from the contents of the following memory locations of the interrupt vector table:-

00008 - IP_L, 00009 - IP_H, 0000A - CS_L, 0000B - CS_H

The first RAM location is 00008h which when divided by 04h gives 2. So, Intel says that this particular type of interrupt will be called by the name 'int 02h'. And accordingly, if an interrupt called 'Overflow Interrupt (due to an arithmetic operation on two unsigned numbers, the result of which exceeds the capacity of the destination register or memory)' is named as 'int 04h', and the starting address of the ISR for this interrupt will be found at the following RAM locations of IVT.

4x4 = 16 decimal : 00010 - IP_L, 00011 - IP_H, 00012 - CS_L, 00013 - CS_H

Hence, if the interrupt type is known, the first point of the vector table can be found by multiplying the 'interrupt type' by 04h. Now, a look may be made at the Interrupt vector Table chart at page-87.

INTERRUPT VECTOR TABLE SPACE DISTRIBUTION

003FF	USER'S DEFINABLE for Software Interrupts	Int FFF
00120		Int 48h of Port-2 Handshake of 8256
00110		Int 47h
00100		Timer-4 or Timer-4/2 or Counter-4/2 of 8256
00090		Int 46h
00080		Serial Transmitt of 8256 Section-6.9.5
00070		Int 45h
00060		Serial Receiver of 8256 Section-6.9.5
00050		Int 44h
00040		Timer-3 or Counter-3 or Timer-5/3 or Counter-5/3 Section-6.9.5
00030		Int 43h
00020		EXNTR of 8256 Section-6.9.5
00010		Int 42h
00000		Timer-2 or Counter-2 or P17 Intr of 8256 Section-6.9.5
00000		Int 41h
00000		Timer-1 of 8256 of IPH Section-6.9.5
00000		Int 40h
00000		Int 3F
Recommended for Defining as Software or Hardware Interrupts		
00014	USER'S DEFINABLE	Int 05h Pre Defined Interrupt Int 04h Overflow Flag
00013	USER'S DEFINABLE	Pre Defined Interrupt Int 03h Break Point Set
00009	USER'S DEFINABLE	Pre Defined Interrupt Int 02h NMI
00007	USER'S DEFINABLE	Pre Defined Interrupt Int 01h Single Step
00005	USER'S DEFINABLE	Pre Defined Interrupt Int 00h Divide-by-zero
00000	USER'S DEFINABLE	Pre Defined Interrupt Int 00h Divide-by-zero

DEPARTMENT OF COMPUTER SCIENCE & ENG : BUET
 Title : 8086 TRAINER: INTERRUPT VECTOR TABLE
 Size Document Number : c:\macthe\SB.sch
 Date: January 4, 1998 Sheet 1 of 1

Fig - 5.8 : Data Structure for Interrupt Vector Table

5.9 Memory Space Map

The break up of the total memory space of 1 Mbytes addressable by the 8086 microprocessor is given below. The tabular form of this break up is shown at page-109.

Memory Space Range	Amount in Bytes	Used By
00000 - 003FF	1024 Bytes	Interrupt vector Table
00400 - 004FF	256 Bytes	Monitor Program]
00500 - 0FBFF	63232 Bytes	Users RAM for writing program
0FC00 - 0FFFF	1024 Bytes	Stack Memory
10000 - EFFFF	851968 Bytes	Available for external decoding
F0000 - FFFFF	65536 Bytes	EPROM Space

EPROM:

A large amount of space is blank. The monitor program and other utility programs have occupied only at best 16kbytes of EPROM space. The blank space can be used by the programmer for storing the control programs and etc.

RAM:

The RAM space has to be used with caution so that no user instruction by mistake address the reserved RAM space. There is no fence register in order to guard such unwanted accesses.

Free Undecoded Memory Space:

A space from 10000h - EFFFFh is available to the user for inserting more memory devices in the system should there arises such requirements. The user has to consult the memory decoder of the trainer (section - 4.3) for designing the new decoder. This is to ensure that the user decoder does decode the undecoded space as shown in the memory space map.

Stack Memory:

The stack memory is allocated from the same RAM chips as for the users memory. The total allocated space is 1024 bytes. The user has to take care of the limited amount of stack while calling subroutines in the program so that the stack does not penetrate the users RAM space.

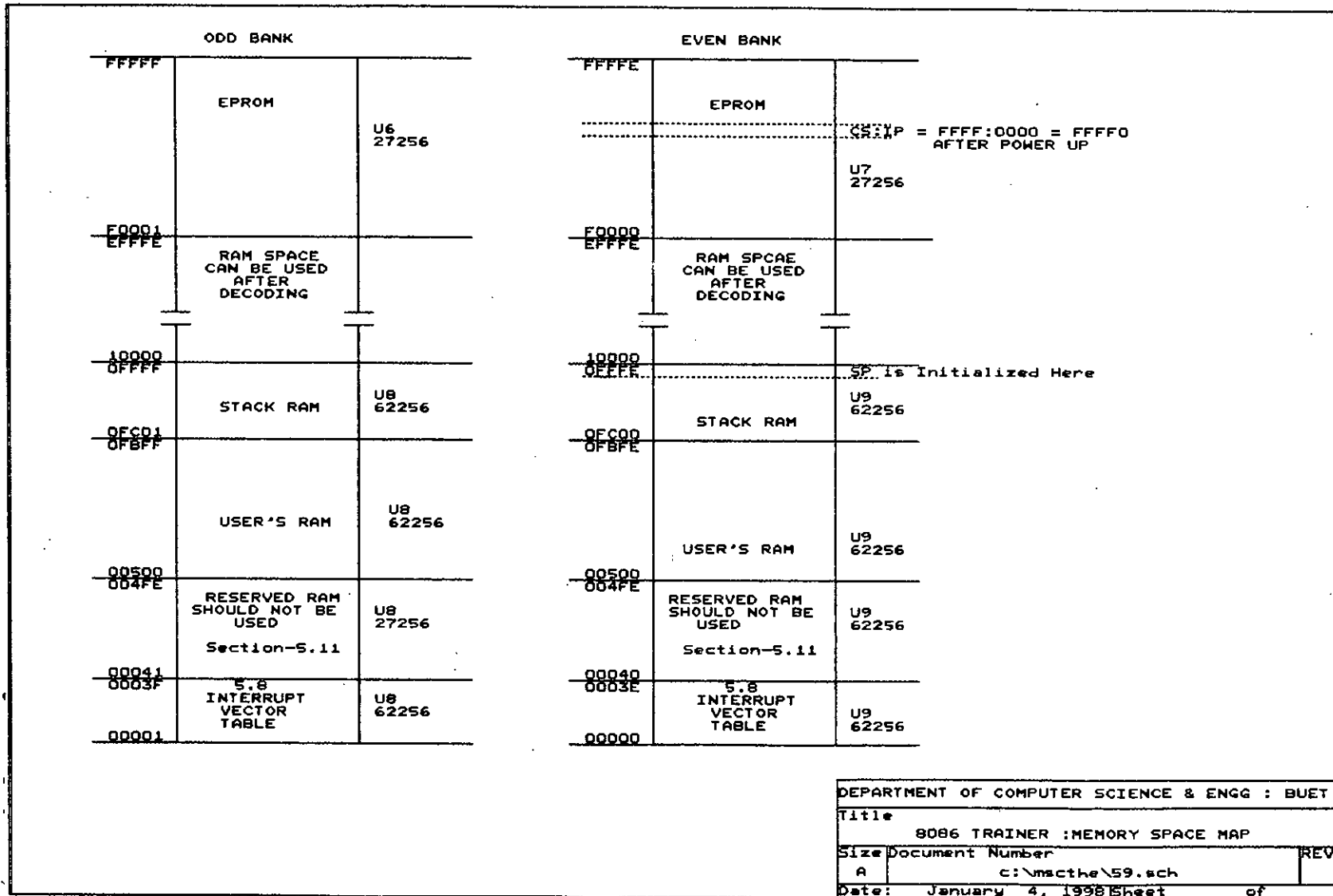


Fig - 5.9 : Memory Map Table

DEPARTMENT OF COMPUTER SCIENCE & ENGG : BUET	
Title	
8086 TRAINER : MEMORY SPACE MAP	
Size	Document Number
A	c:\msc\the\59.sch
Date:	January 4, 1998 Sheet of

5.10 Port Space Map

Refer to Figure-5.10.

Total Ports supported by 8086 microprocessor	: 64 Kbytes
Port Address assuming all the port are variable	: 0000, 0001, 0002,.....,FFFE, FFFF
Even addressed variable Port	: 0000, 0002,.....,FFFC, FFFE
Odd addressed variable Port	: 0001, 0003,....., FFFD, FFFF
Total Fixed Port supported by 8086 microprocessor	: 256 Bytes
Even addressed Fixed Port	: 00, 02,.....,FE
Odd addressed Fixed Port	: 01, 03,.....,FF

On Board Port Status

IC Designation	Type	Purpose	No of Internal Register
U10	8279	Keyboard/Display Interface	2

Port Allocation Status

Port No	Variable/Fixed Port	Assigned to
0000	Variable	Data Register of U10
0001	variable	available for decoding
0002	variable	Control Register or Status register of U10
0004 - 0FFE	even variable	Shadows of U10's registers addresses
0003 - FFFF	odd variable	available for decoding
00 - FE	even addressed Fixed	Can not used. Occupied by U10
01 - FF	odd addressed Fixed	available for decoding
1000 - 1FFE	even addressed variable	decoded and available at J1-15 of the trainer
1001 - 1FFF	odd addressed variable	available for decoding
2000 - 2FFE	even addressed variable	decoded and available at J1-14 of the trainer
2001 - 2FFF	odd addressed variable	available for decoding
3000 - 3FFE	even addressed variable	decoded available at J1-15 of the trainer
3001 - 3FFF	odd addressed variable	available for decoding
4000 - FFFE	even addressed variable	available for decoding
4001 - FFFF	odd addressed variable	available for decoding

ODD BANK PORT		EVEN BANK PORT	
FFFF	AVAILABLE FOR DECODING	FFFF	AVAILABLE FOR DECODING
		4000 3FFE	DECODED C57 / at J1-13
		3000 2FFE	DECODED C56 / at J1-14
		2000 1FFE	DECODED C55 / at J1-15
		1000 0FFE	SHADOWS
0004		0004	U10
0002		0002	CR/SR
0000		0000	DR
0001			8279

DEPARTMENT OF COMPUTER SCIENCE & ENGG : BUET	
Title	8086 TRAINER: PORT SPACE MAP
Size	Document Number
A	c:\macthe\510.sch
Date:	January 4, 1998
Sheet	of
REV	

Fig - 5.10 : Port Space Map Table

5.11 Reserved RAM Space Map

Refer to Figure - 5.11.

Space/ Space Range	Used By	Purpose
00400	Monitor Program (MP)	to store Key's Scan Code
00401 - 00421	Monitor Program	as Flags
00422 - 0043D	Single Step Routine of MP	to store and retrieve registers
0043E	Monitor Program	as Flag
0043F - 00441	Monitor Program	a counters for printing at etc.
00442 - 00445	Monitor Program	holding cc-codes of data for display at D3 - D1
00446 - 0044A	Monitor Program	holding cc-codes of data for display at D9-D4
0044B	None	-----
0044C - 0044D	Printing Routine of MP	contains the printing Position (PP) the offset part of the location of D9-D1 of T1
0044E - 0044F	MP	Hex format of the values D4-D1 of T1 DF = Data Field
00450 - 00452	MP	Hex format of D9-D5 of T1 OF= Offset Field, AF=Address Field, SF=Segment Field
00453	---	----
00454 - 0045C	MP	unpacked Hex format of D9-D1 of T1 SC = Segment Code.
0045D	MP	for conformity
0045E - 00462	MP	for executing a user program PROG EXE = program Execution
00463	--	---
00464 - 00468	Single Step Routine of MP	for keeping track of the next instruction to be executed.
00469	--	--
0046A - 00473	MP	as Flags
00474 - 004FF	restricted use by programmer	for data/code storage

6

**INTEGRATED
PERIPHERAL
MODULE**

6.1 Introduction and Board Layout

Introduction

The trainer board should contain all the common peripheral controllers, and thus the board would certainly become dense. The PCB artwork for this highly denser board could not be manufactured locally, owing to non-availability of the required technology here. Therefore, it has been decided that a separate PCB module can be made which would contain the common peripheral controllers.

Board Layout

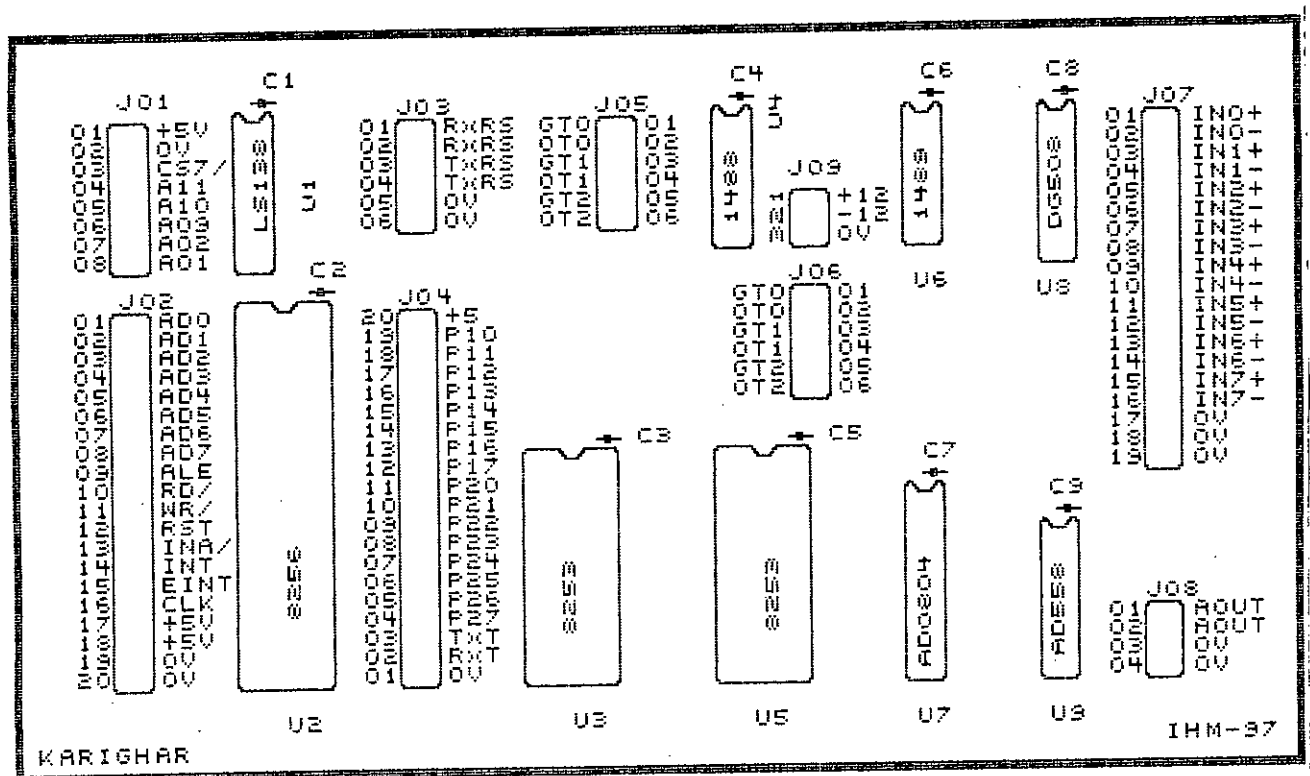


Fig - 6.1 : Board Layout Diagram

6.2 Components Description

Circuit Description	Type	Qty	Part No	Vendor
U1 3-to-8 Decoder	74LS138	1	_____	Signetics
U2 Multi-function UART	8256	1	_____	Siemens
U3,U5 3-Ch Program Interval Timer	8253	2	_____	NEC
U4 TTL-to-RS232 Converter	MC1488	1	_____	Motorola
U6 RS232-to-TTL Converter	MC1489	1	_____	Motorola
U7 Analog-to-Digital Converter	AD0804	1	_____	Signetics
U8 8-Channel Analog Multiplexer	DG508	1	_____	Harrish
U9 Digital-to-Analog Converter	AD558	1	_____	Burr-Brown
C1-C9 Decoupling capacitors	0.1uF/63V	9	_____	_____
J01 Edge Connector	16-Pin DIL	1	_____	_____
J02 Edge Connector	40-Pin DIL	1	_____	_____
J03 Edge Connector	12-Pin DIL	1	_____	_____
J04 Edge Connector	40-Pin DIL	1	_____	_____
J05 Edge Connector	12-Pin DIL	1	_____	_____
J06 Edge Connector	12-Pin DIL	1	_____	_____
J07 Edge Connector	38-Pin DIL	1	_____	_____
J08 Edge Connector	8-Pin DIL	1	_____	_____
J09 Edge Connector	6-Pin DIL	1	_____	_____

6.3 Signal Signatures

Connector	Input/Output	Signal	Remarks
J01-01,02	Input	+5V	Power Supply Live and Return
J01-03	Input	CS7/	Decoded Port Select Line From 8086 Trainer
J01-04	Input	A11	Address Line
J01-05	Input	A10	Address Line
J01-06	Input	A09	Address Line
J01-07	Input	A02	Address Line
J01-08	Input	A01	Address Line
J01-01	Input/Output	AD00	Data Line D0
J02-02	Input/Output	AD01	Data Line D1
J02-03	Input/Output	AD02	Data Line D2
J02-04	Input/Output	AD03	Data Line D3
J02-05	Input/Output	AD04	Data Line D4
J02-06	Input/Output	AD05	Data Line D5
J02-07	Input/Output	AD06	Data Line D6
J02-08	Input/Output	AD07	Data Line D7
J02-09	Input	ALE	Address Latch Signal
J02-10	Input	RD/	Read Activate Signal
J02-11	Input	WR/	Write Activate Signal
J02-12	Input	RST	Hardware RESET Signal
J02-13	Input	INTA/	Interrupt Acknowledge Signal
J02-14	Output	INT	Interrupt Signal from 8256-U2
J02-15	Input	EINT (EXINT)	External Interrupt Signal to 8256-U2
J02-16	Input	CLK	1024 Clock into 8256 (U2),8253(U3,U5),U7
J02-17,18	Input/Output	+5V	_____
J02-19,20	Input/Output	0V	_____
J03-01	Input	RxRS	Receiving Data in RS232 format
J03-02	Input	RxRS	Receiving Data in RS232 Format
J03-03	Output	TxRS	Transmitting Data in RS232 Format
J03-04	Output	TxRS	Transmitting Data in RS232 Format
J03-05	Input/Output	0V	_____
J03-06	Input/Output	0V	_____
J04-01	Input/Output	0V	_____
J04-02	Input	RxT	Receiving Data in TTL Format
J04-03	Output	TxT	Transmitting Data in TTL Format
J04-04	Input./Output	P27	1-Bit Signal from U2
J04-05	Input/Output	P26	“
J04-06	“	P25	“
J04-07	“	P24	“
J04-08	“	P23	“
J04-09	“	P22	“
J04-10	“	P21	“
J04-11	“	P20	“
J04-12	“	P17	“
J04-13	“	P16	“

Connector	Input/Output	Signal	Remarks
J04-14	“	P15	“
J04-15	“	P14	“
J04-16	“	P13	“
J04-17	“	P12	“
J04-18	“	P11	“
J04-19	“	P10	“
J04-20	Input/Output	+5V	Power Supply
J05-01	Input	GT0	Trigger Signal to Timer-0 of U3
J05-02	Output	OT0	Output signal from Timer-0 of U3
J05-03	Input	GT1	Trigger Input Timer-0 of U3
J05-04	Output	OT1	Output Signal from Timer-1 of U3
J05-05	Input	GT2	Trigger Input to Timer-2 of U3
J05-06	Output	OT2	Output signal from Timer-2 of U3
J06-01	Input	GT0	Trigger Signal to Timer-0 of U5
J06-02	Output	OT0	Output signal from Timer-0 of U5
J06-03	Input	GT1	Trigger Input Timer-0 of U5
J06-04	Output	OT1	Output Signal from Timer-1 of U5
J06-05	Input	GT2	Trigger Input to Timer-2 of U5
J06-06	Output	OT2	Output signal from Timer-2 of U5
J07-01	Input	IN0+	Analog Ch-0 Input
J07-02	“	IN0-	Analog Ch-0 Return
J07-03	“	IN1+	Analog Ch-1 Input
J07-04	Input.	IN1-	Analog Ch-1 Return
J07-05	Input	IN2+	Analog Ch-2 Input
J07-06	“	IN2-	Analog Ch-2 Return
J07-07	“	IN3+	Analog Ch-3 Input
J07-08	“	IN3-	Analog Ch-3 Return
J07-09	“	IN4+	Analog Ch-4 Input
J07-10	“	IN4-	Analog Ch-4 Return
J07-11	“	IN5+	Analog Ch-5 Input
J07-12	“	IN5-	Analog Ch-5 Return
J07-13	“	IN6+	Analog Ch-6 Input
J07-14	“	IN6-	Analog Ch-6 Return
J07-15	“	IN7+	Analog Ch-7 Input
J07-16	“	IN7-	Analog Ch-7 Return
J07-17	Input/Output	0V	_____
J07-18	“	0V	_____
J07-19	“	0V	_____
J08-01	Output	AOUT	Analog DC Out from DAC-U9
J08-02	“	“	“
J08-03.04	Input/Output	0V	_____
J09-01	Input	+12V	for RS232 Link
J09-02	Input	-12V	for RS232 Link
J09-03	Input	0V	_____

6.4 Integrated Hardware Block Diagram

Because the 8086 trainer board could not accommodate common I/O functional chips, it is decided to build a separate stand-alone board which can easily be interfaced with the trainer using hook up wires and interface connectors. The board is only 10cm x 18cm size and contains the following peripheral controllers that all we need to build any type of controller. Refer Figure-6.1.

6 - Channel Programmable Interval Timer/Counter	- 2x8253	- M2, M8
8 - Channel 8 - Bit Analog-to-Digital Converter	- AD0804, DG508	- M7, M6
8 - Bit Digital-to-Analog Converter	- AD558	- M3
Multifunction Asynchronous Receiver/Transmitter	- 8256	- M5, M4
16 - Line Parallel I/O		
Async Serial I/O with RS232 Protocol	- 488, 489	- M4
1 or 3 or 5 Channel Timers		
0 or 2 Channel Counters		
8 Level Interrupt Priority Management		

The detailed discussion of each of the above peripheral controllers will be made in the relevant sections. However, a brief coverage is made here referring to the block diagram schematic of Figure-6.6.

M8, M2 : Programmable Interval Timer/Counter

Implemented using Intel's 8253s. Each IC contains three identical 16-bit down counters. Each counter can be configured by software command to work as 'Terminal Counter'/Square Wave Generator/ One Shot. The channels are equipped with input lines for getting startup and clocks. For details, consult the manufacturer data sheet available elsewhere.

M7, M6 : 8 Channel ADC

M7 is the single channel ADC implemented using AD0804. The number of channels have been increased to eight using a multiplexer M6. The mux channel is selected by the programmer using three I/O lines of the M5. The full scale of the ADC is +5V and it is also unipolar.

M3 : Digital-to-Analog Converter

Implemented using industry standard IC of type AD558. This DAC can be operated upto at 12V at lower resolution. The analog output must derive a power section to deliver enough current to the load.

M5 : MUAR - Multifunction Asynchronous Receiver/Transmitter

This is the most and lucrative IC the Intel has ever made containing all common five functions viz., Parallel I/O, Serial I/O, Counting, Timing and Interrupt Management. The 8256 can be configured to various mode of operation and provides an easy support for the designers willing to implement various functions in a small cost effective PCB board. This IC can be used only with multiplexed bus system such as 8085, 8086 and 8751. The 8256 also contains full circuitry of TTL level serial communication.

M4 : RS232-TTL-RS232 Converter

The serial data associated with the 8256 gets converted to the required RS232 format using the Ics 488 and 489. +12V and -12V supplies are needed for the operation of these Ics and are provided using M1.

M9 : Port Decoder

Implemented using 74LS138 which has further decoded the EVEN CS7/ port group line (3000h, ..., 3FFEh) to derive the Variable Port addresses for all the peripherals of this Integrated Hardware Module.

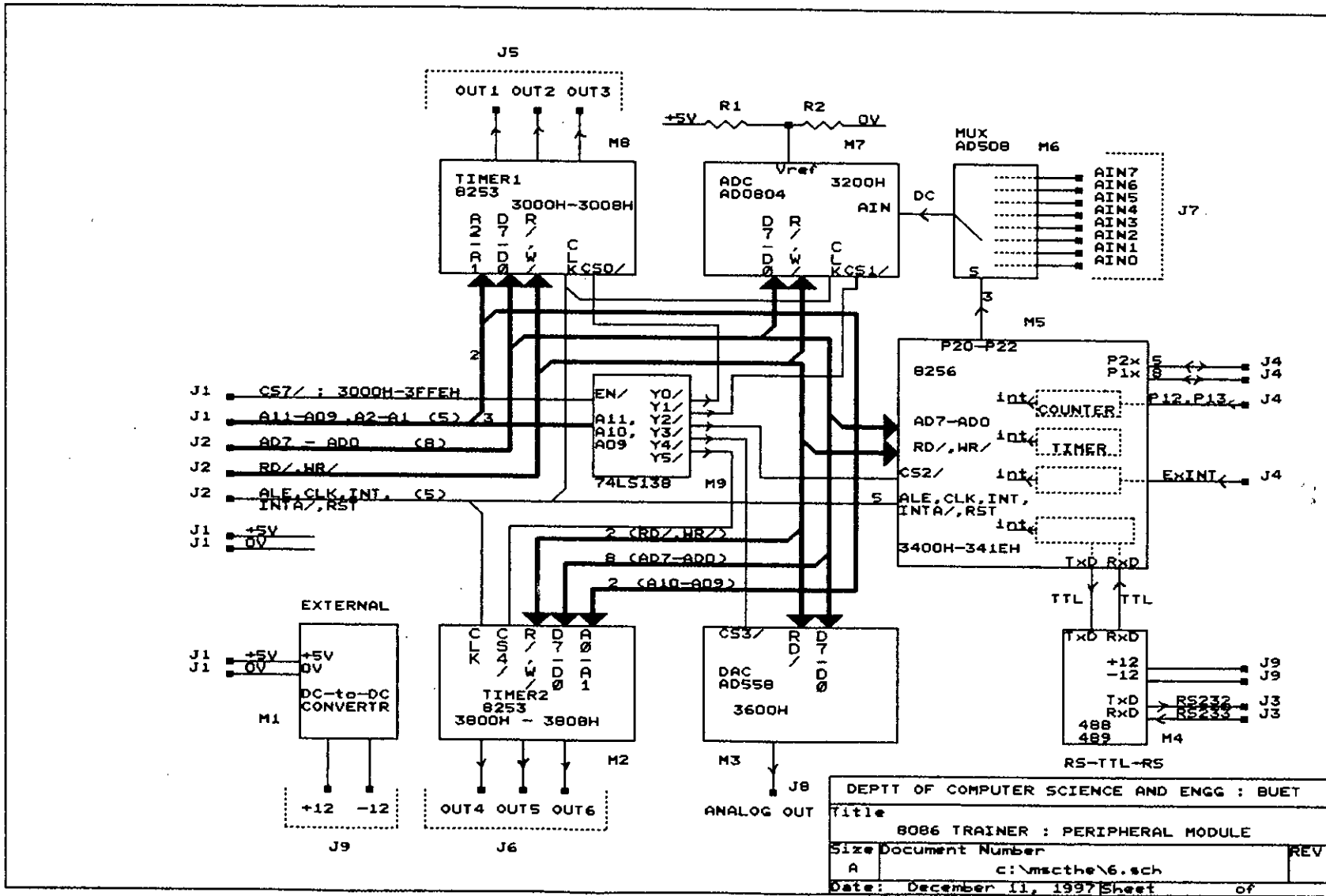


Fig-6.4 : Block Diagram for the Integrated Peripheral Module

6.5 Local Extended Port Decoder

Refer to diagram of Figure-6.5..

U1 : The Decoder

A standard 74LS138 which is a 3-to-8 line decoder. There are only 5 port chips in the peripheral module and therefore, one 74LS138 is just enough to select one port chip out of many. The base address of the decoder is (3000, 3002,....., 3FFC, 3FFE = EVEN addressed) and has come from the 8086 trainer. The base address band has already been conditioned with A0, M-I/O/, BHE/ lines. Therefore, there were not needed to decode here again. The decoder has accepted the A09, A10, A11 lines in order to derive the address band for the various peripheral controllers located in the board.

The decoder is enabled for any EVEN numbered address out of 3000,.....,3FFE. It means that the band of the decoder is 2x1024 Bytes. Each of the decoded line has a band width of 512 Bytes. The truth table of the decoder at page-103 indicates that the U1-15 (pin no 15 of U1) will go low for and address of 3000, 3002,.....,31FC, 31FE.

This is a partial decoder. A partial decoder is one which has not decoded all the unused address lines for the selection of a chip/location. For example, the U3 of page-102 has only four internal registers. So, there should be only four addresses for the chip. In practice, we have assigned 512 byte allocation. This is due to not decoding the unused address lines A3 - A8. If the unused address lines would have been decoded using more 74LS138s, then it would be called a full decoder. Full decoder does not waste memory locations but it does waste money should the shadow memory locations are not to be used.

Shadow Location:

It is created due partial decoding of the address lines. For an example, the decoder presented in this section is a partial decoder and it has created shadow locations in the following way.

A15	A14	A13	A12	A11	A10	A09	X	X	X	X	X	X	A2	A1	A0
0	0	1	1	0	0	0	←	variable				→	x	x	0

The address bits A3-A6 are not decoded neither by U1(page-100) nor by U3 (page-102). Now, the addresses of a register of the U3 will have many port numbers and all these port numbers will hit the same physical register/port.

A2	A1	Register	Port Addresses
0	0	1st	3000, 3008, 3010,.....,31F8
0	1	2nd	3002, 300A, 3012,.....,31FA
1	0	3rd	3004, 300C, 3014,.....,31FC
1	1	4th	3006, 300E, 3016,.....,31FE

Therefore, the following instructions are the same and will hit the 1st register:-

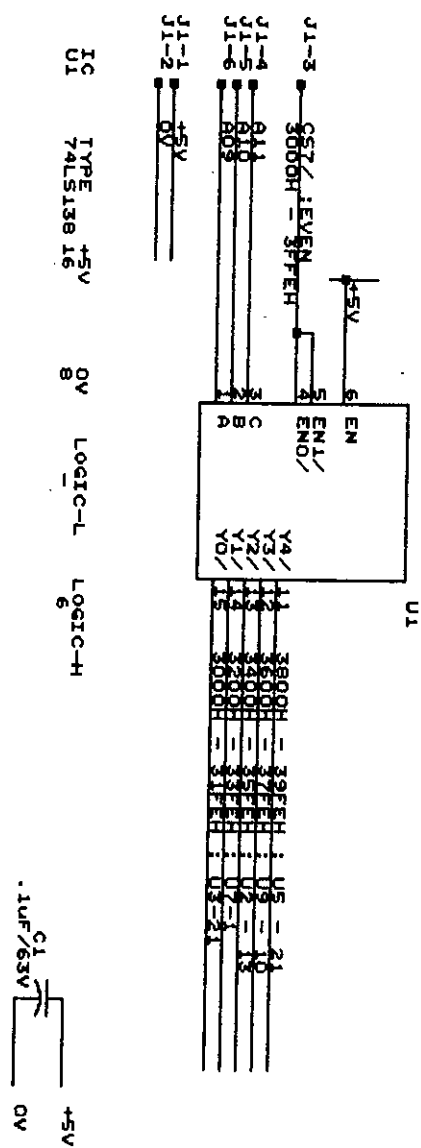
mov dx, 3000h	or	mov dx, 3008h	or	mov dx,31F8h
out dx, al		out dx,al		out dx,al

The same reasoning may also be applied for the registers of other peripheral chips.

While, using the partial decoding scheme, it is a good practice of keeping a decoding table showing the shadow locations.

PORT DECODING TRUTH TABLE

PORT ADDRESS	BASE=3000-3FFF OF U1	PORT RANGE	EN	U1's LOW	SELECTED PORT	DATA XFER	X86 M/C
3000	3000H	3000H - 30FFH	0	Y0	TYPE-1	BYTE	1
3001	3001H	3001H - 30FFH	0	Y1	TYPE-1	BYTE	1
3002	3002H	3002H - 30FFH	0	Y2	TYPE-1	BYTE	1
3003	3003H	3003H - 30FFH	0	Y3	TYPE-1	BYTE	1
3004	3004H	3004H - 30FFH	0	Y4	TYPE-1	BYTE	1
3005	3005H	3005H - 30FFH	0	Y5	TYPE-1	BYTE	1
3006	3006H	3006H - 30FFH	0	Y6	TYPE-1	BYTE	1
3007	3007H	3007H - 30FFH	0	Y7	TYPE-1	BYTE	1
3008	3008H	3008H - 30FFH	0	Y8	TYPE-1	BYTE	1
3009	3009H	3009H - 30FFH	0	Y9	TYPE-1	BYTE	1
300A	300AH	300AH - 30FFH	0	Y10	TYPE-1	BYTE	1
300B	300BH	300BH - 30FFH	0	Y11	TYPE-1	BYTE	1
300C	300CH	300CH - 30FFH	0	Y12	TYPE-1	BYTE	1
300D	300DH	300DH - 30FFH	0	Y13	TYPE-1	BYTE	1
300E	300EH	300EH - 30FFH	0	Y14	TYPE-1	BYTE	1
300F	300FH	300FH - 30FFH	0	Y15	TYPE-1	BYTE	1



DEPT OF COMPUTER SCIENCE AND ENGG : BUET
 Title : 8086 TRAINER : LOCAL EXTENDED PORT DECODER
 Size Document Number : A
 C:\msc\the\62.sch
 Date: December 5, 1997 Sheet 1 of 1

Fig - 6.5 : Schematic Diagram for the Extended Port Decoder

6.6 6 - Channel Programmable Timer

Refer to Figure-6.6 for the following description.

Introduction to 8253:

It has three identical down counters each 16-bit wide. The counter has a clock input for decrementing the counter value. It has also another input to start down counting. There is a output to indicate the user the desired EVENT has occurred after the down counting. This output line can be connected to the interrupt line of the CPU. The down counter of the 8253 can be configured to work in either of the following mode. Since, there are three counters in the single package, three different functions can be obtained simultaneously. The IPM (Integrated Peripheral Module) contains two 8253s and thus will provide 6 available functions. *In the PCB 8253's D7-D0 are wired to D0-D7. So, swap data byte before sending to 8253s. Write a subroutine and call it whenever needed.*

Mode Name	Functions Available	Meaning
0	Interrupt on Terminal Count	Known Time Delay Generation
1	Programmable One Shot	Generation Single Shot or a Series of single shots
2	Rate Generator	Divide by N counter
3	Square Wave Generator	Frequency Division of 50% duty cycle
4	Software Triggered Strobe Gating	Clock generation, one clock out of many
5	Hardware Triggered Strobe	Same as Mode-4 but retriggerable

Circuit Description:

Register Name	Port Address	Operation Mode	IC Designation
Counter - 0	3000	Read/Write	U3
Counter - 1	3002	Read/Write	U3
Counter - 2	3004	Read/Write	U3
Control Register -1	3006	Write Only	U3
Counter -3	3800	Read/Write	U5
Counter - 4	3802	Read/Write	U5
Counter - 5	3804	Read/Write	U5
Control Register - 2	3806	Write Only	U5

Testing Routine for U3:

The following program segment will test the functionality of the Counter - 0 of U3. The test routine is designed to generate 20khz signal out of 1024 Khz signal connected to the input of the counter. The frequency of the output signal can be monitored by connecting a frequency counter or an oscilloscope at terminal OUT0 of the U3.

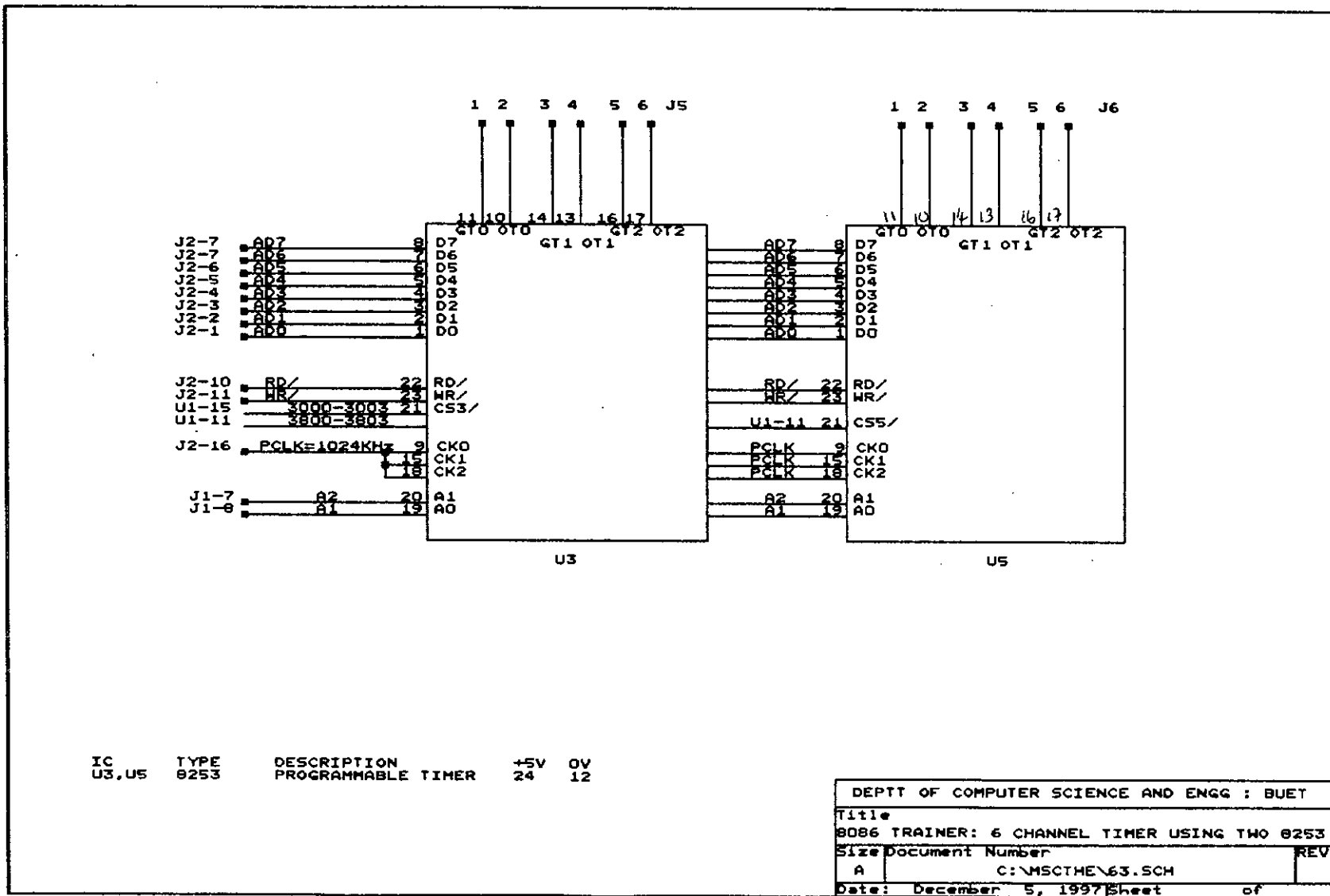
Procedures:

01. Connect the IPM and the 8086 trainer using hook up wires maintaining signal's one-to-one correspondence.
02. Connect GATE0 at +5V
03. Connect a Frequency counter or an oscilloscope at OUT0 terminal
04. Now enter and execute the following instruction codes into the 8086 trainer.

```

07000 - BA 06 30      : mov  dx,3006h          ; pointing at the control register
07003 - B0 36        : mov  al,36h           ; Counter-0 in Mode-3 operation
07005 - EE           : out  dx,al           ; mode is set to generate Squire Wave
07006 - B0 34        : mov  al, 34h         ; data for Lower 8-bit of Counter-0
07008 - BA 00 30    : mov  dx,3000h       ; pointing at Counter-0 data register
0700B - EE           : out  dx,al           ; data is sent
0700C - B0 00        : mov  al, 00h        ; data for upper 8-bit of counter-0
0700E - EE           : out  dx,al           ; now Counter-0 should generate 20khz

```



IC	TYPE	DESCRIPTION	+5V	0V
U3,U5	6253	PROGRAMMABLE TIMER	24	12

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title	
8086 TRAINER: 6 CHANNEL TIMER USING TWO 6253	
Size Document Number	
A	C:\M5CTHE\63.SCH
Date: December 5, 1997	Sheet of

Fig- 6.6 : Schematic Diagram for 6-Channel Programmable Timer

6.7 8-Channel 8-Bit Analog-to-Digital Converter

Refer to diagram of Figure-6.7.

U7 : The ADC - AD0804

It is an industry standard ADC and is fully microprocessor bus compatible. It has a reference input voltage point which must be fed with a regulated voltage and to be half of the full scale. For example, if the $V_{ref}=2.5V$, the this ADC will measure from 0 - 5 volt input range.

