SYSTEM CONTROLLER USING 8088

Project Report

P-129

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OE ENGINEERING IN ELECTRICAL AND ELECTRONICS ENGINEERING OF THE BHARATHIAR UNIVERSITY COIMBATORE - 641 046

SUBMITTED BY

T. S. SARANGARAJAN
N. FATHIMAPARVIN
N. SURESH

UNDER THE GUIDANCE OF

Dr. K. A. PALANISWAMY
B.E., M.Sc. (Eng), Ph D.,
M I S.T.E., C. Eng (I), F.I.E.,

MISS. C. AMUTHAVALLI B.E.

1990-91

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE - 641 006

Department of Electrical and Electronics Engineering

Kumaraguru College of Technology

Coimbatore - 641 006

Certificate

This is to certify that the report entitled System Controller Using uP-8088 has been submitted by

Mr./Miss ______________________________

In partial fulfilment for the award of Bachelor of Engineering in the Electrical and Electronics Engineering branch of the Bharathiar University, Coimbatore-641 046 during the academic year 1990-91

[illegible] 14/3/91

Guide

Dr. K. A. PALANISWAMY, B.E., M.Sc. (Engg), Ph.D., MISTE, C.Engg (I), FIE

Professor and Head

Department of Electrical and Electronics Engineering

Kumaraguru College of Technology,

Coimbatore 641 006

Head of the Dept.

Certified that the candidate was examined by us in Project Work Viva-Voce examinations held on [illegible] and the University Register No was ______________

[illegible]

Internal Examiner

[illegible]

External Examiner

2/4/91

ACKNOWLEDGEMENT

We are profoundly grateful to our project guides **Dr.K.A.Palaniswamy** B.E., M.Sc.(Engg.), Ph.D., M. I. S. T. E., C.Eng(I), F.I.E., Professor and Head of Department of EEE and **Miss C.Amuthavalli** B.E., for their valuable guidance and encouragement which has brought success to this project.

We thank our Principal **Prof. R.Palanivelu** B.E., M.Sc.(Engg.) M. I. S. T. E., MISPRS, for providing all facilities to carry out this project in the College.

We would be failing in our duty if we do not thank **Dr.M.Jayapragasam** B.Sc.(Ag.), M.Sc., Ph.D., of Tamil Nadu G.D.Naidu Agricultural University, Coimbatore who has provided us with his PC for developing PCB circuitry.

Last but not least we would like to thank members of staff and laboratory technicians of EEE Department who helped us directly or indirectly resulting in the completion of this project.

SYNOPSIS

The latest development in Computer technology is the 'MICROPROCESSOR', a device that has all the function of the central processing unit of a computer on one or a few tiny pieces of silicon. Such a device can fetch instruction from memory, decode it and execute them, perform arithmetic and logical operations, accept data from input devices and send results to output devices.

There are many types of microprocessors, 8-bit microprocessors are commonly used for control and measurement applications. In this project an 8-bit microprocessor kit using 8088 chip is designed and fabricated. The monitor program is developed and stored in the EPROM.

This project model can be used as a trainer kit for practicing 8086/8088 software programs. This kit can also be used for control purposes like temperature, flow, pressure etc. The main chip used is Intel 8088.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Microprocessor is now used in every walk of life. It is CPU along with some registers fabricated in a single chip by very large scale integration (VLSI).

As microprocessors are small in size they form a part of the system. Another feature is that as microprocessors are used to perform single task they have fixed program and hence the programs are stored in permanent medium or read only memory (ROM). Microprocessors often perform real-time task like continuous monitoring of frequency and giving alarm when there is a deviation in frequency from the required value in generating station.

1.1 Evolution of Microprocessor:

The first microprocessor was designed by Intel in 1971. It was a 4 bit microprocessor (INTEL 4004). In 1973 Intel released the first 8-bit microprocessor Intel 8080. The improved version of 8080, Intel 8085, was released later by intel. There are many 8 bit microprocessor released by other companies like Z-80, Motorolas 6800 etc.

In 1978 the first 16-bit microprocessor Intel 8086 was released. The intel then released 8088 which is a 8-bit version of 8086. It has only 8-bit output data path but 16 bit internal data path. Hence 8088 microprocessor has power of 16 bit microprocessor but simplified hardware.

The 8088 has many built-in functions like multiplication, division etc. which makes it more easy to be operated than any other 8 bit microprocessor. It is also faster in operation than any other 8 bit microprocessor.

1.2 Advantages of 8088

1. Writing software is easy.
2. As there are many built in functions, number of software steps will be less.
3. It is very fast when compared to other 8 bit microprocessor.
4. Hardware is simpler than 8086.
5. The memory accessing capacity of 8088 is 1 mega byte which is 16 times greater than memory accessing capacity of any other 8 bit microprocessor.

CHAPTER II

8088 MICROPROCESSOR

Intel 8088 is 8 bit version of 8086. It is used as CPU in IBM Personal computers and several compatible personal computers.

2.1 Architecture:

An 8088 has just 8 bit external data bus and a 16 bit internal data path. However in order to have upward compatibility with 8085 and other 8 bit microprocessors both bytes and word operation are possible. Another feature of 8088 is that while instructions are fetched from program memory, these are temporarily kept stored inside its so-called bus interface unit (BIU). Upto four bytes in a queue can be held in it, while the execution of the fetched instructions is taking place. Thus, unlike the 8085, it can fetch an instruction even before a fetched instruction is executed. This is called a 'pipeline' architecture and speeds up the program execution. The 8088 will begin its fetching instruction codes as soon as one or more bytes in the queue gets empty.

The architecture of 8088 is given in Fig.2.1

The maximum memory address range of the 8088 is one megabyte. Address lines A15 to A19 are for this purpose. However at one stretch, one can access only a 64 k within this one megabyte space. But one should not think that this one megabyte is partitioned into 16 blocks of 64 k each. The partitions can be anywhere and may even overlap.

These partitions are called 'Segments'. The beginning address of a segment in this one megabyte space can be chosen by 16 bits. For example, let us choose the beginning of the segment at 12340. Then the extent of this segment is from

Beginning address	1 2 3 4 0 (HEX)
to	+ F F F F
End address	2 2 3 3 F

A segment beginning address must have a 0 at its last digit. In the above, it was 1 2 3 4 0. Thus a segment and the next can overlap upto as small as 16 bytes. Out of the 20-bit memory address, the last four bits are always zero for the beginning address of a segment.

There are four segment registers in the 8088 to contain this information about the beginning address. They are called:

CS	Code segment	This segment is for the program codes.
DS	Data segment	This segment is the area of memory where data resides.
SS	Stack segment	This segment is used as stack.
ES	'Extra' segment	This is an 'Extra' segment used for data in those cases where data has to move from one block to another.

2.1.1 Registers:

Apart from four segment registers, remaining registers are similar to that of 8085 registers. They are:

AX	AL
BX	BL
CX	CL
DX	DL

Each of which are 8 bits wide. For word operation we use the following registers.

AL	+	AH	=	AX
BL	+	BH	=	BX
CL	+	CH	=	CX
DL	+	DH	=	DX

Similar to that of Z80 and 6800, 8088 is also provided with index registers. The 16 bit word size registers used for indexing which are useful for memory reference, are:

Source	Index	register	SI	
Destination	Index	register	DI	and
Base	Pointer	register	BP	

The stack pointer (SP) and the program counter (PC) are similar to that of 8085.

Table 2.1 shows the addressing possibilities using code segment, Base pointer and index registers.

The flag register of 8088 has the following 2 bytes.

X	X	X	X	TF	DF	JF	OF	S	Z	X	AC	X	PE	X	C
				11	10	9	8	7	6		4		2		0

The first 7 bits resembles the corresponding flag bits of 8085.

The bits 8 to 11 of the flag register indicate the following:

8	OF	-	Overflow
9	IF	-	Interrupt enable.
10	DF	-	Direction
11	TF	-	Single step trap.

2.2 Pin out details of 8088:

Pit out diagram is given in Fig. 2.2. The details are given below

Pin 1 and 20	-	ground
Pin 2 to 8	-	address lines A14-A8
Pin 9 to 16	-	address data lines AD_7-AD_0
Pin 17	-	NMI (Non Maskable interrupt)
Pin 18	-	INTR (interrupt Request)
Pin 19	-	Clock (5 MHz from 8284)
Pin 21	-	Reset (5 MHz from 8284)
Pin 22	-	Ready (5 MHz from 8284)
Pin 23	-	TEST (Active Low) Execution continues if Low otherwise, processor waits in an ideal state.
Pin 24	-	INTA (Interrupt Acknowledge) Active, Low

Pin	25	-	ALE (Address Latch Enable)
Pin	26	-	DEN (Data Enable Active LOW)
Pin	27	-	DT/$\overline{R}$ (Data transmit/Receive)
Pin	28	-	IO/$\overline{M}$ If High transfer of data between μp & I/O port If Low with memory
Pin	29	-	WR (Write Active Low)
Pin	30	-	HLDA (Hold Acknowledge)
Pin	30	-	HOLD
Pin	32	-	RD (Read active low)
Pin	33	-	MN/$\overline{MX}$ (If high operates in minimum mode. If LOW operates in maximum mode).
Pin	34	-	SSO
Pin	35-39	-	Address lines A19-A15.
Pin	40	-	VCC (+5V)

2.3 Modes of operation:

There are two modes of operating 8088.

a) Minimum mode

b) Maximum mode

With the pin No., 33 tied to 5V the 8088 works in its so-called 'Minimum system' Hardware configuration i.e., in this mode it works independently like other 8 bit microprocessors.

If pin 33 is low the 8088 operates in maximum mode. In this mode 8088 works as one member of a bigger family of multiple microprocessor chips. The 8088 has built in logics to handle bus access priorities in such multiple chip systems. In such systems each processor will have its own memory and each can also share a common memory.

In this project the 8088 is made to operate in minimum mode.

2.4 Instruction set of 8088:

There are around 75 instruction mnemonics in 8086-8088 set. They can be grouped as

2.4.1 DATA MOVEMENT INSTRUCTIONS:

The instruction is used to transfer data from register to register or register to memory. This instruction has both byte and word operations.

e.g. MOV AX, BX i.e., Move contents of BX to AX

XCHG AX, (0500) Move contents of 0500, 0501 to AX

2.4.2 Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication and division between registers or register and a memory location.

e.g. MUL CL Multiply AL with CL & store the result in AL

2.4.3 Logical Instructions:

These instructions perform logical operations like AND, NOT, OR and EXOR.

e.g. AND AL, OFH

ANDs AL with hexa number OF & stores the result in AL.