The ADC needs a clock of about 600khz for data conversion. The ALE signal of the 8088 is just good enough for this purpose. The ADC starts conversion whenever it receives a low-going pulse at its WR/ pin. The conversion time is about 100uS.

Register Name	Port Address	Operation Mode
Data	3200	Read Only
Convert	3200	Write Only

U8 : Analog Multiplexer

This IC can acquire analog signals from eight inputs. The channels are selected by the control signals S0,S1 and S2. The S0-S2 control signals are being delivered by the output lines of the 8256 under program control. The truth table is given below.

S2	S1	S0	Channel Selected
0	0	0	IN0
0	0	1	IN1
.....			
1	1	1	IN7

Functionality Check of the ADC and MUX

The following program segment will test the functionality of the ADC, multiplexer and the P20-P22 lines of the 8256.

Procedures:

01. Connect the wiper of a 10K potentiometer at J7-2 to feed variable DC via IN0.
02. Enter and execute the following program in the 8086 trainer. The D2D1 positions of the display window of the trainer should indicate the varying value as the pot is varied.

; select ADC channel

using hook up wires connect 0V to all the terminals J4-9, J4-10 and J4-11.

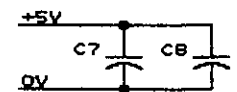
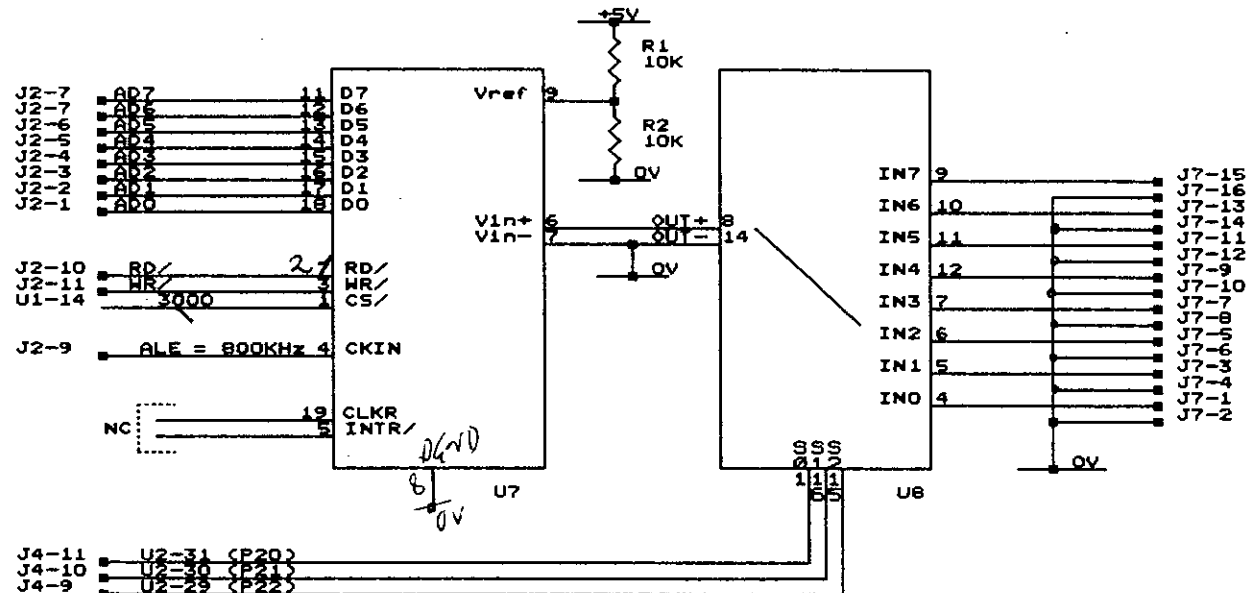
; initiate Convert signal and wait for conversion time

```
06000 - BA 00 32      : mov  dx, 3200h          ; pointing at Convert register
06003 - EE           : out  dx,al             ; to generate a low-going pulse
06004 - 90 90 90 90 : nops                    ; waiting for conversion
```

;acquire digital data and display it

```
06009 - EC           : in   al,dx              ; data is read
0600A - 88 47 4E     : mov  BYTE PTR [bx+4Eh],al ; data at T2 of the Reserved RAM
0600C - 9A 7C F4 00 F0 : call SUR#8                ; to convert T2 to T1 (hex to cc-code)
06011 - 9A B6 FF 00 F0 : call SUR#3                ; transfer T1 to 8279
06016 - B9 FF FF     : mov  cx,0FFFFh           ; delay between sample
06019 - E2 FE       : loop HERE                ;
0601B - EA 00 60 00 00 : jmp  0000:6000           ; take the next sample
```

L-ARRR001-811-ARRR010-111111-00000000



ALL CAPAS ARE .1UF/63V

IC	TYPE	DESCRIPTION	LH	LL	+5V	0V	+12V	-12V
U7	AD0804	ADC	-	-	20	10	-	-
U2	DC508	8-CHANNEL MUX	2	-	-	14	13	3

DEPT OF COMPUTER SCIENCE AND ENGG : BUET	
Title	
8086 TRAINER:8 CHANNEL ADC	
Size	Document Number
A	C:\MSCTHE\67.SCH
Date:	December 5, 1997 Sheet of

Fig - 6.7 : Schematic Diagram of 8-Channel ADC

6.8 8-Bit Digital-to-Analog Converter

Refer to Figure-6.8.

This is an industry standard digital-to-analog converter. This DAC is microprocessor bus compatible. It needs a low-going pulse at its WR/ terminal. The analog output is available at terminal 16 and should drive a power stage for delivering enough current to a load.

Register	Port Address	Mode of Operation
Data/Control	3600	Write Only

Functionality Check of the DAC

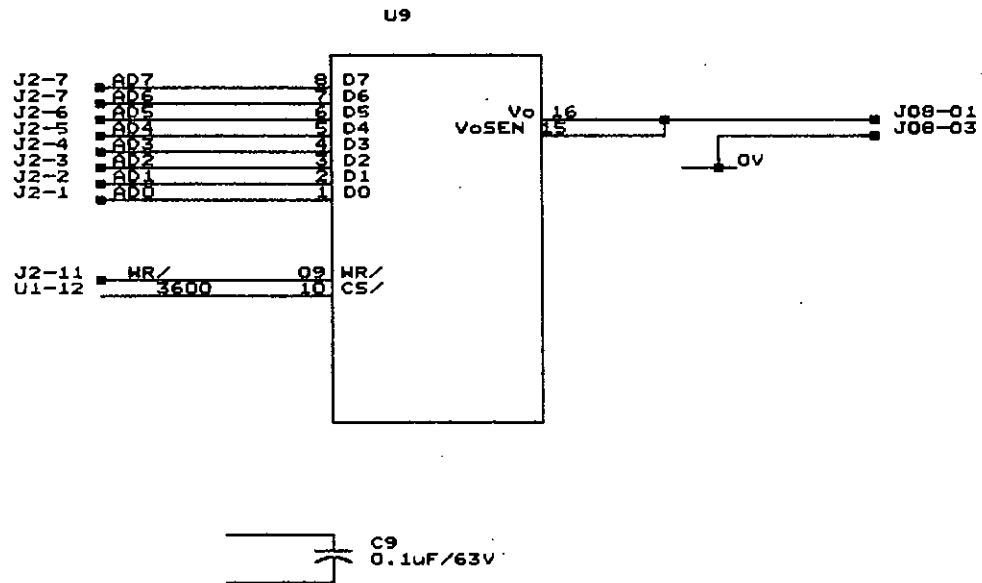
The following program will check that the DAC is working all right. The digital data will be fed by the program to its inputs (AD0-AD7) which will also be outputted at the D2D1 position of the display unit of the 8086 trainer. The user has to connect a suitable DC voltmeter across pin-15 and ground to monitor the converted DC voltage. The reader may observe that the value may saturate before the input data goes to FFh. This is not a problem from the practical point of view as because the interested region remains very near to 00- 3Fh.

; input data display at D2D1

```
05000 - B0 00      : mov   al,00h                ; initial data
05002 - 88 47 4E   : mov   BYTE PTR [bx+4Eh],al    ; data at T2 of the reserved RAM
05005 - 50        : push  ax                    ; saving data
05006 - 9A 7C F4 00 F0 : call  SUR#8                ; to convert hex code to cc-code
0500B - C7 47 44 00 00 : mov   [bx+44h],0000h        ; blanking D4D3
05010 - C7 47 46 00 00 : mov   [bx+46h],0000h        ; blanking D6D5
05015 - C7 47 48 00 00 : mov   [bx+48h],0000h        ; blanking D8D7
0501A - C7 47 4A 00 00 : mov   [bx+4Ah], 0000h       ; blanking D9
0501F - 9A B6 FF 00 F0 : call  SUR#3                ; to transfer T1 into 8279
```

; now inputting data to DAC

```
05024 - 58        : pop   ax                    ; getting data out of stack
05025 - BA 00 36   : mov   dx,3600h             ; pointing at Data/Control Register
05028 - EE        : out   dx,al                ; data is written to DAC
05029 - B3 07     : mov   bl,07h               ; delay counter
0502B - B9 FF FF   : mov   cx, 0FFFFh          ; delay parameter
0502E - E2 FE     : loop  HERE                 ; wait until cx=0
05030 - FE CB     : dec   bl                    ;
05031 - 75 F7     : jnz   F000:502B            ; wait more
05033 - FE C0     : inc   al                    ; next data for input to DAC
05035 - EA 02 50 00 00 : jmp   F000:5002            ; convert the next data byte
```



C9
0.1uF/63V

IC	TYPE	DESCRIPTION	+5V	0V
U9	AD558	8-BIT DAC	11	12,13,14

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title 8086 TRAINER: DIGITAL TO ANALOG CONVERTER	
Size A	Document Number C:\MSCTHE\68.SCH
Date: December 6, 1997	Sheet of

Fig- 6.8 : Schematic for 8-Bit Digital-to-Analog Converter

6.9.1 Parallel I/O Using 8256

Refer to schematic of Figure-6.9.1.

The 8256 is multiplexed bus compatible and it works with both the 8085 and 8086 microprocessors. It has two internal parallel I/O ports each 8-bit wide. The port lines can be individually programmed as either input or output. However, some of the I/O lines may be used for some other purposes depending on the mode of operation of the 8256. The parallel section is just a part out of five parts of the 8256.

All the five functions of the 8256 are completely independent. Therefore it is only enough to initialize the 8256 for the particular function we are interested in. In this example, we will be interested for Port-1 to operate as output. There is no need to initialize the whole chip.

Command words needed to be written into 8256 to make it fully operation as far as Port-1 is concerned. Port-1 is being configured as outputs. The P10-P17 may be connected to the trainer's LEDs for monitoring purposes. Consult the 8256's data sheet at Appendix-D.

A: Initialization Words

<i>Control words</i>	<i>Data</i>	<i>Written to</i>	<i>Address</i>	<i>Purpose</i>
Command Byte-102	Command Register-1	3400		8256 is in 8086 system
Mode Byte	00	Mode Register	3406	Port-2 as I/O, so Port-1 as I/O
Port-1 Control	FF	Port-1 Control Register	3408	Port-1 as outputs
Port-1 Data	xx	Port-1 Data Register	3410	data output

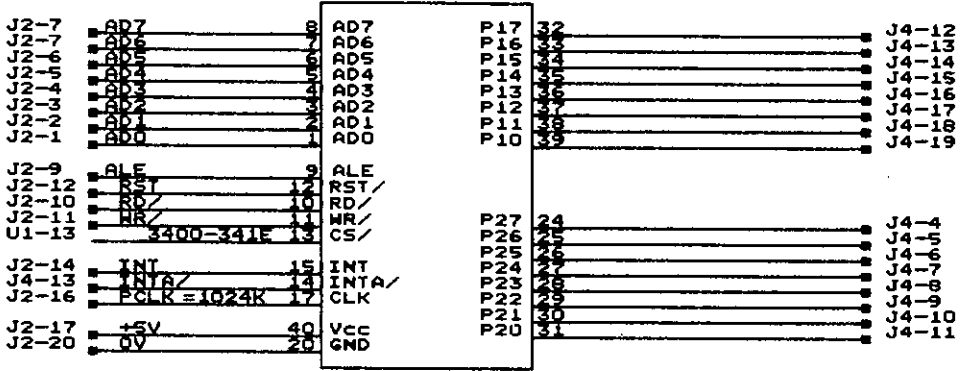
B: Now write data to Port-1

Writing to Port-1 (P17-P10)

01:	02000	- mov	dx,3400h	: BA 00 34	; getting address of Command Register-1
02:	02003	- mov	al, 02h	: B0 02	; the 8256 is in 8086 system
03:	02005	- out	dx,al	: EE	; Command Byte-1 goes to CR-1
04:					
05:	02006	- mov	dx, 3406h	: BA 06 34	; getting address of Mode Register
06:	02009	- mov	al,00h	: B0 00	; set Port-2 as I/O to ensure Port-1 as I/O
07:	0200B	- out	dx,al	: EE	; the configuration is done
08:					
09:	0200C	- mov	dx,3408h	: BA 08 34	; address of PICR (Port-1) Control Reg.
10:	0200F	- mov	al,FFh	: B0 FF	; data to set P17-P10 as outputs
11:	02011	- out	dx,al	: EE	; configuration is done
12:					
13:	02012	- mov	dx,3410h	: BA 10 34	; getting address of PIDR (Port-1 Data Reg)
14:	02015	- mov	al,FFh	: B0 FF	; writing +5V to all the port pins
15:	02017	- out	dx,al	: EE	; the connected LEDs should glow!
16:					
17:	02018	- jmp	HERE (Long Jump)	: EA 18 20 00 00	; program ends, Terminated to a loop at 17:

Execute the program either in DOP or S/S mode. Get the result.

U2



IC	TYPE	DESCRIPTION	+5V	0V	+12V	-12V
U2	8256	MUART (Asynch)	40	20	-	-

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title	
8086 TRAINER: PARALLEL I/O USING 8256	
Size Document Number	
A	C:\MCS\THE\6\1.SCH
Date: December 5, 1997	Sheet of

Fig-6.9.1 : Schematic for Parallel I/O Using 8256

6.9.2 Serial I/O Using 8256

Please refer to Figure-6.9.2 to follow the description given below.

Verification of Serial Communication Link of the 8256

- | | | | | |
|-----|---|-----------|-----|--|
| 01. | Reference circuit diagram | Fig-6.9.2 | 05. | Reset the trainer |
| 02. | Short J3-1 and J3-3 | | 06. | Change the content of location 09021h. |
| 03. | Enter and execute the following Program | | 07. | Repeat steps 03 - 07 |
| 04. | A data value 45 will be displayed at D2D1 positions of the display. | | 08. | Verify that the data is received & displayed |

Working Principle:

The serial section of the 8256 has been initialized for transmitting and receiving serial data at 4800 Baud Rate. The frame length is 11 consisting of 8-character bits, 2-stop bits, 0-parity bit, 1-start bit. The data is transmitted manually through program. The data comes back to the receiver. The receiver makes an interrupt request with IVC of 'int 44h'. The CPU jumps to the corresponding ISR and displays the received data at D2D1 positions of the display window of the trainer.

A: Initialization Words

<i>Control Words</i>	<i>Data</i>	<i>Written to</i>	<i>Address</i>	<i>Purpose</i>
Command Byte-122	Command Register-1	3400	8086 system,	2-stop bits, 8-char.
Command Byte-235	Command Register-2	3402	0-parity,	4800 Bd, CLK prscl 5
Command Byte-3E0	Command Register-3	3404	Rx ,IntACK enabled,	Normal int
Interrupt Related	14	Set Interrupt Register	340A	interrupt enabled for 'int 44h'
Interrupt Related	00	Reset Interrupt Register	340C	enabling interrupt for 'int 44h'.
Data	xx	Transmitter Register	340E	sending data
Data	xx	--	340E	reading received data

B: Now Writing data at Transmitter and reading data from the Receiver

```
ISR setting up for 'int 44h'
                                00110 - IP_L   : 00
                                00111 - IP_H   : 60
                                00112 - CS_L   : 00
                                00113 - CS_H   : 00
```

;initialization

```
09000 - mov dx, 3400h      : BA 00 34
09003 - mov al, 22h       : B0 22
09004 - out dx,al         : EE
                                ;
                                ; 8086 system, 2-stop bits, 8-character
09005 - mov dx, 3402h     : BA 02 34
09008 - mov al, 35h       : B0 35
0900A - out dx,al         : EE
                                ; address for CR-2
                                ; CLK prescaler = 5, Bd = 4800, no-parity
0900B - mov dx,3404h     : BA 04 34
0900E - mov al, E0h       : B0 E0
                                ; address dor CR-3
                                ; Rx Enabled, IntAck enabled, Normal int.
09010 - out dx,ax         : EE
09011 - mov dx, 340Ah     : BA 0A 34
                                ; address for SIR (Set Interrupt Register)
09014 - mov al, 10h       : B0 14
                                ; interrupt Level-4 is enabled for IVC 44h
09016 - out dx,al         : EE
                                ; also for Level-2 is enabled for iVC 42h
09017 - mov dx, 340Ch     : BA 0C 34
                                ; address for RIR (Reset Interrupt Register)
0901A - mov al, 00h       : B0 00
                                ; data relatingto interrupts
0901C - out dx,al         : EE
                                ; done
0901D - sti               : FB
                                ; 8086's interrupt is enabled
0901E - mov dx, 340Eh     : BA 0E 34
                                ; pointing at Transmitter Buffer
09021 - mov al, 23h       : B0 45h
                                ; data to be sent
09023 - out dx,al         : EE
                                ; done
09024 - jmp HERE         : EA 24 90 00 00
                                ; loop here
```

```

ISR for 'int 44h' : The CPU will read the data. It will be displayed at D2D1 position.
06000 - mov dx,340Eh           : BA 0E 34           : pointing at the Receiver
06003 - in al,dx               : EC           : data is in al register
06004 - mov BYTE PTR [bx+4Eh],al : 88 47 4E     : data is in T3 of the reserved RAM
06007 - call SUR#8 (F00:F47C)  : 9A 7C F4 00 F0 : to convert hex-code to cc-codes
0600C - mov [bx+44h],0000h     : C7 47 44 00 00 : blanking D4D3 positions
06011 - mov [bx+46h],0000h     : C7 47 46 00 00 : blanking D6D5 positions
06015 - mov [bx+48h],0000h     : C7 47 48 00 00 : blanking D8D7 positions
0601A - mov [bx+4Ah],0000h     : C7 47 4A 00 00 : blanking D9 position
0601F - call SUR#3 (F00:FFB6)  : 9A B6 FF 00 F0 : transferring T1 to 8279
06024 - iret                   : CF           : returning to Mainline Program

```

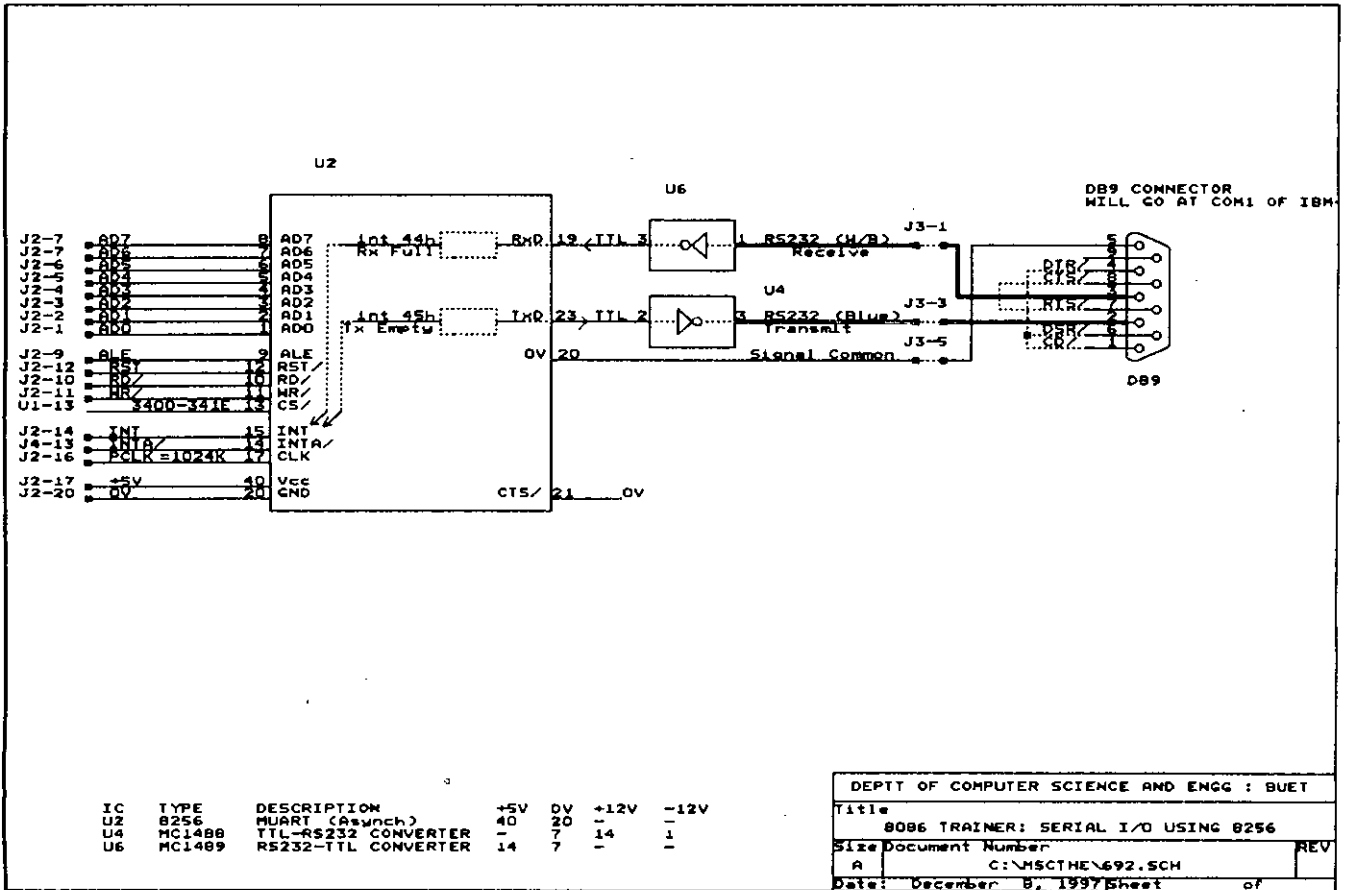


Fig - 6.9.2 : Schematic for Serial I/O Communication Using 8256

6.9.3 Timing Functions Using 8256

Schematic diagram Figure-6.9.3 shows the possible combinations of the Timer configurations and the type of interrupts they produce at terminal counts.

8-Bit Timer : Timer-1 to Timer-4

Timer-1 to Timer-4 are identical in size and operational point of view except that they are independent and produce separate interrupts on terminal counts. These Timers are down counters. They are loaded with known counts which get decremented by the internal clock. The internal clock is some fraction of the CLK input and is determined by the 'System Clock Prescaler' and 'Counter Prescaler'. These prescalers are operated by software commands. The timers are operated by either 1Khz(100uS) or 16Khz(62.5uS) clocks which is derived from the output of 'System Clock Prescaler'.

These counters are always down counting and they produce interrupts whenever the count change from 1 to 0. The initial value is to be loaded first and then enable the corresponding interrupt. These timers are non-retriggerable means they can not be loaded with new values when they are counting down. There is no way of stopping the timers while they are counting down.

8-Bit Timer : Timer-5 with external Control Input

From functional point of view, this timer is same as Timer-1 to Timer-4 except that this timer can be preloaded to an initial value through the use of the P15 line of the 8256. The loading is done at the rising edge of P15 and the timer starts counting down on the detection of falling edge of the P15 line. It is a retriggerable timer. The operation sequence is - Loading the initial value into the save register, Enabling the interrupt and then start the timer.

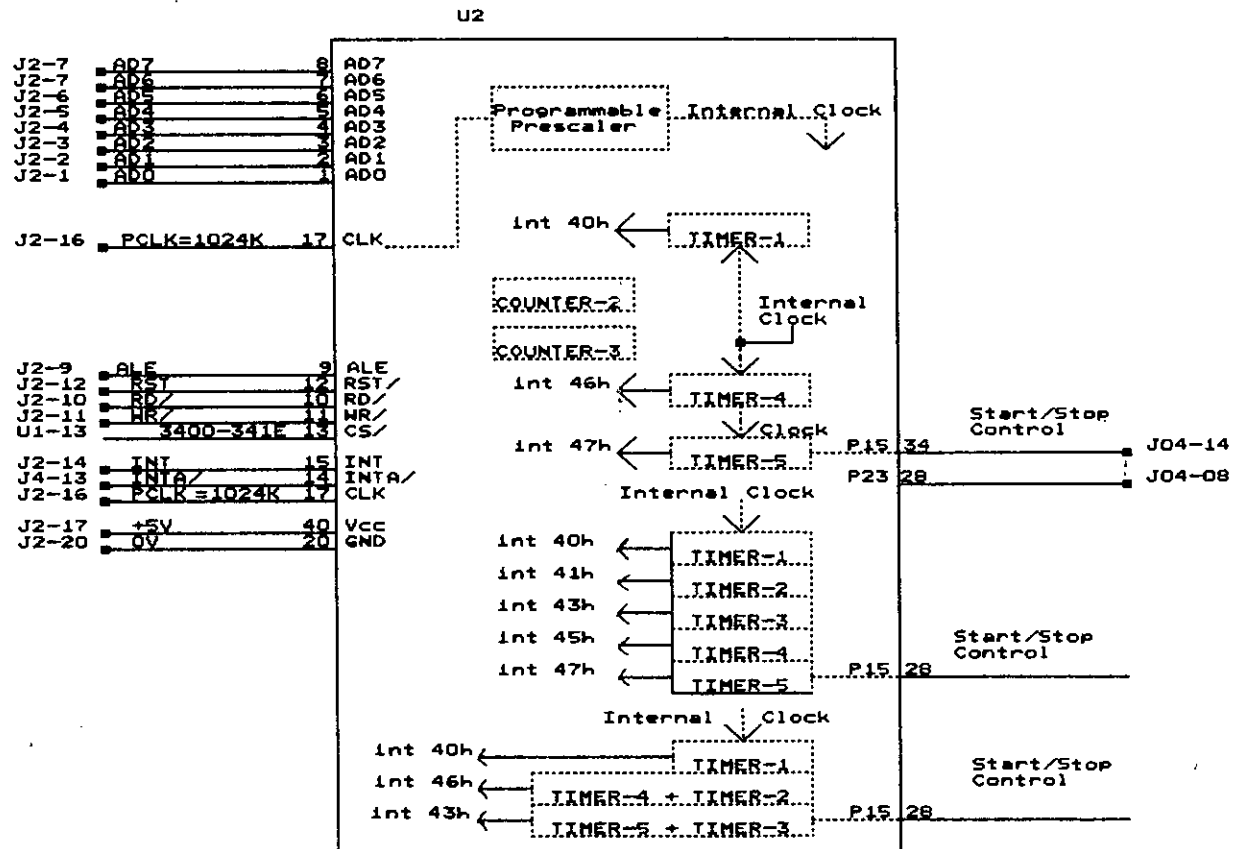
16-Bit Timer : Timer-4 & Timer-2

Timer-4 and Timer-2 can be cascaded together to make a 16-bit timer. The functional features are the same as the Timer-1 to Timer-4.

Time Delay Generation Using Timer-1

;initialization

01.	Time delay required = 50mS.	02.	1khz down counting clock
03.	Timer-1 has to be loaded with value 32h for 50mS time delay.	04.	Let us enter and execute the following program
05.	IVT set up ofr 'int 40h'	00010 - IP_L : 00	, 00011 - IP_H : 80
		00012 - CS_L : 00	, 00013 - CS_H : 00
		08000 - CF	; iret
04000 - BA 00 34	: mov dx, 3400h		; pointing at Command Register - 1
04003 - B0 03	: mov al, 03h		; 8086 system, 1Khz Timer Clock
04005 - EE	: out dx, al		; done
04006 - BA 14 34	: mov dx, 3414h		; pointing at Timer-1
04009 - B0 32	: mov al, 32h		; data to be loaded
0400B - EE	: out dx, al		; done
0400C - BA 04 34	: mov dx, 3404h		; pointing at Command Register - 3
0400F - B0 10	: mov al, 10h		; data for IntACK enabled
04010 - EE	: out dx, al		; done
04010 - BA 0A 34	: mov dx, 340Ah		; point at SIR (Set Interrupt Register)
04013 - B0 00	: mov al, 00h		; interrupt Level-0 for Timer-1 is enabled
04014 - EE	: out dx, al		; done
04015 - FB	: sti		; 8086's interrupt structure is enabled.



IC	TYPE	DESCRIPTION	+5V	0V	+12V	-12V
U2	8256	MUART (Asynch)	40	20	-	-

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title	
8086 TRAINER: TIMING FUNCTIONS USING 8256	
Size	Document Number
A	C:\MSCTHE\693.SCH
Date: December 7, 1997	Sheet of

Fig-6.9.3 : Schematic for the Timing Functions Using 8256

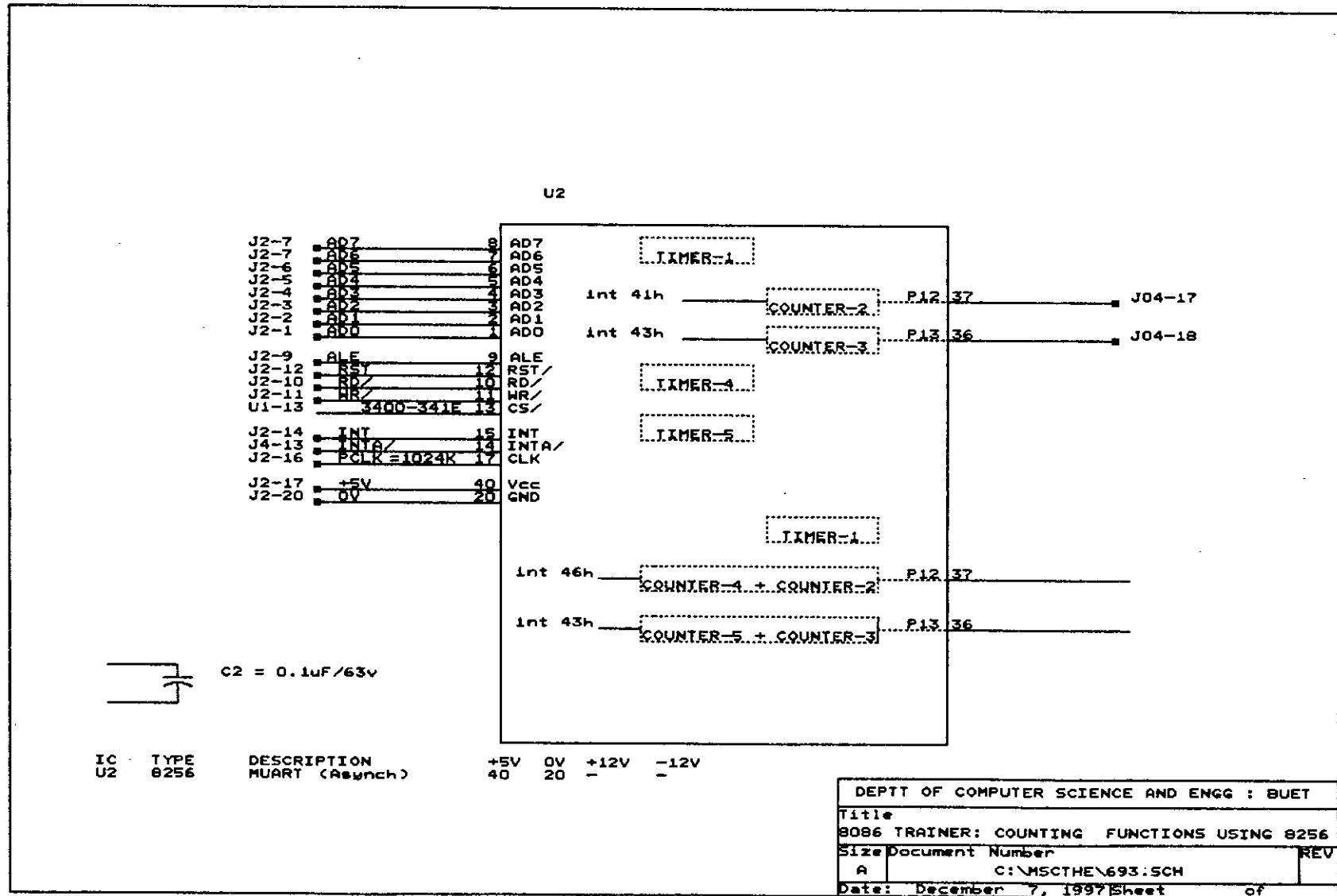


Fig- 6.9.4 : Schematic for Counting Functions Using 8256

6.9.5 Interrupt Priority Management Using 8256

Refer to Figure-6.9.5 at page-118 for the following description.

Assignment of Interrupt Levels to Interrupt Sources

Interrupt Level	Interrupt Vector 8086 Mode	Trigger Mode	Sources (Only one source can be assigned at any time)	Selection By
Highest Priority 0	int 40h	edge	Timer - 1	----
1	int 41h	edge	Counter - 2 OR Timer - 2 OR External Int Request via P17	Command Word - 1 BIT1 (bit - 2)
2	int 42h	edge	External Int Request via EXINT	-
3	int 43h	edge	Timer - 3 OR Counter - 3 OR Timer - 5 & 3 OR Counter - 5 & 3	Mode Word T35 (bit 7)
4	int 44h	edge	Serial Receiver	-
5	int 45h	edge	Serial Transmitter	-
6	int 46h	edge	Timer - 4 OR Timer - 4 & 2 OR Counter - 4 & 2	Mode Word T24 (bit 6)
7 Lowest Priority	int 47h	edge	Timer - 5 OR Port-2 with Handshaking Interrupt Request	Mode Word P2C2 - P2C0 (bits 2,1,0)

8086 - 8256 Interrupt Operation

The 8256 is told by software instruction that it is in an 8086 system. And then the INT and ITTA/ lines are directly tied with the 8086's corresponding lines. Upon receipt an interrupt request from the 8256, the 8086 issues an INTA/ pulse to inform the 8256 that it has accepted the INTR. The 8086 issues a second INTA/ pulse and the 8256 places the interrupt vector bytes 40h through 47h corresponding to the level of the interrupt to be serviced.

Interrupt Registers:

The 8256 has the 'Interrupt Mask', 'Interrupt Address', 'Interrupt Service' and a 'Priority Control' registers associated with the interrupts. Only the IMR (consisting of Set Interrupt Register and Reset Interrupt Register) and Interrupt Address Register can be accessed by the programmer.

Interrupt Priority Logic:

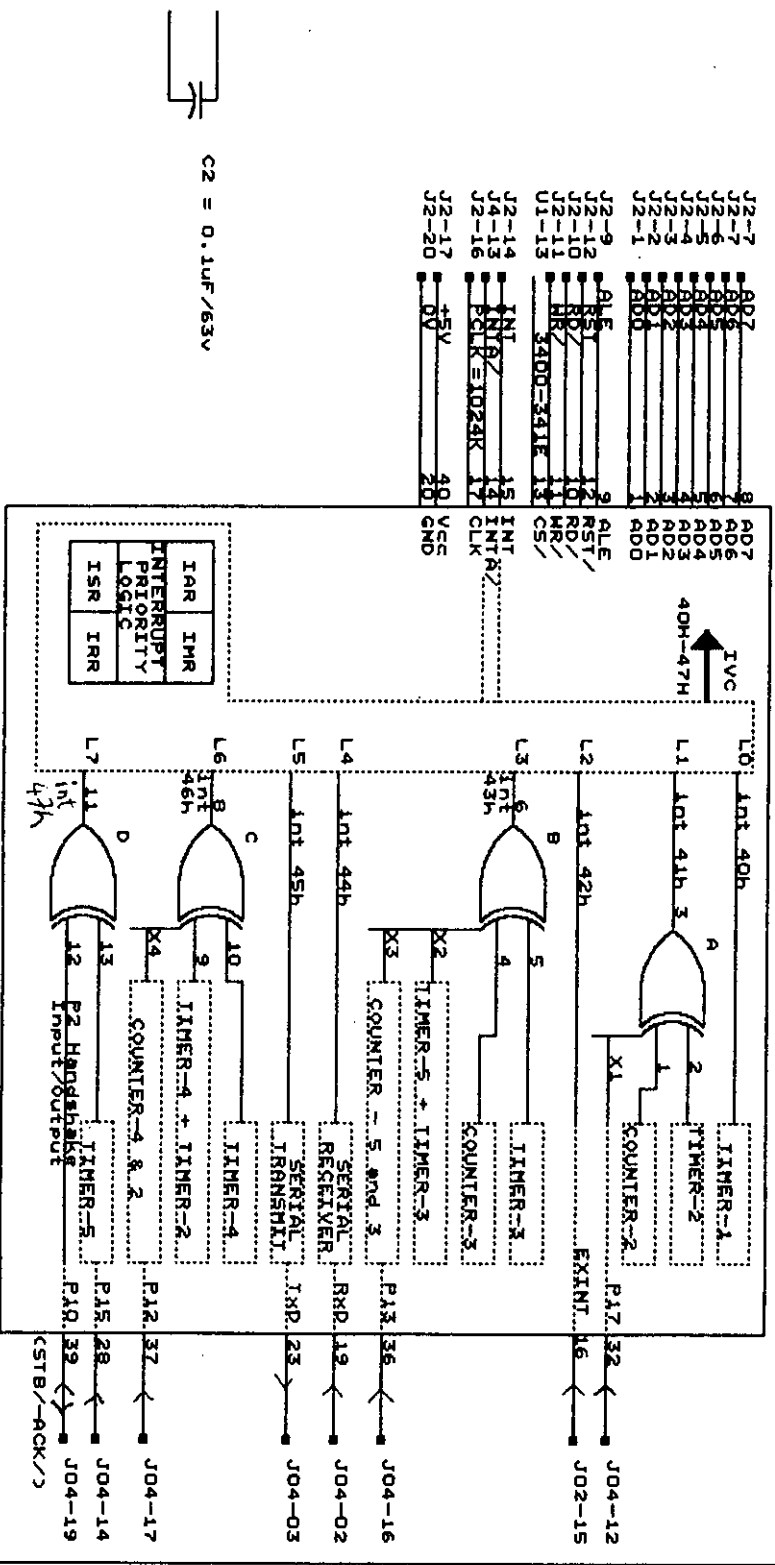
The 8256 contains special circuitry to resolve the conflicts when several interrupt requests occur at the same time, so that when the CPU acknowledges interrupt, the highest priority request is vectored to the CPU.

The individual interrupts can be enabled and disabled by manipulating the contents of the Set Interrupt Register and Reset Interrupt Register.

Interrupt Priority Management Using 8256

IC TYPE DESCRIPTION MOUNT (Asynch)

+5V DV +12V -12V



DEPT OF COMPUTER SCIENCE AND ENGG : BUET
 Title : 8086 TRAINER: PIC MANAGEMENT USING 8256
 Size Document Number : A
 C:\MSCTHE\695.SCH
 Date: December 7, 1997 Sheet 11 of 11

Fig - 6.9.5 : Schematic for Interrupt Priority Management Using 8256

7

APPLICATIONS

7.1.1 Developing and Running Program for Hardware Project

An interfacing experiment based on 8255 programmable peripheral interface is given in Figure-7.1.1. The purpose of this experiment is to give a demonstration that the developed trainer is capable of controlling a user defined hardware.

The user will enter 8-bit binary data using port-A. The data will be acquired and will be displayed to the trainers LEDs using port-B.

Register Name	Port No	Mode
Port - A	3000h	Read/Write
Port-B	3002h	Read/Write
Port-C	3006h	Read/Write
Control Port	3006h	Write Only

Instruction Code:

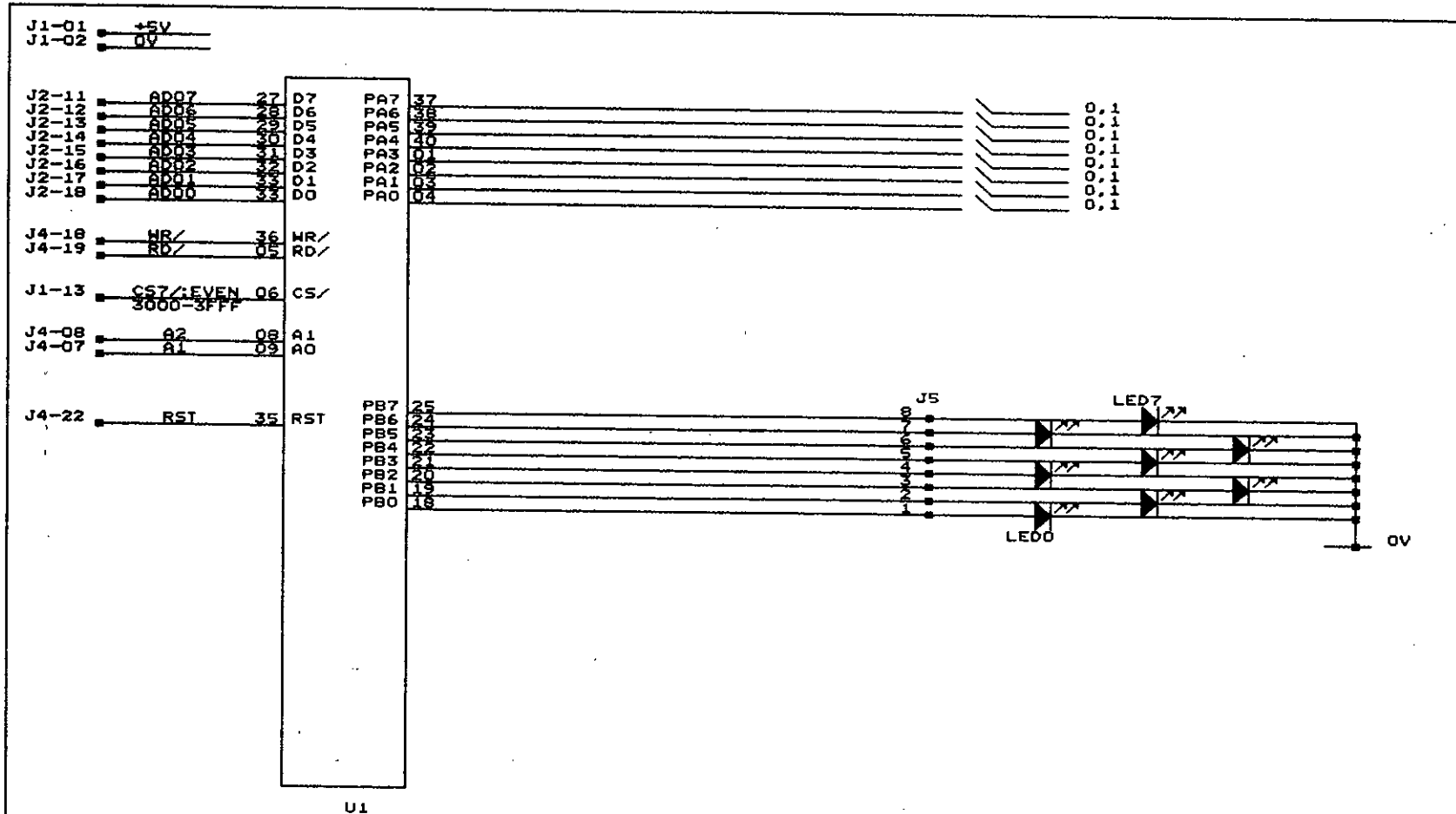
; intializing Port-A as input, Port-B as output

```
09000 - BA 06 30      : mov  dx,3006h      ;pointing at the Control register
09002 - B0 90        : mov  al,90h      ; control byte
09004 - EE           : out   dx,al      ; Port-A as input and Port-B as output
```

;acquire data of Port-A and dump over Port-B

```
09005 - BA 00 30      : mov  dx,3000h      ;pointing at Port-A
09007 - EC           : in   al,dx        ; gettoing the input data
09008 - BA 02 30      : mov  dx,3002h      ; pointing at Port-B
0900B - EE           : out  dx,al        ;writting to Port-B
```

```
0900C - EA 05 90 00 00 : jmp  0000:9005     ;get again
```



IC	TYPE	+5V	+0V
U1	8255	26	7

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title 8086 TRAINER : HARDWARE PROJECT	
Size Document Number A	REV C:\mscthe\711.sch
Date: January 10, 1998	Sheet of

Fig - 7.1.1 : Schematic for Driving LED Arrays Using 8255

7.1.2 Developing and Running Program for Software Project

Bubble Sort:

To sort the numbers in an array into descending order. The flow chart of the bubble sort is given in Figure-7.1.2. In the flow chart, N represents the number of elements in the array. A is the name of the array which has started at memory location 00481h. I is the index within the array. A bubble sort proceeds by starting at the beginning of an array and puts successive elements in descending order. After the first pass through the array the smallest element must be at the end. Therefore, during the second pass, only the N-1 elements are considered, and so on. An assembly program sequence for executing a bubble sort is given below. This program is adopted from [15, page-90] and then coded and executed by the author in the 8086 trainer.

Bubble Sort Program

No of element in the array = N = 0005h
 Elements Counter = cx register
 Array Space = 00481h - 00485h
 Values in the array before sort = 01,02,03,04,05
 Values in the array after sort = 05,04,03,02,01
 Program Execution = 09000h

Before Sort The Array is	After Sort the Array is
00481 - 01	00481 - 05
00482 - 02	00482 - 04
00483 - 03	00483 - 03
00484 - 04	00484 - 02
00485 - 05	00485 - 01

Program Codes:

```

09000 - B9 05 00 : mov  cx,0005h           ;element counter
09003 - 49          : dec  cx           ; N-1
09004 - 8B F9      : mov  di,cx       ; save count at di
09006 - BE 81 00   : mov  si,0081h    ; pointing to 1st element of the array
09009 - 8A 00      : mov  al,BYTE PTR [bx+si] ; 1st element in al register
0900B - 3A 40 01   : cmp  al,BYTE PTR [bx+si+01h] ; comparing present element with the next
0900E - 7D 05      : jge  LB1 (F000: 9015) ; swap if present element < next
09010 - 86 40 01 : xchg al,BYTE PTR [bx+si+01h] ;
09013 - 88 00      : mov  BYTE PTR [bx+si],al ; store the greater number
09015 - 83 C6 01 : add  si,0001h     ; increment index
09018 - E2 EF      : loop LOOP2 (F000:9009) ; if not end, REPEAT
0901A - 8B CF      : mov  cx,di       ; restore Count I for the outer loop
0901C - E2 E6      : loop LOOP1 (F000:9004) ; if not the final PASS, repeat
0901E - EA 00 00 00 F0 : jmp  F000:0000 ; goto keyboard
  
```

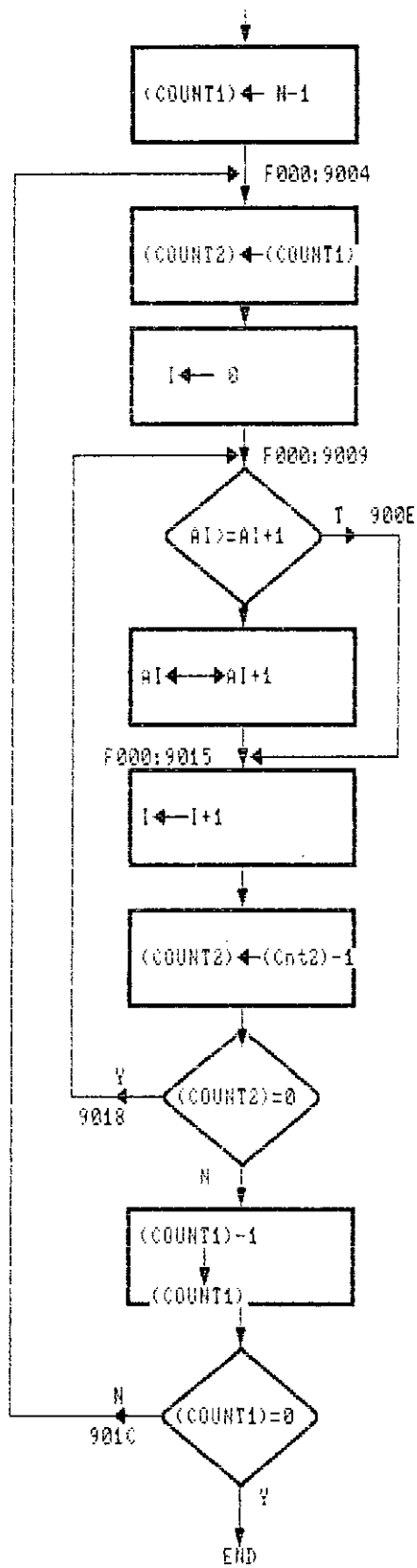


Fig - 7.1.2 : Flow Chart to Implement Bubble Sort Algorithm

7.2.1 Generating Timing Functions for Firing the McMurray-Bedford Inverter

The McMurray-Bedford complementary impulse commutation inverter is well known in the industry for its application in DC-AC converter for supplying shutdown/emergency power. Upto now, there is no recorded information of generating the complex timing functions of this type of inverter using microprocessor. Everything was done using discrete electronics. The author of this paper successfully generated the firing pulses for the thyristor bridge using 8085/8086 microprocessor. The detailed circuit diagram has been shown in Figure-7.2.1. Given below a brief account of the working principles of the interfacing circuit based on 8253 programmable interval timer.

Plaese refer to schematic diagram in Figure-7.2.1.

01. 100Hz (AA) base frequency generated using Counter-0 of 8253.
02. 50 Hz complementay signals (BB, CC) were generated using U2 (7474)
03. Oneshot signal DD generated using Counter-1 in synchronism with 100Hz signal
04. Oneshot signal EE generatd using Counter-2 in synchronism with 100Hz signal.
05. SCR2 is fired by EE.CC.20KHz. SCR4 is fired by DD.CC.20KHz. SCR 3 and SCR1 are OFF becuase BB is low.
06. Polarity changes in real time by the complemenatry signals BB and CC.
07. The timing diagram given at page-123 is self explanatory.
08. Given below the codes for generating the depicted timing functions.

Instruction Codes:

; configuring Counter-0 as square wave generator, Counter-2 and Counter-3 as Oneshots.

```

08000 - BA 06 20      : mov   dx,2006h          ; pointing at control register of 8253
08003 - B0 36        : mov   al,36h             ; Counter-0 as 100.6Hz oscillator
08004 - EE           : out   dx,al             ; done
08005 - BA 00 20     : mov   dx,2000h         ; pointing at Counter-0
08008 - B0 00        : mov   al,00h           ; value for LSByte of C-0
0800A - EE           : out   dx,al             ; done
0800B - B0 78        : mov   al,78h           ; data for MSByte of C-0
0800D - EE           : out   dx,al             ; now C-0 is running at 100.6 Hz.

0800E - BA 06 20     : mov   dx,2006h         ;pointing at Control Register
08011 - B0 72        : mov   al,72h           ; data for configuring Counter-1 as O/S
08013 - EE           : out   dx,al             ; done
08014 - BA 02 20     : mov   dx,2002h         ; pointing at Counter-1
08017 - B0 CC        : mov   al,0CCh          ; data for LSByte of C-1
08019 - EE           : out   dx,al             ; done
0801A - B0 64        : mov   al,64h           ; data for MSByte of C-1
0801C - EE           : out   dx,al             ; now counter-1 is generating DD pulses

0801D - BA 06 20     : mov   dx,2006h         ;pointing at Control Register
08020 - B0 B2        : mov   al,0B2h          ; data for configuring Counter-2 as O/S
08022 - EE           : out   dx,al             ; done
08023 - BA 04 20     : mov   dx,2004h         ; pointing at Counter-2
08026 - B0 CC        : mov   al,0CCh          ; data for LSByte of C-2
08028 - EE           : out   dx,al             ; done
08029 - B0 20        : mov   al,20h           ; data for MSByte of C-2
0802B - EE           : out   dx,al             ; now counter-2 is generating EE pulses

```

The Inverter is now generating the OUTPUT ac voltage as shown in the diagram. The duty cycle of the output voltage can be varied by changing the data contents of the Counter-1 and Counter-2.

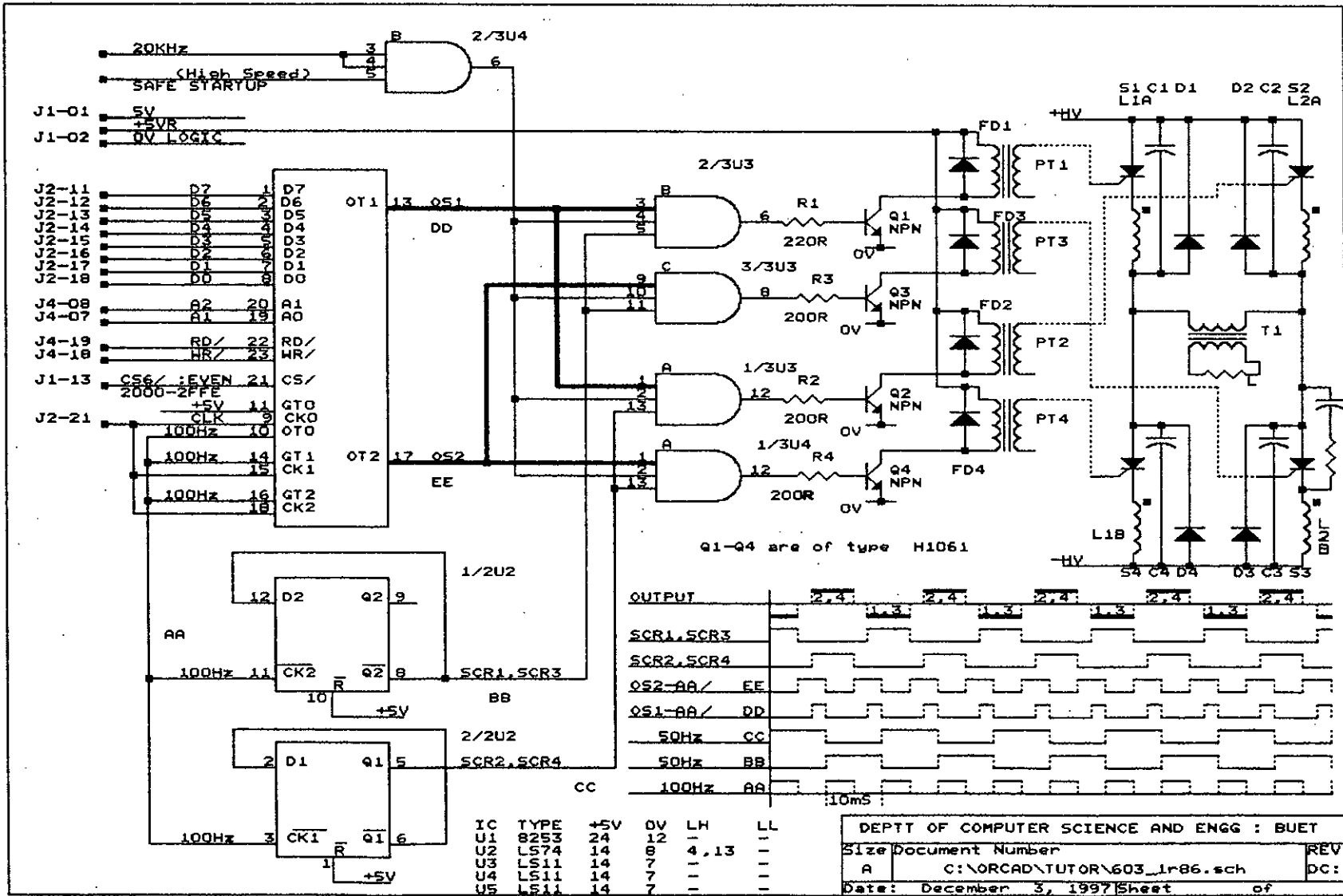


Fig - 7.2.1 : Schematic for Interfacing Bedford Inverter

7.2.2 Developing an EPROM Programmer

The detailed circuit diagram of an EPROM programmer has been given in Figure-7.2.2(d). It supports most of the common EPROMs and the truth table for their programming is given in Figure-7.2.2(a). The hardware functionality of this circuit has been fully tested using the 8086 trainer. There are various switches viz., SW1-SW5 are for connecting various programming voltages and logic levels for different EPROMs. The full documentation of the programming algorithms, the switches, the ZIF socket and etc. are also given. As a demonstration purpose, given below an example program for programming a 2716 EPROM. This routine has been tested and found working all right.

Device	Pin	Capacity	Mode	PB0	PB1	PB2	PB3	PB4	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	ProgAlg
2716	24	2K Bytes	Read	1	0	1	0	0	H/L	1	1	1	1	0	0	1	-----
2716	24	2K Bytes	Prog	0	0	1	0	0	H/L	1	0	1	1	1	0	1	50mS - N
2732A	24	4K Bytes	Read	H/L	0	1	0	0	H/L	1	1	1	1	0	0	1	-----
2732A	24	4K Bytes	Prog	H/L	0	1	0	0	H/L	0	1	0	1	0	0	1	50mS - N
2764X	28	8K Bytes	Read	H/L	H/L	0	1	0	H/L	1	1	1	1	0	0	1	-----
2764	28	8K Bytes	Prog	H/L	H/L	0	0	0	H/L	1	1	0	0	1	0	1	50mS - N
2764A	28	8k Bytes	Prog	H/L	H/L	0	0	0	H/L	1	1	0	0	1	0	0	1ms - I
27128X	28	16K Bytes	Read	H/L	H/L	H/L	1	0	H/L	1	1	1	1	0	0	1	-----
27128	28	16K Bytes	Prog	H/L	H/L	H/L	0	0	H/L	1	1	0	1	1	0	1	50mS - N
27128A	28	16K Bytes	Prog	H/L	H/L	H/L	0	0	H/L	1	1	1	0	1	0	0	1ms - I
27256X	28	32K Bytes	Read	H/L	H/L	H/L	H/L	0	H/L	1	1	1	1	0	0	1	-----
27256	28	32K Bytes	Prog	H/L	H/L	H/L	H/L	0	H/L	1	1	1	0	1	0	0	1ms - I
27128A	28	16K Bytes	Read	H/L	H/L	H/L	0	0	H/L	1	1	1	0	1	0	0	1ms - I

Fig - 7.2.2(a) : PROGRAMMING CHART USING 28-PIN ZIF SOCKET

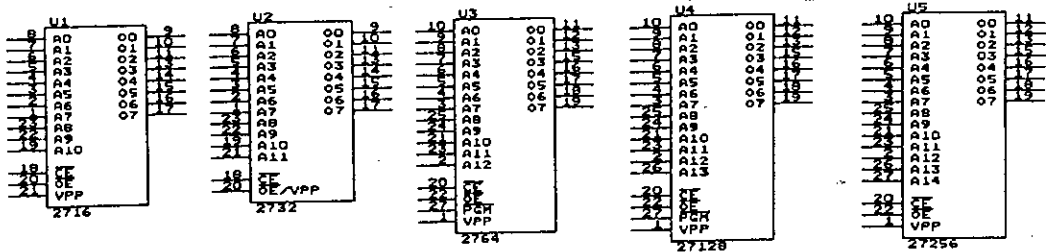


Fig-7.2.2(b) : IC's Pin Diagram

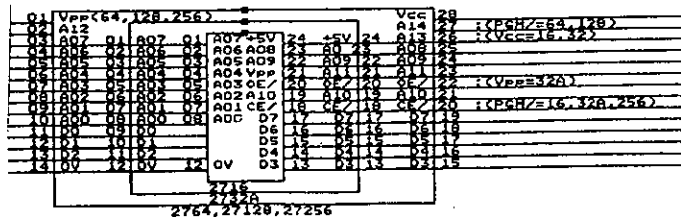


Fig - 7.2.2(c) : Zif Socket Pin Diagram

Programminng a 2716 EPROM

Refer to schematic of Figure-7.2.2(d) at page-127.

; configure PA, PB and PC of the 8255 (U1) as outputs and reset Page Register U2

```

06000 - BA 06 30      : mov  dx, 3006h      ; pointing at Control Register of 8255
06003 - B0 80        : mov  al,80h      ; data to configure PA, Pb, Pc as outputs
06004 - EE           : out  dx,al      ; data is sent at Control Register
06005 - B0 10        : mov  al,10h     ; data to reset U1
06007 - BA 02 30     : mov  dx,3002h   ; pointing at Port-B
0600A - EE           : out  dx,al      ; data is sent
0600B - B0 00        : mov  al,00h     ;
0600D - EE           : out  dx,al      ; U1 is reset and enabled
  
```

; now all controls active except Vpp and Programming Pulse and asserting 0000h memory location

```

0600E - B0 05        : mov  al,05h     ; data for Vcc =connect, Vpp=+5V
06010 - EE           : out  dx,al      ; done
06011 - BA 04 30     : mov  dx,3004h   ; pointing at Port-C
06014 - B0 3E        : mov  al,3Eh     ; data for CE/=0, OE/=1, Vpp=+5V
06016 - EE           : out  dx,al      ; done
  
```

; now assert data (45h for example to get fused inside the 2716 at location 0000h)

```

06017 - BA 00 30     : mov  dx,3000h   ; pointing at Port-A
0601A - B0 45        : mov  al,45h     ; data 45h
0601C - EE           : out  dx,al      ; done
0601D - 00 00 00 00 : nops          ;delat for stabilization
  
```

;now apply Programming Pulse and Programming Voltage

```

06022 - B0 7A        : mov  al,3Ah     ; data for CE/= +5V, Vpp=+24V
06024 - EE           : out  dx,al      ; done
  
```

;50mS delay for getting the data fused inside the 2716

```

06025 - B0 30        : mov  al,30h     ; count for 48d
06027 - B9 F2 23 : mov  cx,23F2h   ; delay loop for 1mS
0602A - E2 FE        : loop 0000:602A  ; loop here until cx=0000
0602C - FE C8        : dec  al         ; count down
0602E - 75 F5        : jnz  0000:6027  ; loop until 50mS is completed
  
```

; 1 byte data is fused. Flag down Programming Pulse and Programming Voltage

```

06030 - B0 3E        : mov  al,3Eh     ;data for CE/=0, Vpp=+5V
06032 - EE           : out  dx,al      ; done
06034 - one byte is done.
  
```

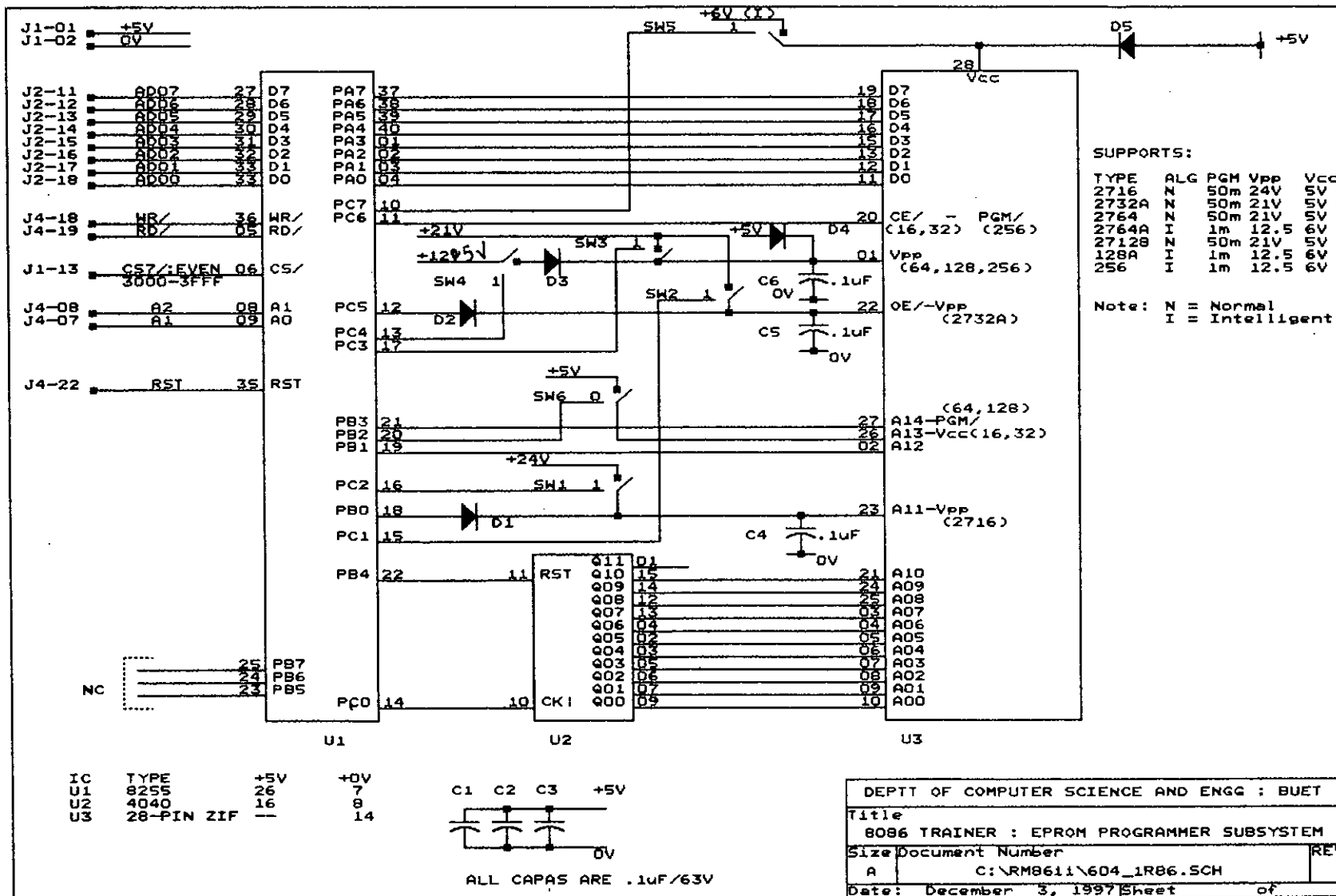


Fig- 7.2.2 (d) : Schematic for the EPROM Programmer

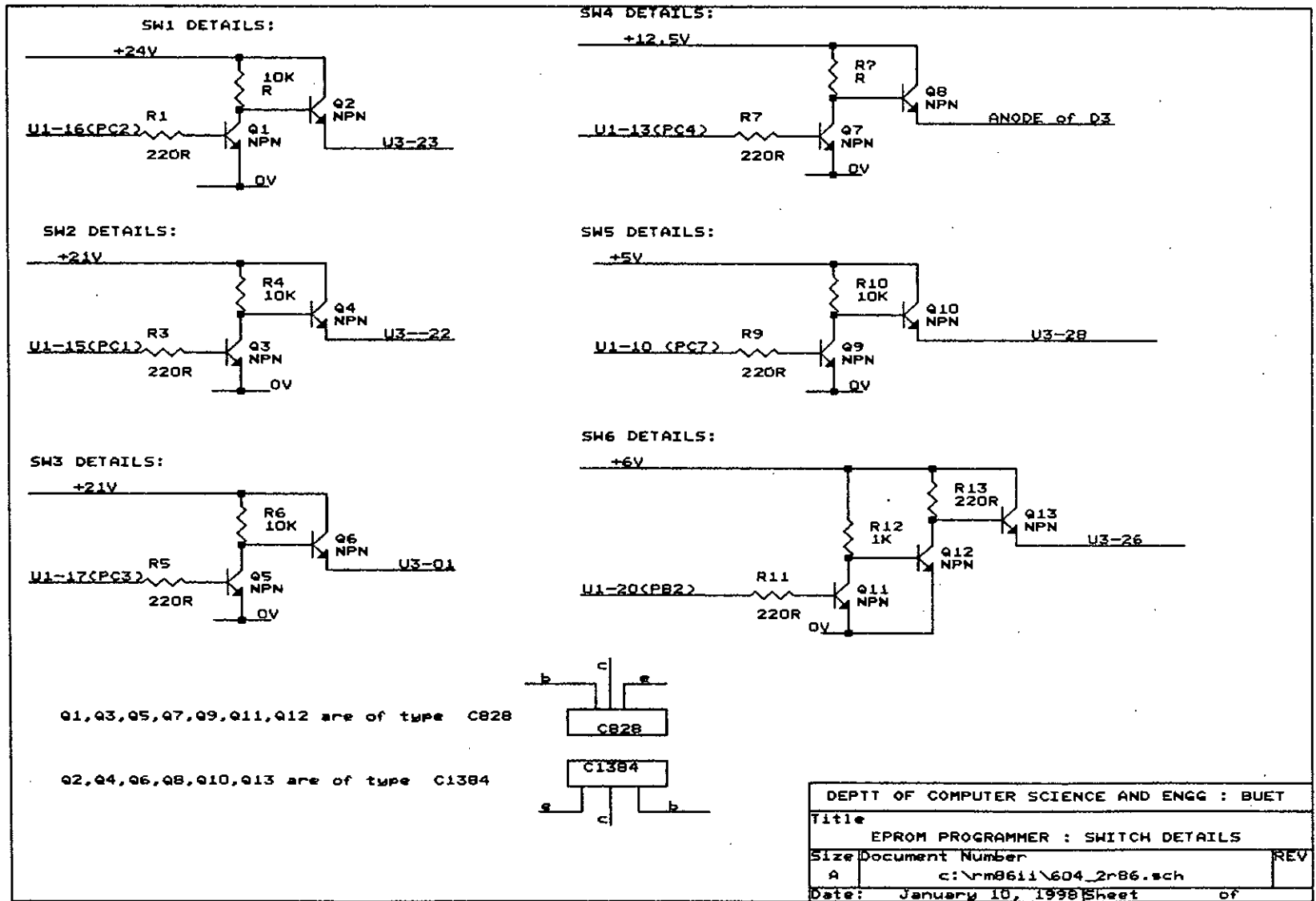


Fig - 7.2.2 (e) : Details of the Switches of the EPROM Programmer Schematic

Table 1. Mode Selection

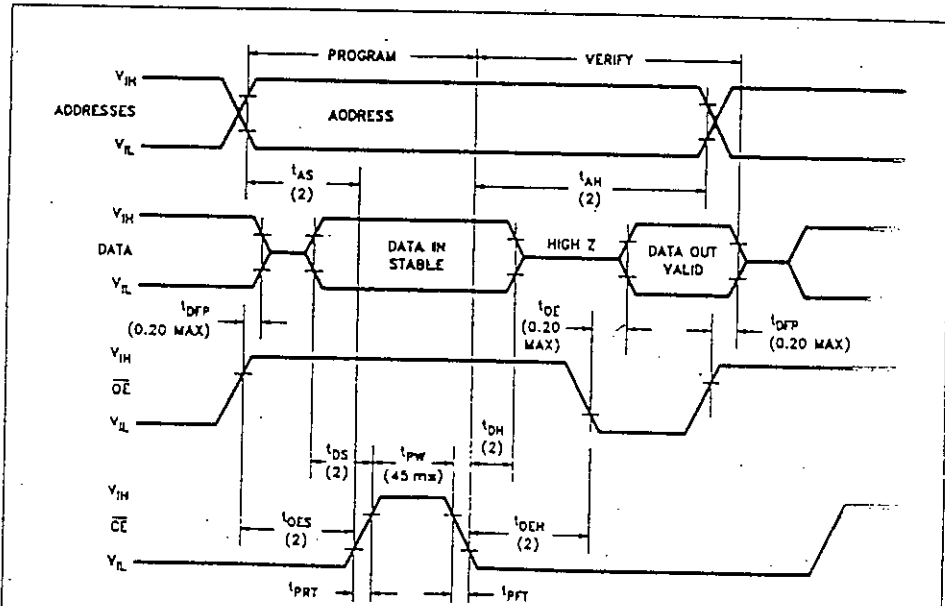
Mode	Pin	CE (18)	OE (20)	V _{pp} (21)	V _{CC} (24)	Outputs (9-11, 13-17)
Read		V _{IL}	V _{IL}	+5	+5	D _{OUT}
Output Disable		V _{IL}	V _{IH}	+5	-5	High Z
Standby		V _{IH}	X	+5	+5	High Z
Program		Pulsed V _{IL} to V _{IH}	V _{IH}	+25	+5	D _{IN}
Venty		V _{IL}	V _{IL}	+25	+5	D _{OUT}
Program Inhibit		V _{IL}	V _{IH}	+25	+5	High Z

NOTE:
1. X can be V_{IL} or V_{IH}.



2716

PROGRAMMING WAVEFORMS



NOTES:
1. All times shown in parenthesis are minimum times and are μ s unless otherwise noted.
2. t_{OE} and t_{DFF} are characteristics of the device but must be accommodated by the programmer.

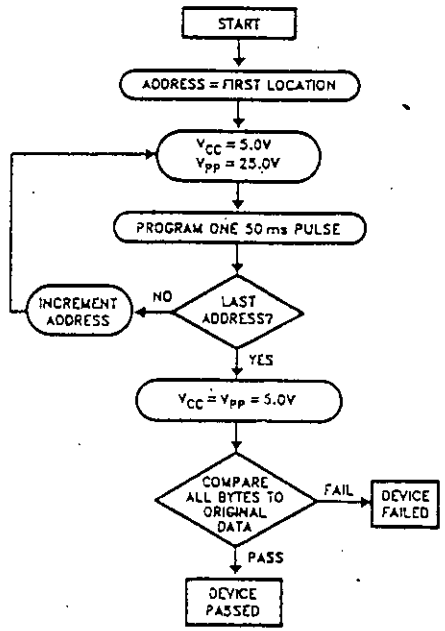


Figure 3. Standard Programming Flowchart

PROGRAMMING CHARACTERISTICS

D.C. PROGRAMMING CHARACTERISTICS $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$, $V_{CC}^{(1)} = 5\text{V} \pm 5\%$, $V_{PP}^{(1,2)} = 25\text{V} \pm 1\text{V}$

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{LI}	Input Current (for Any Input)			10	μA	$V_{IN} = 5.25\text{V}/0.45$
I_{PP1}	V_{PP} Supply Current			5	mA	$\overline{CE} = V_{IL}$
I_{PP2}	V_{PP} Supply Current during Programming Pulse			30	mA	$\overline{CE} = V_{IH}$
I_{CC}	V_{CC} Supply Current			100	mA	
V_{IL}	Input Low Level	-0.1		0.8	V	
V_{IH}	Input High Level	2.0		$V_{CC} + 1$	V	

A.C. PROGRAMMING CHARACTERISTICS $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$, $V_{CC}^{(1)} = 5\text{V} \pm 5\%$, $V_{PP}^{(1,2)} = 25\text{V} \pm 1\text{V}$

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions*
t_{AS}	Address Setup Time	2			μs	
t_{OES}	\overline{OE} Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	2			μs	
t_{OEH}	\overline{OE} Hold Time	2			μs	
t_{DH}	Data Hold Time	2			μs	
t_{OFP}	Output Enable to Output Float Delay	0		200	ns	$\overline{CE} = V_{IL}$
t_{OE}	Output Enable to Output Delay			200	ns	$\overline{CE} = V_{IL}$
t_{PW}	Program Pulse Width	45	50	55	ms	
t_{PRT}	Program Pulse Rise Time	5			ns	
t_{PFT}	Program Pulse Fall Time	5			ns	

***A.C. CONDITIONS OF TEST**

- Input Rise and Fall Times (10% to 90%) 20 ns
- Input Pulse Levels 0.8V to 2.2V
- Input Timing Reference Level 0.8V and 2V
- Output Timing Reference Level 0.8V and 2V

NOTES:

1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP} . The 2716 must not be inserted into or removed from a board with V_{PP} at $25 \pm 1\text{V}$ to prevent damage to the device.
2. The maximum allowable voltage which may be applied to the V_{PP} pin during programming is +26V. Care must be taken when switching the V_{PP} supply to prevent overshoot exceeding this 26V maximum specification.

Fig - 7.2.2 (g): Timing Parameters for 2716 EPROM (Courtesy Intel Corpn.)

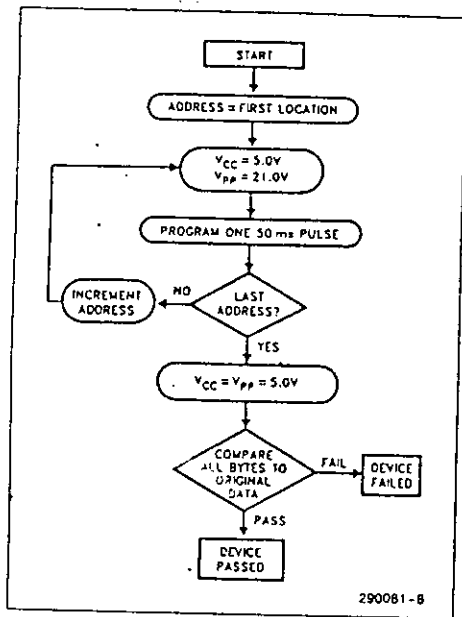


Figure 3. Standard Programming Flowchart

Table 1. Mode Selection

Mode	Pin ^s	CE	OE/V _{PP}	A ₉	A ₀	V _{CC}	Outputs
Read/Program Verify		V _{IL}	V _{IL}	X	X	V _{CC}	D _{OUT}
Output Disable		V _{IL}	V _{IH}	X	X	V _{CC}	High Z
Standby		V _{IH}	X	X	X	V _{CC}	High Z
Program		V _{IL}	V _{PP}	X	X	V _{CC}	D _{IN}
Program Inhibit		V _{IH}	V _{PP}	X	X	V _{CC}	High Z
Intelligent Identifier ⁽³⁾							
—Manufacturer		V _{IL}	V _{IL}	V _H	V _{IL}	V _{CC}	89H
—Device		V _{IL}	V _{IL}	V _H	V _{IH}	V _{CC}	01H

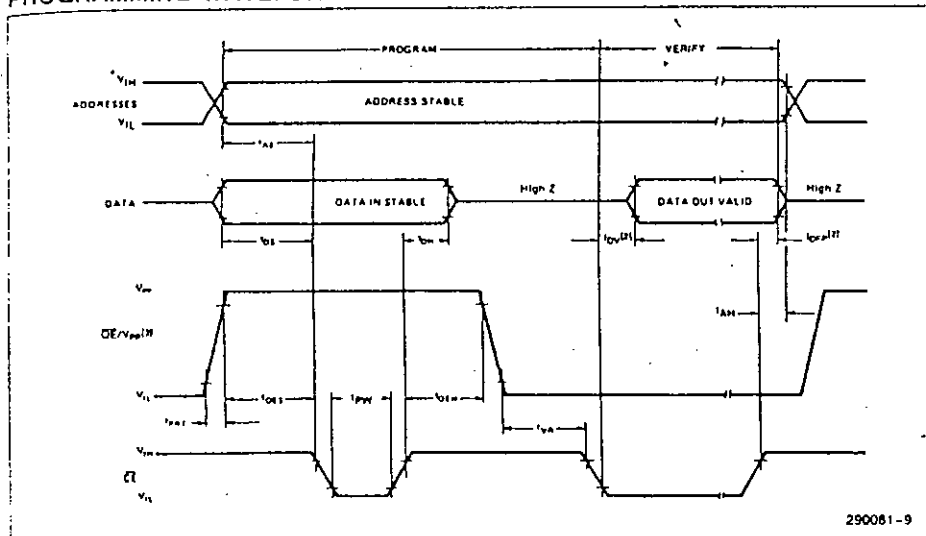
NOTES:

1. X can be V_{IH} or V_{IL}.
2. V_H = 12V ± 0.5V.
3. A₁–A₈, A₁₀, A₁₁ = V_{IL}.

intel

2732A

PROGRAMMING WAVEFORMS



NOTES:

1. The input timing reference level is 0.8V for a V_{IL} and 2V for a V_{IH}.
2. t_{DV} and t_{DPP} are characteristics of the device but must be accommodated by the programmer.
3. When programming the 2732A, a 0.1μF capacitor is required across OE/V_{PP} and ground to suppress spurious voltage transients which can damage the device.

Fig - 7.2.2 (h) : Programming Algorithm for 2732A EPROM

A.C. PROGRAMMING CHARACTERISTICS

$T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$, $V_{CC} = 5\text{V} \pm 5\%$, $V_{PP} = 21\text{V} \pm 0.5\text{V}$

Symbol	Parameter	Limits			Units	Test Conditions (Note 1)
		Min	Typ ⁽³⁾	Max		
I_{LI}	Input Current (All Inputs)			10	μA	$V_{IN} = V_{IL}$ or V_{IH}
V_{IL}	Input Low Level (All Inputs)	-0.1		0.8	V	
V_{IH}	Input High Level (All Inputs Except $\overline{\text{OE}}/V_{PP}$)	2.0		$V_{CC} + 1$	V	
V_{OL}	Output Low Voltage During Verify			0.45	V	$I_{OL} = 2.1\text{ mA}$
V_{OH}	Output High Voltage During Verify	2.4			V	$I_{OH} = -400\ \mu\text{A}$
$I_{CC2}^{(4)}$	V_{CC} Supply Current (Program and Verify)		85	100	mA	
$I_{PP2}^{(4)}$	V_{PP} Supply Current (Program)			30	mA	$\overline{\text{CE}} = V_{IL}$, $\overline{\text{OE}}/V_{PP} = V_{PP}$
V_{ID}	A_9 Intelligent Identifier Voltage	11.5		12.5	V	

intel

2732A

A.C. PROGRAMMING CHARACTERISTICS

$T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$, $V_{CC} = 5\text{V} \pm 5\%$, $V_{PP} = 21\text{V} \pm 0.5\text{V}$

Symbol	Parameter	Limits			Units	Test Conditions* (Note 1)
		Min	Typ ⁽³⁾	Max		
t_{AS}	Address Setup Time	2			μs	
t_{OES}	$\overline{\text{OE}}/V_{PP}$ Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	0			μs	
t_{DH}	Data Hold Time	2			μs	
t_{OFF}	$\overline{\text{OE}}/V_{PP}$ High to Output Not Driven	0		130	ns	(Note 2)
t_{PW}	$\overline{\text{CE}}$ Pulse Width During Programming	20	50	55	ms	
t_{OEH}	$\overline{\text{OE}}/V_{PP}$ Hold Time	2			μs	
t_{DV}	Data Valid from $\overline{\text{CE}}$			1	μs	$\overline{\text{CE}} = V_{IL}$, $\overline{\text{OE}}/V_{PP} = V_{IL}$
t_{VR}	V_{PP} Recovery Time	2			μs	
t_{PRT}	$\overline{\text{OE}}/V_{PP}$ Pulse Rise Time During Programming	50			ns	

NOTES:

- V_{CC} must be applied simultaneously or before $\overline{\text{OE}}/V_{PP}$ and removed simultaneously or after $\overline{\text{OE}}/V_{PP}$.
- This parameter is only sampled and is not 100% tested. Output Float is defined as the point where data is no longer driven—see timing diagram.
- Typical values are for $T_A = 25^\circ\text{C}$ and nominal supply voltages.
- The maximum current value is with outputs O_0 to O_7 unloaded.