2.4.4 Shift and Rotate Instruction:

The rotate instruction is used to shift left or right the contents of any register or even a memory byte. To shift the contents of any register left more than once the CL is loaded with that count. The following instruction performs the required shift.

ROL "Memory/REG", CL

2.4.5 CAll ,Return and Jump Instructions:

CALL instruction is used to call a subroutine. RETURN instruction is the last statement of the subroutine which transfers the control to the next instruction of CALL statement. There are two types of jump instructions.

Unconditional instruction:

This instruction transfers the control to address given along with the opcode for unconditional jump instruction.

e.g JMP 07H

Conditional instruction:

These instructions transfer the control if certain condition is satisfied. There are about 17 conditional jump instructions.

e.g.	JZ	Jump if zero flag is set
	JG	Jump on greater

The opcode & corresponding instruction are given in appendix B

2.5 Addressing Modes:

The 8088 uses 5 addressing modes These are discussed below.

Direct addressing:

In this type of addressing the address is part of instruction.

e.g. MOV A'L, (0500)

This transfers contents of memory address 0500 to register AL.

2.5.2 Register Direct:

In this type of instruction the particular register is directly specified along with the instruction

```
MOV  BL,  CH
```

transfers contents of CH to BL

2.5.3 Register Indirect:

In this type of addressing the register containing the address, where the data is present is given along with the instruction .

```
ADD  AL,  (BX)
```

This instruction adds contents of AL with the contentents of memory address given in BX.

2.5.4 Immediate Addressing:

The actual data forms a part of instruction in immediate addressing.

e.g. MUL C & OFH

This instruction multiplies contents of CL with hexa number & stores the result in CL.

2.5.5 Index Addressing:

Indexed addressing means that the CPU adds the contents of an index register to the address supplied with the instruction in the order to find the effective address.

e.g. MOV CL, (BX)+(SI)

This instruction transfers contents of memory address given by the sum of contents of registers BX & SI to register CX.

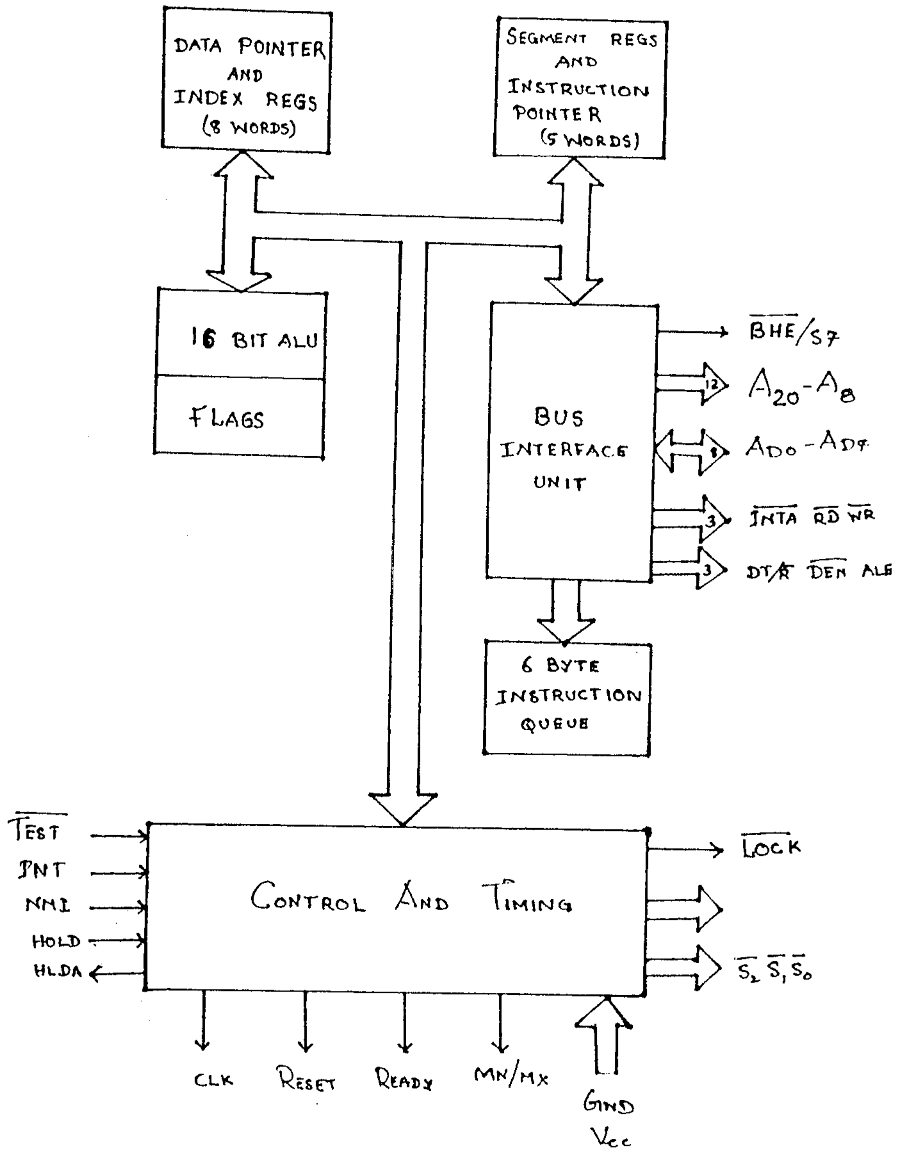

Fig. 2.1. Block Diagram Of 8088 Architecture.

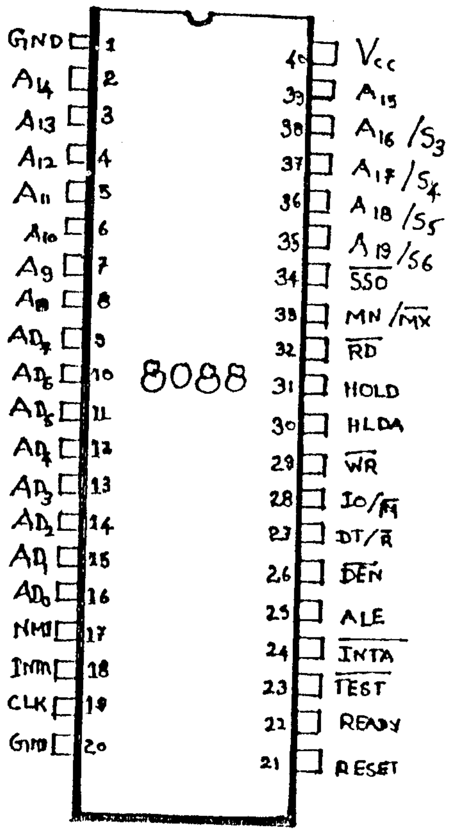

FIG: 2.2 : PINOUT DIAGRAM OF 8088.

TABLE 2.1 : ADDRESSING POSSIBILITIES USING CODE SEGMENT BASE POINTER AND INDEX REGISTERS.

Memory Reference	Segment Register	Base Register	Index Register	Possible Displacements			Assembly Language Operand Mnemonic
				16-Bit Unsigned	8-Bit High-order Bit Extended	None	
Normal Data Memory Reference	DS (Alternate* CS, SS or ES)	None	SI	*	*	*	SEE INSTRUCTION SET
			DI	*	*	*	
		BX	SI	*	*	*	
			DI	*	*	*	
			None	*	*	*	
	DS	None	None	*			
	SS (Alternate* CS, DS or ES)	BP	SI	*	*	*	
			DI	*	*	*	
			None	*	*		
Stack	SS	SP	None				
String Data	DS	None	SI				
	ES	None	DI				
Instruction Fetch	CS	PC	None				
Branch	CS	PC	None		*		
I/O Data	DS	DX	None				

CHAPTER III

HARDWARE

The hardware part consists of designing the circuit and fabrication of a PCB for the circuit with selected support chips.

3.1 Chips Used:

Main chip

Intel 8088 with clock rating 5 MHz. The detailed specification are given in Table 3.1

Support chips

8284 - Clock generator

The 8088 does not have an internal clock. So an external clock generator 8284 is used. A 15 MHz crystal oscillator is used. This 15 MHz is divided by three and three 5 MHz signals are fed as input to the Reset, RDY and CLK of 8088. The pin 2 of 8284 gives a signal of 2.5 MHz which is used for other peripheral devices such as 8279.

Fig. 3.1 illustrates Timing Circuit for 8088 using 8284.

74 LS 245:

This is used as a Buffer for I/O lines from the main IC 8088.

74 LS 373:

This is used for latching the lower 8 address bits (AD_0-AD_7).

The Fig 3.2 gives the buffering & latching circuitry for 8088 using LS 245 & LS 373.

74 LS 155:

This is 1 out of 8 decoder used as memory chip selector.

2732:

4 KB EPROM used to store monitor program.

6116:

RAM 4KB User Free.

The memory connection for 8088 is given in Fig.3.3

8253:

TIMER

8255

Programmable peripheral Interface

The Fig 3.4 illustrates how 8255 is connected

8279:

For key board and display routines.

FND 507:

7 segment LED display.

The keyboard & display interfacing circuit is given in Fig.3.5

3.2 Reset:

The reset key causes a hardware reset and starts the monitor. If this key is pressed it passes a low signal to pin 11 of the 8284. This resets 8088 i.e. the code segment is intialized to FFFF. Hence the monitor program is written from FFFF. On pressing Reset Key "8088 μ P." sign is displayed.

3.3 Key Board and Display Interface:

The 8088 keyboard and display is controlled by IC8279. It is programmed to operate in encoded keyboard with two key lockout and 8 bit character with left entry mode. The keyboard reading is implemented through polling the 8279 status. The 8279 can be addressed as follows:

8279 control Address-0E19H

8279 Data Address-0E18H

The display consists of 6 number of seven segment LED displays separated into two fields. The left field comprising 4 LEDs is called as address field and the right field comprising two LEDs is called as data field. The 4 bit address is displayed in address field and the opcode or data is displayed in data field.

The correlation between the individual bits and the display segments is given in table 3.2.

3.4 Program Testing

For testing programs using this kit the program is loaded in RAM. To load the program into RAM, "EXAM MEM" key is first pressed and then starting address is loaded. The data i.e. opcode is loaded after pressing "NEXT" key. The data or opcode is entered in the subsequent address field after pressing "NEXT" key. On pressing "PRV" key the address is decreased by 1.

For execution of program following keys are to be pressed.

GO	STARTING ADDRESS	EXEC

3.5 Keyboard Monitor Command Supply:

COMMAND	FUNCTION/FORMAT
Examine/modify memory	Displays/Modifies the contents of memory location EXAM MEM "ADDR" NEXT

Examine/Modify Register	Displays/Modifies 8088 register contents EXAM REG "HEXPAD"
GO	Transfer control from monitor to user program GO (ADDR) EXEC

3.6 Programmable Peripheral Interface Devices:

The kit has two number of 8255A. Each 8255A consists of a command port and three 8 bit programmable input/output ports called PortA, PortB, and PortC. These 8255 are completely available to user. The port signals are available on connector J1 and J2.