*A.C. TEST CONDITIONS

Input Rise and Fall Time (10% to 90%) $\leq 20\text{ ns}$
 Input Pulse Levels 0.45V to 2.4V
 Input Timing Reference Level 0.8V and 2.0V
 Output Timing Reference Level 0.8V and 2.0V

Fig - 7.2.2 (I) : Timing Parameters for 2732A EPROM

Table 2

D.C. PROGRAMMING CHARACTERISTICS $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$.

Symbol	Parameter	Limits			Test Conditions (see Note 1)
		Min	Max	Unit	
I_{LI}	Input Current (All Inputs)		10	μA	$V_{IN} = V_{IL}$ or V_{IH}
V_{IL}	Input Low Level (All Inputs)	-0.1	0.8	V	
V_{IH}	Input High Level	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage During Verify		0.45	V	$I_{OL} = 2.1 \text{ mA}$
V_{OH}	Output High Voltage During Verify	2.4		V	$I_{OH} = -400 \mu\text{A}$
$I_{CC2}^{(4)}$	V_{CC} Supply Current (Program & Verify)		75	mA	
$I_{PP2}^{(4)}$	V_{PP} Supply Current (Program)		50	mA	$\overline{CE} = V_{IL}$
V_{ID}	A_9 intelligent Identifier Voltage	11.5	12.5	V	
V_{PP}	intelligent Programming Algorithm	12.0	13.0	V	$\overline{CE} = \text{PGM} = V_{IL}$
V_{CC}	intelligent Programming Algorithm	5.75	6.25	V	

A.C. PROGRAMMING CHARACTERISTICS

$T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ (see table 2 for V_{CC} and V_{PP} voltages)

Symbol	Parameter	Limits				Test Conditions* (see Note 1)
		Min	Typ	Max	Unit	
t_{AS}	Address Setup Time	2			μs	
t_{OES}	\overline{OE} Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	0			μs	
t_{DH}	Data Hold Time	2			μs	
t_{OFP}	\overline{OE} High to Output Float Delay	0		130	ns	(See Note 3)
t_{VPS}	V_{PP} Setup Time	2			μs	
t_{VCS}	V_{CC} Setup Time	2			μs	
t_{CES}	\overline{CE} Setup Time	2			μs	
t_{PW}	PGM Initial Program Pulse Width	0.95	1.0	1.05	ms	
t_{OPW}	PGM Overprogram Pulse Width	2.85		78.75	ms	(see Note 2)
t_{OE}	Data Valid from \overline{OE}			150	ns	

*A.C. CONDITIONS OF TEST

Input Rise and Fall Times (10% to 90%) 20 ns
 Input Pulse Levels 0.45V to 2.4V
 Input Timing Reference Level 0.8V and 2.0V
 Output Timing Reference Level 0.8V and 2.0V

NOTES:

- V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP} .
- The length of the overprogram pulse may vary from 2.85 msec to 78.75 msec as a function of the iteration counter value X.
- This parameter is only sampled and is not 100% tested. Output Float is defined as the point where data is no longer driven—see timing diagram.
- The maximum current value is with Outputs O_0 to O_7 unloaded.

Table 1. Mode Selection

Mode	Pin	\overline{CE}	\overline{OE}	PGM	A_9	A_0	V_{PP}	V_{CC}	Outputs
Read		V_{IL}	V_{IL}	V_{IH}	X ⁽¹⁾	X	V_{CC}	5.0V	O_{OUT}
Output Disable		V_{IH}	V_{IH}	V_{IH}	X	X	V_{CC}	5.0V	High Z
Standby		V_{IH}	X	X	X	X	V_{CC}	5.0V	High Z
Programming		V_{IL}	V_{OH}	V_{IL}	X	X	(4)	(4)	D_{IN}
Program Verify		V_{IL}	V_{IL}	V_{IH}	X	X	(4)	(4)	O_{OUT}
Program Inhibit		V_{HI}	X	X	X	X	(4)	(4)	High Z
intelligent Identifier ⁽²⁾ —manufacturer		V_{IL}	V_{IL}	V_{HI}	V_{HI} ⁽²⁾	V_{IL}	V_{CC}	5.0V	89H
—device		V_{IL}	V_{IL}	V_{HI}	V_{HI} ⁽²⁾	V_{IH}	V_{CC}	5.0V	08H

NOTES:

- X can be V_{IH} or V_{IL} .
- $V_{HI} = 12.0V \pm 0.5V$.
- $A_1 - A_8, A_{10} - A_{12} = V_{IL}$.
- See Table 2 for V_{CC} and V_{PP} voltages.

Fig - 7.2.2 (k) : Timing parameters for 2764A EPROM

DEVICE OPERATION

The modes of operation of the 27128A are listed in Table 1. A single 5V power supply is required in the read mode. All inputs are TTL levels except for V_{PP} and 12V on A₉ for intelligent Identifier.

Table 1. Modes Selection

Mode	Notes	CE	OE	PGM	A ₉	A ₀	V _{PP}	V _{CC}	Outputs	
Read	1	V _{IL}	V _{IL}	V _{IH}	X	X	V _{CC}	5.0V	D _{OUT}	
Output Disable		V _{IL}	V _{IH}	V _{IH}	X	X	V _{CC}	5.0V	High Z	
Standby		V _{IH}	X	X	X	X	V _{CC}	5.0V	High Z	
Programming	4	V _{IL}	V _{IH}	V _{IL}	X	X	V _{PP}	6.0V	D _{IN}	
Program Verify	4	V _{IL}	V _{IL}	V _{IH}	X	X	V _{PP}	6.0V	D _{OUT}	
Program Inhibit	4	V _{IH}	X	X	X	X	V _{PP}	6.0V	High Z	
Intelligent Identifier	Manufacturer	2, 3	V _{IL}	V _{IL}	V _{IH}	V _H	V _{IL}	V _{CC}	5.0V	89 H
	Device	2, 3	V _{IL}	V _{IL}	V _{IH}	V _H	V _{IH}	V _{CC}	5.0V	89 H

- NOTES:
 1. X can be V_{IL} or V_{IH}
 2. V_H = 12.0V ± 0.5V
 3. A₁-A₈, A₁₀-A₁₂ = V_{IL}
 4. See Table 2 for V_{CC} and V_{PP} voltages.

DC PROGRAMMING CHARACTERISTICS T_A = 25°C ± 5°C

Symbol	Parameter	Limits			Test Conditions (Note 1)
		Min	Max	Unit	
I _{LI}	Input Current (All Inputs)		10	µA	V _{IN} = V _{IL} or V _{IH}
V _{IL}	Input Low Level (All Inputs)	-0.1	0.8	V	
V _{IH}	Input High Level	2.0	V _{CC} + 1	V	
V _{OL}	Output Low Voltage During Verify		0.45	V	I _{OL} = 2.1 mA
V _{OH}	Output High Voltage During Verify	2.4		V	I _{OH} = -400 µA
I _{CC2} (4)	V _{CC} Supply Current (Program & Verify)		100	mA	
I _{PP2}	V _{PP} Supply Current (Program)		50	mA	CE = V _{IL}
V _{ID}	A ₉ intelligent Identifier Voltage	11.5	12.5	V	
V _{PP}	intelligent Programming Algorithm	12.0	13.0	V	CE = PGM = V _{IL}
V _{CC}	intelligent Programming Algorithm	5.75	6.25	V	

AC PROGRAMMING CHARACTERISTICS

T_A = 25°C ± 5°C (See Table 2 for V_{CC} and V_{PP} voltages.)

Symbol	Parameter	Limits				Conditions* (Note 1)
		Min	Typ	Max	Unit	
t _{AS}	Address Setup Time	2			µs	
t _{OES}	OE Setup Time	2			µs	
t _{OS}	Data Setup Time	2			µs	
t _{AH}	Address Hold Time	0			µs	
t _{DH}	Data Hold Time	2			µs	
t _{DFP}	OE High to Output Float Delay	0		130	ns	(Note 3)
t _{VPS}	V _{PP} Setup Time	2			µs	
t _{VCS}	V _{CC} Setup Time	2			µs	
t _{CES}	CE Setup Time	2			µs	
t _{PW}	PGM Initial Program Pulse Width	0.95	1.0	1.05	ms	
t _{OPW}	PGM Overprogram Pulse Width	2.85		78.75	ms	(Note 2)
t _{OE}	Data Valid from OE			150	ns	

***AC CONDITIONS OF TEST**

- Input Rise and Fall Times (10% to 90%) 20 ns
 Input Pulse Levels 0.45V to 2.4V
 Input Timing Reference Level 0.8V and 2.0V
 Output Timing Reference Level 0.8V and 2.0V

NOTES:

- V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
- The length of the overprogram pulse may vary from 2.85 msec to 78.75 msec as a function of the iteration counter value X.
- This parameter is only sampled and is not 100% tested. Output Float is defined as the point where data is no longer driven—see timing diagram.
- The maximum current value is with outputs O₀-O₇ unloaded.

Fig - 7.2.2 (m) : Timing Parameters for 27128A EPROM

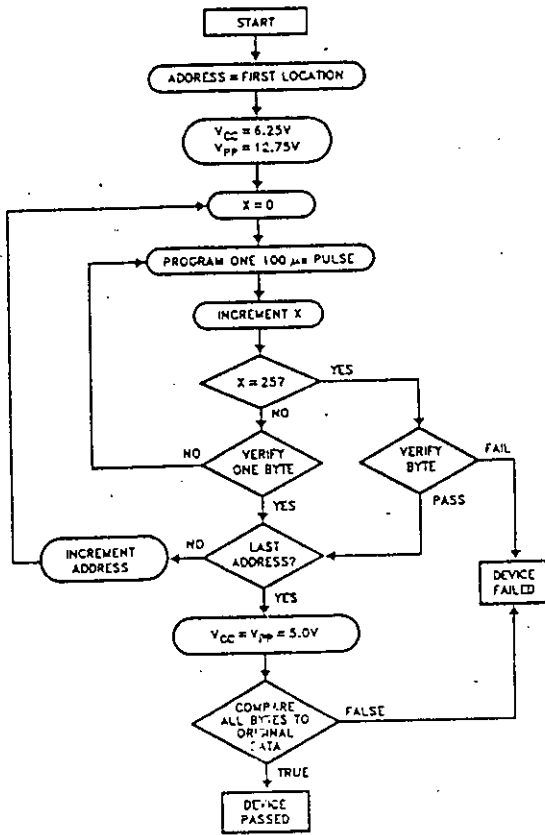


Figure 3. Quick-Pulse Programming™ Algorithm

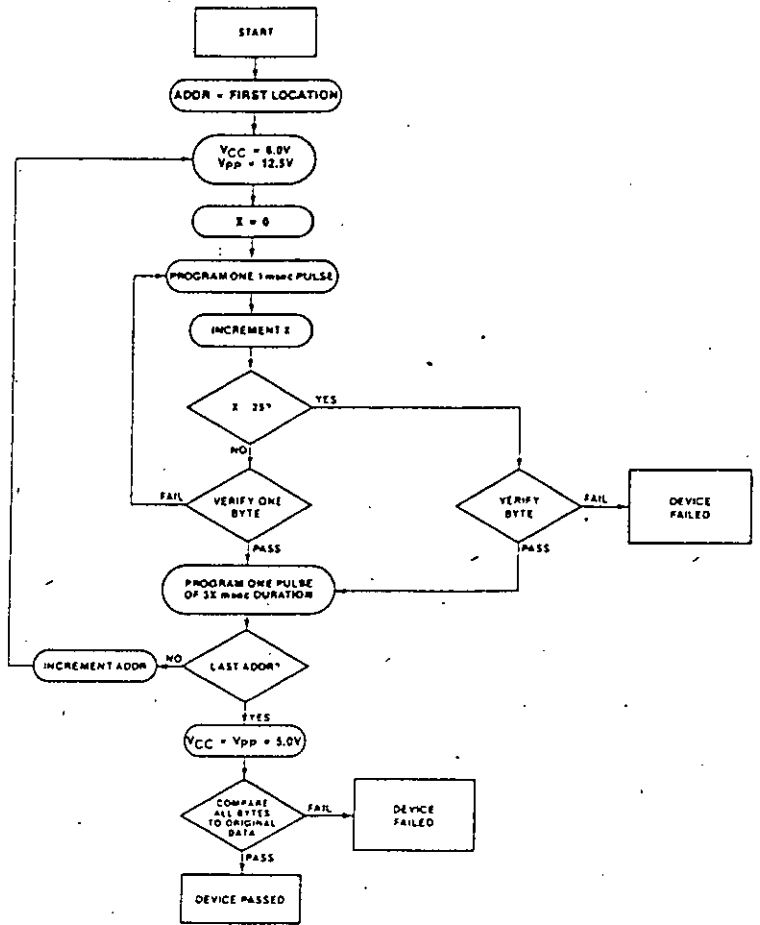


Figure 4. Intelligent Programming™ Flowchart

Fig - 7.2.2 (n) : Programming Algorithm for 27256 EPROM

Table 1. Operating Modes

Mode	CE	OE	A ₉	A ₀	V _{PP}	V _{CC}	Outputs
Read	V _{IL}	V _{IL}	X ⁽¹⁾	X	V _{CC}	5.0V	D _{OUT}
Output Disable	V _{IL}	V _{IH}	X	X	V _{CC}	5.0V	High Z
Standby	V _{IH}	X	X	X	V _{CC}	5.0V	High Z
Programming	V _{IL}	V _{IH}	X	X	(4)	(4)	D _{IN}
Program Verify	V _{IH}	V _{IL}	X	X	(4)	(4)	D _{OUT}
Optional Program Verify	V _{IL}	V _{IL}	X	X	V _{CC}	(4)	D _{OUT}
Program Inhibit	V _{IH}	V _{IH}	X	X	(4)	(4)	High Z
Intelligent Identifier ⁽³⁾							89H ⁽⁵⁾
—manufacturer	V _{IL}	V _{IL}	V _H ⁽²⁾	V _{IL}	5.0V	5.0V	88H ⁽⁵⁾
—device	V _{IL}	V _{IL}	V _H ⁽²⁾	V _{IH}	5.0V	5.0V	04H

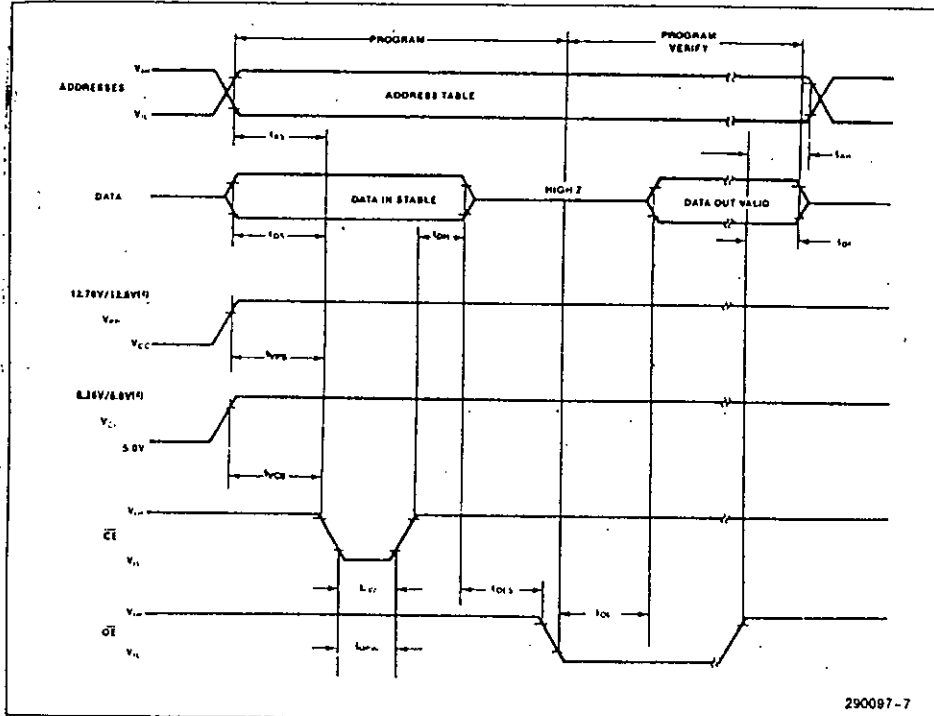
NOTES:

- X can be V_{IH} or V_{IL}.
- V_H = 12.0V ± 0.5V.
- A₁-A₉, A₁₀-A₁₃ = V_{IL}, A₁₄ = V_{IH}.
- See Table 2 for V_{CC} and V_{pp} voltages.
- The manufacturer's identifier reads 89H for Cerdip EPROMs; 88H for Plastic EPROMs.



27256

PROGRAMMING WAVEFORMS



NOTES:

- The input timing reference level is 0.3V for a V_{IL} and 2V for a V_{IH}.
- Timing parameters are characteristics of the device but must be accommodated by the programmer.
- A 0.1 μF capacitor is required across V_{pp} and ground to suppress spurious voltage transients when programming.
- 12.75V V_{pp} & 6.25V V_{cc} for Quick-Pulse Programming Algorithm. 12.5V V_{pp} & 6.0V V_{cc} for intelligent Programming Algorithm.

Fig - 7.2.2 (o) : Timing Parameters for 27256 EPROM

TABLE 2. D.C. PROGRAMMING CHARACTERISTICS $T_A = 25 \pm 5^\circ\text{C}$

Symbol	Parameter	Limits			Test Conditions (see Note 1)
		Min	Max	Unit	
I_{LI}	Input Current (All Inputs)		10	μA	$V_{IN} = V_{IL}$ or V_{IH}
V_{IL}	Input Low Level (All Inputs)	-0.1	0.8	V	
V_{IH}	Input High Level	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage During Verify		0.45	V	$I_{OL} = 2.1 \text{ mA}$
V_{OH}	Output High Voltage During Verify	2.4		V	$I_{OH} = -400 \mu\text{A}$
$I_{CC2}^{(4)}$	V_{CC} Supply Current (Program & Verify)		125	mA	
$I_{PP2}^{(4)}$	V_{PP} Supply Current (Program)		50	mA	$\overline{CE} = V_{IL}$
V_{ID}	A_9 intelligent Identifier Voltage	11.5	12.5	V	
V_{PP}	intelligent Programming Algorithm	12.0	13.0	V	$\overline{CE} = V_{IL}$
	Quick-Pulse Programming Algorithm	12.5	13.0	V	$\overline{CE} = V_{IL}$
V_{CC}	intelligent Programming Algorithm	5.75	6.25	V	
	Quick-Pulse Programming Algorithm	6.0	6.5	V	

A.C. PROGRAMMING CHARACTERISTICS

$T_A = 25 \pm 5^\circ\text{C}$ (see table 2 for V_{CC} and V_{PP} voltages)

Symbol	Parameter	Limits				Test Conditions* (Note 1)
		Min	Typ	Max	Unit	
t_{AS}	Address Setup Time	2			μs	
t_{OES}	\overline{OE} Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	0			μs	
t_{DH}	Data Hold Time	2			μs	
t_{DFP}	\overline{OE} High to Output Data Float Delay	0		130	μs	(Note 3)
t_{VPS}	V_{PP} Setup Time	2			μs	
t_{VCS}	V_{CC} Setup Time	2			μs	
t_{PW}	\overline{CE} Initial Program Pulse Width	0.95	1.0	1.05	ms	intelligent Programming
		95	100	105	μs	Quick-Pulse Programming
t_{OPW}	\overline{CE} Overprogram Pulse Width	2.85		78.75	ms	(Note 2)
t_{OE}	Data Valid from \overline{OE}			150	ns	

*A.C. CONDITIONS OF TEST

Input Rise and Fall Times
(10% to 90%) 20 ns
 Input Pulse Levels 0.45V to 2.4V
 Input Timing Reference Level 0.8V and 2.0V
 Output Timing Reference Level 0.8V and 2.0V

NOTES:

- V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP} .
- The length of the overprogram pulse may vary from 2.85 msec to 78.75 msec as a function of the iteration counter value X (intelligent Programming Algorithm only).
- This parameter is only sampled and is not 100% tested. Output Data Float is defined as the point where data is no longer driven—see timing diagram on the following page.
- The maximum current value is with outputs O_0 to O_7 unloaded.

Fig - 7.2.2 (p) : Timing Parameters for 27256 EPROM

8

**IBM-PC
TO
TRAINER
DOWN LOADING
SOFTWARE**

8.1 Introduction

The IBM-PC and the 8086 trainer can be connected together by a 4800-Bd RS232 serial link on COM1 port. An ASCII (non document text) of the following format can be down loaded into the RAM of the 8086 trainer. The ASCII file will usually be prepared from the LIST file produced by the Macro Assembler. This link will be used mainly to down load program codes which will be executed out of the RAM of the trainer. The following program will drive an ADC connected with the trainer as per schematic diagram of page-81 RUT-1. The program codes are developed using MASM. The LST file is given the following format according to the RAM locations of the trainer. Every instruction line must be terminated by ; (semicolon). Also, the last character should be a * (star) which is used to indicate the end of transmission. Now, these codes will be transferred to the trainer. Execution will be done at the local keyboard of the trainer at location 08000h.

08000	BA 00 30	; mov dx, 3000h	pointing at Command Register
08000	EE	; out dx,al	Convert command to ADC
08003	B9 02 00	; mov cx, 0002h	delay parameter
08004	E2 FE	; loop HERE1 (0000:8007)	Conversion delay
08007	EC	; in al, dx	data is read
08009	88 47 4E	; mov BYTE PTR [bx+4Eh],al	data is xferred to T2
0800A	9A 7C F4 00 F0	; call SUR#8 (F000:F47C)	translating T2 to T1 (hex to cc-codes)
0800D	C7 47 44 00 00	; mov [bx+44h], 0000h	blanking D4D3 positions of the display
08012	C7 47 46 00 00	; mov [bx+46h], 0000h	blanking D6D5 positions of the display
08017	C7 47 48 00 00	; mov [bx+48h], 0000h	blanking D8D7 positions of the display
0801C	C7 47 4A 00 00	; mov [bx+4Ah], 0000h	blanking D9 position of the display
08021	9A B6 FF 00 F0	; call SUR# 3 (F000:FFB6)	transferring T1 to 8279
08026	B9 FF FF	; mov cx, 0FFFFh	display update delay
0802B	E2 FE	; loop HERE2 (0000:082F)	wait
0802F	EA 00 80 00 00	; jmp 0000:8000	get the next sample
08030			

*

The content of the 1st small box (08000) is transmitted first to the trainer as the memory location from where the program codes will be stored. The contents of the second box is then transmitted character by character. The PC xmits a character and then waits for the acknowledgment form the trainer. If no ACK is received, no transmission. The communication software has the following parts and will be documented in the next few pages.

01. Program for IBM-PC (SCOM86.EXE)

- Checking if the 8086 trainer has been powered up.
- Asks the user to enter the ASCII file name from the keyboard
- The ASCII file is opened for READ only.
- The Header (RAM starting location) is transmitted. Xmission is indicated by sending code 01h.
- Then the instruction codes are transmitted. Code 01h is xmitted after the end of each instruction codes transmission.
- End of file transmission is marked by sending character 03h.

02. 8086 Trainer Firmware

- Receives the header. The ASCII is processed to retrieve the 20-bit starting address of the RAM.
- Receives the instruction codes in ASCII. These are processed to reconstruct the hex codes and one instruction at a time. The codes are also placed in the appropriate RAM locations.

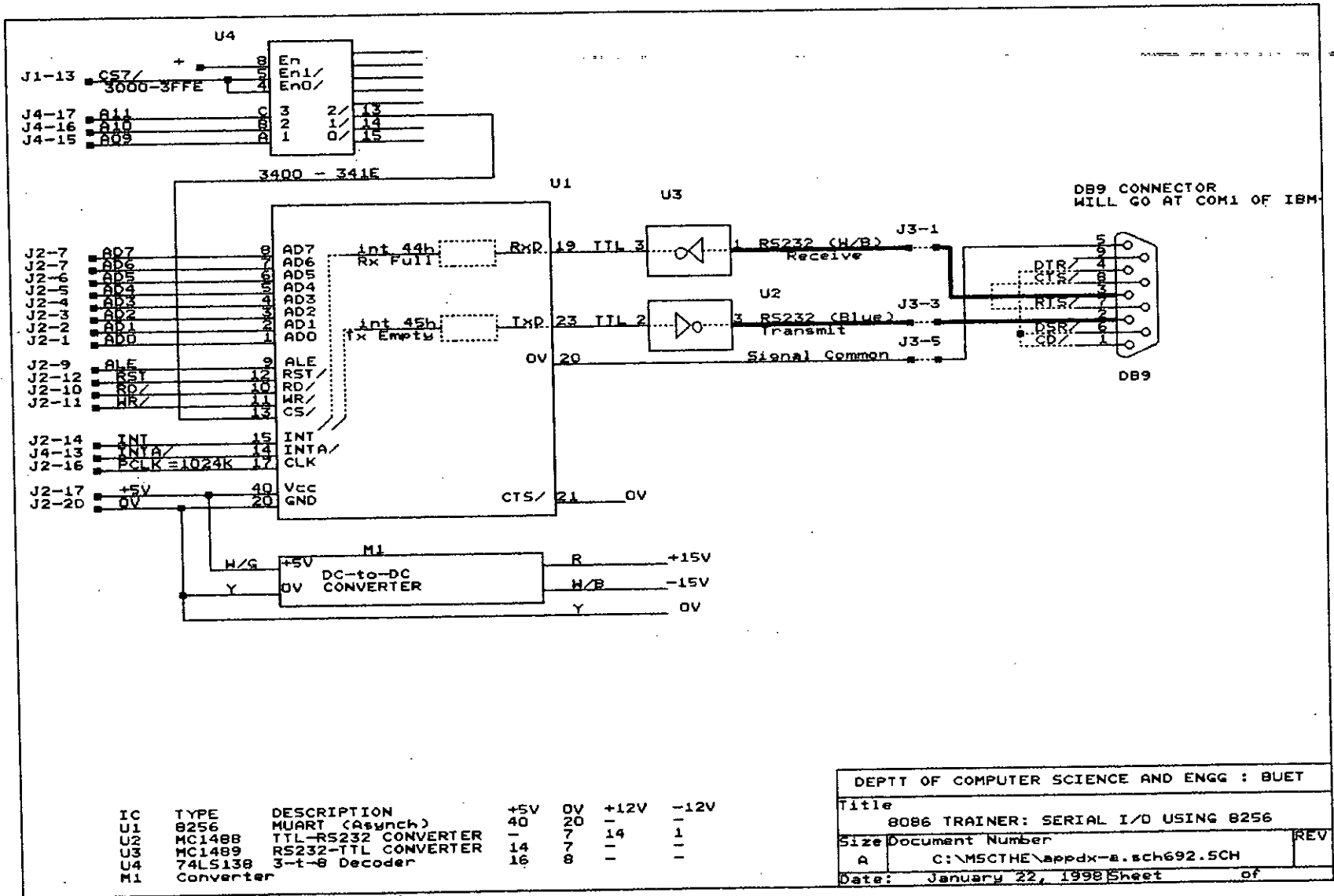


Fig - 8.1(a) : Hardware Interface Circuit Bet[®] Trainer and IBM-PC

DEPTT OF COMPUTER SCIENCE AND ENGG : BUET	
Title 8086 TRAINER: SERIAL I/O USING 8256	
Size	Document Number
A	C:\MSCTHE\appdx-a.sch692.SCH
Date:	January 22, 1998 Sheet of

8.2 Assembly Source Code Listing for IBM-PC's EXE program

```

STACK SEGMENT      para stack 'stack'
      DW           2048 dup(0)
STKTOP LABEL      word
STACK ENDS

DATA SEGMENT para public 'data'
FLAG1 DB 00 ; used by COM1_IN procedure
FLAG2 DB 00
FLAG3 DB 00
FLAG4 DB 03 ; for retry to enter valid filename
FLAG5 DB 00
FLAG6 DB 00
PMT   DB 0Dh,0Ah ; PMT = PRoMt, 0D=Carrriage Retux, 0A=Line Feed
      DB 'Enter Capital (L) for Loading ASCII File.....!',0Dh,0Ah,24h
ENQ   DB 0Dh,0Ah
      DB 'Now Roll Calling for 8086 Trainer.....!',0Dh,0Ah,0Dh,0Ah,24h
ACK1  DB '8086 Trainer is Ready.....!',0Dh,0Ah,0Dh,0Ah,24h
ACK2  DB 0Dh,0Ah
      DB 'Now Transmitting Starting Location of RAM.....!',0Dh,0Ah,24h
      DB 0Dh,0Ah
ACK3  DB 0Dh,0Ah
      DB 'Now Transmitting Program/Data Code.....!',0Dh,0Ah,24h
      DB 0Dh,0Ah
ACK4  DB 'Transmission Ends.....!',0Dh,0Ah,24h
      DB 0Dh,0Ah
TI_OUTM DB 'Transmit Timeout - Check Hardware',0Dh,0Ah,24h
PROMPT DB 0Dh,0Ah
      DB 'Please Type in ASCII Filename',0Dh,0Ah
      DB 'Filename Format is: path\Filename.TXT',0Dh,0Ah,24h
FNAME DB 81
      DB 0 ; number of characters in file path excluding CR
      DB 81 dup(0) ; all characters of path including CR
BUFF  DB 16 dup(0)
FHANDLDW 0
MSG1  DB 'File Successfully Opened.',0Dh,0Ah,24h
MSG2  DB 0Dh,0Ah
      DB 'Enter the File Pathname in the Right Format.....!',0Dh,0Ah,24h
MSG3  DB 0Dh,0Ah
      DB 'Bad Pathname.....!',0Dh,0Ah,24h
WARN  DB 'BE SURE THE TXT FILE DOESN'T CONTAIN ANY DASH @ AFTER SEMICOLON @...!',0Dh,0Ah,24h
QUEUE DB 1000 dup(0)
SBUFF DB 00h
DATA ENDS

CODE SEGMENT para public 'code'
      ASSUME cs:MYCODE,ds:MYDATA,ss:MYSTACK

START: mov ax,STACK
      mov ss,ax
      mov sp,offset STKTOP

      mov ax,0000h ;init of COM1's IVT
      mov es,ax ;00030h-00033h
      mov WORD PTR es:0030h,offset COM1_IN
      mov WORD PTR es:0032h,seg COM1_IN ;init IVT for int 0Ch

      mov WORD PTR es:008Ch,offset CTLC
      mov WORD PTR es:008Eh,seg CTLC ; IVT for ^C detection during I/O

COMMENT *
      These codes do not work to set the IVT for int 0Ch
      mov ah,25h

```

```

mov  al,0Ch
mov  dx,seg COM1_IN ;
mov  ds,dx
mov  dx,offset COM1_IN
int  21h ; does not work

in   al,21h      ;Master 8259 (20-3F) IMR addr = 21h
and  al,0ECh    ;al && 11101100b ,COM1's Rx interrupts CPU on IR4
out  21h,al     ;IR4 line (int 0Ch, IR0=int 08h) is enabled

mov  ah,00h     ;init of comm protocol
mov  dx,0000h   ;pointing at COM1 port
mov  al,11000111b ;4800Bd,NP,2-SB,8-DB
int  14h       ;ROM BIOS function call

mov  dx,03FBh   ;LCP =Line Control Port addr =03FBh
in   al,dx     ;RxDY line can generate interrupt
and  al,7Fh
out  dx,al
mov  al,01h
mov  dx,03F9h   ; pointing at IER = Interrupt Enable Register
out  dx,al     ; no need to assert DTR, RTS/ which are already
               ; hardware programmed by jumpers at COM1 connector
               ; pointing at Modem Control Register

sti                     ;enable 8086's interrupt

mov  ah,09h     ;message Now Roll Calling for 8086 Trainer...!
mov  dx,seg ENQ
mov  ds,dx
mov  dx,offset ENQ
int  21h       ;send prompt message to CRT

LB2:  mov  al,05h   ;LB2=Label-2 Roll Calling 8086 Trainer
      call XMIT    ; Procedure to write data at Transmitter of 8250

CDAG: mov  bl,45h   ;wait for the trainer to digest 05h code.
      call DELAY
      dec  bl
      jnz CDAG    ; CDAG = Count Down Again

THERE: mov ah,01h  ;getting keyboard status
      int 16h
      jnz RDKY   ;ax = not zero, key waiting

BB:   cmp  FLAG1,01h ; 01 is given by COM1's Rx in COM1_IN SUR when ACK is received
      jnz LB2    ; 05 code was not received by 8086 trainer
      jmp  FRL   ; trainer is ready and now transmit ASCII file

RDKY: mov  ah,00h
      int 16h
      cmp  al,51h ;check if Quit (Q) command
      jnc BB     ;recheck Acknow of 06h
      jmp  QUIT  ;goto DOS

FRL:  mov  ah,09h   ; 8086 Trainer is Ready.....!
      mov  dx,seg ACK1
      mov  ds,dx
      mov  dx,offset ACK1
      int 21h

      mov  ah,09h   ; Press L for Loading ASCII file

```

```

    mov dx,seg PMT
    mov ds,dx
    mov dx,offset PMT
    int 21h

CHKLK: mov ah,00h    ; check for L key
       int 16h
       cmp al,4Ch
       jz GRDTX
       cmp al,51h
       jz QUIT
       jmp CHKLK

GRDTX: call RDTX    ; go ahead read and writing ASCII file
       jmp QUIT    ; transmit finishes

QUIT:  in  al,21h   ;unmask UART for COM1 to prevent it
       or  al,10h  ;from disrupting DOS
       out 21h,al
       mov ax,4C00h
       int 21h

CTLC   PROC NEAR
       mov ax,4C00h
       int 21h
       iret

CTLC   ENDP

COM1_IN PROC NEAR    ; data is read from Rx buffer. Put in circuit
       sti          ;queue QUEUE and then CHK_DIS routine prints it on CRT.
       push ax
       push bx
       push dx
       push di
       push ds
       mov ax,MYDATA
       mov ds,ax
       mov dx,03F8h ;Receiver Buffer Address of 8250
       in  al,dx
       mov SBUFF,al

       cmp al,06h
       jnz LB4      ;no ack og 06h
       mov FLAG1,01h ;ack is received
LB4:   nop
       mov al,20h   ; non specific End of Interrupt for 8259
       out 20h,al
       pop ds
       pop di
       pop dx
       pop bx
       pop ax
       iret
COM1_IN ENDP

XMIT PROC NEAR
       push bx
       push cx
       push dx
       push ax
       mov cx,0010h

RECHK: mov dx,03FDh ;check if Tx buffer is ready
       in  al,dx

```

```

    and    al,20h
    jz     NO_RDY
    pop    ax          ;Tx is READY!
    mov    dx,03F8h   ; pointing at UART
    out    dx,al
    cld
    jmp    DONE

NO_RDY: loop RECHK    ; check Tx buffer for readiness again
    mov    dx,OFFSET TI_OUTM
    mov    ah,09h
    mov    bh,00h
    int    21h
    stc

DONE:   pop    ax
    pop    dx
    pop    cx
    pop    bx
    ret

XMIT    ENDP

RDTX    PROC    NEAR    ; ASCII file read and XMIT
    sti
    push  bx          ;prompt to enter filename from keyboard

    mov    ah,09h
    mov    bh,00h
    mov    dx,seg PROMPT
    mov    ds,dx
    mov    dx,offset PROMPT
    int    21h

    call  SPACE

    mov    ah,09h    ; Warning message not to include - after ; in ASCII file
    mov    bh,00h
    mov    dx,seg WARN
    mov    ds,dx
    mov    dx,offset WARN
    int    21h

GFIL:   mov    ah,0Ah    ;GFIL=GetFILE get filename from keyboard
    mov    dx,seg FNAME
    mov    ds,dx
    mov    dx,offset FNAME
    int    21h

    mov    b[FNAME+1] ;length of filename for C:\TXFILE.TXT it is 13d
    mov    bh,00h     ;bx is a pointer
    mov    FNAME+2[bx],00h ;place 00 (ASCIIz) to replace CR(0Dh) at end.

    mov    ah,3Dh    ; ASCII file opening
    mov    al,00h
    mov    dx,seg FNAME+2
    mov    ds,dx
    mov    dx,offset FNAME+2
    int    21h

    mov    FHANDL,ax ;File Handle is saved
    jnc   FIOK      ; file opened

    dec   FLAG4     ; 3 times chance if wrong path for filename
    jz    BELOW

```

```

        mov ah,09h
        mov dx,seg MSG2
        mov ds,dx
        mov dx,offset MSG2
        int 21h
        jmp GFIL

BELOW:  mov FLAG4,03h ; not correct filename
        mov ah,09h
        mov dx,seg MSG3 ; message Bad pathname....
        mov ds,dx
        mov dx,offset MSG3
        int 21h
        mov al,2Ah ; xmit * to trainer to indicate end of xmission
        call XMIT

        jmp EXIT1

FIOK:   mov ah,09h ; file opening success message
        mov dx,seg MSG1
        mov ds,dx
        mov dx,offset MSG1
        int 21h

        mov ah,42h ;File Pointer Position
        mov al,00h
        mov bx,FHANDL
        mov cx,0000h
        mov dx,0000h
        int 21h

        mov al,09h ; header will be sent RAM address
        call XMIT
        call DELAY

PM:     cmp FLAG1,01h ; ack 06 is received
        jz OKX ; go ahead
        call CTLQ
        jmp PM

OKX:    mov FLAG1,00h ; to sense next acknowledgement

        mov ah,09h ; message Transmitting RAM starting location
        mov dx,seg ACK2
        mov ds,dx
        mov dx,offset ACK2
        int 21h

GCHA:   call NXTCHA ; reading next character from File
        cmp BUFF,20h ; space is not printed
        jz GCHA ;GCHA=Get CHAracter

        cmp BUFF,2Dh ; - will not be printed
        jz PEXIT ; proceed to EXIT

        mov ah,02h ; printing the start of RAM XXXXXX -
        mov dl,BUFF
        int 21h

        mov al,BUFF ;xmit start of RAM XXXXXX
        call XMIT
        call DELAY

ACMP:   cmp FLAG1,01h ; checking if ACK (06h) has arrived from trainer.
        jz OK1

```

```

        call CTLQ
        jmp ACMP
OK1:    mov FLAG1,00h

        jmp GCHA

PEXT:   mov al,01h      ;end of header transmission, RAM start XXXXX
        call XMIT
        call DELAY
CM:     cmp FLAG1,01h
        jz OK2          ;ack OK of 06h
        call CTLQ
        jmp CM          ;waiting for ack or Q=Quit
OK2:    mov FLAG1,00h   ;flag reset to sense next ack
        nop             ;Header Tx done
        call SPACE

        mov ah,09h     ; message transmitting Program/data codes
        mov dx,seg ACK3
        mov dx,dx
        mov dx,offset ACK3
        int 21h

AGN:    call NXTCHA
        mov ah,02h     ; printing Offset of the RAM locations
        mov d,IBUFF   ; no xmission
        int 21h

        cmp BUFF,2Dh  ;
        jnz AGN

AGN1:   call NXTCHA    ; now time to xmit program data/codes

        cmp BUFF,2Ah  ; if * end of xmit
        jz EXITP      ; sensed *

        mov ah,02h    ; print all program codes
        mov d,IBUFF
        int 21h

        cmp BUFF,3Bh  ;; detecting
        jz A1

        cmp BUFF,20h  ; space is not being transmitted
        jz AGN1

        mov d,IBUFF   ; valid program codes are being xmitted
        call XMIT
        call DELAY
CMP1:   cmp FLAG1,01h
        jz OK3        ; ack OK
        call CTLQ
        jmp CMP1
OK3:    mov FLAG1,00h ; to sense next ack
        jmp AGN1     ;remaining Characters of the Line

A1:     mov al,01h    ; one instruction xmit done
        call XMIT
        call DELAY
FCHK:   cmp FLAG1,01h
        jz OK4        ;ack OK
        ; call CTLQ
        jmp FCHK     ; FlagCHecK
OK4:    mov FLAG1,00h

```



```

        jmp AGN      ; next Line

EXITP:  mov  al,BUFF ; xmit * to trainer
        call XMIT
        call SPACE
        call SPACE
        mov  ah,09h ; Transmission Ends Message
        mov  dx,seg ACK4
        mov  ds,dx
        mov  dx,offset ACK4
        int  21h

EXIT:   mov  ah,3Eh ; ASCII file close
        mov  bx,FHANDL
        int  21h

EXIT1:  mov  ax,4C00h
        int  21h
        ret

RDTX   ENDP

CTLQ   PROC NEAR
        mov  ah,00h
        int  16h
        cmp  al,51h
        jnz  OUTX

        mov  ah,3Eh ; ASCII file close
        mov  bx,FHANDL
        int  21h

        mov  ax,4C00h
        int  21h

OUTX:   ret
CTLQ   ENDP

NXTCHA PROC NEAR
        mov  ah,3Fh ;reading one byte from the current file pointer
        mov  bx,FHANDL
        mov  cx,0001h
        mov  dx,seg BUFF
        mov  ds,dx
        mov  dx,offset BUFF
        int  21h
        ret
NXTCHA ENDP

SPACE  PROC NEAR
        mov  ah,02h
        mov  dl,0Dh
        int  21h
        mov  ah,02h
        mov  dl,0Ah
        int  21h
        ret
SPACE  ENDP

DELAY  PROC NEAR
        mov  cx,0FFFFh
HERE:   loop  HERE
        ret
DELAY  ENDP
CODE   ENDS
        END  START

```

8.3 C Source Code Listing for IBM-PC's EXE Program

```
#include <stdio.h>
#include <bios.h>

#define MSG1 "\nNow Roll Calling for 8086 Trainer.....!\n"
#define STCMD (bioscom(2,0x00,0)) && 0x0006 != 0x0006
#define TXCMD 1,ch,0
#define MSG2 "\nAcknowledgment Not Received from the Trainer...!\n"

main ()
{
    char fname [80],ch;
    FILE *fp;
    int y;

    bioscom (0,0xC7,0); /* COM1 initialize at 4800Bd,NP,2-SB */
    printf (MSG1);

    do
    {
        bioscom (1,0x05,0); /* sending ASCII=05h=ENQ=Enquiry */
        delay (100); /* delay to absorb the data byte by the trainer */
    }
    while ((bioscom(2,0x00,0)) != 0x06);

    printf ("\nTrainer is Ready.....!\n");

    printf ("\nPress Capital-L to Load and Xmit ASCH File\n");
    printf ("\nEnter Pathname for the File.\n");
    gets (fname);

    fp = fopen (fname,"r");

    bioscom (1,0x09,0); /* 09h = ASCII Header Transmit (HT) */
    if (STCMD) /*checking for the ACK coming from Trainer */
    {
        printf(MSG2);
        exit(1);
    }

    printf ("\nNow Transmitting the RAM Starting Address....!\n");

    do
    {
        putchar(ch=getc(fp));
        if(ch != '\n')
        {
            if (ch != '-')
            {
                bioscom(TXCMD);
                delay (10);
                if(STCMD) /* ACK (06h) is checked and not received*/
                {
                    printf(MSG2);
                    exit(2);
                }
            }
        }
    }
}
```

```
while(ch != '\n');
```

```
putchar (getc(fp));  
bioscom(1,0x01,0);
```

/ End of RAM address Transmission */*

```
printf("\nNow Transmitting ONLY the Instruction Codes.\n");
```

```
do
```

```
{  
    putchar(ch=getc(fp));  
    if (ch == '\n')
```

```
    {  
        do
```

```
        {  
            if((ch=getc(fp)) != '\n')
```

```
            {  
                if(ch != ';')
```

```
                {  
                    bioscom(TXCMD);  
                    delay(10);  
                    putchar(ch);  
                    if(ch == '*')
```

```
                    {  
                        printf("\nTransmission Ends.....!\n");  
                        exit (0);
```

/ xmit of * marks the end of transmission */*

```
                    }  
                else
```

```
                {  
                    if (STCMD) /* ACK not received */
```

```
                    {  
                        printf (MSG2);  
                        exit(3);
```

```
                }  
            }  
        }  
    }  
    else /* one instruction xmission is ended */
```

```
    {  
        putchar(ch);  
        bioscom(1,0x01,0); /* 01h = SOH = start of header */  
        if (STCMD)  
            exit(4);
```

*end of one instruction transmission */*

```
    }  
    else
```

```
        putchar (ch);
```

```
    }  
    while (ch != ';');
```

```
while (ch != '*');
```

```
return (0);
```

8.4 Assembly Source Code Listing for Trainer's Firmware

8086 Trainer's Firmware Program Listing

Mainline Program Starts at F1600h.

```

F1600 - C7 47 70 00 00      : mov     [bx+70h],0000h      ; f70=00470h, f71=00471h
F1605 - C7 47 72 00 00      : mov     [bx+72h],0000h      ; f72=00472h, f73=00473h
F160A - C7 47 74 00 00      : mov     [bx+74h],0000h      ; f74=00474h, f75=00475h
F160F - BB 00 01             : mov     bx,0100h           ; new value for bx to set vector for int 44h
F1612 - C7 47 10 00 17      : mov     [bx+10h],1700h      ; (00110) = 00, (00111) = 17 for int 44h
F1617 - C7 47 12 00 F0      : mov     [bx+12h],0F000h     ; (00112) = 00, (00113) = F0 for int 44h
F161C - BB 00 04             : mov     bx,0400h           ; getting back bx's value

;now initializing the 8256's Rx/Tx
F161F - BA 00 34             : mov     dx,3400h           ; pointing at CR-1
F1622 - B0 22               : mov     al,22h             ; 8086 system, 2-SB, 8-Character
F1624 - EE                  : out     dx,al              ; done
F1625 - BA 02 34             : mov     dx,3402h           ; pointing at CR-2
F1628 - B0 35               : mov     al,35h             ; CLK prescaler=5 for 1024K,4800-Bd
F162A - EE                  : out     dx,al              ; done
F162B - BA 04 34             : mov     dx,3404h           ; pointing at CR-3
F162F - B0 E0               : mov     al,0E0h           ; Rx Enabled, Normal Int., IntAck enabled
F1630 - EE                  : out     dx,al              ; done
F1631 - BA 0A 34             : mov     dx,340Ah           ; pointing at Set Interrupt Register
F1634 - B0 14               : mov     al,14h             ; Interrupt enabled for L-4 of int 44h
F1636 - EE                  : out     dx,al              ; done
F1637 - BA 0C 34             : mov     dx,340Ch           ; pointing at Reset Interrupt register
F163A - B0 00               : mov     al,00h             ; data relating to interrupt
F163E - EE                  : out     dx,al              ; done
F163D - BE 78 00            : mov     si,0078h           ; si as pointer to save RAM start address
F1640 - FB                  : sti                     ; 8086's interrupt is enable
F1641 - EA 41 16 00 F0      : jmp     F000:1641          ; initialization done. Wait for interrupt.

```

Interrupt Service Routine for int 44h

```

F1700 - 80 7F 73 00        : cmp     BYTE PTR [bx+73h],00h ; if f73=1?
F1704 - 74 05              : jz      F000:170B           ;
F1706 - EA 00 70 00 00     : jmp     0000:7000          ; goto LBI and develop codes in RAM
F170B - 80 7F 74 00        : cmp     BYTE PTR [bx+74h],00h ; if f74=1?
F170F - 74 05              : jz      F000:1716           ;
F1711 - EA 60 17 00 F0     : jmp     F000:1760          ; follow LB2
F1716 - 80 7F 75 00        : cmp     BYTE PTR [bx+75h],00h ; if f75=1?
F171A - 74 05              : jz      F000:1721           ;
F171C - EA 40 17 00 F0     : jmp     F000:1740          ; follow LB3
F1721 - BA 0E 34           : mov     dx,340Eh           ; pointing at Rx buffer of 8256
F1724 - EC                 : in     al,dx               ; get the ASCII byte
F1725 - 3C 05              : cmp     al,05h             ; check if Roll Calling by the IBM-PC?
F1727 - 74 05              : jz      F000:172E           ; yes Roll Calling
F1729 - EA 35 17 00 F0     : jmp     F000:1735          ; no Roll Calling, just IRET
F172E - C6 47 75 01        : mov     BYTE PTR [bx+75h],01h ; allow LB3
F1732 - B0 06              : mov     al,06h             ; code for Acknowledgement to IBM-PC
F1734 - EE                  : out     dx,al              ; ACK sent to IBM-PC
F1735 - CF                 : ired                       ; ISR done for one byte ASCII character.

```

Codes for LB3:

```

F1740 - BA 0E 34           : mov     dx,340Eh           ; pointing at Rx
F1743 - EC                 : in     al,dx               ; reading code
F1744 - 3C 09              : cmp     al,09h             ; check if Marker for RAM Loc is coming?
F1746 - 74 05              : jz      F000:174D           ; yes Marker has arrived
F1748 - EA 54 17 00 F0     : jmp     F000:1754          ; no Marker, just iret
F174D - C6 47 74 01        : mov     [bx+74h],01h       ; allow LB2 to process ASCII for RAM Loc
F1751 - B0 06              : mov     al,06h             ; data for ACK
F1753 - EE                  : out     dx,al              ; done
F1754 - CF                 : ired                       ; ISR done

```

Codes for LB2 and LB10:

```

F1760 - BA 0E 34      : mov     dx,340Eh      ; pointing at Rx
F1763 - EC           : in     al,dx         ; get ASCII
F1764 - 3C 01       : cmp    al,01h        ; check if RAM Loc receive done
F1766 - 74 05       : jz     F000:176D     ; reception done
F1768 - EA 72 17 00 F0 : jmp    F000:1772     ; still to receive the RAM Loc codes
F176D - EA 00 60 00 00 : jmp    0000:60000    ; LB11-to mem to process RAM Loc ASCII
F1772 - 88 00       : mov    [bx+si],al    ; LB10-save the ASCII at RAM 00478b.....
F1774 - 56         : push  si            ; saving the si pointer
F1775 - 88 47 4E     : mov    [bx+4Eh],al   ; placinf at T2 of Reserved RAM
F1778 - 9A 7C F4 00 F0 : call  SUR#8         ; xrite T2(hex) to T1 (cc-code)
F177D - 9A B6 FF 00 F0 : call  SUR#3         ; xfer T1 to 8279 to display ASCII
F1782 - 5E         : pop    si           ; get si back
F1783 - 46         : inc   si            ; increment the memory pointer
F1784 - BA 0E 34     : mov    dx,340Eh     ; pointing at Tx
F1787 - B0 06       : mov    al,06h       ; code for ACK
F1789 - EE         : out   dx,al         ; done
F178A - CF         : ret                    ; ISR finished

```

Codes for LB11:

```

06000 - EA 60 16 00 F0 : jmp    F000:1660     ; linking to ROM from RAM
; process 1st ASCII to reconstruct SEGMENT Base of the Form X000
F1660 - 8A 47 78     : mov    al,BYTE PTR [bx+78h]; get 1st ASCII of RAM start address
F1663 - 9A B0 16 00 F0 : call  SUR*          ; to get Hex single digit from ASCII
F1668 - B4 00       : mov    ah,00h        ; to put 00 fo gettinh SEG=XD 00
F166A - 89 47 76     : mov    [bx+76h],ax   ; SEGMENT value at (00477)(00476)

```

;process next four ASCII codes to reconstruct OFFSET of the form XXXX

```

F166D - 8B 47 79     : mov    ax,[bx+79h]   ; getting next two ASCII codes
F1670 - 9A B0 16 00 F0 : call  SUR*
F1675 - 8A D0       : mov    dl,al         ; saving the 1st Hex digit
F1677 - 8A C4       : mov    al,ah         ; ASCII for the other hex digit
F1679 - 9A B0 16 00 F0 : call  #              ; to get next hex and form XX hex
F167E - B1 04       :
F1680 - D2 C8       :
F1682 - 24 0F       :
F1684 - 0A 02       :
F1686 - 88 47 78     : mov    [bx+78h],al   ; MSByte of OFFSET at (00479)(00478)

```

;process next two ASCII codes for LSByte of OFFSET

```

F1681 - 8B 47 7B     : mov    ax,[bx+7Bh]   ; getting next two ASCII codes
F1684 - 9A B0 16 00 F0 : call  SUR*
F1689 - 8A D0       : mov    dl,al         ; saving the 1st Hex digit
F168B - 8A C4       : mov    al,ah         ; ASCII for the other hex digit
F168D - 9A B0 16 00 F0 : call  #              ; to get next hex and form XX hex
F1692 - B1 04       :
F1694 - D2 C8       :
F1696 - 24 0F       :
F1698 - 0A C2       :
F169A - 88 47 79     : mov    [bx+79h],al   ; MSByte of OFFSET at (0047A)(00479)
F169A - 88 47 79     :
F169D - BA 0E 34     :
F16A4 - B0 06       :
F16A6 - EE         :
F16A7 -             : CF

```

SUR* Codes:

```

F16B0 - 8A E8       : mov    ch,al         ; temporary storage of the ASCII code
F16B2 - 24 10       : and   al,10h        ; checking if ASCII of (31 - 39)
F16B4 - 74 09       : jz     F000:         ; not ASCII for 0 -9, it is for A - F
F16B6 - B1 04       : mov    cl,04h       ; counter for rotation
F16B8 - 8A C5       : mov    al,ch         ; get the uncorrupted ASCII
F16BA - D2 C0       : rol   al,cl         ; processing to get hex digit

```

```

F16BC- 24 F0      : and    a10F0h      ; processing to get hex digit
F16BE- CB        : ret                      ; done

; hex digit is A - F
F16BF- 8A C5      : mov    al, ch      ; get the uncorrupted ASCII
F16C1- 04 09      : add    al, 09h     ; processing to get hex digit
F16C3- B1 04      : mov    cl, 04h     ; counter for rotation
F16C5- D2 C0      : rol    al, cl      ; processing to get Hex digit
F16C7- 24 F0      : and    a10F0h     ; hex got for A - F
F16C9- CB        : ret                      ; done

SUR # codes:
F16D0- 24 0F      : and    a10Fh      ; process to get hex digit
F16D2- 0A C2      : or     al, dl      ; got of the form xx
F16D4- CB        : ret                      ; done

```

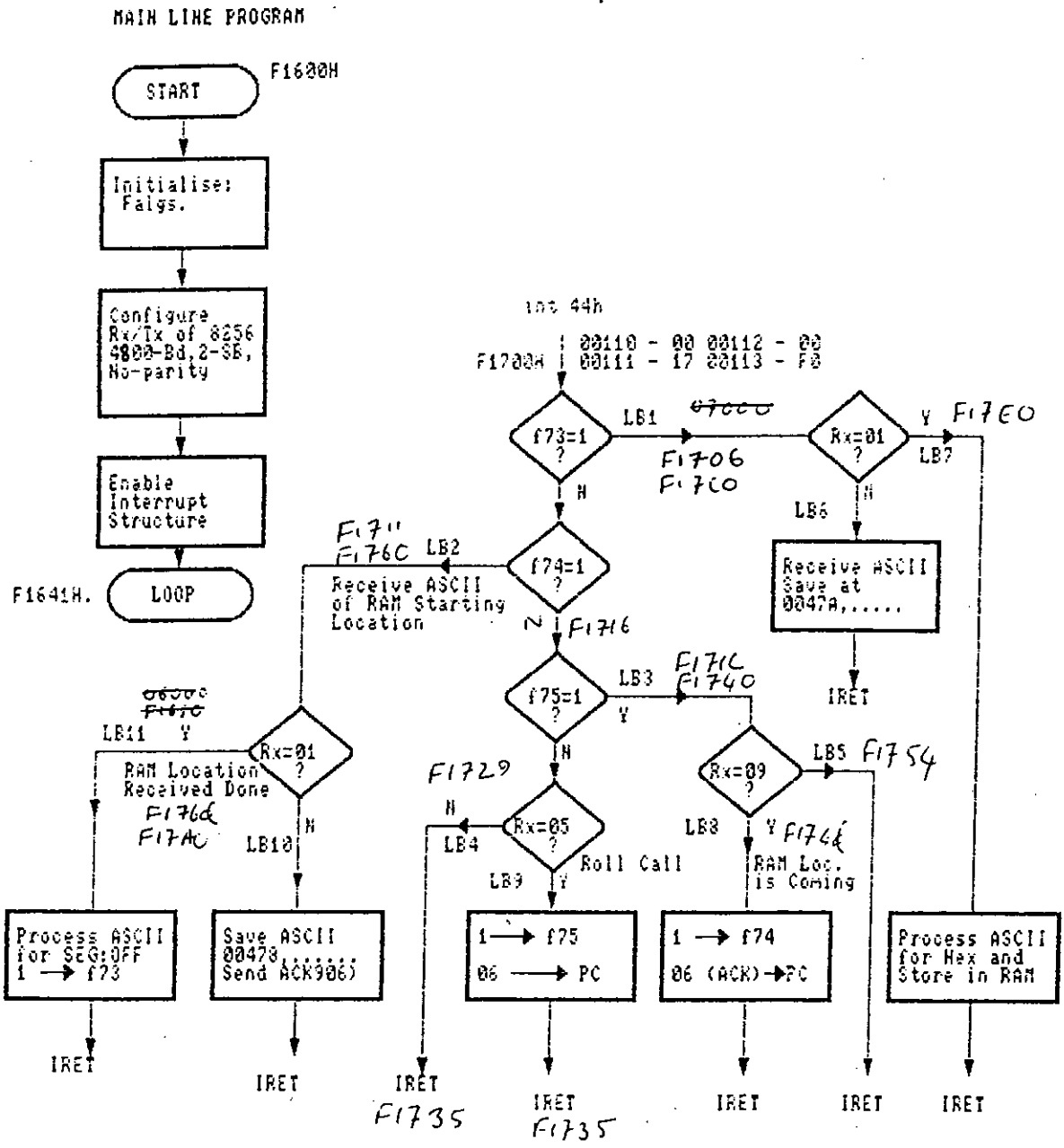


Fig - 8.4 : Software Flow Chart for Trainer's Firmware

9

RESULTS AND DISCUSSION

RESULTS

Given below a short table indicating the Actual Result versus the Expected Result. The reasons for the anomalies have been discussed in the next page.

Item	Expected Result	Actual Result
01.	Driving the on-board memory devices without using bi-directional data buffers like 74LS245/244.	Working fine. At present there are 5 memory chips on the bus.
02.	EPROM based composite memory and port decoder in order to reduce the components counts and increase the organizational beauty of the board.	One 2716 EPROM is found to be just good for the purpose. Unused decoded lines are for EVEN ports.
03.	Getting all the bus signals available at edge connectors for Interfacing Experiments	They are available
04.	Getting available at edge connector the multiplexed (A3-A0)(B3-B0) display signals along with the corresponding scan signals for driving extra (16-9) seven-segment display devices.	They are available. But seems to be not working as expected.
05.	HLDA signal implementation to disable the bus devices like 74LS373s to allow DMA device	Can not be easily connected a DMA device in the system.
06.	Every key should have double functions without any Shift/Control Key.	Successfully done
07.	Modularity in he design of the Monitor Program.	Not fully achieved.
08.	Typing error correction both in address and data field.	Fully achieved
09.	Generalized subroutine for printing digit at any position of the display unit.	Implemented but limited utility.
10.	The content of bx register could not be changed during single stepping.	Changeable by changing bl, bh.
11.	Memory Location can be examined and changed without exiting the single step environment.	Not implemented
12.	Port location can be examined and changed during program debugging.	Not implemented
13.	Separate Integrated Peripheral Module containing all the common I/O functions.	Developed

DISCUSSION

Item - 01:

The 8086's line can source about 400uA current, That means that it can drive only one TTL gate. At present, the 8086 will at best drive two CMOS gates (either two EPROMs or two RAMs at the same time for word operation) which will not overload the bus. It has been experimentally verified that the bus can be connected to the IPM of section-6 without any data buffers. However, if the question of connecting TTL RAM (74S189) occurs, then one has to condition the bus lines using suitable data buffers like 74LS245.

Item - 02:

Thanks to 2716 EPROM for having 11 address lines. It has made it possible to decode the memory chips of 32Kbyte size with full decoding. The decoder has 8 active low outputs. 5 of them have gone to 5 on-board memory/port chips. The remaining 3 are available at edge connector as decoded lines for accessing EVEN addressed byte-oriented ports. If someone wishes to access ports at ODD address, then the contents of the EPROM has to be changed to meet the requirements. This can be easily done by consulting the section-4.3.

Item - 04:

The 8279 can drive 16 seven-segment display devices. In the trainer, only 9 are used. The remaining 7 can be added by the user to monitor some of the data of interest. The data and the scan lines of the 8279 are made available at edge connectors J7 and J6. The functionality of this scheme has been verified. But, the data does not remain in the added display device when the control is transferred to the monitor program. It is good as long as the control remains in the executing program. The beginning part of the monitor program needs to be read and analyzed properly for finding out the clue of this problem.

Item - 07:

Full modularity in the design and implementation of the monitor program could not be achieved and it is mainly due to manual coding of the instructions. A survey of the flow charts at section-5.1 indicates that the program development is tree structure. This is good for understanding the program logic but creates problem to maintain the program codes. There is also difficulty to modify the program should a need arises. However, still a good amount of modularity has been achieved by creating many subroutines.

Item - 09 :

Digit printing in the display unit will occur while examing/editing memory/register contents. There is no separate 'Digit Printing Routin' for each situation. Rather, a generalised 'Printing Subroutine' has been developed which is being called with appropriate parameters.

Item - 10:

The bx register is used by the single stepping mechanism as a pointer. If the content of this register is changable in the Single Stepping environment, the Single Step routine fails. Therefore, the content of the bx register can only be examined. However, due to an error, the content can be changed using CHG command, while the bx register is examined as bl and bh. The users need to remain aware about it.

Item - 11 and Item -12 :

It is a necessary routine that allows checking the memory content without exiting the single stepping routine. However, due to time constraint, this routine could not be developed and implemented. The key labeled as 0/PRT may be used to examine/edit port contents. The routine is yet to be developed.

10

**CONCLUSION
AND
FUTURE SCOPE
OF
WORKS**

Conclusion:

Design, development and the construction of a microprocessor trainer requires the same setup as required for any other PCB assembly line. The set up consists of :-

01. Trained engineers and technicians
02. PCB (at least double layer) manufacturing facilities
03. Development Tools and software

In spite of the lacking of the minimum requirements of the above support, the 8086 trainer has been realized, and it has been functioning properly.

The advent of this trainer has established a theory that 'Creative Idea' always gets transformed into reality regardless of the degree of obstacles and it is attributed to the 'Strong Desire' of the team working behind the project.

This work will motivate the researchers of the various organizations to transform their 'Hookup Wiring' circuitry into manually PTHed double layer PCBs.

There are virtually endless number of consumer products and industrial controllers that could be developed using programmable devices like Microprocessors. However, due to non-availability of the trainers, the concerned designers could not work on 'Microprocessor Based Design'. The locally developed 8086 trainer would help the engineers designing their products based on microprocessor. A microprocessor product requires less maintenance. It is low cost and highly reliable.

Microprocessor is a common subject to offer in the curriculum of electrical/electronic/computer engineering and other disciplines of applied sciences in home and abroad. In Bangladesh, the practical classes of the microprocessor courses are conducted based on a few imported 'Trainers' for which there is no Laboratory Manuals. The developed trainer, if adopted by the educational institutes of Bangladesh would allow the respective teachers to write and develop Microprocessor Laboratory manuals and make the teaching more practical and complete.

The success of this work makes an indication that all types of educational trainers at least of electrical/electronic type may be developed at home.

To make a programmer, there is a need of PC which must be available as a low cost consumer product. Likewise, to make an engineer (who will be making a machine - the computer), there is a need of low cost microprocessor trainer. The realization of this trainer has opened the door for a graduate to become an engineer by exercising his theoretical knowledge on this trainer.

Having a look at the trainer, it can be said that designing a computer does not require the so called talents. Rather, it requires 'A File' that contains the systematic methods of developing a product. People can be trained to follow the steps of the 'File'.

It is believed that the present thesis will serve as such a 'File'.

Future Scope of Work

The following areas have been identified as potential areas where good technical work can be carried out based on this trainer.

On-board I/O Peripherals:

This is a basic trainer. It is built keeping in mind that it will be used by the students for learning purposes. The trainer does not contain any on-board I/O peripherals. Therefore, the product designers will face some difficulties in developing a controller. A new PCB may be designed incorporating the Intel's MUART chip 8256 which contains all the five common functions (Parallel I/O, serial I/O, Timing Functions, Counting Functions and Interrupt Priority management) in a 40-pin IC.

On-board 8087 Math Coprocessor:

Many applications may require the number crunching jobs where the 8086 will certainly be slow. Therefore, incorporating an 8087 chip on the board might be good idea. A new PCB could be designed to add an 8087 and operate the 8086 in Maximum Mode. (Refer to Appendix-A).

ROM-BASED Assembler:

The present trainer will require an IBM-PC and the Macro-assembler for getting the machine codes for the instructions to be entered into the trainer for execution. This can be easily eliminated if some one comes forward to write a ROM based assembler for this trainer. The author has given some hints in this regard in Appendix-B.

80286 Trainer:

8086 is an advanced processor, but it does not include the PVAM (Protected Virtual Addressing Mode) concepts. 80286 and later microprocessors support this concept. These processors always boots up in 8086 mode. Therefore, an attempt could be made to develop an 80286 trainer based on the 8086 trainer for the understanding of the PVAM concepts.

Improving the Monitor Program:

The monitor program of the trainer could be improved in many areas. Moreover, many routines are yet to be developed and fused into the EPROMs. The following areas are being pointed:-

Treating a memory Location as a Simple Register in the Single Step

At present, there is no provision for checking the content of a memory location after the execution of a memory reference instruction in the single step mode. To do it, one has to EXIT from the single step, check the memory content and then re-enter manually. This can easily be done still being in the Single Step environment if the RAM location could be treated as simple register. The author has given some in this regards in section-5.4.9.

Examining/Editing Port Content:

This is an useful routine which the trainer does not have. A key has been assigned for this purpose but there is no software routine to response to the key's request. Some works may be done to write and implement this routine. For hints please see section - 5.4.10

Software Documentation:

The full documentation of the monitor program has not been done. Some more works could be done in this regard because 'A software exists by its documentation'. Without systematic documentation of the monitor program, future modification of the monitor program would be simply impossible.

APPENDIX - A

MAXIMUM MODE OPERATION OF 8086 MICROPROCESSOR WITH 8087 FLOATING POINT UNIT

A.1 Schematic of 8087 with 8086

The design of the 8086 CPU is not optimized to carry out fast mathematical calculations. Virtually, any kind of mathematical calculations can be carried out using 8086 instructions, but they will run very slow. Therefore, a specialized processor named as math coprocessor or Floating Point Unit of the type 8087 has been designed to carry out all types of number crunching tasks. Since, the FPU is not a general purpose processor, it has to be operated in parallel with a general purpose processor like 8086.

The 8086 CPU is housed in a 40-pin package and is usually used in its minimum mode where there is no need of connecting any other co-processor like 8087. The schematic diagram of Figure-A.1 shows actual implementation of the 8086's maximum mode having connection with an 8087 and an 8288. A clock generator chip of type 8284 has also been shown.

U2 : 8087 Math Co-processor

Double arrows near the A16/S3 - A19/S6 lines indicate that the 8087 can assert address information while operating on the memory devices. It can also monitor the status signals S3-S6 being asserted by the 8086 when the bus is under 8086's control. The same reasoning holds good for the BHE/-S7 line.

Double arrows near the S0/-S2/ lines also indicate that the 8087 asserts the encoded signals while it is the master of the bus. It also monitors the 8086's status signals S0/-S2/ while 8086 is the bus master.

The 8087 gains the bus mastership from the 8086 using the RQ/-GT0/ lines. This is also a bi-directional line and the single line furnishes the 'Bus Request', 'Bus Grant' and 'Bus Release' operations between the 8086 and 8087. The RQ/-GT1/ line is strapped to +5V meaning that no other coprocessor can be cascaded in the system.

The 8087 uses the QS0 and QS1 lines to monitor the status of the Instruction Prefetch Queues of the 8086 for the purpose of copying the instruction bytes of those queues .

The 8087 has an interrupt line labeled as 'INT'. This pin goes high whenever there occurs an error inside the 8087. This line can be routed to the 8086 CPU via an optional 8259. Or, it can be directly connected to the NMI line of the CPU.

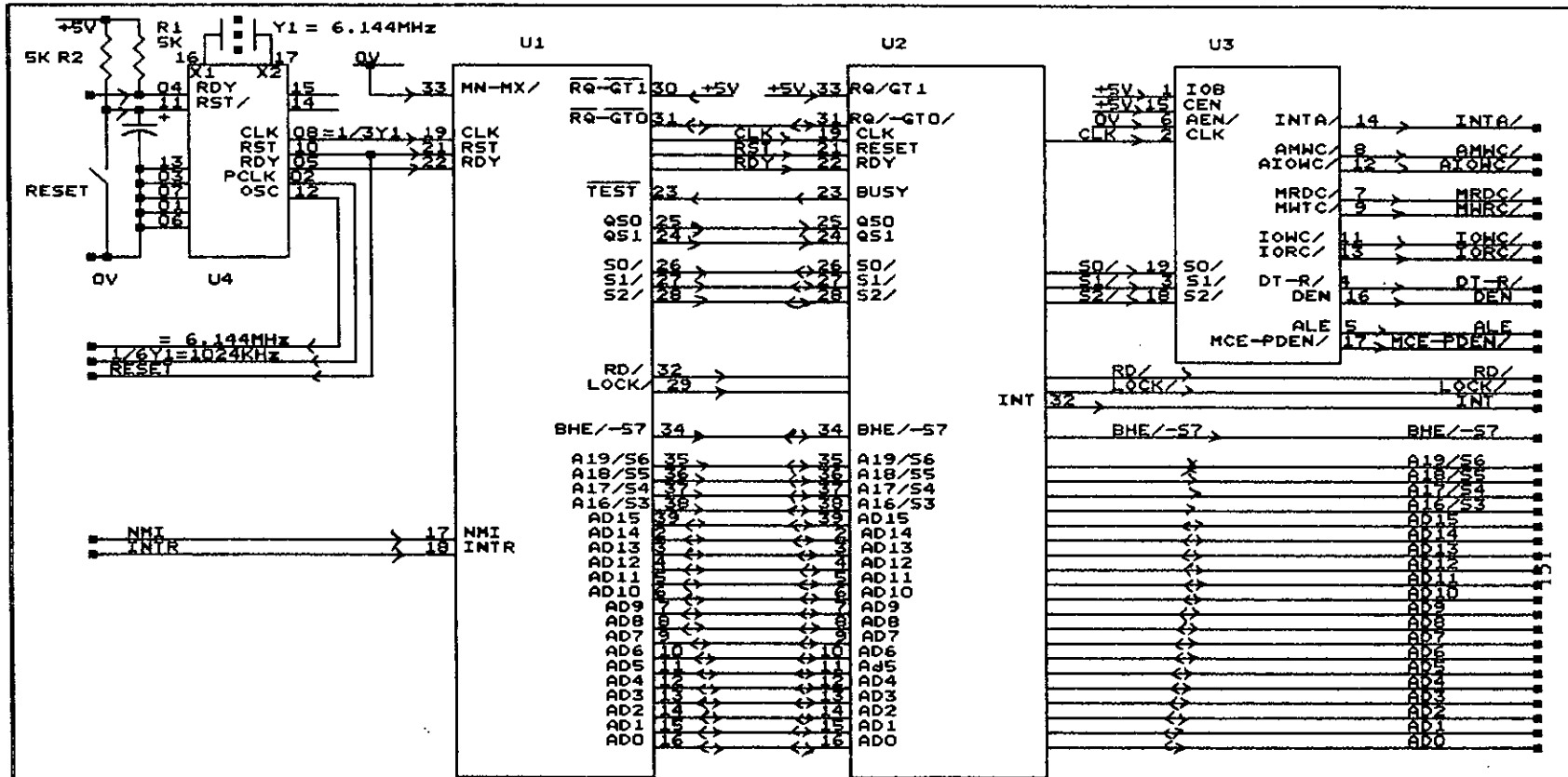
U1 : 8086 Microprocessor

MN-MX/ pin strapped to +5V in order to allow it for generating the signals shown against its pins. The CPU generates the encoded status signals S0/-S2/ which get decoded by the bus controller 8288. The output of the 8288 chip indicate the various timing functions required for interfacing RAMs, ROMs, Interrupt Controller and I/O devices. It also generates the ALE signal for sampling the address information from multiplexed bus system carrying composite signals like AD15-D00.

The 8086 has two bus transfer line viz., RQ/-GT0/ and RQ/-GT1/. In the present case only one has been used and the other one has been disabled by strapping at +5V.

Bus Arbiter?

Do we need a Bus Arbiter here? Why not? Or why? There are two processors 8086 and 8087. They will share the same data memory. Therefore, there may be a chance of bus conflict! If this would be the case, then the bus arbiter 8289 could be employed in the system to generate the AEN/ and CEN signals of the 8288. Under this circumstances, the IOB input line of the 8288 has to be strapped to +0V indicating the Bus System.



IC	TYPE	DESCRIPTION	+5V	0V
U1	8086	Microprocessor	40	20, 1
U2	8087	Math Coprocessor	40	20, 1
U3	8289	Bus Controller	20	10
U4	8284	Clock Generator	20	10

DEPARTMENT OF COMPUTER SCIENCE & ENGG :BUET	
Title	
8086 Trainer with 8087 FPU	
Size	Document Number
A	c:\msctyhe\b.sch
Date:	January 3, 1998 Sheet 1 of 1

D&D By: Golam Mostafa

Fig - A.1 : Schematic of 8087 with 8086

A.2 Demonstration Example

The prototype 8087 trainer is built using the 8086 trainer. The 8086, 8087 and 8288 are placed on the bread board. The circuit as per Figure-A.1 has been implemented using hook up wires. The system has come up in maximum mode and allowed execution of all 8086 instructions. To verify that the 8087 is also working, the following program codes are entered in the trainer and executed. The result is found as expected.

The following program employs both the 8086 and 8087 to add two numbers viz., 03h and 04h. If the program is executed successfully, the result 07h will be stored at memory location 0047Ch by the 8087.

```
05000 - C7 47 74 03 00      : mov     WORD PTR [bx+74h],0003h      ; 8086 is storing data 0003h at 004745 and 00474      ;01
05 005 - C7 47 76 00 00      : mov     WORD PTR [bx+76h],0000h      ; 8086 is storing data 0000h at 00477 and 00476      ;02
0500A - 9B                  : wait                                     ; 8086 is waiting, next instruction for 8087      ;03
0500B - D9 47 74            : fld     DWORD PTR [bx+74h]           ;8087 getting data into its ST(0) from 00474 - 00477 ;04
0500E - 9B                  : wait                                     ; 8086 is waiting                                     ;05
0500F - DD D1              : fst     ST(1)                        ; 8087 is moving content of ST(0) into ST(1)         ;06
05011 - C7 47 78 04 00      : mov     WORD PTR [bx+78h],0004h      ; 8086 is putting data 0004h at 00479 and 00478      ;07
05016 - C7 47 7A 00 00      : mov     WORD PTR [bx+7Ah],0000h      ; 8086 is putting data 0000h at 0047B and 0047A      ;08
0501B - 9B                  : wait                                     ; 8086 is waiting, next instruction is for 8087      ;09
0501C - D9 47 78            : fld     DWORD PTR [bx+78h]           ; 8087 is getting data into its ST(0) from 00478 - 0047B;10
0501F - 9B                  : wait                                     ; 8086 is waiting, next instruction is for 8087      ;11
05020 - D8 C1              : fadd   ST(0), ST(1)                  ; 8087 is doing the addition                          ;12
05022 - 9B                  : wait                                     ; 8086 is waiting, next instruction is for 8087      ;13
05023 - D9 5F 7C            : fstp   DWORD PTR [bx+7Ch]           ; 8087 is storing data at 0047F - 0047C = 00 00 00 07 ;14
05025 - EA 00 00 00 F0      : jmp     F000:0000                    ; goto prompt and display message 8086 UP            ;15
```

APPENDIX - B

8086 TRAINER WITH BUILT-IN ASSEMBLER

B.1 Introduction

The 8086 trainer introduced in this thesis is a very basic one. It has only a hex keyboard and a 9-digit 7-segment display unit with a friendly and powerful EPROM-based monitor program. The trainer board does not contain any port devices. The trainer will allow any user to study, analyze and experiment any of the 8086 instructions and the addressing modes. The trainer is equipped with edge connectors for conducting interfacing experiments.

Because of so many (about 24) addressing modes of the 8086 microprocessor, it is not very easy to code the assembly mnemonics into binary values by hand. Therefore, the common practice is to use the Assembler package and get the machine codes for the instructions of the program intended to be executed in the trainer. This requires the need of IBM-PC.

The aim here is to work on the upgrading of the basic trainer so that it becomes independent of IBM-PC for coding the assembly mnemonics. It means that a EPROM-based assembler is to be designed. The trainer must contain an alphabetic keyboard and an alphanumeric display unit so that instructions can be entered in plain text. The trainer should retain the original hex key pad in order to program in machine codes.

The author claims that he has conducted sufficient works and experiments in order to test the possibility of realizing such a sophisticated trainer with the available hardware. There is a good sign of getting a trainer of this kind in near future if a suitable candidate could be found to carry out the remaining jobs which is mainly developing the translation and code generation routines. The summary of the works so far conducted is given below:-

<i>Works Conducted</i>	<i>Implemented Using</i>	<i>Result</i>	<i>Reference</i>
Alphanumeric Display	MAN2815 (15 Segment),8279	OK	C.2: U20,U13; C.3 : D1 - D16, U13
Characters ROM	2716	OK	C.2 : U14-U15; C.3 : U14 - U15
Keyboard Interrupt	8256	OK	C.2 : U11; C.3 : U13-4 (IRQ)
Alphabetic Keyboard	8279	OK	C.2 : C.3 : U13, K11 - K88
Sample Line Editor	Machine Codes	OK	--- C.3 : Page-156

Section - B.2 depicts the board layout of the proposed assembler-based trainer. The trainer now has used the Intel's versatile MUART chip of type 8256. It is a Multifunction Universal Asynchronous Receiver/Transmitter and contains all the five common functions usually required in a microcomputer applications viz., (I) Parallel I/O, (II) Serial I/O, (III) Counting Functions, (IV) Timing Functions and (V) Interrupt Priority Management. Some additional Ics viz., MC1488 and MC1489 have been used to make the serial communication to RS232 standard in order to communicate with an IBM-PC over the COM1/COM2 port should there arises a need of recording data in the hard disk or on the color monitor.

Section - B.3 contains detailed schematic of the alphanumeric display unit and the keyboard. The drawing also contains the detailed structure of the key matrix along with their scan codes.

Section - B.4 depicts the internal diode matrix structure of the alphanumeric display unit of the type MAN2815. It is to be noted that the display unit is of multiplexed type. The display must be driven by a character ROM.

Section - B.5 is the data table of the character ROMs which are designated as U14 and U15 in the schematic diagram in Figure-B.2. The data table has been prepared by hand first and then were fused in the EPROMs of type 2716. They work all right.

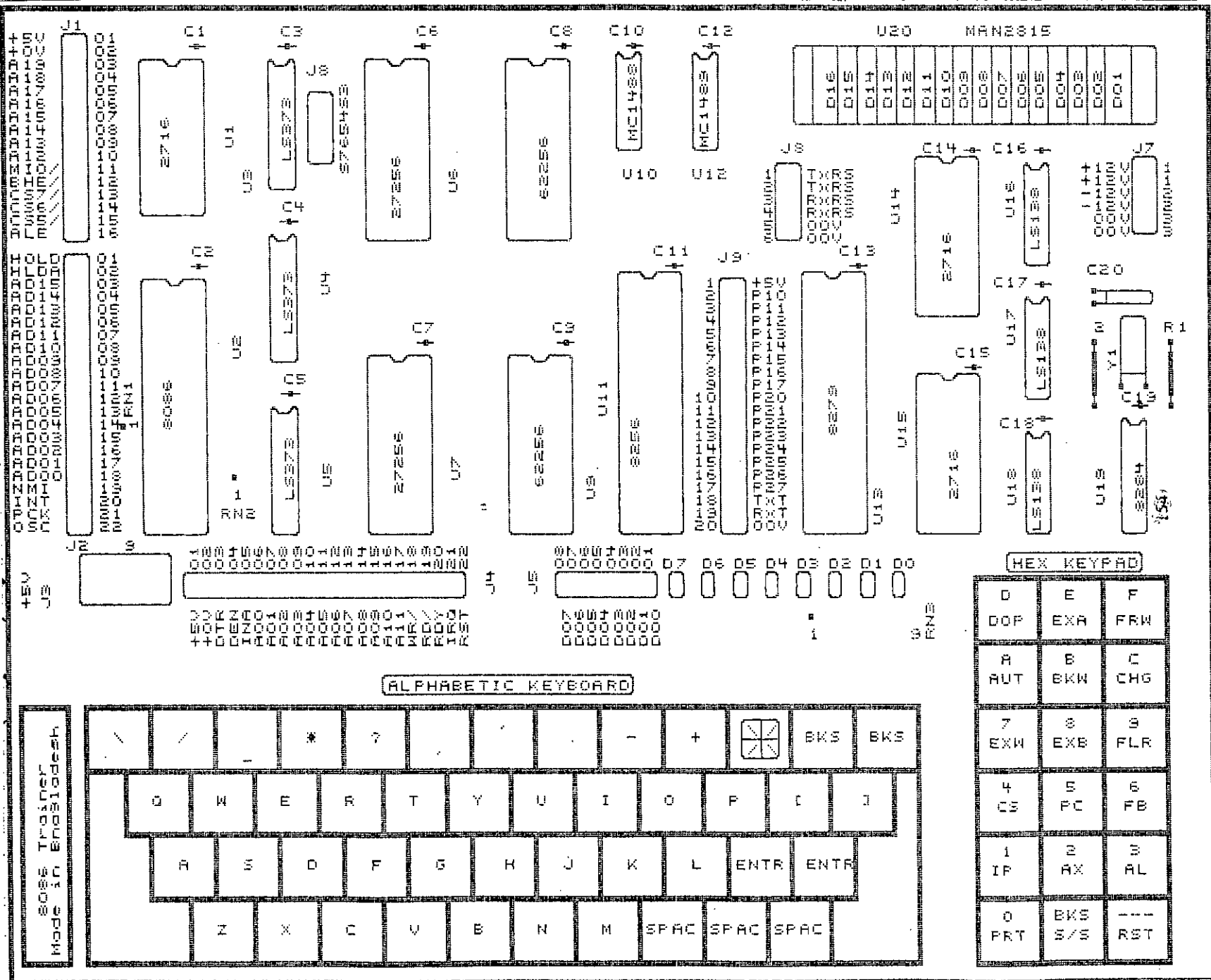


Fig - B.1 : Component Layout of the Proposed Trainer

B.3 Alphanumeric Display/Keyboard

U20: Multiplexed Display - Type MAN2815

This is a 15-segment display device. Please see section-B.4. To activate the 15-segments, the 8-bit user data lines of the 8279 have been expanded to 15-lines by incorporating the chips U14 and U15. U15 drives the segments a - h and U14 drives the segments i-dp. The common cathodes of the display devices are driven by the decoders U16 and U17.