3.7 Interrrupts:

There are two hardware interrupts INTR and NMI both built within the chip.

NMI - Non Maskable Interrupt

INTR - Interrupt request

An interrupt caused by signal applied to one of these inputs is referred to as hardware interrupt.

A second source of interrupt is software interrupt. This is got by executing the interrupt instruction INT.

Another type of software interrupt is 'Interrupt on overflow'. The mnemonic for this instruction is INTO. This interrupt is enabled if overflow flag is set to 1. There is another interrupt called as 'TYPE 0 INTERRUPT' which is enabled when a number is tried to be divided by zero.

A programmable interrupt controller 8259 can be connected to this kit to have more hardware interrupts.

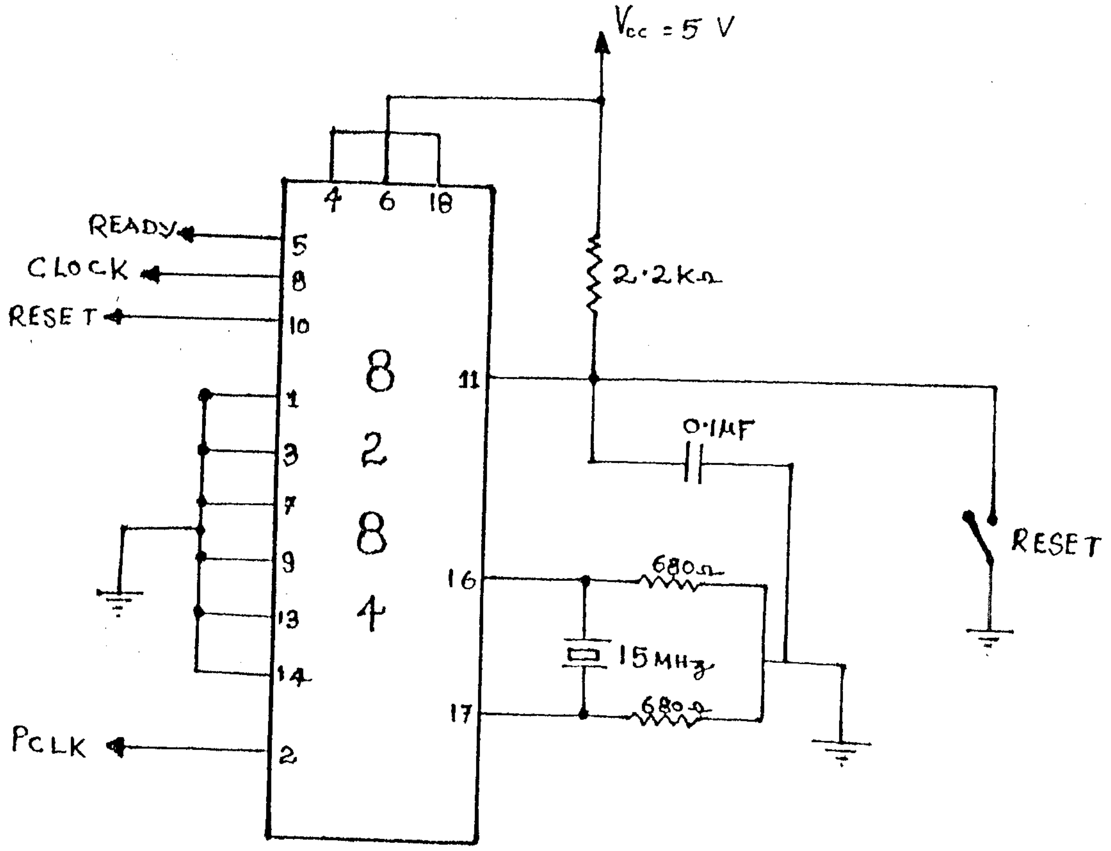

FIG: 3·1 TIMING CIRCUIT FOR 8088

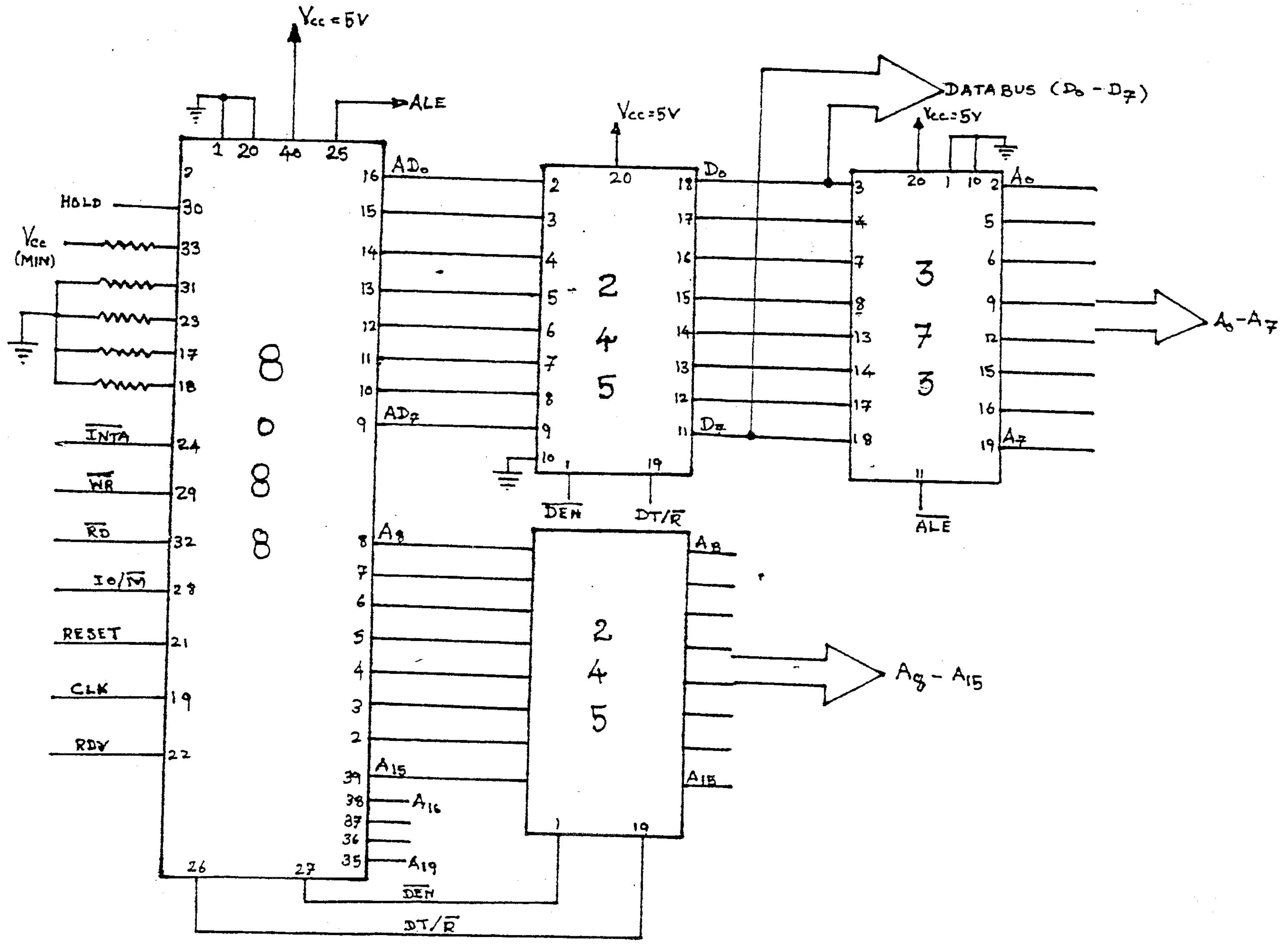

Fig 3.2 Buffering And Latching Of 8088 Address And Data Lines

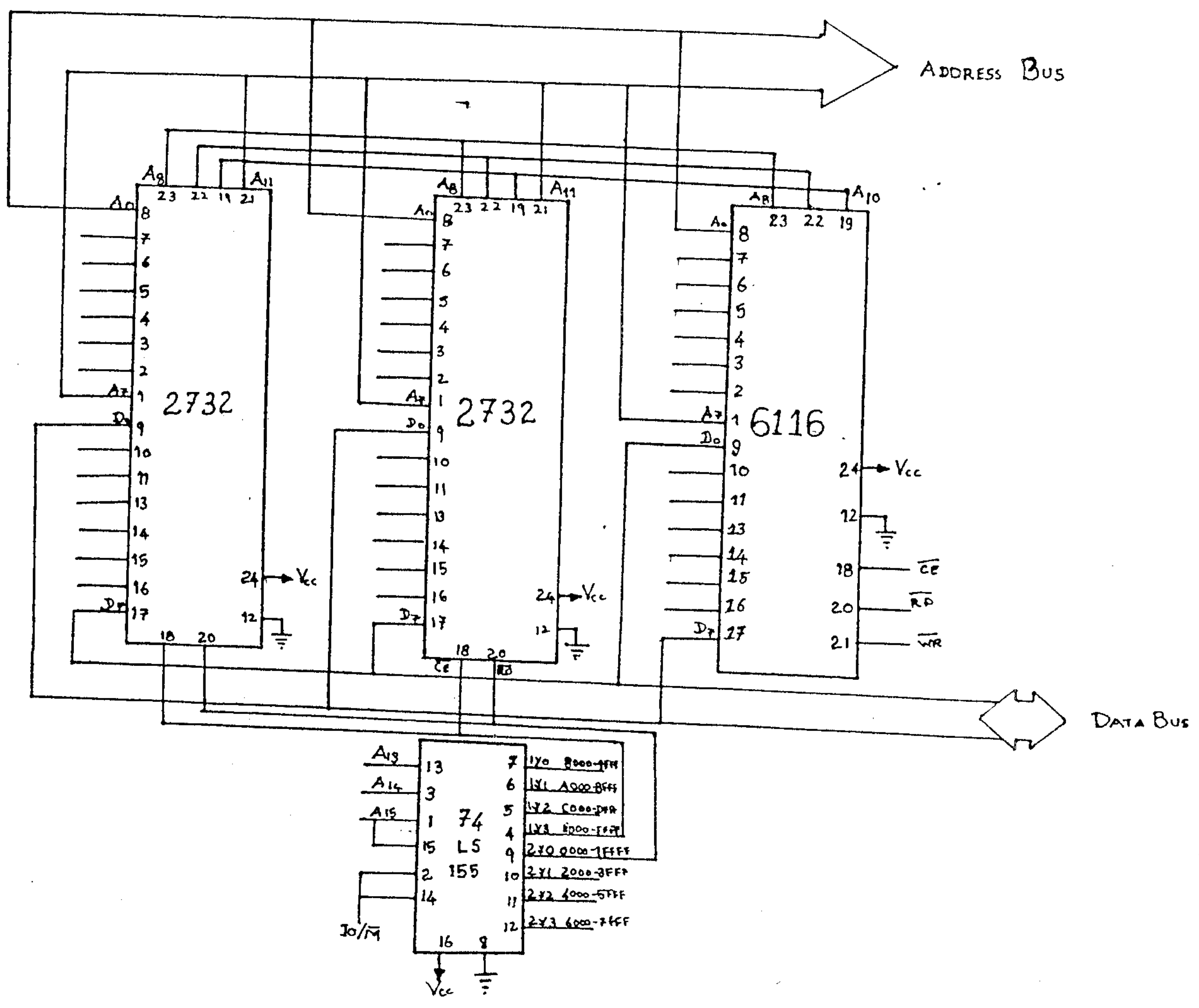

Fig. 3·3: Memory Connection For 8088.