U13: Keyboard/Display Controller

It is initialized to drive 16-digits left entry and a 2-key lockout keyboard.

U14,U15 : ROM Characters

Please see section-C.5 for the details of data fused in these two EPROMs. Say, we wish to display character 'A' at D16 position. To activate the required segments, we consult the data chart of page-158 and see that U14 should output 00h and U15 should output F7h. The data chart also indicates that these two data bytes are fused at locations 00h of the U14 and U15 which are essentially EPROMs. Therefore, in order to get these two data bytes out, we need to output a data value 00h at A3-A0 and B3-B0 lines of the 8279. That means that writing a data value 00h at Display RAM location 01h of the 8279 will accomplish everything wanted.

Demonstration Routine:

;initializing the 8279

assumed to be already done at power up.

;displaying 'A' at D16 position

```
0500C - mov    al,80h          : B0 80          ; control byte to be able to write data at display
0500E - out    dx,al          : EE            ; RAM location 00h corresponding to D16
0500F - mov    dx,0000h       : BA 00 00     ; pointing at data register of the 8279
05012 - mov    al,01h         : B0 01          ; data byte for RAM location 00h
05014 - out    dx,al          : EE            ; data is written
05015 -                                     ; we shall see character 'A' at D16 position.
```

In real situation, there will be a blinking cursor probably of the type * (star). The CPU will fully remain busy in blinking the cursor. Therefore all other tasks including the keyboard entry would be served on interrupt basis. Thus the IRQ line of the 8279 has been funneled to the INTR line of the 8086 via the ExINT line of the 8256.

Full syntax rules of the commercial assembly language like Macro-assembler could not be followed in this case due to lack of sufficient symbols in the keyboard. Even there is no ; (semi colon) sign in the keyboard. The cursor * (star) itself can be used as the line termination symbol of an assembly instruction. The following type of instructions can be entered using the alphanumeric features of the keyboard. For fully syntaxed system, the display unit has been converted to a single/multiline graphics/dot matrix one. Our one is a simple 16-digit display unit. However, it should be good enough to implement a workable system at least for learning purposes.

Maximum Length Instruction in Assembly : L2: mov WORD PTR cs:[bx][di+1234h],al ;

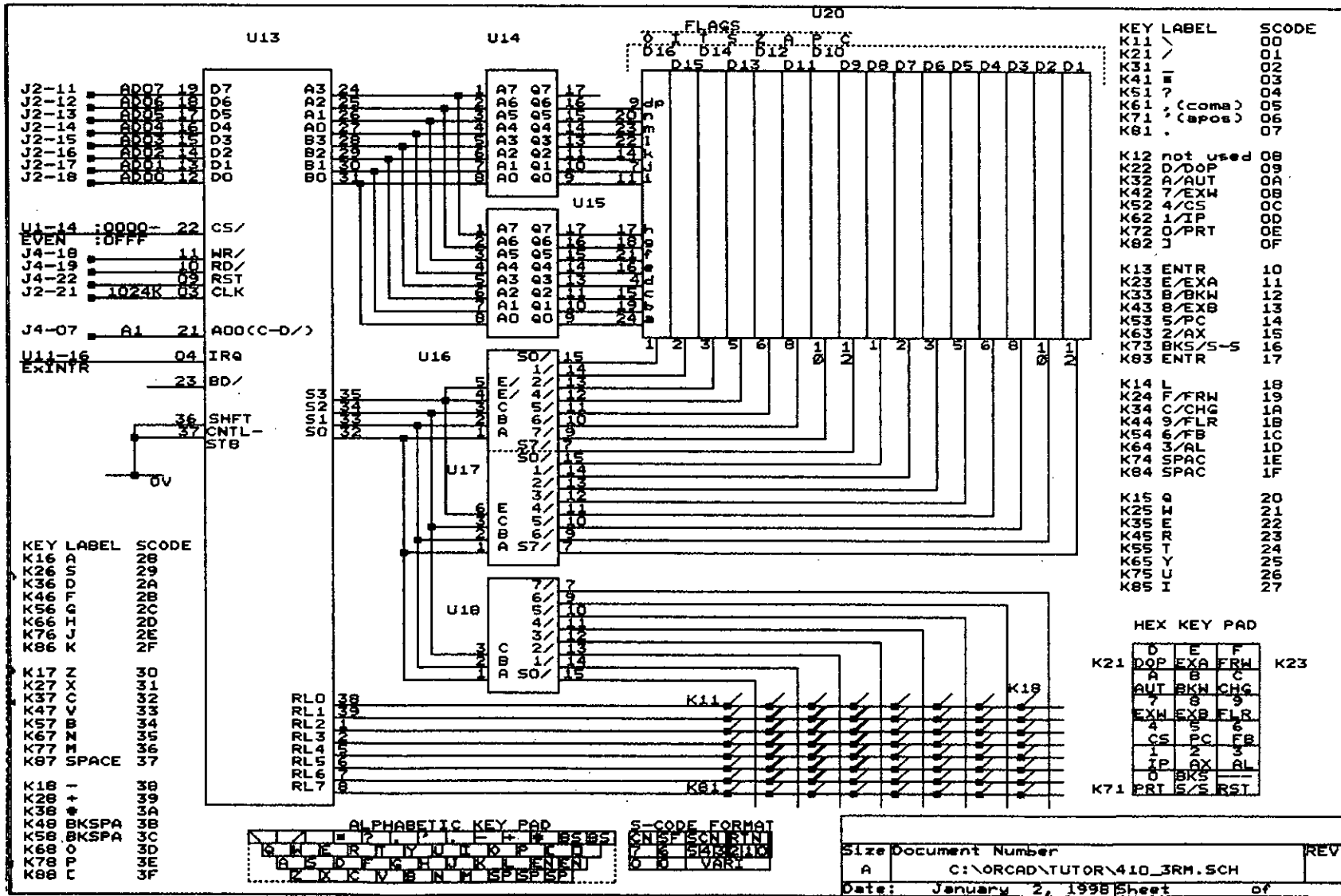
Implementation in the Proposed Trainer : L2\ MOV BYTE PTR CS.[BX][DI+1234H],AL*

\ - backward slash will work as : (colon) for LABEL

. - full stop will work for the : (segment override prefix sign)

* - star sign will work for the end of the instruction line

Because there is only 16-digits, the entry will shift to left. It will be retained in RAM.



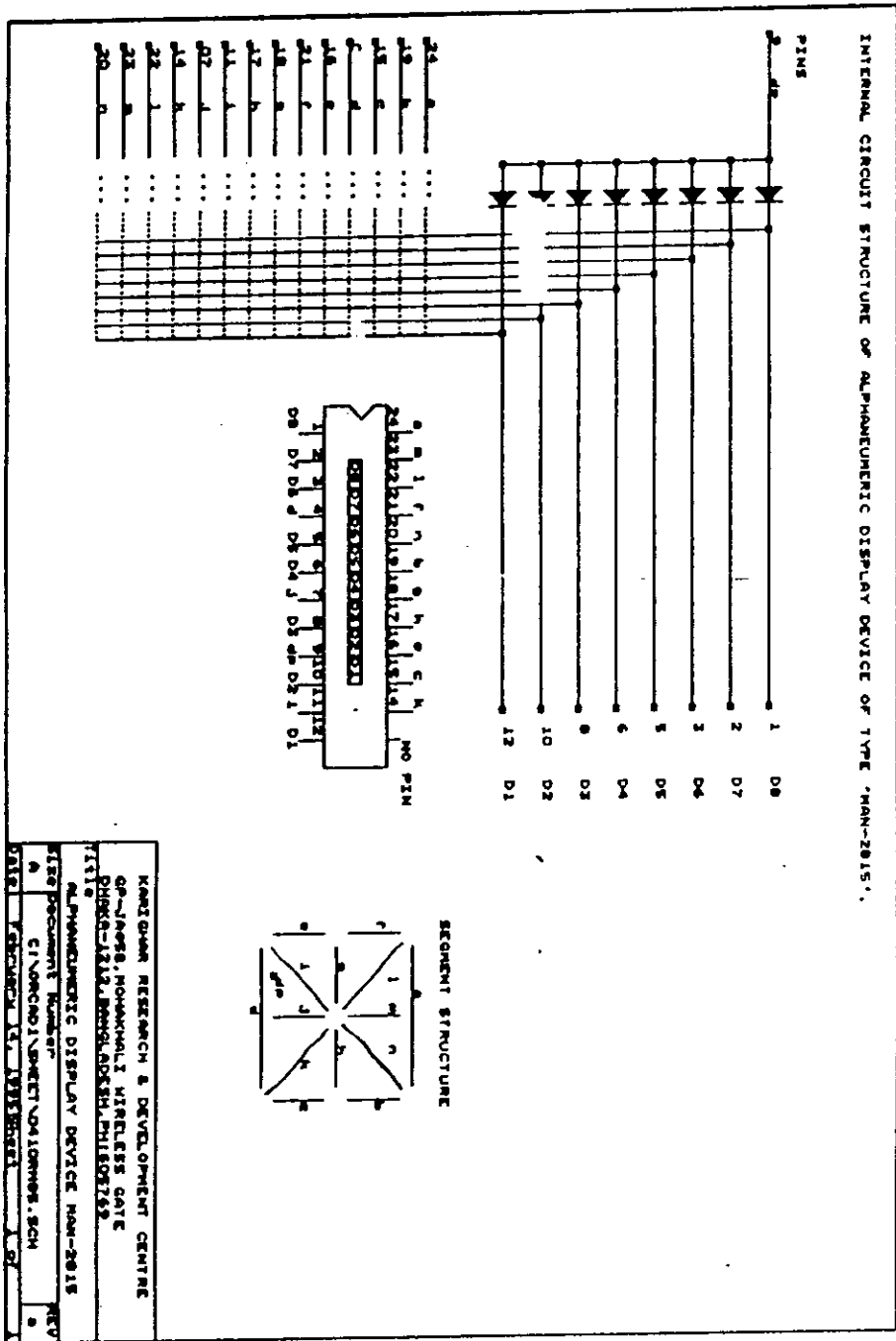


Fig - B.2(b) : Internal of MAN2815 Display Device

APPENDIX - C

SELECTED DATA SHEETS



8279/8279-5 PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

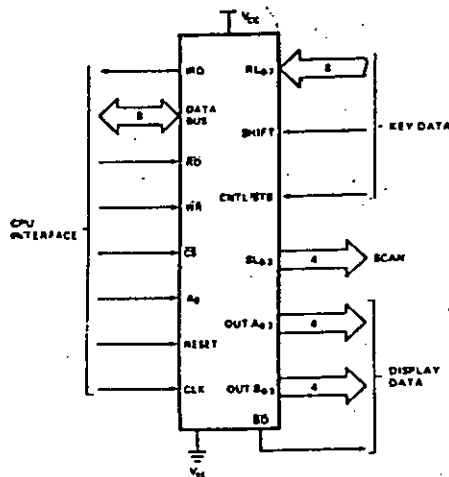


Figure 1. Logic Symbol

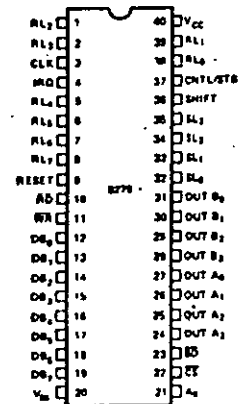


Figure 2. Pin Configuration

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Descriptions

Symbol	Pin No.	Name and Function
DB ₀ -DB ₇	8	Bi-directional data bus: All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	1	Clock: Clock from system used to generate internal timing.
RESET	1	Reset: A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display—left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	1	Chip Select: A low on this pin enables the interface functions to receive or transmit.
A ₀	1	Buffer Address: A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
RD, WR	2	Input/Output Read and Write: These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	1	Interrupt Request: In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V _{as} , V _{cc}	2	Ground and power supply pins.
SL ₀ -SL ₃	4	Scan Lines: Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL ₀ -RL ₇	8	Return Lines: Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed input mode.

Symbol	Pin No.	Name and Function
SHIFT	1	Shift: The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure puts it low.
CNTL/STB	1	Control/Strobed Input Mode: For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure puts it low.
OUT A ₀ -OUT A ₃ OUT B ₀ -OUT B ₃	4 4	Outputs: These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL ₀ -SL ₃) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
BD	1	Blank Display: This output is used to blank the display during digit switching or by a display blanking command.

FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

Input Modes

- Scanned Keyboard — with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0, A_3 = D_7$).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display Interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the \overline{CS} , A_0 , \overline{RD} and \overline{WR} lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by \overline{CS} . The character of the information, given or desired by the CPU, is identified by A_0 . A logic one means the information is a command or status. A logic zero means the information is data. \overline{RD} and \overline{WR} determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS} = 1$), the devices are in a high impedance state. The drivers input during $\overline{WR} = \overline{CS}$ and output during $\overline{RD} = \overline{CS}$.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with $A_0 = 1$ and then sending a \overline{WR} . The command is latched on the rising edge of \overline{WR} .

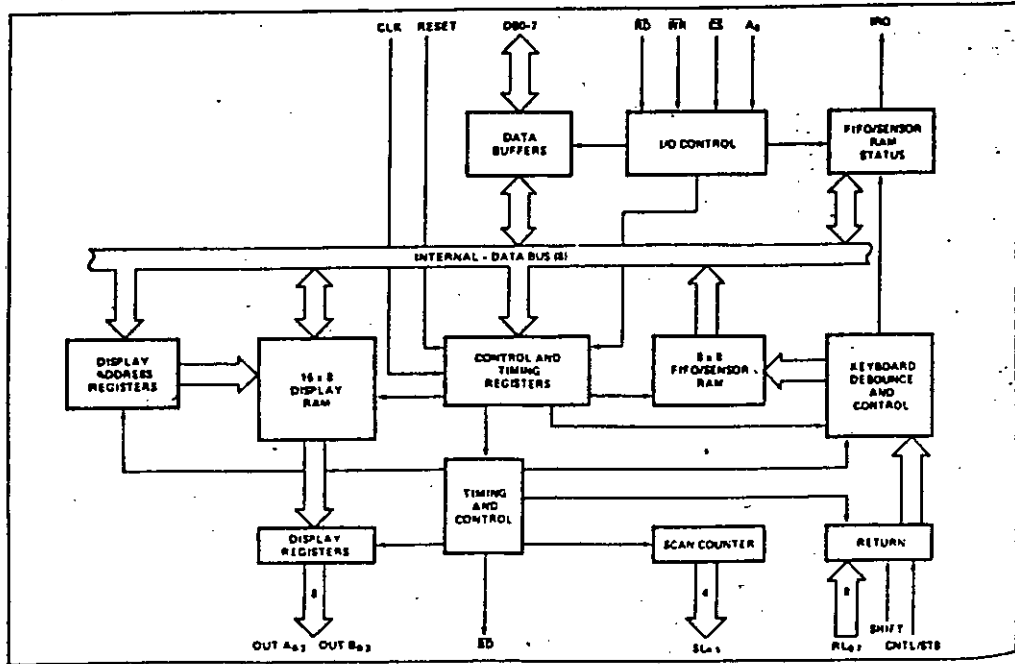


Figure 3. Internal Block Diagram

6-404

APN 007478

The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a ÷ N prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note that when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row. If the debounce circuit detects a closed switch, it waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM. In Keyboard or Strobed Input modes, it is a FIFO. Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an RD with CS low and A0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

SOFTWARE OPERATION

8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with CS low and A0 high and are loaded to the 8279 on the rising edge of WR.

Keyboard/Display Mode Set



Where DD is the Display Mode and KKK is the Keyboard Mode.

DD

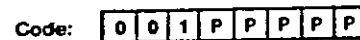
- 0 0 8 8-bit character display — Left entry
- 0 1 16 8-bit character display — Left entry*
- 1 0 8 8-bit character display — Right entry
- 1 1 16 8-bit character display — Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

KKK

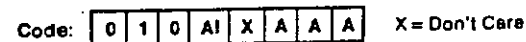
- 0 0 0 Encoded Scan Keyboard — 2 Key Lockout*
- 0 0 1 Decoded Scan Keyboard — 2-Key Lockout
- 0 1 0 Encoded Scan Keyboard — N-Key Rollover
- 0 1 1 Decoded Scan Keyboard — N-Key Rollover
- 1 0 0 Encoded Scan Sensor Matrix
- 1 0 1 Decoded Scan Sensor Matrix
- 1 1 0 Strobed Input, Encoded Display Scan
- 1 1 1 Strobed Input, Decoded Display Scan

Program Clock



All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM



The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-

*Default after reset.

board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are Irrelevant. The 8279 will automatically drive the data bus for each subsequent read ($A_0=0$) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set ($AI=1$), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code:

0	1	1	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set ($AI=1$), this row address will be incremented after each following read or write to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read or write address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code:

1	0	0	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with $A_0=1$, all subsequent writes with $A_0=0$ will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

Code:

		A	B	A	B		
1	0	1	X	IW	IW	BL	BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag ($IW=1$) for one of the ports, the port becomes masked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A_3 corresponds to bit D_7 .

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code:

1	1	0	C_D	C_D	C_D	C_F	C_A
---	---	---	-------	-------	-------	-------	-------

The C_D bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

C_D	C_D	C_D	
0	X		All Zeros (X = Don't Care)
1	0		AB = Hex 20 (0010 0000)
1	1		All Ones

Enable clear display when = 1 (or by $C_A = 1$)

During the time the Display RAM is being cleared (~160 μ s), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted ($C_F=1$), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A , the Clear All bit, has the combined effect of C_D and C_F ; it uses the C_D clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code:

1	1	1	E	X	X	X	X
---	---	---	---	---	---	---	---

 X = Don't care.

For the sensor matrix modes this command lowers the IRO line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset).

For the N-key rollover mode — if the E bit is programmed to "1" the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

Status Word

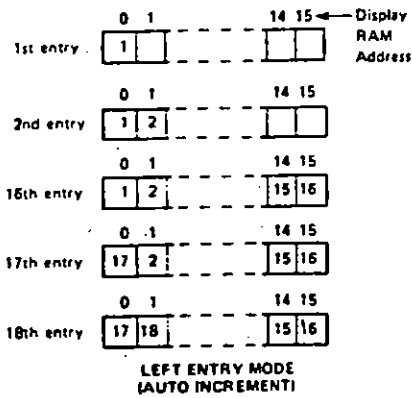
The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A_0 is high and \overline{CS} and \overline{RD} are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A_0 , \overline{CS} and \overline{RD} are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of \overline{RD} will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

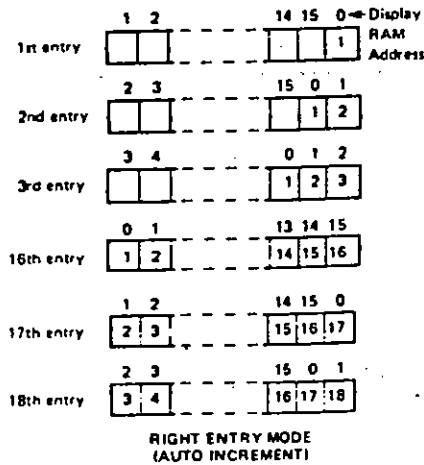
Data Write

Data that is written with A_0 , \overline{CS} and \overline{WR} low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of \overline{WR} occurs if AI set by the latest display command.



Right Entry

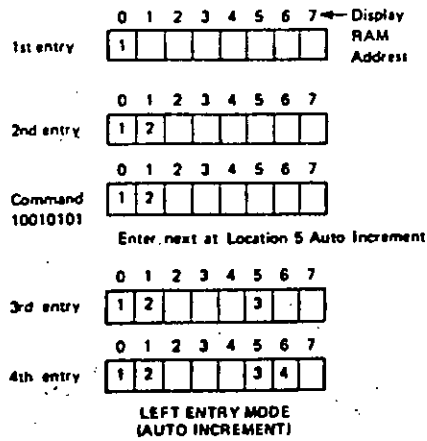
Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.



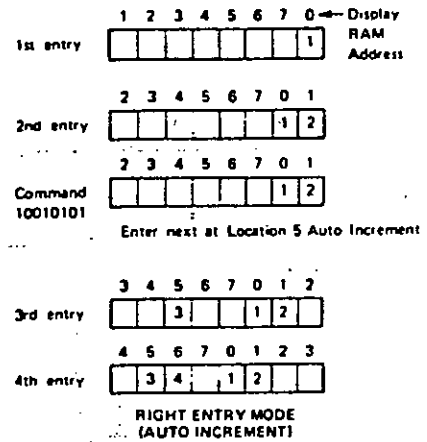
Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 5 with sequential entry is recommended.

Auto Increment

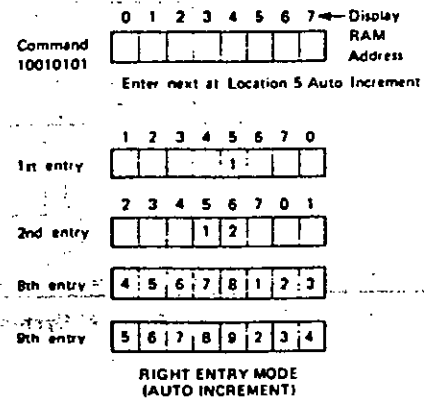
In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted:



Starting at an arbitrary location operates as shown below:



Entry appears to be from the initial entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

G. FIFO Status

FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.

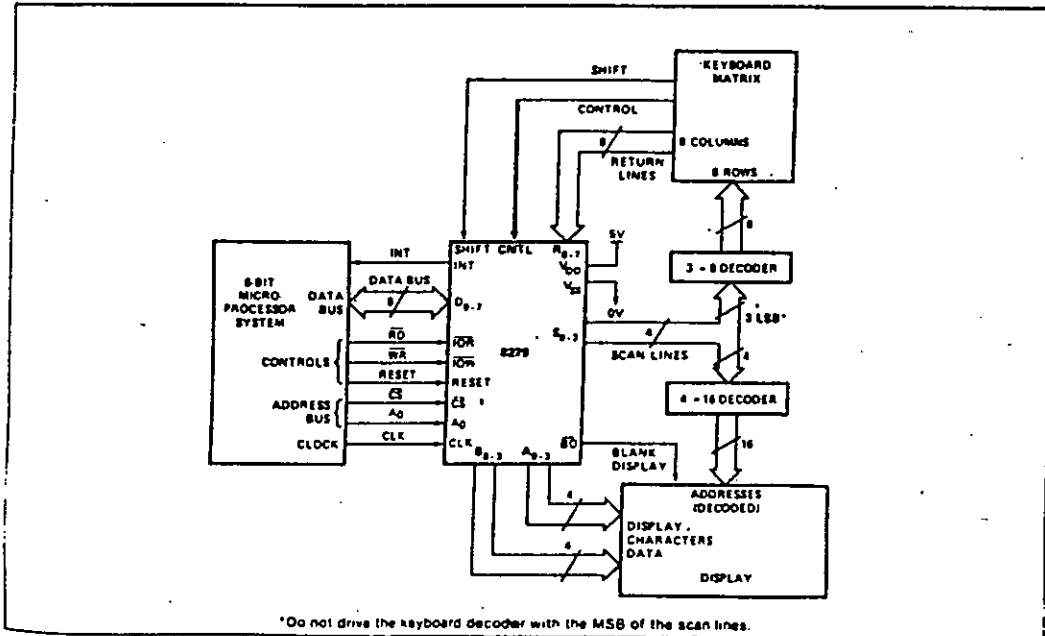
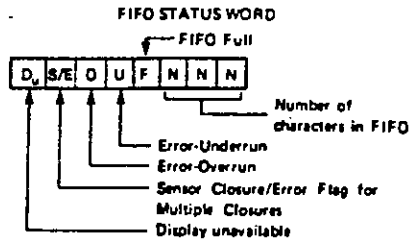


Figure 4. System Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature	0°C to 70°C
Storage Temperature	-65°C to 125°C
Voltage on any Pin with Respect to Ground	-0.5V to +7V
Power Dissipation	1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (NOTE 3)]*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V _{IL1}	Input Low Voltage for Return Lines	-0.5	1.4	V	
V _{IL2}	Input Low Voltage for All Others	-0.5	0.8	V	
V _{IH1}	Input High Voltage for Return Lines	2.2		V	
V _{IH2}	Input High Voltage for All Others	2.0		V	
V _{OL}	Output Low Voltage		0.45	V	Note 1
V _{OH1}	Output High Voltage on Interrupt Line	3.5		V	Note 2
V _{OH2}	Other Outputs	2.4			I _{OH} = -400 μA 8279-5 -100 μA 8279
I _{IL1}	Input Current on Shift, Control and Return Lines		+10 -100	μA	V _{IN} = V _{CC} V _{IN} = 0V
I _{IL2}	Input Leakage Current on All Others		±10	μA	V _{IN} = V _{CC} to 0V
I _{OFL}	Output Float Leakage		±10	μA	V _{OUT} = V _{CC} to 0.45V
I _{CC}	Power Supply Current		120	mA	

CAPACITANCE

Symbol	Parameter	Typ.	Max.	Unit	Test Conditions
C _{IN}	Input Capacitance	5	10	pF	f _C = 1 MHz Unmeasured pins returned to V _{SS}
C _{OUT}	Output Capacitance	10	20	pF	

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (Note 3)]*
Bus Parameters
READ CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t _{AR}	Address Stable Before <u>READ</u>	50		0		ns
t _{RA}	Address Hold Time for <u>READ</u>	5		0		ns
t _{RR}	<u>READ</u> Pulse Width	420		250		ns
t _{RD} ^[4]	Data Delay from <u>READ</u>		300		150	ns
t _{AD} ^[4]	Address to Data Valid		450		250	ns
t _{DF}	<u>READ</u> to Data Floating	10	100	10	100	ns
t _{RCY}	Read Cycle Time	1		1		μs

A.C. CHARACTERISTICS (Continued)
WRITE CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before WRITE	50		0		ns
t_{WA}	Address Hold Time for WRITE	20		0		ns
t_{WW}	WRITE Pulse Width	400		250		ns
t_{DW}	Data Set Up Time for WRITE	300		150		ns
t_{WD}	Data Hold Time for WRITE	40		0		ns
t_{WCY}	Write Cycle Time	1		1		μ s

OTHER TIMINGS

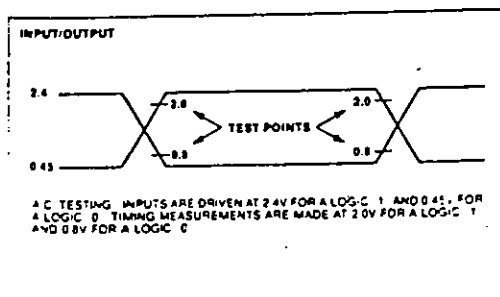
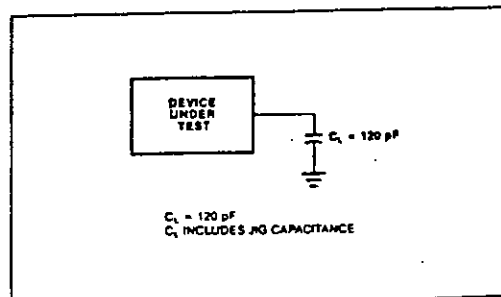
Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{\phi W}$	Clock Pulse Width	230		120		nsec
t_{CY}	Clock Period	500		320		nsec

Keyboard Scan Time 5.1 msec
 Keyboard Debounce Time 10.3 msec
 Key Scan Time 80 μ sec
 Display Scan Time 10.3 msec

Digit-on Time 480 μ sec
 Blanking Time 160 μ sec
 Internal Clock Cycle⁽⁵⁾ 10 μ sec

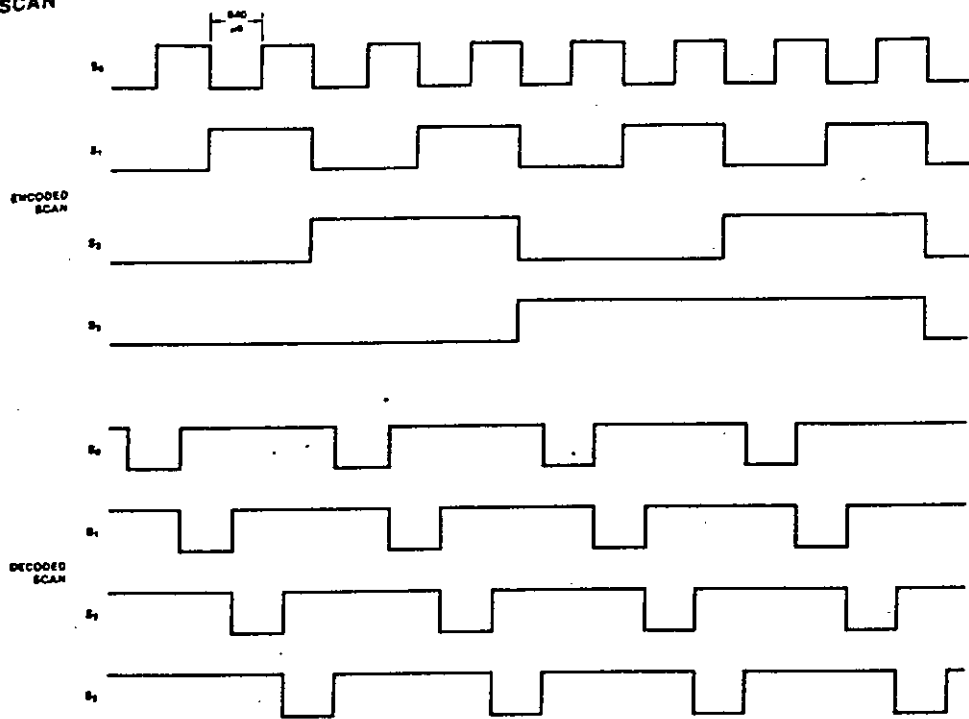
NOTES:

1. 8279, $I_{OL} = 1.6\text{mA}$; 8279-5, $I_{OL} = 2.2\text{mA}$.
2. $I_{OH} = -100\ \mu\text{A}$
3. 8279, $V_{CC} = +5V \pm 5\%$; 8279-5, $V_{CC} = +5V \pm 10\%$.
4. 8279, $C_L = 100\text{pF}$; 8279-5, $C_L = 150\text{pF}$.
5. The Prescaler should be programmed to provide a 10 μ s internal clock cycle.
 * For Extended Temperature EXPRESS, use M8279A electrical parameters.

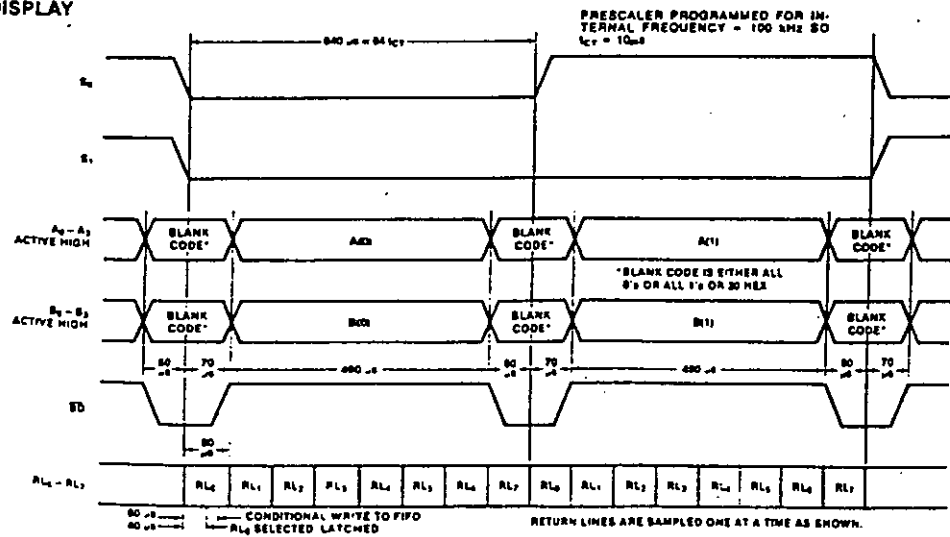
A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT


WAVEFORMS (Continued)

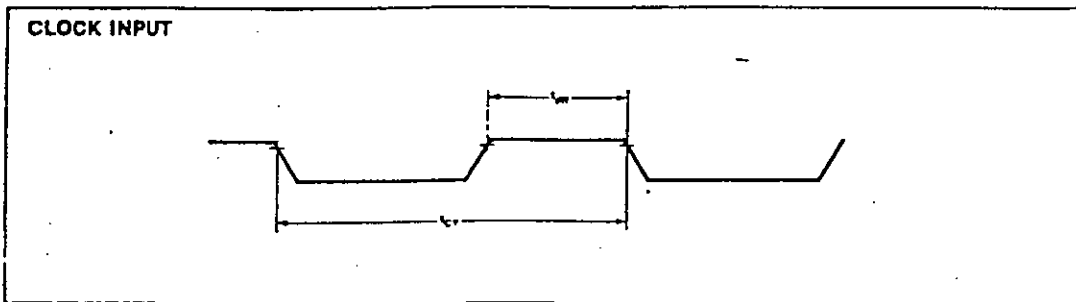
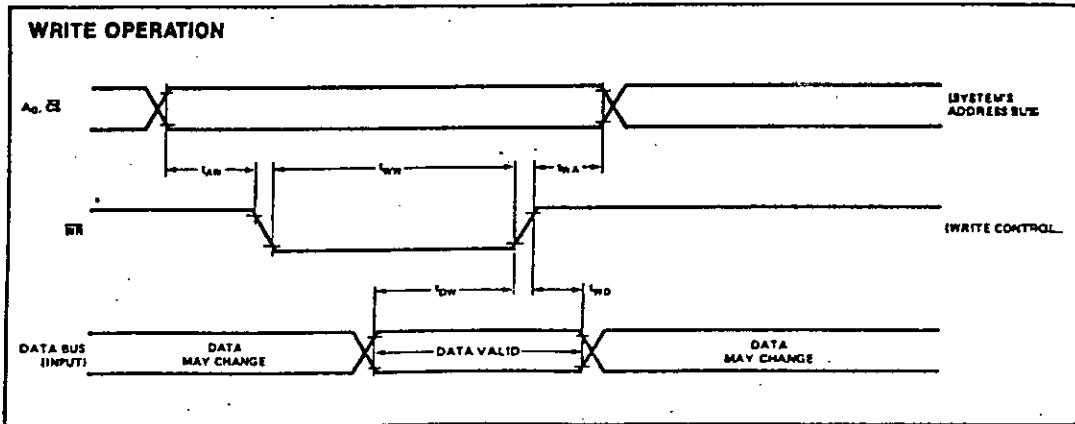
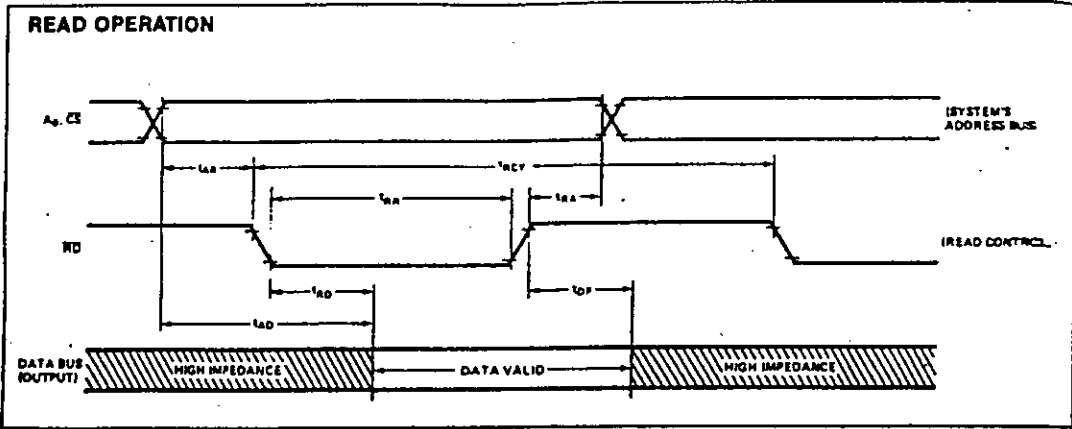
SCAN



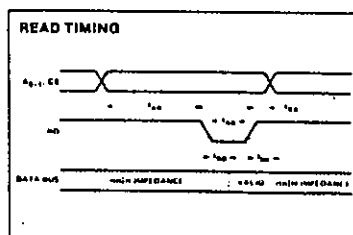
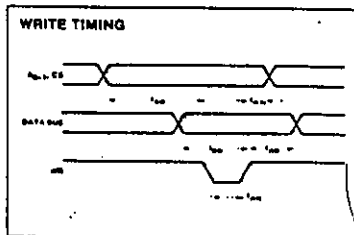
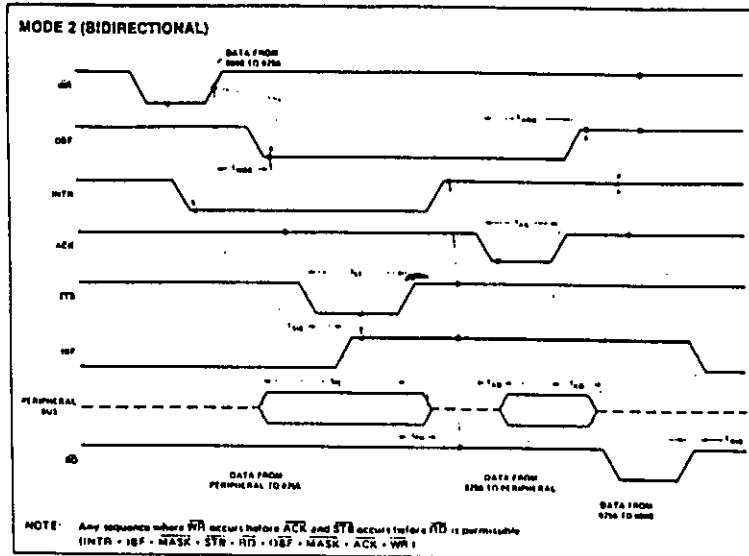
DISPLAY



WAVEFORMS



WAVEFORMS (Continued)



6-378

474-0274-01

8256AH MULTIFUNCTION UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER (MUART)

- Programmable Serial Asynchronous Communications Interface for 5-, 6-, 7-, or 8-Bit Characters, 1, 1½, or 2 Stop Bits, and Parity Generation
- On-Board Baud Rate Generator Programmable for 13 Common Baud Rates up to 19.2K Bits/second, or an External Baud Clock Maximum of 1M Bit/second
- Five 8-Bit Programmable Timer/Counters; Four Can Be Cascaded to Two 18-Bit Timer/Counters
- Two 8-Bit Programmable Parallel I/O Ports; Port 1 Can Be Programmed for Port 2 Handshake Controls and Event Counter Inputs
- Eight-Level Priority Interrupt Controller Programmable for 8085 or IAPX 86, IAPX 88 Systems and for Fully Nested Interrupt Capability
- Programmable System Clock to 1 x, 2 x, 3 x, or 5 x 1.024 MHz

The Intel® 8256AH Multifunction Universal Asynchronous Receiver-Transmitter (MUART) combines the commonly used functions into a single 40-pin device. It is designed to interface to the 8086/88, IAPX 186/188, and 8051 to perform serial communications, parallel I/O, timing, event counting, and priority interrupt functions. All of these functions are fully programmable through nine internal registers. In addition, the five timer/counters and two parallel I/O ports can be accessed directly by the microprocessor.