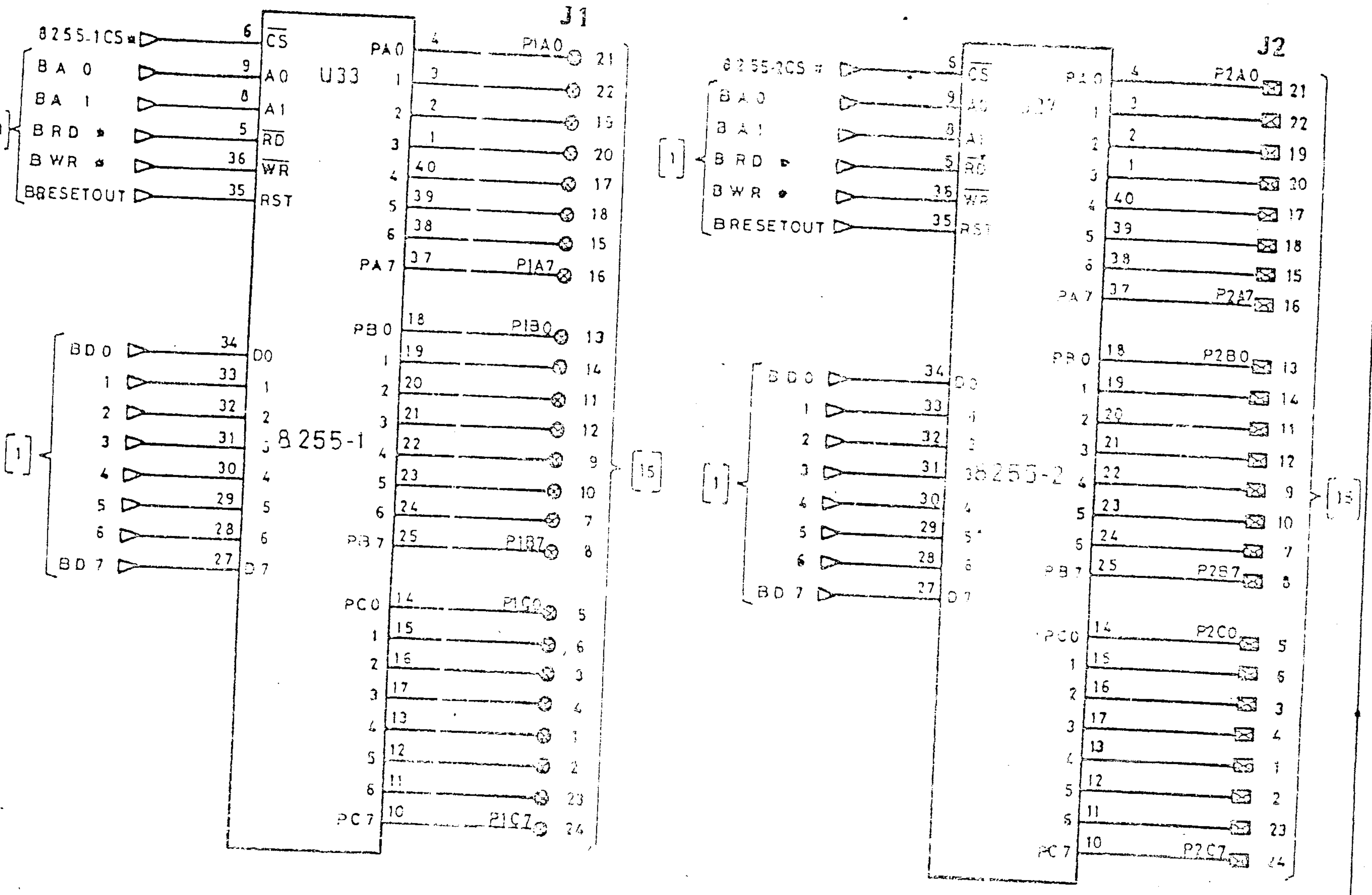

Fig. 3·4 Programmable Peripheral Interface

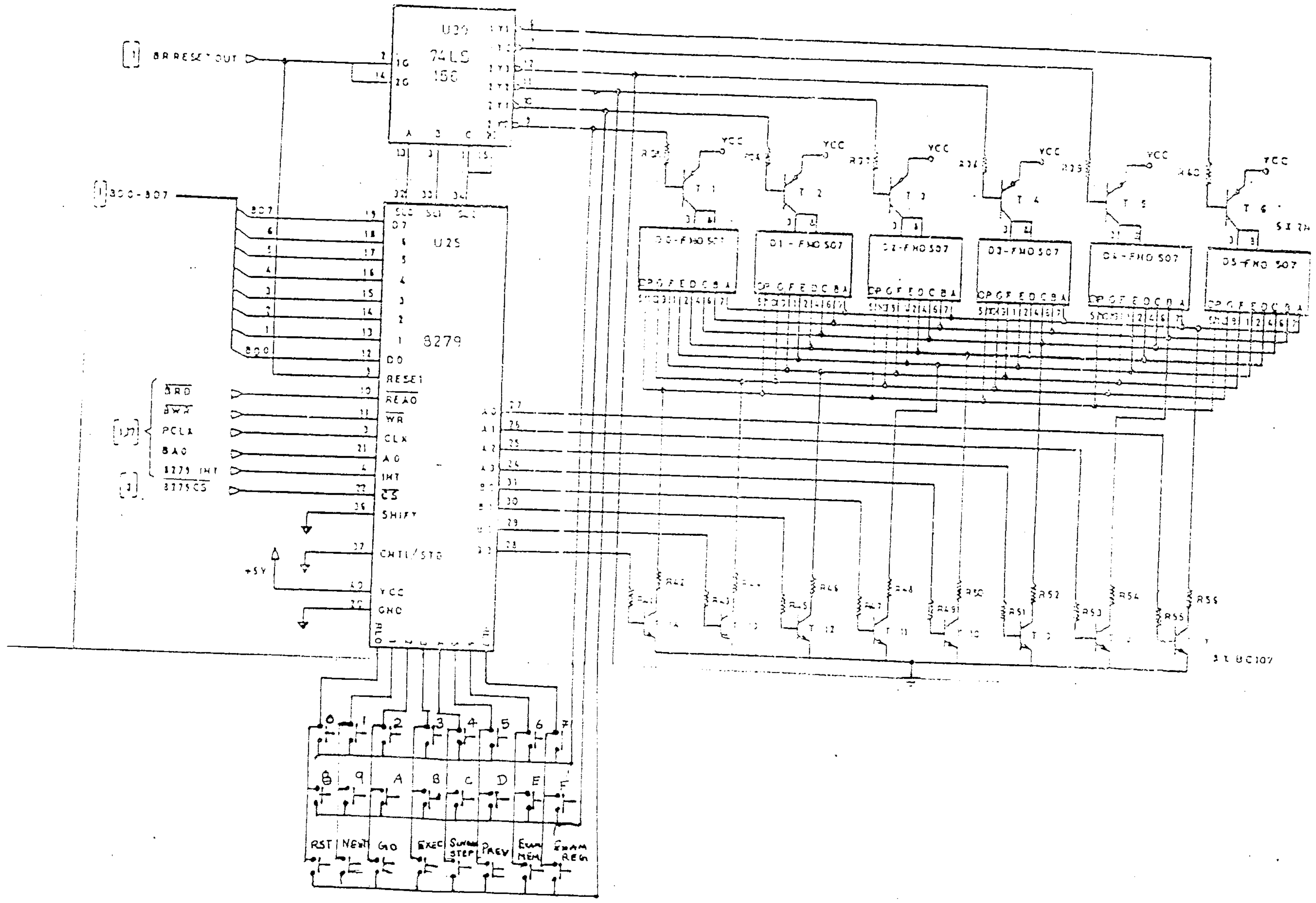

Fig. 3.5. Keyboard And Display Interface

Table 3·1 Display Segment Control

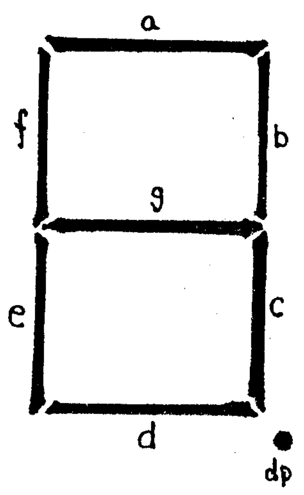	CPU DATA BUS	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	8279 OUTPUT	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0
	SEGMENT ENABLED	d	c	b	a	dp	g	f	e
	Bit = 1 Corresponding Segment is ON = 0 Corresponding Segment is OFF								

Table 3.3 Key and Its Corresponding Code

Serial No.	Key	Corresponding Code (HEXA)
1	0	00
2	1	01
3	2	02
4	3	03
5	4	04
6	5	05
7	6	06
8	7	07
9	8	08
10	9	09
11	A	0A
12	B	0B
13	C	0C
14	D	0D
15	E	0E
16	F	0F
17	RESET	10
18	NEXT	11
19	PRV	12
20	GO	13
21	EXEC	14
22	BLK MOV	15
23	EXAM MEM	16
24	EXAM REG	17

Table 3.1 Specification

CPU :-	8088 operated at 5 MHz
Memory:-	4KB FIRMWARE in 2732 4KB USER FREE RAM 6116 4KB EMPROM 2732
Peripherals:-	8279:- To control 24 keys keyboard and 6 digit 0.5" seven segment LED display. 8253:- Programmable interval timer 8255:- (2 numbers) These are available to user.
Power supply:-	5V (±0.1V), 1A

CHAPTER IV

PROGRAMMING EXAMPLES

Five programs illustrating typical 8088 instructions are given in this chapter.

4.1 Binary Multiplication:

The binary multiplication and division can be done directly in 8088 as they have built in function codes for these operations.

The following program performs 16 bit multiplication

Address	OPcode	Mnemonic
0400	BB 00 05	MOV BX,0500
0403	8B 07	MOV AX, (BX)
0405	43	INC BX
0406	43	INC BX
0407	F7 27	MUL AX, (BX)
0409	86 C4	XCHG AX, AL
040B	86 D6	XCHG DX, DL
040D	89 16 04 05	MOV 0504, DX
0411	A3 06 05	MOV 0506, AX
0414	F4	HLT

The 16 bit multiplier and multiplicand are to be entered from the memory address 0500.

The result will be in memory locatios 0504, 0505, 0506 and 0507.

Sample problem

(0500) = E8 (0501) = 03 (0502) = E8 (0503) = 03

Result

(0504) = 00 (0505) = 0F (0506) = 42 (0507) = 40

Replacing instruction MUL by DIV in the above program 16 bit division can be performed.

5.2 Transfer of One Page of FF Bytes:

This program transfers the contents of 05 page in RAM to 06 page.

Address		**OPcode**	**Mnemonic**
0400		BE 00 05	MOV SI, 0500
0403		BF 00 06	MOV DI, 0600
0406		B9 FF 00	MOV CX, 00FF
0409		FC	CLD
040A	pt P	A4	MOV DI, SI
040B		E2 FD	LOOP P
040D		F4	HLT

4.3 Unpacked BCD Numbers Division:

This program divides an unpacked BCD number with another unpacked BCD number.

Address	OPcode	Mnemonic
0400	B8 06 05	MOV AX, 0605
0403	B3 08	MOV BL, 08
0405	D5 0A	AAD
0407	F6 F3	DIV BL
0409	A3 00 05	MOV 0500, AX
040C	F4	HLT

Result wili be in 0500

(0500) = 07

4.4 Time Delay Routine:

This is subroutine when called will delay the execution.

Address		OPcode	Mnemonic
0400		50	PUSH AX
0401		9C	PUSH F
0402		B8 09 FF	MOV AX, FF09
0405	pt P	48	DEC AX
0406		75 FD	JNZ to pt A
0408		9D	POP F
0409		58	POP AX
040A		C3	RET

4.5 To Find the Average:

Address	OPcode	Mnemonic
0400	BE 00 05	MOV SI, 0500
0403	B9 0B 00	MOV CX, 0B00
0406	51	PUSH CX

```
0407    TOP    02 04          ADD AL, (SI)
0409           80 D4 00       ADC AH, 00
040C           46             INC SI
040D           E2 F8          LOOP to TOP
040F           59             POP CX
0410           F6 F1          DIV CL
0412           A3 50 05       MOV AX, 0550
0415           F4             HLT
```

The numbers to be averaged are stored in 0500 onwards.

Address

```
0500           01 02 03 04 05 06 07 08 09 0A 0B
```

The result 41H is stored in memory location 0550.

To use this kit for any application the relevent software program is to be written in EPROM with starting address anywhere between 8000 to 9FFF. The corresponding interfacing unit are to be interfaced.

CONCLUSION

An 8088 system controller has been designed and fabricated successfully. Number of programs have been tested on this kit.

The kit developed can be used as a trainer kit to practice 8086/8088 software programs. The program is to be loaded in RAM with starting address anywhere between 0000 to 1FFF and executed by pressing relevent keys as discussed in chapter III.

To use this kit for control applications the software is to be written in EPROM chip with starting address anywhere between 8000 to 9FFF. The EPROM is to be placed in the socket provided in the kit and corresponding interfacing unit is to be interfaced.

The advantage of 8088 is that there are more built in function in 8088 compared with any other 8 bit microprocessors. This reduces the number of software steps and hence speeds up the execution. The clock frequency of 8088 is also greater than clock frequency of any other 8 bit microprocessor. This reduces the time require to execute a program.

REFERENCES

1. DOUGLAS V.HALL "Microprocessor Interfacing Programming and Hardware", McGraw Hill 1986.

2. K.PADMANABHAN AND S.ANANTHI "An 8088 Microprocessor Board", Electronics for yout Jan. 1986. p. 50-66.

3. ALBERT PAUL MALVINO AND DONALD P. LEACH "Digital Principles and Applications", McGraw HIll 1985.

4. DOUGLAS V. HALL "Microprocessors and Digital Systems", McGraw Hill 1983.

5. ALBERT PAUL MALVINO "Digital Computer Electronics An Introduction to Microcomputers" Tata McGraw Hill 1983.

6. LANCE A LEVENTHAL " Introduction to Microprocessor Software, Hardware and Programming", Prentince Hall of India 1983.

7. ADITYA P MATHUR "Introduction to Microprocessors" Tata McGraw Hill 1984.

8. HARRY GARLAND " Introduction to Microprocessor System Design", McGraw Hill 1979.

9. LIU GIBSON "Microprocessor and Microcomputing The 8086, 8088 Family Architecture, Designing and Programming". PTH 1987.

APPENDIX A

MONITOR PROGRAM

```
Lable

            Initialization of 8279:-
            MOV DX, 0E19
            MOV AL, 00000000B
            OUT DX, AL
            MOV AL, 00111001B
            OUT DX, AL
            MOV AL, 11000000B
            OUT DX, AL
                         Display of message "8088 µP"
            MOV CH, 8088
            CALL ADDISP
            MOV AL, 90H
            MOV DX, 0E19H
            OUT DX, AL
            MOV DX, 0E18H
            MOV AL, E4
            OUT DX, AL
            MOV AL, EC
            OUT DX, AL
                       Examine memory and GO
START       MOV AL, 01000000B
            MOV DX, 0E19
            OUT DX, AL
            MOV DX, 0E18
```

```
           IN AL, DX
           CMP AL, 16H
           JZ, EXAM MEM
           CMP AL, 12H
           JZ, GO
           JMP START
                                Examine memory
           CALL ADDRESS
NEXT       MOV BX, AX
           MOV CL, (BX)
           MOV CH, CL
           MOV AL, CL
           MOV CL, 04H
           ROL CH, CL
           AND CH, 0FH
           MOV CL, AL
           AND CL, 0FH
           CALL DATADISP
LOOP       CALL KEY
           CALL NPCMP
           MOV CH, CL
           MOV CL, AL
           CALL DATADISP
           JMP LOOP1
                                Compare
COMPARE    CMP AL, 10H
           JS RETURN
           JMP ERROR
RETURN     RET
```