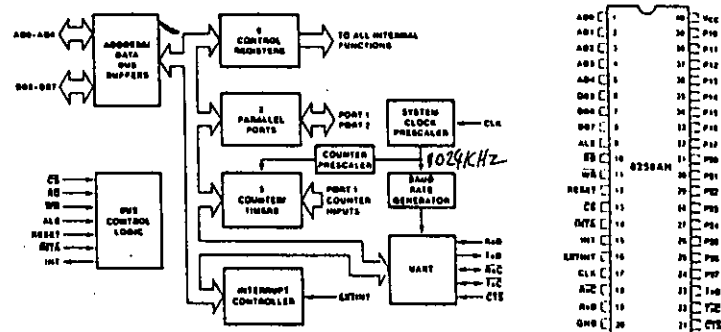


Figure 1. MUART Block Diagram

Figure 2. MUART Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuit Other Than Circuit Embodied in an Intel Product. No Other Circuit Patent, Copyright or Infringement is intended.

© INTEL CORPORATION, 1983

6-379

AUGUST 1983
ORDER NUMBER: 316740-001

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
ADO-AD4 DB5-DB7	1-5 6-8	VO	ADDRESS/DATA: Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, ADO-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while ADO in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	ADDRESS LATCH ENABLE: Latches the 5 address lines on ADO-AD4 and CS on the falling edge.
RD	10	I	READ CONTROL: When this signal is low, the selected register is gated onto the data bus.
WR	11	I	WRITE CONTROL: When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	RESET: An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
CS	13	I	CHIP SELECT: A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and RD and WR have no effect unless CS was latched low during the ALE cycle.
INTA	14	I	INTERRUPT ACKNOWLEDGE: If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	INTERRUPT REQUEST: A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	EXTERNAL INTERRUPT: An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an INTA occurs or the interrupt address register is read.
CLK	17	I	SYSTEM CLOCK: The reference clock for the baud rate generator and the timers.
RxC	18	VO	RECEIVE CLOCK: If the baud rate bits in the Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	RECEIVE DATA: Serial data input.
GND	20	PS	GROUND: Power supply and logic ground reference.

Table 1. Pin Description (continued)

Symbol	Pin	Type	Name and Function
CTS	21	I	CLEAR TO SEND: This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected CTS is level sensitive. As long as CTS is low, any character loaded into the transmitter buffer register will be transmitter serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-0FH is selected, CTS must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If CTS is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on CTS occurs. If 0.75 stop bits is chosen, the CTS input is edge sensitive. A negative edge on CTS results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on CTS. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	VO	TRANSMIT CLOCK: If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1 1/2 or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	TRANSMIT DATA: Serial data output.
P27-P20	24-31	VO	PARALLEL I/O PORT 2: Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	VO	PARALLEL I/O PORT 1: Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
V _{cc}	40	PS	POWER: +5V power supply.

FUNCTIONAL DESCRIPTION

The 8256AH Multi-Function Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. The MUART performs asynchronous serial communications, parallel I/O, timing, event counting, and interrupt control. For detailed application information, see Intel Ap Note #153, Designing with the 8256.

Serial Communications

The serial communications portion of the MUART contains a full-duplex asynchronous receiver-transmitter (UART). A programmable baud rate generator is included on the MUART to permit a variety of operating speeds without external components. The UART can be programmed by the CPU for a variety of character sizes, parity generation and detection, error detection, and start/stop bit handling. The receiver checks the start and stop bits in the center of the bit, and a break hails the reception of data. The transmitter can send breaks and can be controlled by an external enable pin.

Parallel I/O

The MUART includes 16 bits of general purpose parallel I/O. Eight bits (Port 1) can be individually changed from input to output or used for special I/O functions. The other eight bits (Port 2) can be used as nibbles (4 bits) or as bytes. These eight bits also include a handshaking capability using two pins on Port 1.

Counter/Timers

There are five 8-bit counter/timers on the MUART. The timers can be programmed to use either a 1 kHz or 16 kHz clock generated from the system clock. Four of the 8-bit counter/timers can be cascaded to two 16-bit counter/timers, and one of the 8-bit counter/timers can be reset to its initial value by an external signal.

Interrupts

An eight-level priority interrupt controller can be configured for fully nested or normal interrupt priority. Seven of the eight interrupts service functions on the MUART (counter/timers, UART), and one external interrupt is provided which can be used for a particular function or for chaining interrupt controllers or more MUARTs. The MUART will support 8085 and 8086/88 systems with direct interrupt vectoring, or the MUART can be polled to determine the cause of the interrupt. If additional interrupt control capability is needed, the MUART's interrupt controller can be cascaded into

another MUART, into an Intel 8259A Programmable Interrupt Controller, or into the interrupt controller of the IAPX 186/188 High-Integration Microprocessor.

INITIALIZATION

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Byte 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

```
Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set interrupts
```

The modification register may be loaded if required for special applications; normally this operation is not necessary. The MUART should be reset before initialization. (Either a hardware or a software reset will do.)

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 80186 microprocessor. Figure 3 displays the block diagram for this interface. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines; for 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte,

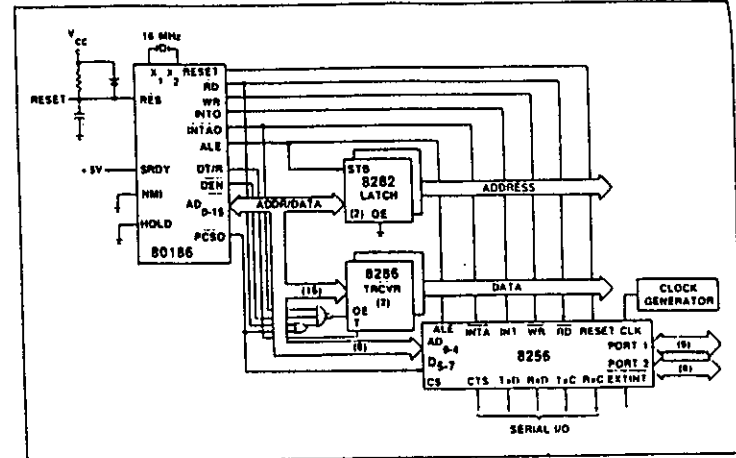


Figure 3. 80186/8256 Interface

the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 2 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
				(0R)	(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16 kHz (62.5µs). If FRQ = 1, the timer input frequency is 1 kHz (1 ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first INTA. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to INTA is for 8086 interrupts where the first INTA is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second INTA.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

Table 2. UART Registers

Read Registers										Write Registers										
8085 Mode:					AD3 AD2 AD1 AD0					8086 Mode:					AD4 AD3 AD2 AD1					
L1	L0	S1	S0	BRK1	BIT1	5066	FRQ	0	0	0	0	L1	L0	S1	S0	BRK1	BIT1	5066	FRQ	
Command 1																				
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2																				
0	RAE	IAE	NIE	0	5BRK1	IBRK1	0	0	0	1	0	SET	RAE	IAE	NIE	END	5BRK1	IBRK1	RS1	
Command 3																				
135	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	135	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	
Mode																				
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10	
Port 1 Control																				
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0	
Interrupt Enable																				
07	06	05	04	03	02	01	00	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0	
Interrupt Address																				
07	06	05	04	03	02	01	00	0	1	1	0	07	06	05	04	03	02	01	00	
Receiver Buffer																				
07	06	05	04	03	02	01	00	0	1	0	0	07	06	05	04	03	02	01	00	
Transmitter Buffer																				
07	06	05	04	03	02	01	00	0	1	0	0	07	06	05	04	03	02	01	00	
Port 1																				
07	06	05	04	03	02	01	00	1	0	0	1	07	06	05	04	03	02	01	00	
Port 2																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	
Timer 1																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	
Timer 2																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	
Timer 3																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	
Timer 4																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	
Timer 5																				
07	06	05	04	03	02	01	00	1	0	1	0	07	06	05	04	03	02	01	00	

BRK1 — Break-In Detect Enable

If this bit equals 0, Port 1 P18 is a general purpose I/O port. When BRK1 equals 1, the Break-In Detect feature is enabled on Port 1 P18. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P18 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input CTS is discussed in the Pin Description section under "CTS".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(11H)							

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

vide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalars are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1,024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	TxC, RxC	1
0	0	0	1	TxC/64	64
0	0	1	0	TxC/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input: 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise, high level.
3, 0, F	Output: baud rate divider ratio	Output: baud rate divider ratio

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIW	END	SBRK	TBRK	RST
(2R)				(2W)			

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change oc-

curs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1. All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2. The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
3. The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4. If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does not alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of CTS is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit reset the currently served interrupt level in the Interrupt Service Register. This command must occur at the end of each interrupt service routine during fully nested interrupt mode. END is automatically cleared when the Interrupt Service Register (Internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to INTA. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

TJ5	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)				(3W)			

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value > 3H.

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction	
				Upper	Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake		Input
1	0	1	Byte Handshake		Output
1	1	0	DO NOT USE		
1	1	1	Test		

NOTE: If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 38). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

185

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T3C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value. But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 3.

NOTE:

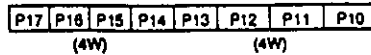
Interrupt levels assigned to single counters are partly not occupied if event counter/timers are cascaded. Level 2 will be vacated if event counter/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counter/timers 3 and 5 are cascaded.

Single event counter/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 001H to 000H.

Table 3. Event Counters/Timers Mode of Operation

Event Counter/Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	—	Internal clock
2	8-bit timer	T24=0, CT2=0	Internal clock
	8-bit event counter	T24=0, CT2=1	P12 pin 37
2	8-bit timer	T35=0, CT3=0	Internal clock
	8-bit event counter	T35=0, CT3=1	P13 pin 38
4	8-bit timer	T24=0	Internal clock
5	8-bit timer, normal mode	T35=0, T5C=0	Internal clock
	8-bit timer, retriggeable mode	T35=0, T5C=1	Internal clock
2 and 4 cascaded	16-bit timer	T24=1, CT2=0	Internal clock
	16-bit event counter	T24=1, CT2=1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35=1, T5C=0, CT3=0	Internal clock
	16-bit event counter, normal mode	T35=1, T5C=0, CT3=1	P13 pin 36
	16-bit timer, retriggeable mode	T35=1, T5C=1, CT3=0	Internal clock
	16-bit event counter, retriggeable mode	T35=1, T5C=1, CT3=1	P13 pin 36

Port 1 Control Register



Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as STB/ACK handshake control input, and Port 11 is programmed as IBF/OBF handshake control output.

If byte handshake mode is enabled for output on Port 2 OBF indicates that a character has been loaded

into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving ACK low. OBF is set low by writing to Port 2 and is reset by ACK.

If byte handshake mode is enabled for input on Port 2, STB is an input. IBF is driven low after STB goes low. On the rising edge of STB the data from Port 2 is latched.

IBF is reset high when Port 2 is read.

Port 12, 13 — Counter, 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggeable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

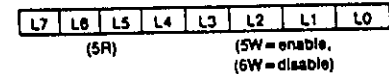
Port 16 — Break-in Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request. Priority Level 7 is assigned to this interrupt.

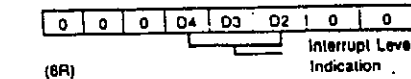
Interrupt Enable Register



Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt enable Register (5R).

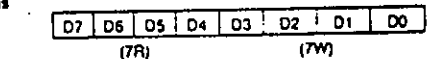
Priority	Source
Highest	L0 Timer 1
	L1 Timer 2 or Port Interrupt
	L2 External interrupt (EXTINT)
	L3 Timer 3 or Timers 3 & 5
	L4 Receiver Interrupt
	L5 Transmitter Interrupt
	L6 Timer 4 or Timers 2 & 4
Lowest	L7 Timer 5 or Port 2 Handshaking

Interrupt Address Register



Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge INTA; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer



Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are double buffered.

disruptive

100

NOTE:

The modification register cannot be read. Reading from address 0FH, 8086 I/O gates the contents of the status register onto the data bus.

A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:

- The start bit check is enabled.
- Status Register Bit 0 (FE) indicates framing error.
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1. Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.
2. Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.
3. The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT IS INACTIVE (LOW).
4. The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.
5. The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.
6. Status Register Bit 0 implies framing error.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length
0	1	1	1	1	1 (Start of Bit)
0	1	1	1	0	2
0	1	1	0	1	3
0	1	1	0	0	4
0	1	0	1	1	5
0	1	0	1	0	6
0	1	0	0	1	7
0	1	0	0	0	8
0	0	1	1	1	9
0	0	1	1	0	10
0	0	1	0	1	11
0	0	1	0	0	12
0	0	0	1	1	13
0	0	0	1	0	14
0	0	0	0	1	15
0	0	0	0	0	16 (Bit center)
1	1	1	1	1	17
1	1	1	1	0	18
1	1	1	0	1	19
1	1	1	0	0	20
1	1	0	1	1	21
1	1	0	1	0	22
1	1	0	0	1	23
1	1	0	0	0	24
1	0	1	1	1	25
1	0	1	1	0	26
1	0	1	0	1	27
1	0	1	0	0	28
1	0	0	1	1	29
1	0	0	1	0	30
1	0	0	0	1	31

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
 Storage Temperature -65°C to -150°C
 Voltage On Any Pin With Respect to ground -0.5V to -7V
 Power Dissipation 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V _{IL}	Input Low Voltage	-0.5	0.8	V	
V _{IH}	Input High Voltage	2.0	V _{CC} + 0.5	V	
V _{OL}	Output Low Voltage		0.45	V	I _{OL} = 2.5 mA
V _{OH}	Output High Voltage	2.4		V	I _{OH} = -400 μA
I _L	Input Leakage		10 -10	μA	V _{IN} = V _{CC} V _{IN} = 0V
I _O	Output Leakage		10 -10	μA	V _{OUT} = V _{CC} V _{OUT} = 0.45V
I _{CC}	V _{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C _{IN}	Input Capacitance		10	pF	f _c = 1 MHz
C _{IO}	I/O Capacitance		20	pF	Unmeasured pins returned to V _{SS}

Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming CTS is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
(BR)				(SW)			

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
(BR)				(SW)			

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
(OA ₁₋₅ , OE ₁₋₅ , R)				(OA ₁₋₅ , OE ₁₋₅ , W)			

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while RD is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X

leads to a count of $X * 256 + 255$. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BO	PE	OE	FE
(OF, R)							

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the first stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section of AP-153 under "Receive Break Detect" and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If CTS is low, the Transmitter Register will be loaded during the transmission of the start bit. If CTS is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until CTS goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by INTA or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC
(OF, W)							

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RS_n alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is not clocked by RxC.

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIW	END	SBRK	TBRK	RST
(2R)					(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change oc-

curs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

- All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
- The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
- The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
- If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does not alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of CTS is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit reset the currently served interrupt level in the Interrupt Service Register. This command must occur at the end of each interrupt service routine during fully nested interrupt mode. END is automatically cleared when the Interrupt Service Register (Internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to INTA. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the Status Register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)				(3W)			

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value > 3H.

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction	
				Upper	Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake		Input
1	0	1	Byte Handshake		Output
1	1	0	DO NOT USE		
1	1	1	Test		

NOTE:

If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

A.C. CHARACTERISTICS

 $(T_A = 0^\circ\text{C to } 70^\circ\text{C}, V_{CC} = +5.0\text{V} \pm 10\%, \text{GND} = 0\text{V})$

BUS PARAMETERS

Symbol	Parameter	8256AH		Units
		Min.	Max.	
tLL	ALE Pulse Width	70		ns
tCSL	$\overline{\text{CS}}$ to ALE Setup Time	0		ns
tAL	Address to ALE Setup Time	20		ns
tLA	Address Hold Time After ALE	30		ns
tLC	ALE to RD/WR	20		ns
tCC	RD, WR, INTA Pulse Width	200		ns
tRD	Data Valid from RD (1)		150	ns
tDF	Data Float After RD (2)		70	ns
tDW	Data Valid to WR	200		ns
tWD	Data Valid After WR	50		ns
tCL	RD/WR Control to Latch Enable	25		ns
tLDR	ALE to Data Valid		180	ns
tRST	Reset Pulse Width	500		ns
tRV	Recovery Time Between RD/WR	500		ns

TIMER/COUNTER PARAMETERS

tCPI	Counter Input Cycle Time (P12, P13)	2.2		μs
ICPWH	Counter Input Pulse Width High	1.1		μs
ICPWL	Counter Input Pulse Width Low	1.1		μs
ITPI	Counter Input to INT1 at Terminal Count		2.5	μs
tTIH	LOAD Pulse High Time Counter 5	1.1		μs
tTIL	LOAD Pulse Low Time Counter 5	1.1		μs
tPP	Counter 5 Load Before Next Clock Pulse on P13	1.1		μs
tCR	External Count Clock to RD1 to Ensure Clock is Reflected in Count	2.2		μs
tRC	RD1 to External Count Clock to Ensure Clock is not Reflected in Count	0		ns
tCW	External Count Clock to WR1 to Ensure Count Written is Not Decrementd	2.2		μs
tWC	WR1 to External Count Clock to Ensure Count Written is Decrementd	0		ns

INTERRUPT PARAMETERS

IDEX	EXTINT1 to INT1		200	ns
IDPI	Interrupt request on P171 to INT1		21CY + 500	ns
tPI	Pulse Width of Interrupt Request on P17	1CY + 100		ns
IHEA	INTA1 or RD1 to EXTINT1	30		ns
IHA	INTA1 or RD1 to INT1		300	ns

A.C. CHARACTERISTICS (continued)

SERIAL INTERFACE AND CLOCK PARAMETERS

Symbol	Parameter	8256AH		Units
		Min.	Max.	
tCY	Clock Period	195	10,000	ns
tCLKH	Clock High Pulse Width	65		ns
tCLKL	Clock Low Pulse Width	65		ns
tR	Clock Rise Time		30	μs
tF	Clock Fall Time		30	ns
tSCY	Serial Clock Period (4)	975		ns
tSPD	Serial Clock High (4)	350		ns
tSPW	Serial Clock Low (4)	350		ns
tSTD	Internal Status Update Delay From Center of Stop Bit (5)		300	ns
tDTX	TxD to TxD Data Valid		300	ns
tIRBF	INT Delay From Center of First Stop Bit		21CY + 500	ns
tITBE	INT Delay From Falling Edge of Transmit Clock at end of Start Bit		21CY + 500	ns
tCTS	Pulse Width for Single Character Transmission (6)			

PARALLEL I/O PORT PARAMETERS

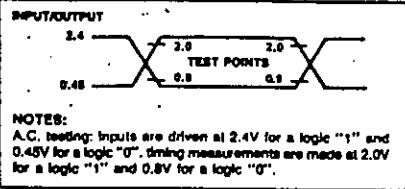
tWP	WR1 to P1/P2 Data Valid		0	ns
tPR	P1/P2 Data Stable Before RD1 (7)	300		ns
tRP	P1/P2 Data Hold Time	50		ns
tAK	ACK Pulse Width	150		ns
tST	Strobe Pulse Width	tSIB		ns
tPS	Data Setup to STB1	50		ns
tPH	Data Hold After STB1	50		ns
tWOB	WR1 to OBF1		250	ns
tAOS	ACK1 to OBF1		250	ns
tSIB	STB1 to IBF1		250	ns
tRI	RD1 to IBF1		250	ns
tSIT	STB1 to INT1		21CY + 500	ns
tAIT	ACK1 to INT1		21CY + 500	ns
tAED	OBF1 to ACK1 Delay	0		ns

NOTES:

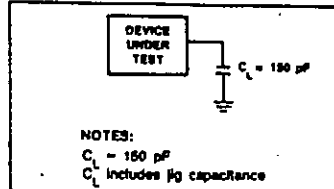
- $C_L = \text{pF}$ all outputs.
- Measured from logic "one" or "zero" to 1.5V at $C_L = 150 \text{ pF}$.
- P12, P13 are external clock inputs.
- Note that RxC may be used as an input only in 1X mode, otherwise it will be an output.
- The center of the Stop Bit will be the receiver sample time, as programmed by the modification register.
- 1/16th bit length for 32X, 64X; 100 ns for 1X.
- To ensure t_{acc} spec is met.

WAVEFORMS

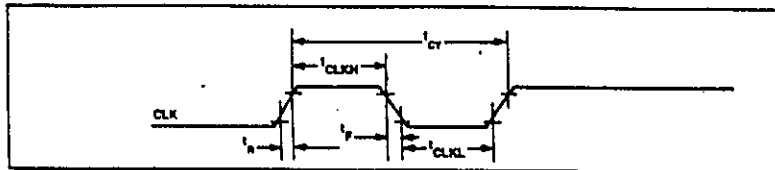
A.C. TESTING INPUT, OUTPUT WAVEFORM



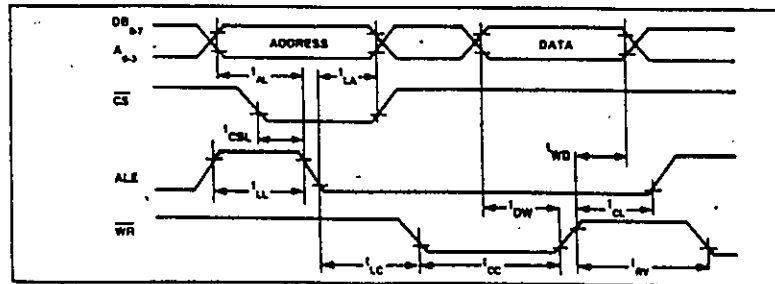
A.C. TESTING LOAD CIRCUIT



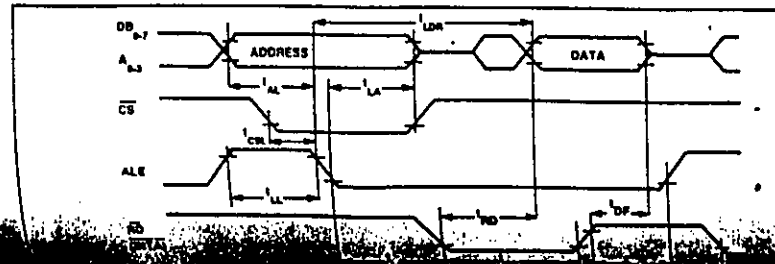
SYSTEM CLOCK



WRITE CYCLE

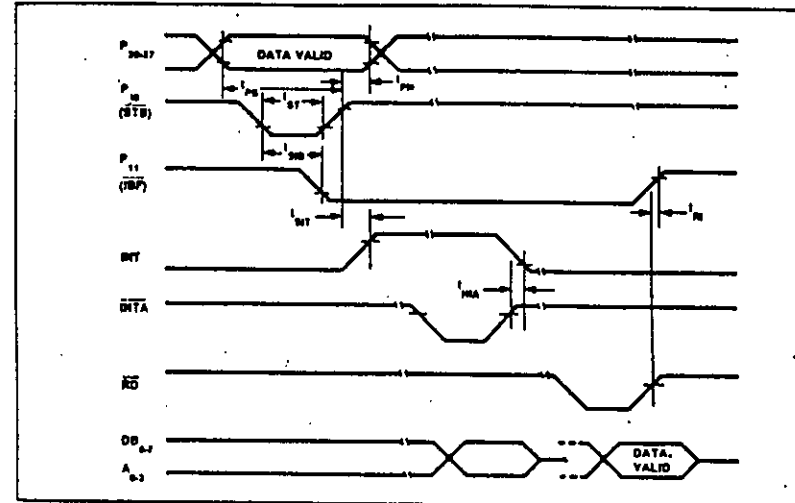


READ CYCLE

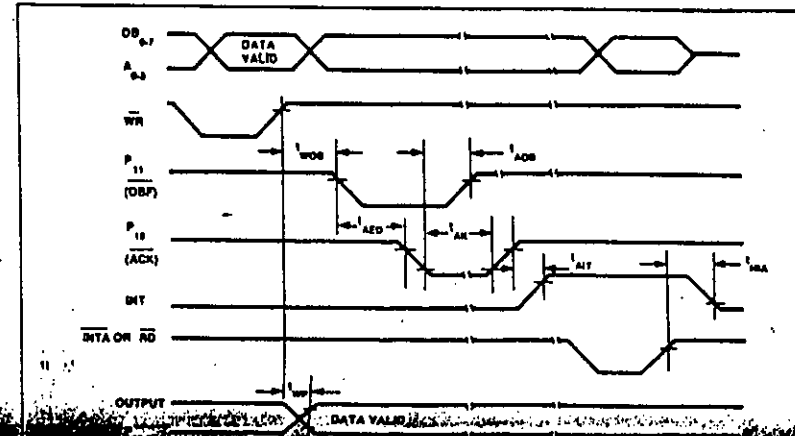


WAVEFORMS (Continued)

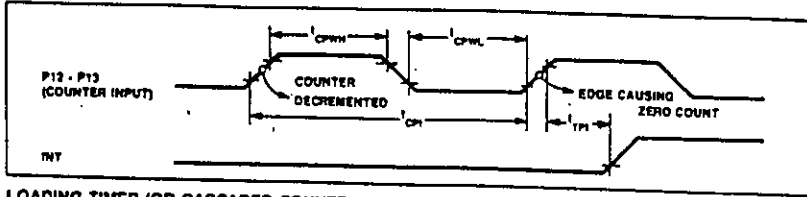
PARALLEL PORT HANDSHAKING - INPUT MODE



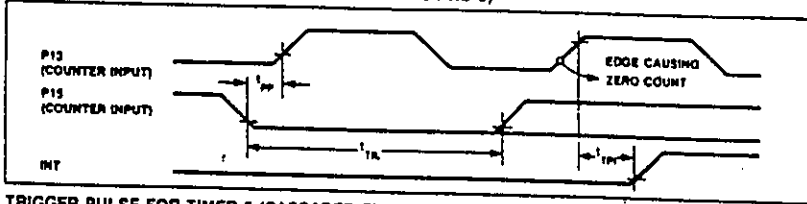
PARALLEL PORT HANDSHAKING - OUTPUT MODE



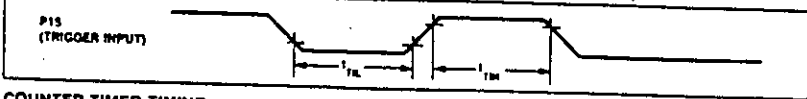
COUNT PULSE TIMINGS



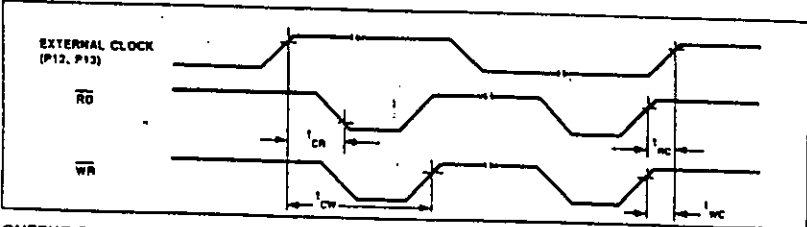
LOADING TIMER (OR CASCADED COUNTER/TIMER 3 AND 5)



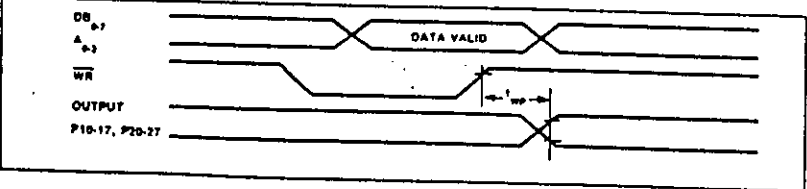
TRIGGER PULSE FOR TIMER 5 (CASCADED EVENT COUNTER/TIMER 3 AND 5)



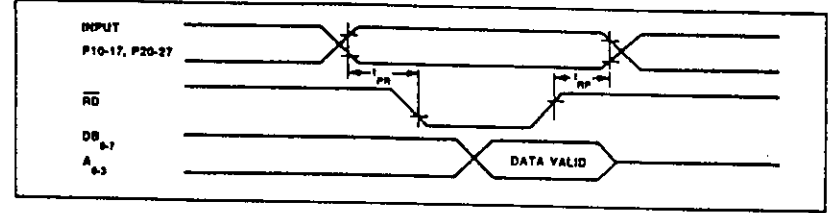
COUNTER TIMER TIMING



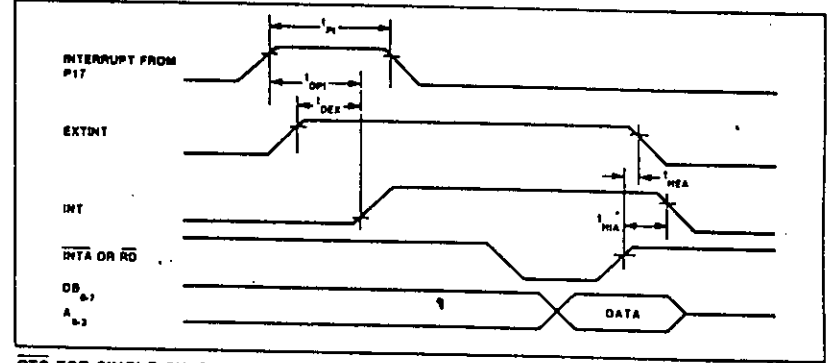
OUTPUT FROM PORT 1 AND PORT 2



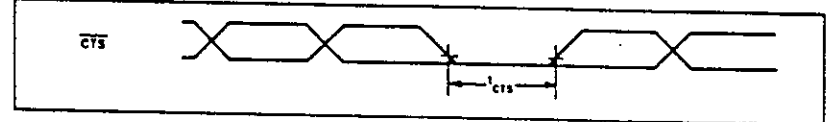
INPUT FROM PORT 1 AND PORT 2



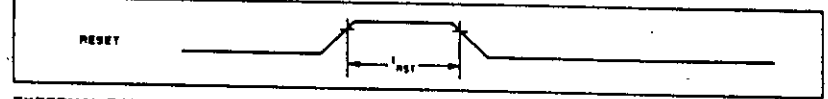
INTERRUPT TIMING



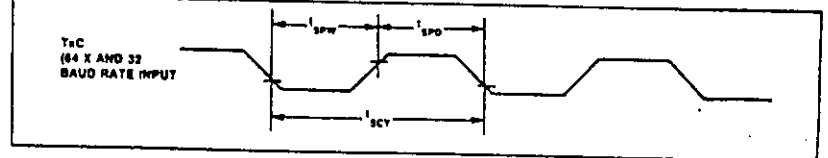
CTS FOR SINGLE CHARACTER TRANSMISSION



RESET TIMING

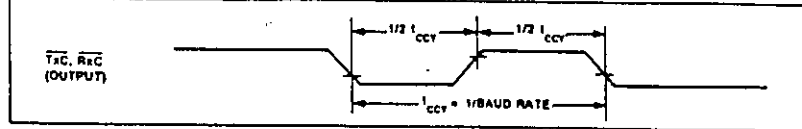


EXTERNAL BAUD RATE CLOCK FOR SERIAL INTERFACE

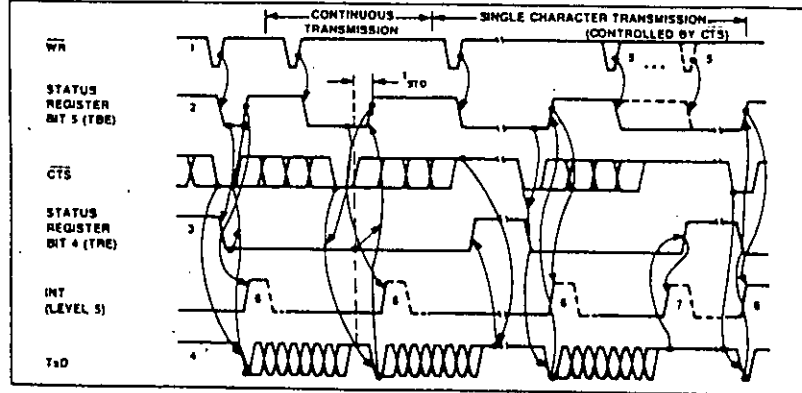


192

TRANSMITTER AND RECEIVER CLOCK FROM INTERNAL CLOCK SOURCE



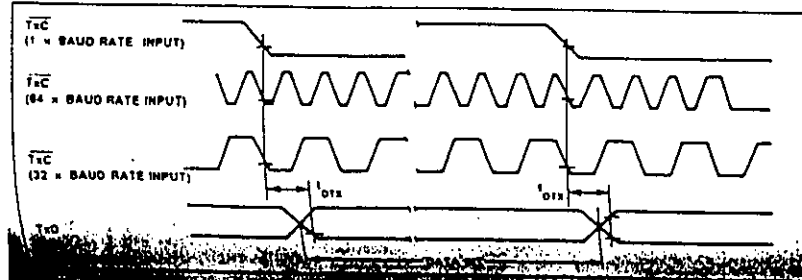
TRANSMISSION OF CHARACTERS ON SERIAL INTERFACE



- NOTES:
1. Load transmitter buffer register.
 2. Transmitter buffer register is empty.
 3. Transmitter register is empty.
 4. Character format for this example: 7 Data Bits with Parity Bit and 2 Stop Bits.
 5. Loading of transmitter buffer register must be complete before CTS goes low.
 6. Interrupt due to transmitter buffer register empty.
 7. Interrupt due to transmitter register empty.

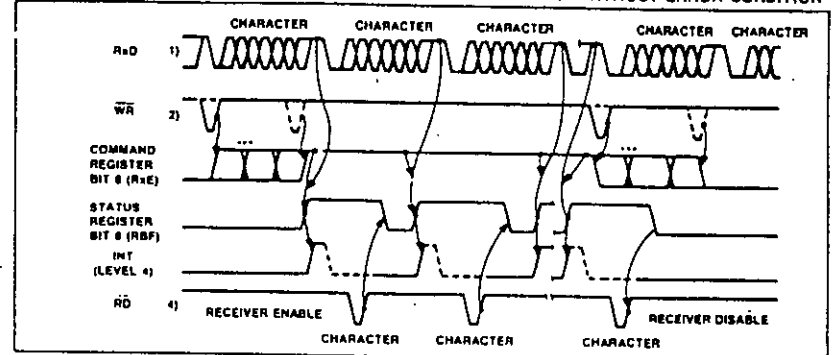
No Status bits are altered when \overline{RD} is active.

DATA BIT OUTPUT ON SERIAL INTERFACE



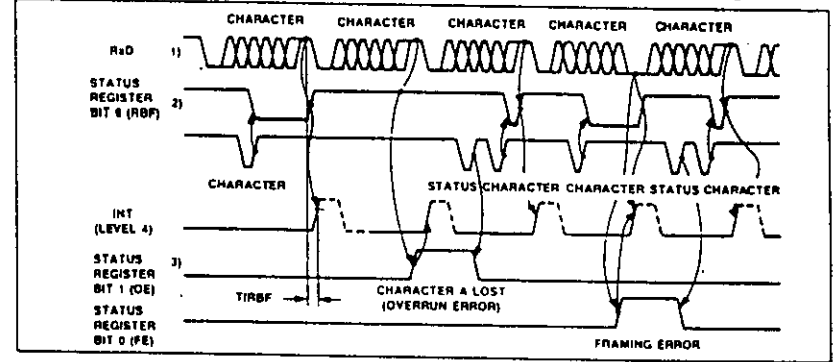
193

CONTINUOUS RECEPTION OF CHARACTERS ON SERIAL INTERFACE WITHOUT ERROR CONDITION



- NOTES:
1. Character format for this example: 6 data bits with parity bit and one stop bit.
 2. Set or reset bit 6 of command register 3 (enable receiver).
 3. Receiver buffer located.
 4. Read receiver buffer register.

ERROR CONDITIONS DURING RECEPTION OF CHARACTERS ON THE SERIAL INTERFACE



- NOTES:
1. Character format for this example: 6 data bits without parity and one stop bit.
 2. Receiver buffer register loaded.
 3. Overrun error.
 4. Framing error.
 5. Interrupt from receiver buffer register loading.
 6. Interrupt from overrun error.
 7. Interrupt from framing error and loading receiver buffer register.

APPENDIX - D

HADWARE DEVELOPMENT TECHNOLOGY

D.1 Introduction to Procedure/Methodology

Brief Description on the Procedure / Methodology:

1. Minimum system with the following components were built in a grid-board by wire-wrap. Page-195, 196

8086 (CPU)	8284 (Clock Chip)
2x27256 (Two EPROMs)	3x74LS373 (Three Latch)
8279 (To Control Display & Keyboard)	1-CC 7-Seg Display
2716 (EPROM as Decoder)	74LS154 (To Drive Display)

2. Just minimum amount of Boot Codes will be written into 2x27256 EPROMs so that the character A appears in the 7-segment display.
3. A key will be connected to the system via the 8279 chip and an additional 74LS138 chip. (74LS138 is a 3-to-8 line decoder).
4. Codes will be added to the 27256 EPROMs, so that now the character A appears on the 7-segment display when the key is pressed.
5. Now, the core hardware is ready to accept more functional ICs and software.
6. Display will be expanded to 16-digit (still by wire-wrapping).
7. Keyboard will be expanded to 18-keys. 2x62256 RAMs will be added.
8. The Monitor program's Flow Chart (minimum) will be developed.
9. The flow chart will now be slowly and carefully coded.
10. The binary codes will be entered into the 27256 EPROMs, tested and debugged.
11. The flow chart will slowly be expanded as much as possible and will be coded and written to EPROM.
12. Using SmartWork software, the PCB artwork of for the schematic will be developed.
13. The results, the difficulties and the ways difficulties were solved, will be documented.
14. Finally, the product will be physically made. The documents will be organized as per rules of Thesis preparation.

Components Organization:

Because the 8086 system is an educational trainer, its components are visible. Therefore, the arrangement of the various ICs have to be well placed relative each other so that the board looks pretty without sacrificing the optimum routes of their interconnections using copper tracks.

All the components including keys, connectors and etc. are gathered. A holed grid board is arranged. The components are placed on the grid board. Various positions are tried until a good looking and well organized lay out is found. The obtained lay out is shown to non-electronic people to assess the organizational beauty of the board. Refer at page-187 for the component lay out that we have chosen for the 8086 trainer. It is a good practice to keep a model of this one if possible. For our case, we don't have except the engineering prototype.

D.2 Wire Wrapping Accessories:

Wire Wrap Socket:

Allows developing and testing circuits based on pure thought. If the circuit does not work due to faulty design or faulty connections, then it is very easy and quick to modify the wiring just by unwrapping the connection.

Manual Wrap Tool:

Type P160-2B tool will make wrapped wire connections manually with No. 26 through 30 gauge wire.

Manually Unwrap Tool:

Type P160-1B slips over terminal and is turned to unwrap 26 to 30 gauge wires wrapped either right or left on 0.025" or 0.028" square posts.

Powered Wrap Tool

Type P148-7-30 tool comes complete with external power supply. The tool makes standard wrap joints for solderless wrapped terminations. With an 03 bit this power tool wraps No. 26 through 30 gauge wires.

Wire Wrap Wire:

P160-6A package is a tension regulator wire spool bracket. Usually comes with 200 feet of spool of No. 30 gauge silver plated wire with Tefzel insulation.

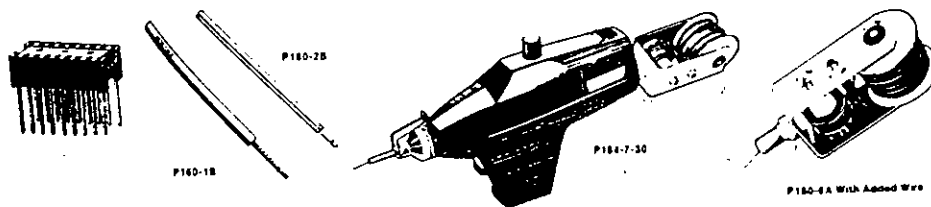


Fig - D.2 : Wire Wrap Tools and Accessories

Local Vendor:

Sabium Electronics
34, Stadium Swimming Pool
Dhaka Stadium, Dhaka - 1000
BNAGLADESH

Foreign Vendor:

Network Electronics
4801 N. RAVENSWOOD AVE.
CHICAGO, IL. 60640 - 4496
U.S.A

D.3 PCB Assembly Manufacturing

The industry standard procedures for making a quality graded PCB (Printed Circuit Board) assembly are :-

01. Use of a flexible software for developing the PCB artwork. Flexible means that the software will allow easy changing of the width of the PCB track and the inter/external diameters of the pad as per the diameter of the component leads. For example, the BM (Board Maker) software.
02. Use of plotter to obtain industry standard good quality 2X (Two Times) print of the PCB artwork.
03. Good camera works facility on the premises to obtain real size negative-positive films of the PCB artwork.
04. Solder Mask including PTH (printed through hole) facilities.
05. Industry standard Screen Print facility or Photo Resist facility to transfer the PCB artwork impression on the copper clad board.
06. Industry standard temperature controlled itching plant.
07. Industry standard CNC (Computer based Numeric Controlled) drilling facilities.
08. Finally, wave soldering facilities for high quality soldering.

While building the 8086 trainer, we had none of the facilities stated above prevailing in the market. But, yet the product has been realized and available for learning purposes. This section will discuss how the job gets done inspite of lacking of the required services.