SUBROUTINES

```
NPCMP     CMP AL, 11H
          JZ NEXT1
          CMP AL, 15H
          JZ PREV1
          CMP AL, 12H
          JZ GO
          CALL COMPARE
          RET

NEXT1     MOV AL, CL
          MOV CL, 04
          ROL CH, CL
          ADD AL, CH
          MOV (BX), AL
          INC BX
          CALL ADDISP
          JMP NEXT

PREV1     MOV AL, CL
          MOV CL,04
          ROL CH, CL
          ADD AL, CH
          MOV (BX), AL
          DEC BX
          CALL ADDISP
          JMP NEXT
```

```
ADDRESS    MOV CX, 0000H
           CALL ADDISP
           CALL DATADISP
           CALL KEY
           CALL COMPARE
           ADD CL, AL
           CALL ADDISP
           MOV BH, CL
           CALL KEY
           CALL COMPARE
           MOV BL, CL
           MOV CL, 04
           ROL BL, CL
           MOV CL, BL
           AND CL, 0F
           ADD CL, AL
           CALL ADDISP
           CALL KEY
           CALL COMPARE
           MOV CH, BH
           MOV BL, CL
           MOV CL, 04
           ROL BL, CL
           MOV CL, BL
           ADD CL, AL
           CALL ADDISP
           RET
```

```
ERROR       MOV DX, 0E19H
            MOV AL, 93H
            OUT DX, AL
            MOV DX, 0E18H
            MOV AL, 97H
            OUT DX, AL
            MOV AL, 05H
            OUT DX, AL
            MOV AL, 05H
            OUT DX, AL
            HLT

                              GO and Execute
GO          CALL ADDRESS
            MOV BX, CX
            CALL KEY
            CMP 13
            JNZ ERROR
            MOV AL, 97H
            OUT DX, AL
            JMP (BX)

                             To Display address in address field
ADDISP      PUSH F
            PUSH DS
            PUSH AX
            PUSH BX
            PUSH CX
            PUSH DX
```

```
MOV DX, 0E19H
MOV AL, 92H
OUT DX, AL
MOV BX, SEVEN SEGMENT
MOV DX, 0E18H
MOV AL, CL
AND AL, 0FH
XLATB
OUT DX, AL
MOV AL, CL
MOV CL, 04H
ROL AL, CL
AND AL, 0FH
XLATB
OUT DX, AL
MOV AL, CH
AND AL, 0FH
XLATB
MOV CL, BH
MOV CL, 04
ROL AL, CL
AND AL, 0FH
XLATB
OUT DX, AL
POP DX
POP CX
POP BX
POP AX
POP DS
POP F
RET
```

```
                              Displays Data in Data Field
DATADISP   PUSH F
           PUSH DS
           PUSH AX
           PUSH BX
           PUSH CX
           PUSH DX
           MOV DX, 0E19H
           MOV AL, 90H
           OUT DX, AL
           MOV BX, SEVEN SEGMENT
           MOV DX, 0E18H
           MOV AL, CL
           AND AL, 0FH
           XLATB
           OUT DX, AL
           MOV AL, CH
           AND AL, 0FH
           XLATB
           OUT DX, AL
           POP DX
           POP CX
           POP BX
           POP AX
           POP DS
           POP F
           RET

DB: CF 06 AD 2F 66 6B EB 0E EF 6F EE E3 C9 A7 E9 E8
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

Instruction Set Summary

DATA TRANSFER

MOV Move

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to register	1 0 1 1 w reg	data	data if w 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr low	addr high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr low	addr high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

PUSH Push

Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment register	0 0 0 reg 1 1 0			

POP Pop

Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment register	0 0 0 reg 1 1 1			

XCHG Exchange

Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with accumulator	1 0 0 1 0 reg			

IN=Input from

Fixed port	1 1 1 0 0 1 0 w	port		
Variable port	1 1 1 0 1 1 0 w			

OUT - Output to

Fixed port	1 1 1 0 0 1 1 w	port		
Variable port	1 1 1 0 1 1 1 w			
XLAT Translate byte to AL	1 1 0 1 0 1 1 1			
LEA-Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES-Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF-Load AH with flags	1 0 0 1 1 1 1 1			
SAHF Store AH into flags	1 0 0 1 1 1 1 0			
PUSHF-Push flags	1 0 0 1 1 1 0 0			
POPF-Pop flags	1 0 0 1 1 1 0 1			

ARITHMETIC

ADD - Add

Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w 01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w 1	

ADC - Add with carry:

Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w-01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w 1	

INC - Increment:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA-ASCII adjust for add	0 0 1 1 0 1 1 1			
DAA-Decimal adjust for add	0 0 1 0 0 1 1 1			

SUB - Subtract:

Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w-01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w 1	

SBB - Subtract with borrow

Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w-01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w-1	

DEC Decrement

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		

CMP Compare

Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w 01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w 1	
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1			
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1			
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW Convert byte to word	1 0 0 1 1 0 0 0			
CWD Convert word to double word	1 0 0 1 1 0 0 1			

LOGIC

NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		

AND And

Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w 1	

TEST And function to flags, no result

Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w 1	

OR Or

Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w 1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w 1	

XOR Exclusive or

Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w 1	

STRING MANIPULATION

REP-Repeat	1 1 1 1 0 0 1 z
MOVS-Move byte/word	1 0 1 0 0 1 0 w
CMPS-Compare byte/word	1 0 1 0 0 1 1 w
SCAS-Scan byte/word	1 0 1 0 1 1 1 w
LODS-Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS-Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Instruction Set Summary (Continued)

CONTROL TRANSFER

CALL · Call:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	
JMP · Unconditional Jump:			
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within segment-short	1 1 1 0 1 0 1 1	disp	
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET · Return from CALL:			
Within segment	1 1 0 0 0 0 1 1		
Within seg. adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment, adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ · Jump on equal/zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE · Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG · Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE · Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA · Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp	
JP/JPE · Jump on parity/parity even	0 1 1 1 1 0 1 0	disp	
JO · Jump on overflow	0 1 1 1 0 0 0 0	disp	
JS · Jump on sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ · Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE · Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG · Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp	

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JNB/JAE Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp
JNBE/JA Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JNP/JPO Jump on not par/par odd	0 1 1 1 1 0 1 1	disp
JNO Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS Jump on not sign	0 1 1 1 1 0 0 1	disp
LOOP Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE Loop while not zero/equal	1 1 1 0 0 0 0 0	disp
JCXZ Jump on CX zero	1 1 1 0 0 0 1 1	disp
INT Interrupt		
Type specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INTO Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

CLC Clear carry	1 1 1 1 1 0 0 0	
CMC Complement carry	1 1 1 1 0 1 0 1	
STC Set carry	1 1 1 1 1 0 0 1	
CLD Clear direction	1 1 1 1 1 1 0 0	
STD Set direction	1 1 1 1 1 1 0 1	
CLI Clear interrupt	1 1 1 1 1 0 1 0	
STI Set interrupt	1 1 1 1 1 0 1 1	
HLT Halt	1 1 1 1 0 1 0 0	
WAIT Wait	1 0 0 1 1 0 1 1	
ESC Escape (to external device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK Bus lock prefix	1 1 1 1 0 0 0 0	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive,
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

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

if s:w = 01 then 16 bits of immediate data form the operand
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with Z.F FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)		8-Bit (w = 0)		Segment	
000	AX	000	AL	00	ES
001	CX	001	CL	01	CS
010	DX	010	DL	10	SS
011	BX	011	BL	11	DS
100	SP	100	AH		
101	BP	101	CH		
110	SI	110	DH		
111	DI	111	BH		