PCB Artwork Making :

Software Used : SmartWork Software (Old version). Capable for putting only two sizes copper tracks and two sizes pad.
No of Layers : Solder side Layer page - 188
: Component side Layer page - 189

PCB Artwork Print:

PCB Artwork Size : 9.45" x 7.65"
Printer Used : EPSON LX - 800 8" and 9-pin Dot Printer
Print Difficulties : The software produces 2X artwork PCB. The printer is only 8" span. Therefore, the print of each layer had two pieces and glued together. This has created mechanical misalignment while adjusting the solder side and component side layers.

PTH Solution:

In Bangladesh, there is no commercial use of double layer PCBs. Therefore, the relevant infra structure did not get developed. The design of the 8086 trainer could not be realized using single layer PCB. We were well aware about the non-existence of PTH facility. The PTH has been done manually in the following ways:

Refer to PCB artworks at page- 187, 188, 189.

Pin - 10 of U2 (page-197) will have copper tracks at both layers. So, the pin-10 should have a PTH (Printed Through Hole). Since, we could not place a PTH there, still we managed to get copper tracks from pin-10 at both sides of the PCB in the following way:-

A hole is drilled near pin-10. Copper track has been made at solder side from pin-10 upto the new hole. At the component side, copper track has also been made from the new hole upto the target. A wire is inserted in the hole and it is soldered at both sides with the pads.

PCB Manufacturer:

Design Group Limited, 65-66, Khilgaon Taltola uper market, 1st Floor.Dhaka, Phone: 416178, 415772

D. 4 Soldering techniques

Manual Soldering:

The IC socket pins and others are soldered manually. Manual solder causes no problem if done following the prescribed procedures. Otherwise, there would remain many cold solders leading to intermittent problem throughout the life of the product. There are about 2700 points in the 8086 trainer PCB which are to be soldered by hand. Fortunately, the author possesses 13 years of soldering experience along with professional training on soldering, to finish the soldering job of the trainer. Given below, the standard procedures of Hand Soldering which needs to be exercised on regular basis till the soldering proficiency is achieved by an worker.

Type of Soldering :

Temperature of the Soldering Iron Bit:

Surface Soldering means Romm Temperature soldering: Should be around 600°F

Procedures:

01. Place the soldering iron tip at 45° against both the lead and the circuit board foil as shown in the picture to the right. Let both be heated for two to three seconds.
02. Now apply solder to the other side of the connection as shown in the picture to the right. Allow the hetaed lead and the circuit board foil to melt the solder.
03. When the solder begins to melt, let it to flow around the connection. And then remove the solder and the iron. Allow the the connectin to cool down. Figure to the right.
04. Trim the lead lengths close to the connection. Clip the leads to prevent them from flying toward eyes.

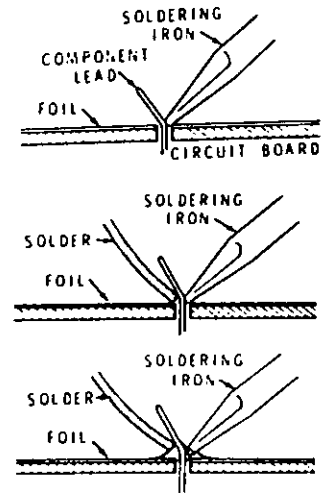


Fig-D.4(a) : Soldering Techniques

A Good Solder Connection:

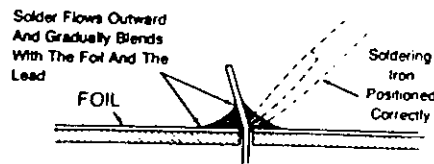
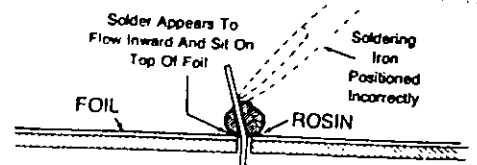
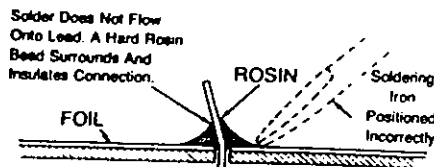
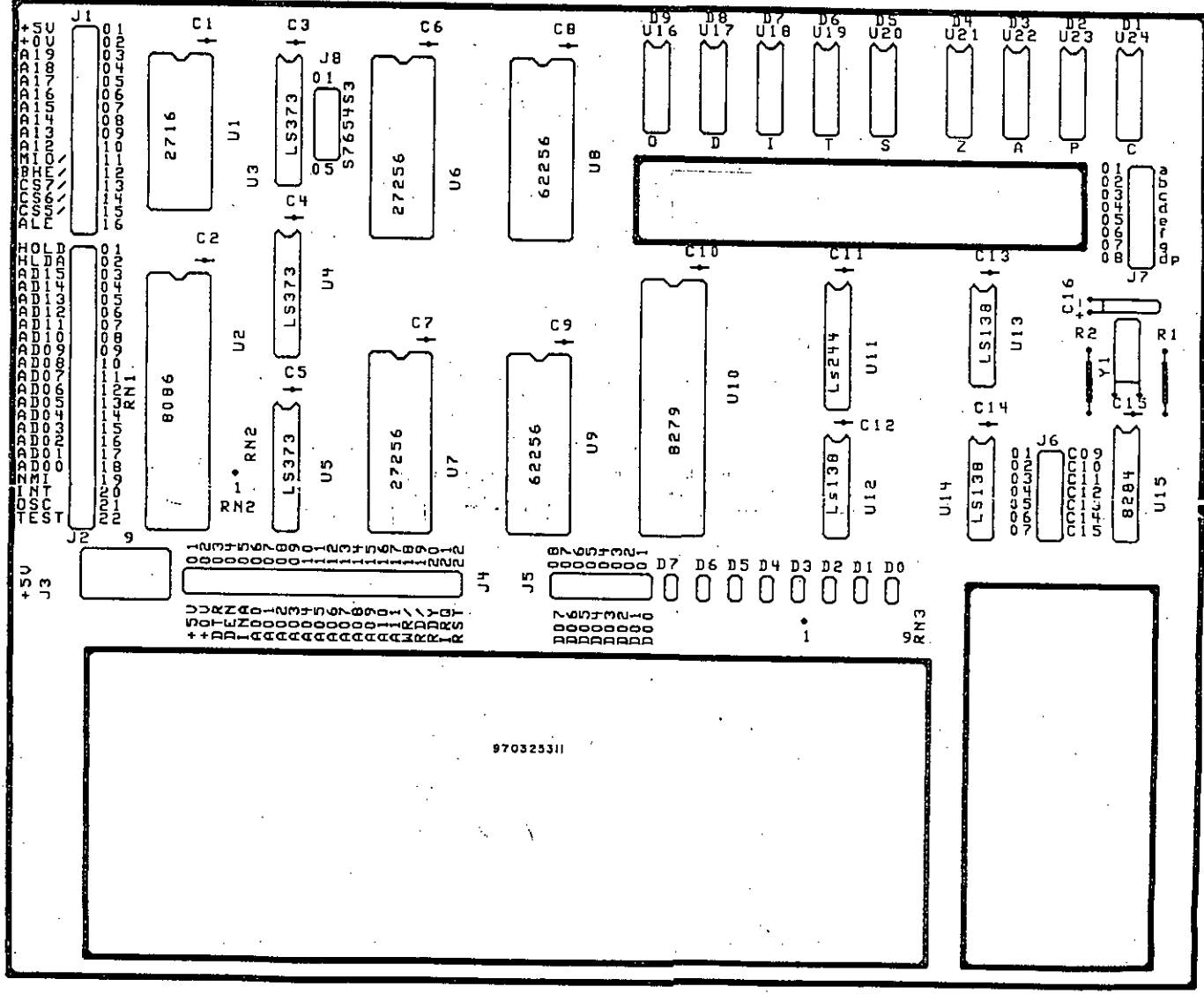


Fig-D.4(b) : Good Soldering

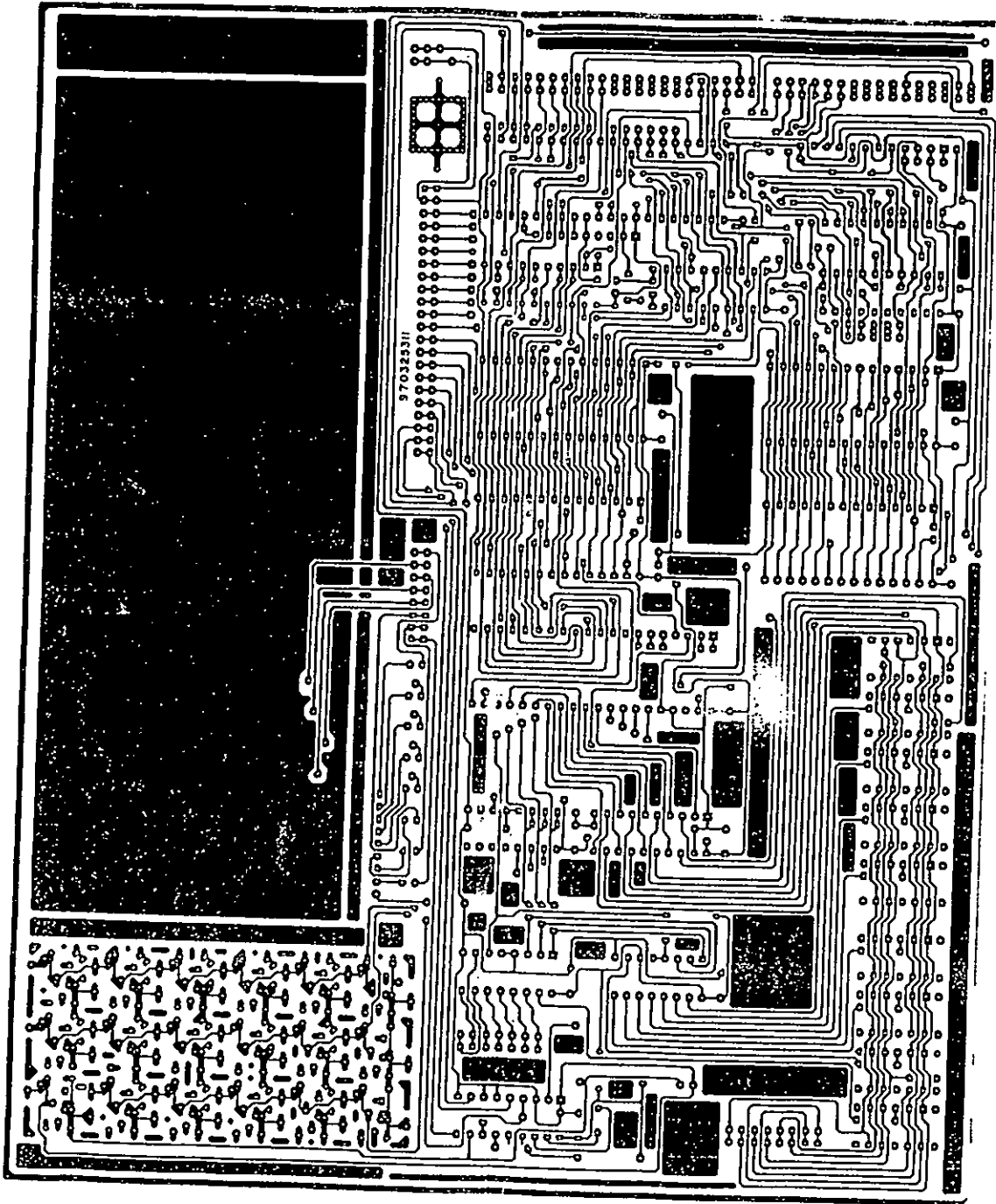
Poor Solder Connection: Fig - D.4 (c)



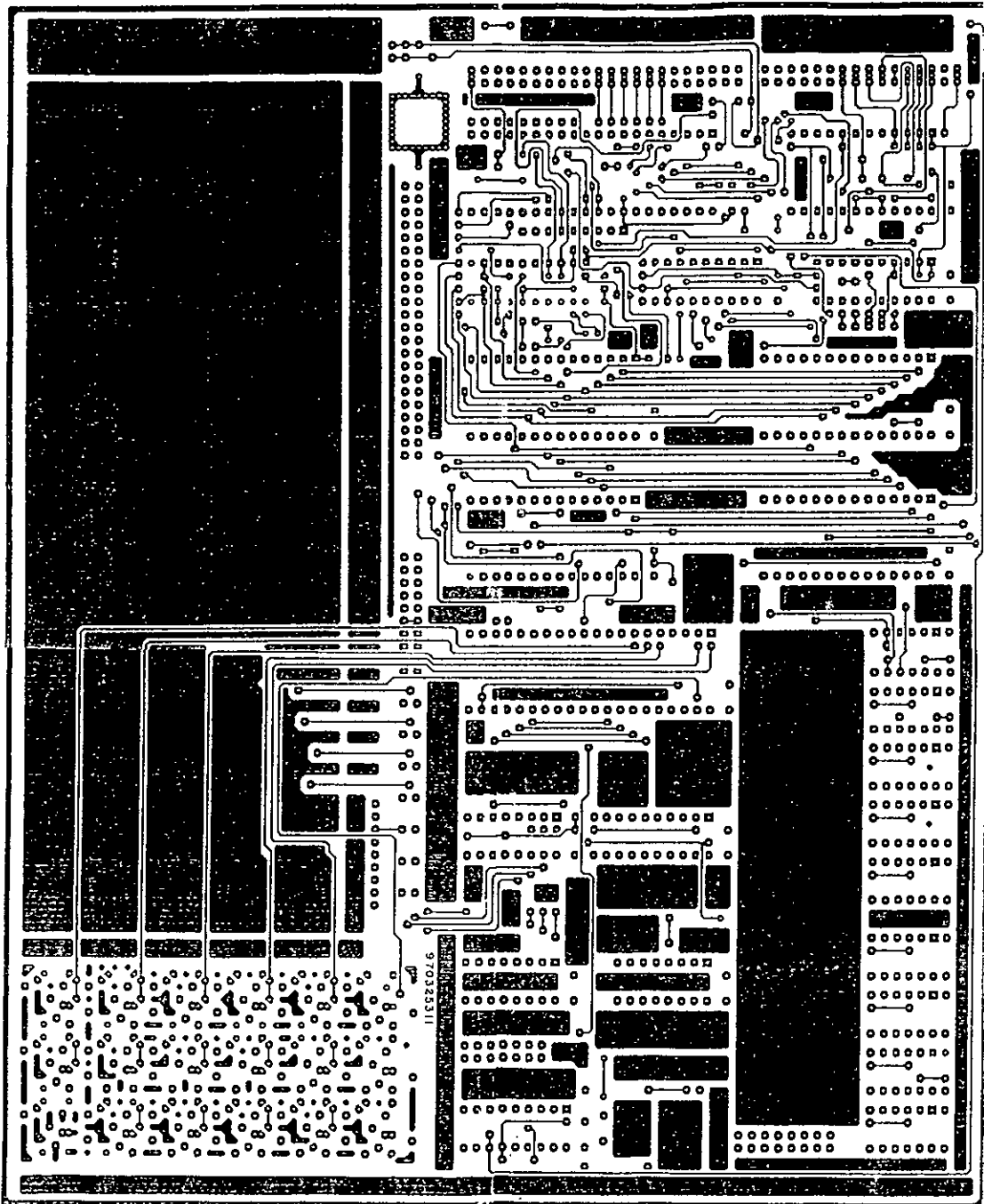
D.5 View of the Trainer Board as seen from TOP



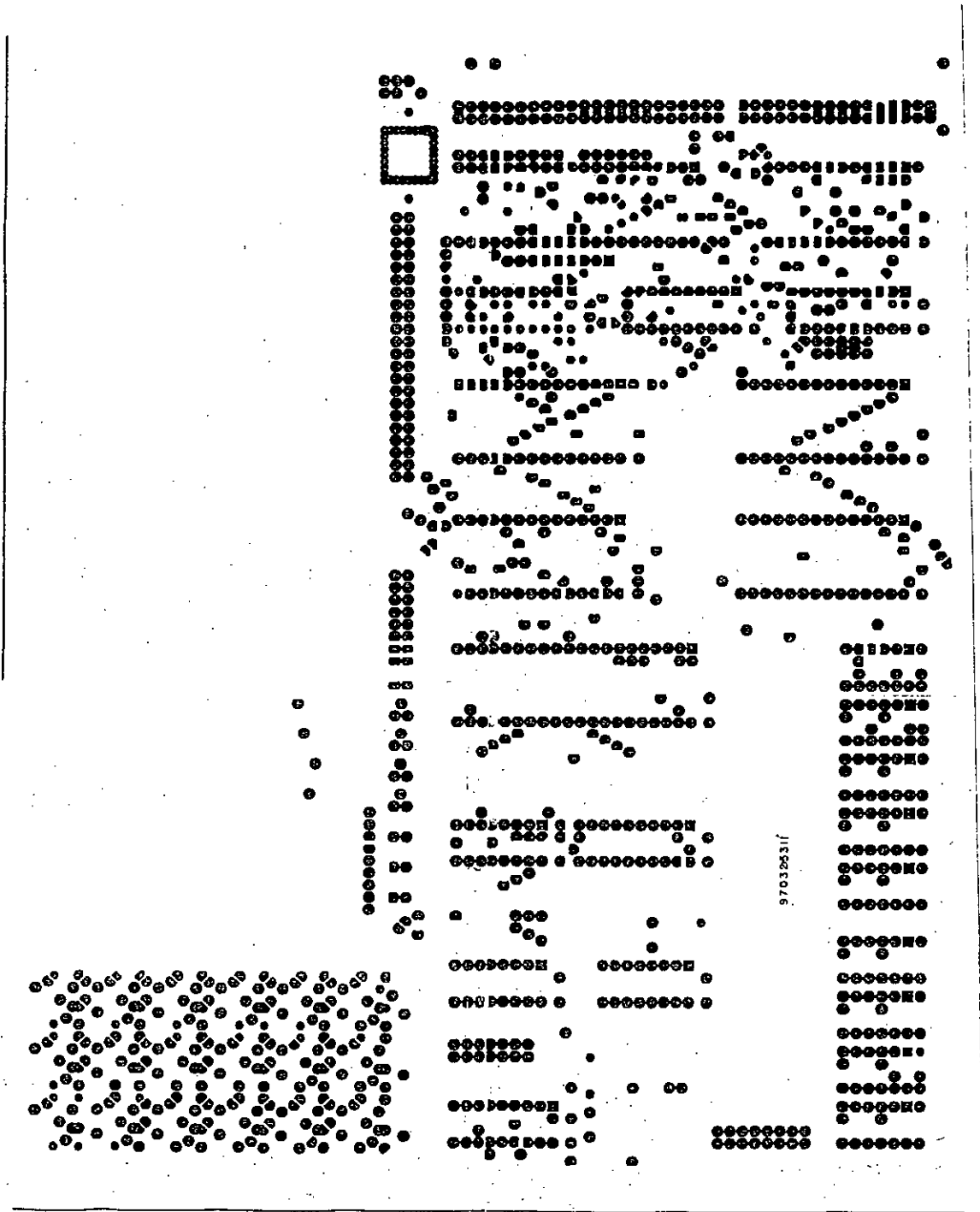
D.6 Solderside PCB Artwork Looking from Bottom



D.7 Component side PCB Artwork Looking form Top



D.8 Solder Mask including PTH (Printed Through Hole Mask) of the PCB Artwork



APPENDIX - E

MONITOR PROGRAM DEVELOPMENT TECHNOLOGY

Introduction

The monitor program is the mind of the trainer. It regulates all possible behavior of the trainer. The size of the monitor program is about 12Kbytes including the kernel, the subroutines, the stand-alone routines and the data tables. The code/data bytes are fused in two EPROMs arranged in ODD and EVEN banks.

This section will make an attempt to document the methods adopted to write and fuse the program codes/data into the EPROMs.

Let us begin with an example to demonstrate how has the job been accomplished. The example is the Subroutine-4 (SUR#4). This subroutine allows to print a hex digit at any position of the 7-segment display unit..

01. Preparation of the Source Codes using EDIT program of the DOS Package.

```
MYCODE    SEGMENT                ; 01
           ASSUME    cs:MYCODE    ; 02

           mov     al, BYTE PTR [bx]    ; 03
           mov     bx, BYTE PTR [bx]    ; 04
           add     bl, al                ; 05
           mov     al, cs:[di]          ; 06
           mov     si, 044Ch            ; 07    register si to be loaded with
4C04h
           mov     di, [si]             ; 08
           mov     [di], al             ; 09
           dec     di                   ; 10
           mov     [si], di             ; 11
           mov     bx, 0004h            ; 12    bx is to be loaded with 0400h
           call    SUR#3                ; 13    calling at F000:FFB6
           dec     BYTE PTR [bx+40h]    ; 14
           ret                               ; 15
MYCODE    ENDS                        ; 16
           END                                ; 17
```

Note that at line-12, the instruction is 'mov bx, 0004h'. Actually, the standard way of writing is 'mov bx, 0400h' while using MASM and LINK. The MASM produces codes 'BB 04 00' for the instruction 'mov bx, 0400h' and the LINK transposes these codes into 'BB 00 04' to satisfy the condition that for Intel processor the lower byte comes first. But, since we did not use LINK program, we had to write the instruction in such way so that when the program is processed by MASM, we get the right codes - lower byte comes first and ready to enter in the trainer manually.

02. Now assemble the above source codes using MASM and prepare the following SUR4.lst file.

The SUR4.ASM file has to be repeatedly assembled until all the errors have gone except the errors for the undefined symbols. In the case of this example program, the only undefined symbol is SUR#3. We know the calling address of the SUR#3 (Subroutine - 3) which we could not enter into the source codes due to certain limitations of the Macro-Assembler package.

```

0000 -      8A 07
0002 -      8B 1F
0004 -      02 D8
0006 -      2E 8A 05
0009 -      BE 4C 04
000C -      8B 3C
000E -      88 05
0010 -      4F
0011 -      89 3C
0013 -      BB 00 04
0016 -      9A B6 FF 00 F0 ; 20-bit address of SUR#3 is know. So, we use FAR call
                                ; The opcode for FAR call is '9A'. The instruction size ;
                                ; is 5 bytes. Therefore, the next instruction would start at
                                ; offset 001B.

001B -      FE 4F 40
001E -      CB ; the opcode for 'ret' instruction due to FAR call is 'CB'.
                                ; C3 is the opcode for the 'ret' instruction of NEAR call.

```

03. Now, choose suitable EPROM space (20-bit system address) from which the codes will be stored.

Let us choose memory location FF400h.

Now, rewrite the program codes of step-02 as follows with respect to base address FF400.

```

FF400 -      8A 07
FF402 -      8B 1F
FF404 -      02 D8
FF406 -      2E 8A 05
FF409 -      BE 4C 04
FF40C -      8B 3C
FF40E -      88 05
FF410 -      4F
FF411 -      89 3C
FF413 -      BB 00 04
FF416 -      9A B6 FF 00 F0 ; 20-bit address of SUR#3 is know. So, we use FAR call
                                ; The opcode for FAR call is '9A'. The instruction size ;
                                ; is 5 bytes. Therefore, the next instruction would start at
                                ; offset 001B.

FF41B -      FE 4F 40
FF41E -      CB ; the opcode for 'ret' instruction due to FAR call is
                                ; 'CB'. ; C3 is the opcode for the 'ret' instruction of
                                ; NEAR call.

```

04. Splitting the Codes into EVEN and ODD Locations

The instruction codes of section-03 will be fused into two EPROMs. The EVEN address numbered bytes will be fused into EVEN banked EPROM and the ODD address numbered bytes will be fused into ODD banked EPROM. Therefore, r-arrange them manually as follows:-

EVEN address Bytes

FF400 -	8A
FF402 -	8B
FF404 -	02
FF406 -	2E
FF408 -	05
FF40A -	4C
FF40C -	8B
FF40E -	88
FF410 -	4F
FF412 -	3C
FF414 -	00
FF416 -	9A
FF418 -	FF
FF41A -	F0
FF41C -	4F
FF41E -	CB

ODD address Bytes

FF401 -	07
FF403 -	1F
FF405 -	D8
FF407 -	8A
FF409 -	BE
FF40B -	04
FF40D -	3C
FF40F -	05
FF411 -	89
FF413 -	BB
FF415 -	04
FF417 -	B6
FF419 -	00
FF41B -	FE
FF41D -	40
FF41F -	FF

05. Fusing EVEN addressed Bytes into EVEN banked EPROM

The EPROM type is 27256 of capacity 32kbytes. Its unit address is 0000h- 7FFFh. Now, we need to determine which unit address of this ROM does correspond to the system address FF400h.

Procedures:

- a. Get the 20-bit physical address written here. And enter the 1st location bits

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0

- b. Now, we discard the lowest significant bit. Write down below what we have got:-

1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- c. Now, get the hex format of the bit pattern of step-02. We get:-

7A00

- d. So, the data value 8A will be fused at location 7A00h, 8B at location 7A01 and so on.

- e. Now, re-arrange EVEN address data bytes of section - 04 as follows:-

EVEN address Bytes

7A00 -	8A
7A01 -	8B
7A02 -	02
7A03 -	2E
7A04 -	4C
7A05 -	8B
7A06 -	88
7A07 -	4F
7A08 -	3C
7A09 -	00

7A0A	-	9A
7A0B	-	FF
7A0C	-	F0
7A0D	-	4F
7A0E	-	CB

06. Actual Fusing of the Data into EPROM

We use the IBM-PC based DOS dependent EPROM Programmer of the following particulars:-

01. EPROM Programmer : Model : EW - 901 BN
02. Manufacturer : Shunshine Company of Taiwan.
03. Local Vendor : Sabium Electronics
34, Stadium Swimming Pool
Dhaka - 1000, Bangladesh

Procedures:

01. Execute the EPROM1.exe program.
02. Select the EPROM as 27256 for $V_{pp} = 12.5V$
03. Select the programming pulse algorithm as 'Quick'
04. Insert a blank EPROM in the ZIF socket.
05. Make a Read Operation
06. Choose the Edit option
07. The screen should show all FFs or at least that part where we want to fuse our codes
08. Now, enter the codes of step-04 into the specified memory locations.
09. Now program the (fusing) the codes into the EPROM.

07. The procedures applied for EVEN address bytes fusing equally applicable for ODD address bytes fusing.

APPENDIX - F

COLOR PLATES

F.1 Component Side of the Prototype 8086 Trainer

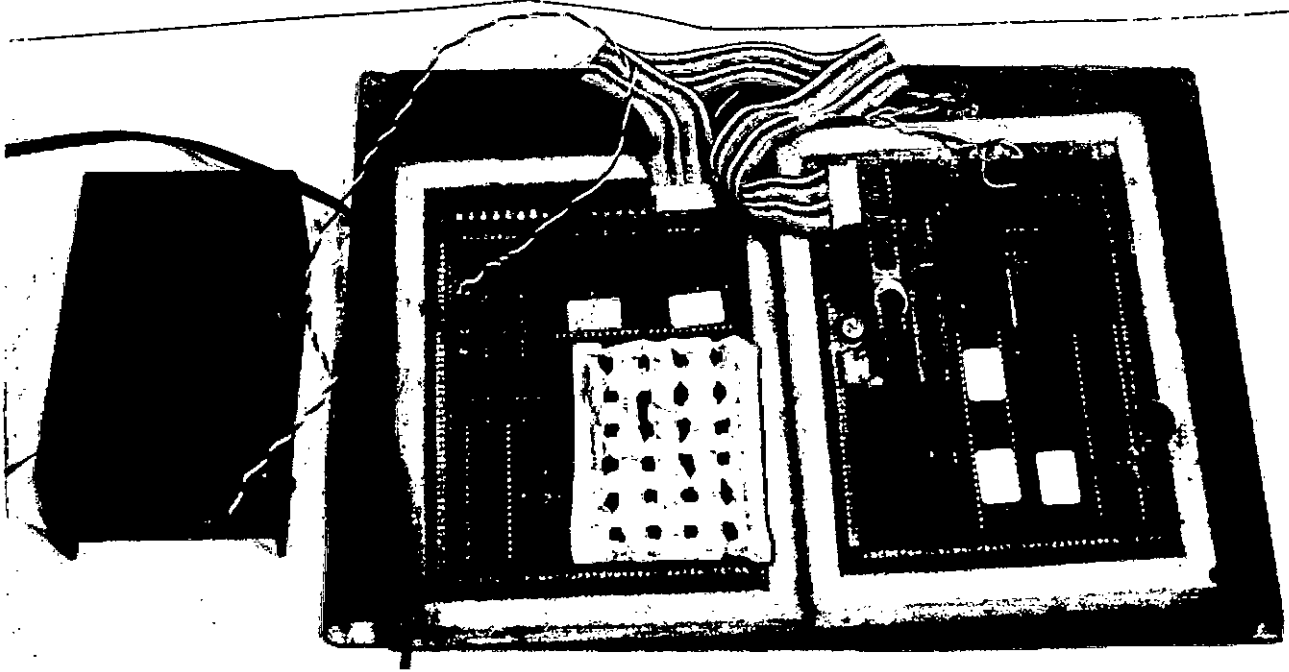


Fig-F.1: Component Side of the Prototype 8086 Trainer

F.2 Wire-wrap Side of the 8086 Trainer

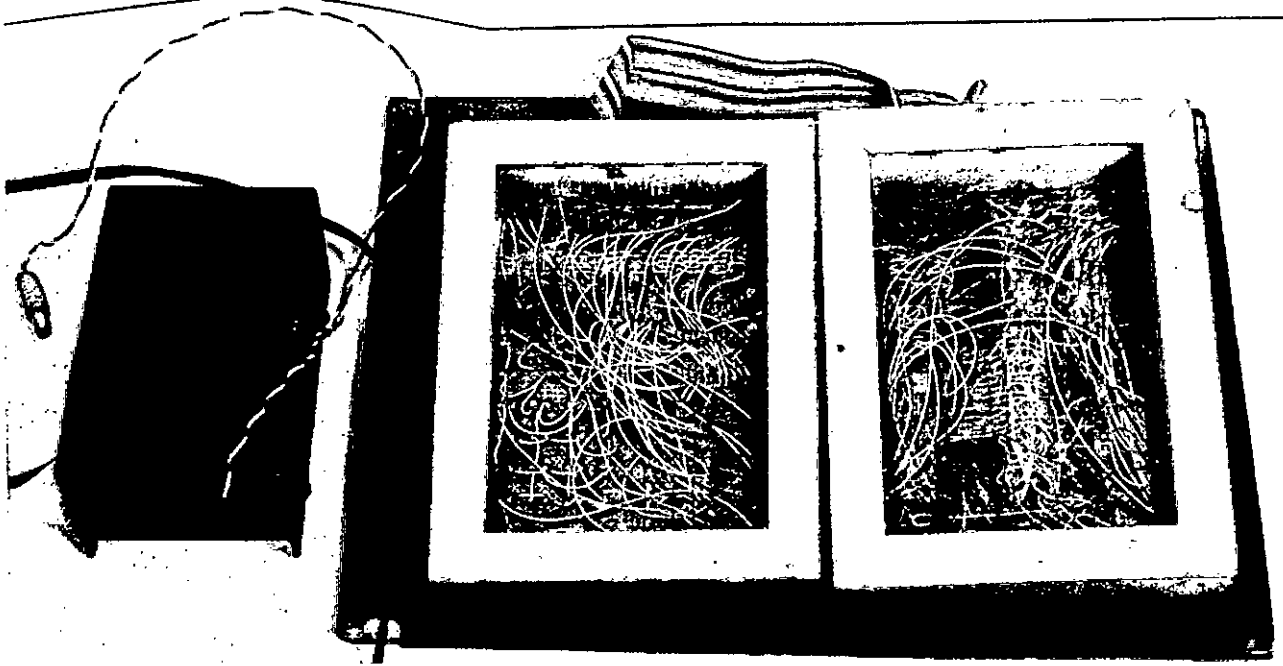
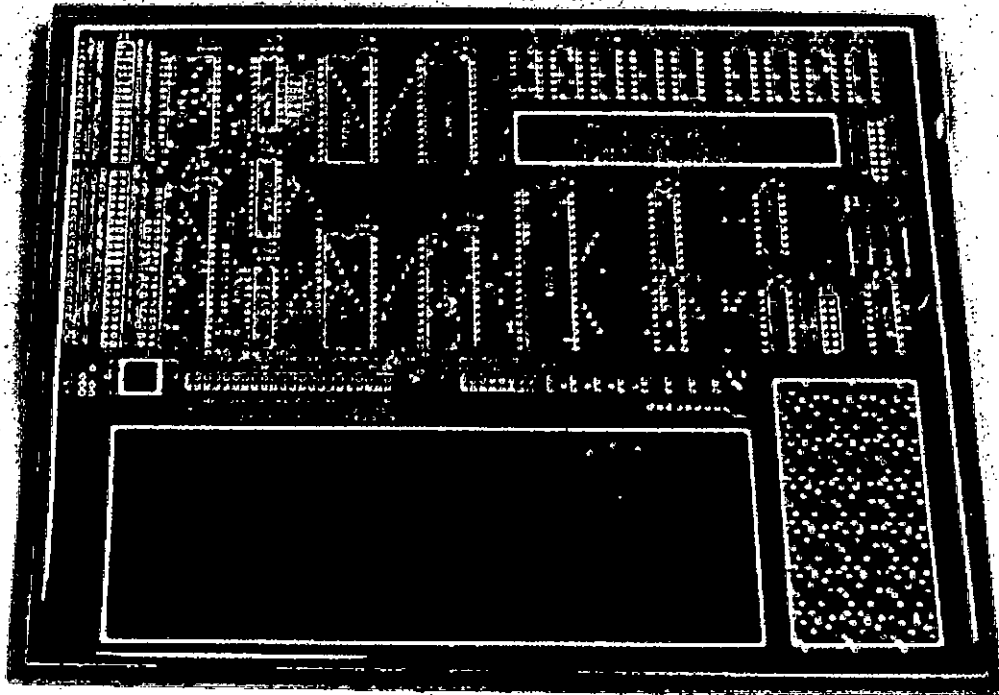


Fig-F.2: Wire-wrap Side of the 8086 Trainer

F.3 Component Side of the PCB of the 8086 Trainer



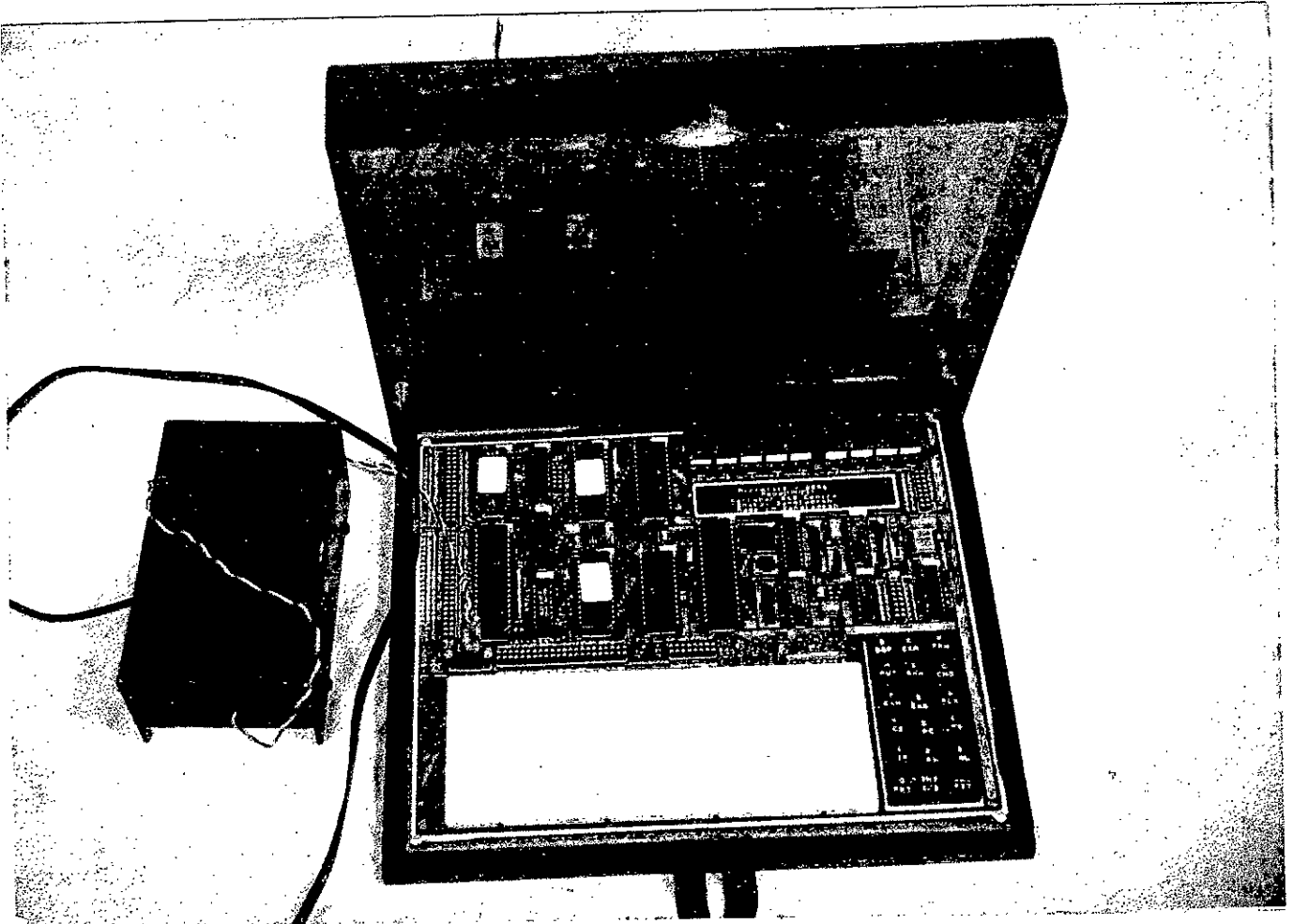
F.3 Component Side of the PCB of the 8086 Trainer

F.4 Solder Side of the PCB of the 8086 Trainer



F.4 Solder Side of the PCB of the 8086 Trainer

F.5 Pictorial View of the 8086 Trainer



F.5 Pictorial View of the 8086 Trainer

REFERENCES

- [1] Bartee, T.C., Digital Computer Fundamentals, 6th ed., McGraw-Hill Book Company, 1987.
- [2] Brumm, P. and Brumm, D., 80386 : A Programming and Design Hand Book, 2nd ed., TAB Books Inc., 1989.
- [3] Duncan, R., MSDOS Programming, 2nd ed., Microsoft Press, 1988.
- [4] Flight electronics Limited : UK, Catalog - 1990 ed.
- [5] Gibson, G.A. and Liu, Y., Microcomputer Systems : The 8086/8088 Family, 2nd ed., Prentice-Hall of India Private Limited, 1994.
- [6] Gilmore, C.M., Microprocessors : Principles and Applications, 2nd ed., Glencoe-McGraw-Hill, 1995.
- [7] Ginsberg, G.L., Printed Circuits Design : Featuring Computer Aided Technologies, 1st ed., 1990.
- [8] Givone, D.D. and Roesser, R.P., Microprocessors/Microcomputers: An Introduction, 1st ed., McGraw-Hill Book Company, 1985.
- [9] Hall, D.V., Microprocessors and Digital Systems, 2nd ed., McGraw-Hill Book Company, 1987.
- [10] Hall, D.V., Microprocessors and Interfacing : Programming and Hardware, 1st ed., McGraw-Hill Book Company, 1987.
- [11] Hayes, J.P., Digital System Design and Microprocessors, 1st ed., McGraw-Hill Book Company, 1987.
- [12] Heath User Group : USA : catalog 1990-1991 ed.
- [13] Intel Corporation, Microprocessors Data Book, 1990.
- [14] Intel Corporation : USA, SDK-85 System Design Kit - Users Manual
- [15] Intel Corporation, Memory Data Book, 1990.
- [16] LS Z80 Operations manual : Binary systems : UK.
- [17] Middendorf, W.H., Design of Devices and Systems, 1st ed., Marcel Dekker, Inc, 1986.
- [18] Miller, M.A., The 68000 Microprocessor : Architecture, Programming and Applications, 1st ed., Universal Book Stall, 1989.
- [19] Mostafa, G., Reference Manual : MicroTalk-8085, 3rd ed., Karighar R&D Centre, 1996.
- [20] Mostafa, G., reference Manual : MicroTalk-8751, 1st ed., Karighar R&D Centre, 1995.
- [21] Peatman, J.B., Microcomputer-Based Design, 1st ed., McGraw-Hill Book company, 1988.
- [22] Rafiquzzaman, M., Microprocessors and Microcomputer-Based System Design, 2nd ed., CRC Press, 1995.
- [23] Ramshaw, R.S., Power Electronics: Thyristor controlled power for electric motors, 1st ed., ELBS Book Societ, 1979.
- [24] Scanlon, L.J., 8086/8088/80286 Assembly Language, 1st ed., A Brady Book, 1988.
- [25] Tedeschi, F.P. and Kueck, G., 101 Microprocessor Software and Hardware Projects, 1st ed., TAB Bokks Inc., 1982.
- [26] The Computer Application Journal : USA, December - 1993 ed.
- [27] Wiatrowski, C.A. and House, C.H., Logic Circuits and Microcomputer Systems, 1st ed., McGraw-Hill Book Company, 1984.
- [28] 6802 Microprocessor Trainer manual : Heath Company : USA.