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X X X X (OF) (DF) (IF) (TF) (SF) (ZF) X (AF) X (PF) X (CF)

APPENDIX B

8086/8088 Instructions

Notes for 8086/8088 Instructions

The individual instruction descriptions are shown by a format box such as the following:

Opcode	m/op/r/m			Data	

These are byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- Opcode is the 8-bit opcode for the instruction. The actual opcode generated is defined in the "Opcode" column of the instruction table that follows each for mat box.
- m/op/r/m is the byte that specifies the operands of the instruction. It contains a 2-bit mode field (m), a 3-bit register field (op), and a 3-bit register or memory (r/m) field.
- Dashed blank boxes following the m/op/r/m box are for any displacement required by the mode field.
- Data is for a byte of immediate data.
- A dashed blank box following a Data box is used whenever the immediate operand is a word quantity.

Operand Summary

"reg" field Bit Assignments:

Word Operand	Byte Operand	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Second Instruction Byte Summary

mod xxx r/m

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(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.

Flags

AF: AUXILIARY CARRY — BCD
CF: CARRY FLAG
DF: DIRECTION FLAG (STRINGS)
IF: INTERRUPT ENABLE FLAG
OF: OVERFLOW FLAG (CF SF)
PF: PARITY FLAG
SF: SIGN FLAG
TF: TRAP (SINGLE STEP FLAG)
ZF: ZERO FLAG

Instructions that reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

15							8								0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

X = Don't Care

Segment Override Prefix

0 0 1 reg 1 1 0

Timing: 2 clocks

Use of Segment Override

Operand Register	Default	With Override Prefix
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr for strings)	DS	ES, SS, or CS
DI (implicit dest addr for strings)	ES	Never

Operand Address (EA) Timing (Clocks):

Add 4 clocks for word operands at ODD ADDRESSES.
Immed Offset = 6
Base (BX, BP, SI, DI) = 5
Base + DISP = 9
Base + Index (BP + DI, BX + SI) = 7
Base + Index (BP + SI, BX + DI) = 8
Base + Index (BP + DI, BX + SI) + DISP = 11
Base + Index (BP + SI, BX + DI) + DISP = 12

AAA = ASCII Adjust for Addition

Opcode

Opcode	Clocks	Operation
37	4	adjust AL, flags, AH

AAD = ASCII Adjust for Division

Long——Opcode

Opcode	Clocks	Operation
D5,0A	60	Adjust AL, AH prior to division

AAM = ASCII Adjust for Multiplication

Long——Opcode

Opcode	Clocks	Operation
D4,0A	83	Adjust AL, AH after multiplication

AAS = ASCII Adjust for Subtraction

Opcode

Opcode	Clocks	Operation
3F	4	adjust AL, flags, AH

ADC = Integer Add with Carry

Memory/Reg + Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	12	3	Reg8 ← CF + Reg 8 + Reg8
	12	9 + EA	Reg8 ← CF + Reg8 + Mem8
	10	16 + EA	Mem8 ← CF + Mem8 + Reg8
Word	13	3	Reg16 ← CF + Reg16 + Reg16
	13	9 + EA	Reg16 ← CF + Reg16 + Mem16
	11	16 + EA	Mem16 ← CF + Mem16 + Reg16

Immed to AX/AL

Opcode	Data	

	Opcode	Clocks	Operation
Byte	14	4	AL ← CF + AL + Immed8
Word	15	4	AX ← CF + AX + Immed16

Immed to Memory/Reg

Opcode	mod 010 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← CF + Reg8 + Immed8
	80	17 + EA	Mem8 ← CF + Mem8 + Immed8
Word	81	4	Reg16 ← CF + Reg16 + Immed16
	81	17 + EA	Mem16 ← CF + Mem16 + Immed16
	83	4	Reg16 ← CF + Reg16 + Immed8
	83	17 + EA	Mem16 ← CF + Mem16 + Immed8

ADD = Integer Addition

Memory/Reg + Reg

Opcode	mod reg r/m				

	Opcode	Clocks	Operation
Byte	02	3	Reg8 ← Reg8 + Reg8
	02	9 + EA	Reg8 ← Reg8 + Mem8
	00	16 + EA	Mem8 ← Mem8 + Reg8
Word	03	3	Reg16 ← Reg16 + Reg16
	03	9 + EA	Reg16 ← Reg16 + Mem16
	01	16 + EA	Mem16 ← Mem16 + Reg16

Immed to AX/AL

Opcode	Data	

Opcode	Clocks	Operation
04	4	AL ← AL + Immed8
05	4	AX ← AX + Immed16

Immed to Memory/Reg

Opcode	mod 000 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 + Immed8
	80	17 + EA	Mem8 ← Mem8 + Immed8
Word	81	4	Reg16 ← Reg16 + Immed16
	81	17 + EA	Mem16 ← Mem16 + Immed16
	83	4	Reg16 ← Reg16 + Immed8
	83	17 + EA	Mem16 ← Mem16 + Immed8

AND = Logical AND

Memory/Reg with Reg

Opcode	mod reg r/m				

	Opcode	Clocks	Operation
Byte	22	3	Reg8 ← Reg8 AND Reg8
	22	9 + EA	Reg8 ← Reg8 AND Mem8
	20	16 + EA	Mem8 ← Mem8 AND Reg8
Word	23	3	Reg16 ← Reg16 AND Reg16
	23	9 + EA	Reg16 ← Reg16 AND Mem16
	21	16 + EA	Mem16 ← Mem16 AND Reg16

Immed to AX/AL

Opcode	Data	

	Opcode	Clocks	Operation
Byte	24	4	AL ← AL AND Immed8
Word	25	4	AX ← AX AND Immed16

Immed to Memory/Reg

Opcode	mod 100 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 AND Immed8
	80	17 + EA	Mem8 ← Mem8 AND Immed8
Word	81	4	Reg16 ← Reg16 AND Immed16
	81	17 + EA	Mem16 ← Mem16 AND Immed16

CALL = Call

Within segment or group, IP relative

Opcode	DispL	DispH

Opcode	Clocks	Operation
E8	19	IP ← IP + Disp16 -- (SP) ← return link

Within segment or group, Indirect

Opcode	mod 010 r/m		

Opcode	Clocks	Operation
FF	16	IP ← Reg16--(SP) ← return link
FF	21 + EA	IP ← Mem16--(SP) ← return link
FF	21 + EA	IP ← Mem16--(SP) ← return link

Inter-segment or group, Direct

Opcode	offset	offset	segbase	segbase	segbase

Opcode	Clocks	Operation

9A	28	CS ← segbase
		IP ← offset

Inter-segment or group, Indirect

Opcode	mod 011 r/m		

Opcode	Clocks	Operation

FF	37 + EA	CS ← segbase
		IP ← offset

CBW = Convert Byte to Word

Opcode

Opcode	Clocks	Operation
98	2	convert byte in AL to word in AX

CLC = Clear Carry Flag

Opcode

Opcode	Clocks	Operation
F8	2	clear the carry flag

CLD = Clear Direction Flag

Opcode

Opcode	Clocks	Operation
FC	2	clear direction flag

CLI = Clear Interrupt Enable Flag

Opcode	Clocks	Operation
FA	2	clear interrupt flag

CMC = Complement Carry Flag

Opcode

Opcode	Clocks	Operation
F5	2	complement carry flag

CMP = Compare Two Operands

Memory/Reg with Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	38	3	flags ← Reg8 - Reg8
	38	9 + EA	flags ← Reg8 - Mem8
	3A	9 + EA	flags ← Mem8 - Reg8
Word	39	3	flags ← Reg16 - Reg16
	39	9 + EA	flags ← Reg16 - Mem16
	3B	9 + EA	flags ← Mem16 - Reg16

Immed to AX/AL

Opcode	Data	

	Opcode	Clocks	Operation
Byte	3C	4	flags AL - Immed8
Word	3D	4	flags AX - Immed16

Immed to Memory/Reg

Opcode	mod 111 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	flags ← Reg8 - Immed8
	80	10 + EA	flags ← Mem8 - Immed8
Word	81	4	flags ← Reg16 - Immed16
	81	10 + EA	flags ← Mem16 - Immed16
	83	4	flags ← Reg16 - Immed8
	83	10 + EA	flags ← Mem16 - Immed8

CWD = Convert Word to Doubleword

Opcode

Opcode	Clocks	Operation
99	5	convert word in AX to doubleword in DX:AX

DAA = Decimal Adjust for Addition

Opcode

Opcode	Clocks	Operation
27	4	adjust AL, flags, AH

DAS = Decimal Adjust for Subtraction

Opcode

Opcode	Clocks	Operation
2F	4	adjust AL, flags, AH

DEC = Decrement by 1

Word Register

Opcode + reg

Opcode	Clocks	Operation
48 + reg	2	Reg16 ← Reg16 - 1

Memory/Byte Register

Opcode	mod 001 r/m		

	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 - 1
	FE	15 + EA	Mem8 ← Mem8 - 1
Word	FF	15 + EA	Mem16 ← Mem16 - 1

DIV = Unsigned Division

Memory/Reg with AX or DX:AX

Opcode	mod 110 r/m		

	Opcode	Clocks	Operation
Byte	F6	80-90	AH,AL ← AX / Reg8
	F6	(86-96) + EA	AH,AL ← AX / Mem8
Word	F7	144-162	DX,AX ← DX:AX / Reg16
	F7	(150-168) + EA	DX,AX ← DX:AX / Mem16

ESC = Escape

Opcode + i	mod xxx r/m		

Opcode	Clocks	Operation
D8 + i	8 + EA	data bus ← (EA)
D8 + i	2	data bus ← (EA)

HLT = Halt

Opcode

Opcode	Clocks	Operation
F4	2	halt operation

IDIV = Signed Division

Memory/Reg with AX or DX:AX

Opcode	mod 111 r/m		

	Opcode	Clocks	Operation
Byte	F6	101-112	AH,AL ← AX / Reg8
	F6	(107-118) + EA	AH,AL ← AX / Mem8
Word	F7	165-184	DX,AX ← DX:AX / Reg16
	F7	(171-190) + EA	DX,AX ← DX:AX / Mem16

IMUL = Signed Multiplication

Memory/Reg with AL or AX

Opcode	mod 101 r/m		

	Opcode	Clocks	Operation
Byte	F6	80-98	AX ← AL*Reg8
	F6	(86-104) + EA	AX ← AL*Mem8
Word	F7	128-154	DX:AX ← AX*Reg16
	F7	(134-160) + EA	DX:AX ← AX*Mem16

IN = Input Byte, Word

Fixed port

Opcode	Port

	Opcode	Clocks	Operation
Byte	E4	10	AL ← Port8
	E5	10	AX ← Port8

Variable port

Opcode

	Opcode	Clocks	Operation
Word	EC	8	AL ← Port16(in DX)
	ED	8	AX ← Port16(in DX)

INC = Increment by 1

Word Register

Opcode + reg

Opcode	Clocks	Operation
40 + reg	2	Reg16 ← Reg16 + 1

Memory/Byte Register

Opcode	mod 000 r/m		

	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 + 1
	FE	15 + EA	Mem8 ← Mem8 + 1
Word	FF	15 + EA	Mem16 ← Mem16 + 1

INT / INTO = Interrupt

Opcode	type

Opcode	Clocks	Operation
CC	52	Interrupt 3
CD	51	Interrupt 'type'
CE	53 or 4	Interrupt4 if FLAGS OF = 1, else NOP

IRET = Return from Interrupt

Opcode

Opcode	Clocks	Operation
CF	24	Return from interrupt

Jcond = Jump on Condition

Operation

```
if condition is true then do,
  sign-extend displacement to 16 bits;
  IP ← IP + sign-extended displacement,
  end if;
```

Format

Opcode	Disp

Opcode	Clocks	Operation	cond
77	16 or 4	jump if above	JA
73	16 or 4	jump if above or equal	JAE
72	16 or 4	jump if below	JB
76	16 or 4	jump if below or equal	JBE
72	16 or 4	jump if carry set	JC
74	16 or 4	jump if equal	JE
7F	16 or 4	jump if greater	JG
7D	16 or 4	jump if greater or equal	JGE
7C	16 or 4	jump if less	JL
7E	16 or 4	jump if less or equal	JLE
76	16 or 4	jump if not above	JNA
72	16 or 4	jump if neither above nor equal	JNAE
73	16 or 4	jump if not below	JNB
77	16 or 4	jump if neither below nor equal	JNBE
73	16 or 4	jump if no carry	JNC
75	16 or 4	jump if not equal	JNE
7E	16 or 4	jump if not greater	JNG
7C	16 or 4	jump if neither greater nor equal	JNGE
7D	16 or 4	jump if not less	JNL
7F	16 or 4	jump if neither less nor equal	JNLE
71	16 or 4	jump if no overflow	JNO
7B	16 or 4	jump if no parity	JNP
79	16 or 4	jump if positive	JNS
75	16 or 4	jump if not zero	JNZ
70	16 or 4	jump if overflow	JO
7A	16 or 4	jump if parity	JP
7A	16 or 4	jump if parity even	JPE
7B	16 or 4	jump if parity odd	JPO
78	16 or 4	jump if sign	JS
74	18 or 6	jump if zero	JZ
E3	18 or 6	jump if CX is zero (does not test flags)	JCXZ

JMP = Jump

Within segment or group, IP relative

Opcode	DispL	DispH

Opcode	Clocks	Operation
E9	15	IP ← IP + Disp16
EB	15	IP ← IP + Disp8 (Disp8 sign-extended)

Within segment or group, Indirect

Opcode	mod 100 r/m		

Opcode	Clocks	Operation
FF	11	IP ← Reg16
FF	18 + EA	IP ← Mem16
FF	18 + EA	IP ← Mem16

Inter-segment or group, Direct

Opcode	offset	offset	segbase	segbase

Opcode	Clocks	Operation

EA	15	CS ← segbase
		IP ← offset

Inter-segment or group, Indirect

Opcode	mod 101 r/m		

Opcode	Clocks	Operation

FF	24 + EA	CS ← segbase
		IP ← offset

LAHF = Load AH from Flags

Opcode

Opcode	Clocks	Operation
9F	4	copy low byte of flags word to AH

LDS/LES = Load Pointer to DS/ES and Register

Opcode	mod reg r/m		

Opcode	Clocks	Operation
C4	16 + EA	dword pointer at EA goes to reg16 (1st word) and ES (2nd word)
C5	16 + EA	dword pointer at EA goes to reg16 (1st word) and DS (2nd word)

LEA = Load Effective Address

Opcode	mod reg r/m		

Opcode	Clocks	Operation
8D	2 + EA	Reg16 ← EA

LOCK = Assert Bus Lock

Opcode

Opcode	Clocks	Operation
F0	2	assert the bus lock next instruction

LOOP*xx* = Loop Control

Opcode	Disp

Opcode	Clocks	Operation	*xx* =
E1	18 or 6	dec CX; loop if equal and CX not 0	LOOPE
E0	19 or 5	dec CX; loop if not equal and CX not 0	LOOPNE
E1	18 or 6	dec CX; loop if zero and CX not 0	LOOPZ
E0	19 or 5	dec CX; loop if not zero and CX not 0	LOOPNZ
E2	17 or 5	dec CX; loop if CX not 0	LOOP

MOV = Move Data

Memory/Reg to or from Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	88	9 + EA	Mem8 ← Reg8
	88	2	Reg8 ← Reg8,
	8A	8 + EA	Reg8 ← Mem8
Word	89	9 + EA	Mem16 ← Reg16
	89	2	Reg16 ← Reg16
	8B	8 + EA	Reg16 ← Mem16

Direct-Addressed Memory to or from AX/AL

Opcode	AddrL	AddrH

	Opcode	Clocks	Operation
Byte	A0	10	AL ← Mem8
	A2	10	Mem8 ← AL
Word	A1	10	AX ← Mem16
	A3	10	Mem16 ← AX

Immed to Reg

Opcode	Data	

	Opcode	Clocks	Operation
Byte	B0 + reg	4	Reg 8 ← Immed8
Word	B8 + reg	4	Reg16 ← Immed16

Immed to Memory/Reg

Opcode	mod 000 r/m			Data	

Opcode	Clocks	Operation
C6	4	Reg8 ← Immed8
C6	10 + EA	Mem8 ← Immed8
C7	4	Reg16 ← Immed16
C7	10 + EA	Mem16 ← Immed16

Memory/Reg to or from SReg

Opcode	mod sreg r/m		

	Opcode	Clocks	Operation
Word	8C	9 + EA	Mem16 ← SReg
	8C	2	Reg16 ← SReg
	8E	8 + EA	SReg ← Mem16
	8E	2	SReg ← Reg16

MUL = Unsigned Multiplication

Memory/Reg with AL or AX

Opcode	mod 100 r/m		

	Opcode	Clocks	Operation
Byte	F6	70-77	AX ← AL·Reg8
	F6	(76-83) + EA	AX ← AL·Mem8
Word	F7	118-133	DX:AX ← AX·Reg16
	F7	(124-139) + EA	DX:AX ← AX·Mem16

NEG = Negate an Integer

Memory/Reg

Opcode | mod 011 r/m

Opcode	Clocks	Operation
F6	3	Reg8 ← 00H - Reg 8
F7	3	Reg16 ← 0000H - Reg16
F6	16 + EA	Mem8 ← 00H - Mem8
F7	16 + EA	Mem16 ← 0000H - Mem16

NOP = No Operation

Opcode

Opcode	Clocks	Operation
90	3	no operation

NOT = Form One's Complement

Memory/Reg

Opcode | mod 010 r/m

	Opcode	Clocks	Operation
Byte	F6	3	Reg8 ← 0FFH - Reg8
	F6	16 + EA	Mem8 ← 0FFH - Mem8
Word	F7	3	Reg16 ← 0FFFFH - Reg16
	F7	16 + EA	Mem16 ← 0FFFFH - Mem16

OR = Logical Inclusive OR

Memory/Reg with Reg

Opcode | mod reg r/m

	Opcode	Clocks	Operation
Byte	0A	3	Reg8 ← Reg8 OR Reg8
	0A	9 + EA	Reg8 ← Reg8 OR Mem8
	08	16 + EA	Mem8 ← Mem8 OR Reg8
Word	0B	3	Reg16 ← Reg16 OR Reg 16
	0B	9 + EA	Reg16 ← Reg16 OR Mem16
	09	16 + EA	Mem16 ← Mem16 OR Reg16

Immed to AX/AL

Opcode | Data

Opcode	Clocks	Operation
0C	4	AL ← AL OR Immed8
0D	4	AX ← AX OR Immed16

Immed to Memory/Reg

Opcode | mod 001 r/m | Data

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 OR Immed8
	80	17 + EA	Mem8 ← Mem8 OR Immed8
Word	81	4	Reg16 ← Reg16 OR Immed16
	81	17 + EA	Mem16 ← Mem16 OR Immed16

OUT = Output Byte, Word

Fixed port

Opcode | Port

	Opcode	Clocks	Operation
Byte	E6	10	Port8 ← AL
	E7	10	Port8 ← AX

Variable port

Opcode

	Opcode	Clocks	Operation
Word	EE	8	Port16 (in DX) ← AL
	EF	8	Port16 (in DX) ← AX

POP = Pop a Word from the Stack

Word Memory

Opcode | mod 000 r/m

Opcode	Clocks	Operation
8F	17 + EA	Mem16 ← (SP) + +

Word Register

Opcode + reg

Opcode	Clocks	Operation
58 + reg	8	Reg16 ← (SP) + +

Segment Register

Opcode + SReg

Opcode	Clocks	Operation
07 + SReg	8	SReg ← (SP) + +

POPF = Pop the TOS into the Flags

Opcode

Opcode	Clocks	Operation
9D	8	FLAGS ← (SP) + +

PUSH = Push a Word onto the Stack

Memory/Reg

Opcode | mod 110 r/m

Opcode	Clocks	Operation
FF	16 + EA	--(SP) ← Mem16

Word Register

Opcode + reg

Opcode	Clocks	Operation
50 + reg	11	--(SP) ← Reg16

Segment Register

Opcode + SReg

Opcode	Clocks	Operation
06 + SReg	10	--(SP) ← SReg

PUSHF = Push the Flags to the Stack

Opcode

Opcode	Clocks	Operation
9C	10	--(SP) ← FLAGS

RCL = Rotate Left Through Carry

Memory or Reg by 1

Opcode | mod 010 r/m

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg 8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg 16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 010 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

RCR = Rotate Right Through Carry

Memory or Reg by 1

Opcode | mod 011 r/m

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 011 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

REP*x* = Repeat Prefix

Opcode

Opcode	Clocks	Operation	REP*x* =
F3	2	repeat next instruction until CX = 0	REP
F3	2	repeat next instruction until CX = 0 or ZF ≠ 1	REPE REPZ
F2	2	repeat next instruction until CX = 0 or ZF ≠ 0	REPNE REPNZ

RET = Return from Subroutine

Opcode

	Opcode	Clocks	Operation
	C3	8	intra-segment return
	CB	18	inter-segment return

Return and add constant to SP

Opcode | DataL | DataH

	Opcode	Clocks	Operation
	C2	12	intra-segment ret and add
	CA	17	inter-segment ret and add

ROL = Rotate Left

Memory or Reg by 1

Opcode | mod 010 r/m

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 010 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + Ea + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

ROR = Rotate Right

Memory or Reg by 1

Opcode | mod 011 r/m

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 011 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

SAHF = Store AH in Flags

Opcode

	Opcode	Clocks	Operation
	9E	4	copy AH to low byte of flags word

SAL/SHL = Arithmetic/Logical Left Shift

Memory or Reg by 1

Opcode | mod 100 r/m

	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 100 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + EA + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

SAR = Arithmetic Right Shift

Memory or Reg by 1

Opcode | mod 111 r/m

	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 111 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + EA + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

SBB = Integer Subtraction with Borrow

Memory/Reg with Reg

Opcode | mod reg r/m

	Opcode	Clocks	Operation
Byte	1A	3	Reg8 ← Reg8 - Reg8 - CF
	1A	9 + EA	Reg8 ← Reg8 - Mem8 - CF
	18	16 + EA	Mem8 ← Mem8 - Reg8 - CF
Word	1B	3	Reg16 ← Reg16 - Reg16 - CF
	1B	9 + EA	Reg16 ← Reg16 - Mem16 - CF
	19	16 + EA	Mem16 ← Mem16 - Reg16 - CF

Immed from AX/AL

Opcode | Data

	Opcode	Clocks	Operation
	1C	4	AL ← AL - Immed8 - CF
	1D	4	AX ← AX - Immed16 - CF

Immed from Memory/Reg

Opcode | mod 011 r/m | | Data

	Opcode	Clocks	Operation
	80	4	Reg8 ← Reg8 - Immed8 - CF
	80	17 + EA	Mem8 ← Mem8 - Immed8 - CF
	81	4	Reg16 ← Reg16 - Immed16 - CF
	81	17 + EA	Mem16 ← Mem16 - Immed16 - CF
	83	4	Reg16 ← Reg16 - Immed8 - CF
	83	17 + EA	Mem16 ← Mem16 - Immed8 - CF (Immed8 is sign-extended before subtract)

SHR = Logical Right Shift

Memory or Reg by 1

Opcode | mod 101 r/m

	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL

Opcode | mod 101 r/m

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + Ea + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

STC = Set Carry Flag

Opcode

	Opcode	Clocks	Operation
	F9	2	set the carry flag

STD = Set Direction Flags

Opcode

	Opcode	Clocks	Operation
	FD	2	set direction flag

STI = Set Interrupt Enable Flag

Opcode

	Opcode	Clocks	Operation
	FB	2	set interrupt flag

String = String Operations

Opcode

Opcode	Clocks	Operation
A6	22	flags ← (SI) - (DI)

String CMPS

A7	22	flags ← (SI) · (DI)	CMPS
A4	18	(DI) ← (SI)	MOVS
A5	18	(DI) ← (SI)	MOVS
AE	15	flags ← (DI) · AL	SCAS
AF	15	flags ← (DI) · AX	SCAS
AC	12	AL ← (SI)	LODS
AD	12	AX ← (SI)	LODS
AA	11	(DI) ← AL	STOS
AB	11	(DI) ← AX	STOS

SUB = Integer Subtraction

Memory/Reg with Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	2A	3	Reg8 Reg8 · Reg8
	2A	9 + EA	Reg8 Reg8 · Mem8
	28	16 + EA	Mem8 Mem8 · Reg8
Word	2B	3	Reg16 Reg16 - Reg16
	2B	9 + EA	Reg16 Reg16 - Mem16
	29	16 + EA	Mem16 Mem16 - Reg16

Immed to AX/AL

Opcode	Data	

	Opcode	Clocks	Operation
Byte	2C	4	AL ← AL - Immed8
Word	2D	4	AX ← AX - Immed16

Immed to Memory/Reg

Opcode	mod 101 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 - Immed8
	80	17 + EA	Mem8 ← Mem8 - Immed8
Word	81	4	Reg16 ← Reg16 - Immed16
	81	17 + EA	Mem16 ← Mem16 - Immed16
	83	4	Reg16 ← Reg16 - Immed8
	83	17 + EA	Mem16 ← Mem16 - Immed8

TEST = Logical Compare

Memory/Reg with Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	84	3	flags ← Reg8 AND Reg8
	84	9 + EA	flags ← Reg8 AND Mem8
Word	85	3	flags ← Reg16 AND Reg16
	85	9 + EA	flags ← Reg16 AND Mem16

Immed to AX/AL

Opcode	Data	

	Opcode	Clocks	Operation
Byte	A8	4	flags ← AL AND Immed8
Word	A9	4	flags ← AX AND Immed16

Immed to Memory/Reg

Opcode	mod 000 r/m			Data	

	Opcode	Clocks	Operation
Byte	F6	5	flags ← Reg8 AND Immed8
	F6	11 + EA	flags ← Mem8 AND Immed8
Word	F7	5	flags ← Reg16 AND Immed16
	F7	11 + EA	flags ← Mem16 AND Immed16

WAIT = Wait While TEST Pin Not Asserted

Opcode

Opcode	Clocks	Operation
9B	3 + 5n	none

XCHG = Exchange Memory/Register with Register

Memory/Reg with Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	86	4	Reg8 ↔ Reg8
	86	17 + EA	Mem8 ↔ Mem8
Word	87	4	Reg16 ↔ Reg16
	87	17 + EA	Mem16 ↔ Mem16

Word Register with AX

Opcode + Reg

Opcode	Clocks	Operation
90 + Reg	3	AX ↔ Reg16

XLAT XLATB = Table Look-up Translation

Opcode

Opcode	Clocks	Operation
D7	11	replace AL with table entry

XOR = Logical Exclusive OR

Memory/Reg with Reg

Opcode	mod reg r/m		

	Opcode	Clocks	Operation
Byte	32	3	Reg8 ← Reg8 XOR Reg8
	32	9 + EA	Reg8 ← Reg8 XOR Mem8
	30	16 + EA	Mem8 ← Mem8 XOR Reg8
Word	33	3	Reg16 ← Reg16 XOR Reg16
	33	9 + EA	Reg16 ← Reg16 XOR Mem16
	31	16 + EA	Mem16 ← Mem16 XOR Reg16

Immed to AX/AL

Opcode	Data	

Opcode	Clocks	Operation
34	4	AL ← AL XOR Immed8
35	4	AX ← AX XOR Immed16

Immed to Memory/Reg

Opcode	mod 110 r/m			Data	

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 XOR Immed8
	80	17 + EA	Mem8 ← Mem8 XOR Immed8
Word	81	4	Reg16 ← Reg16 XOR Immed16
	81	17 + EA	Mem16 ← Mem16 XOR Immed16

Appendix B

MOD R/M	00	01	10	11	
				W = 0	W = 1
000	(BX) + (SI)	(BX) + (SI) + d8	(BX) + (SI) + d16	AL	AX
001	(BX) + (DI)	(BX) + (DI) + d8	(BX) + (DI) + d16	CL	CX
010	(BP) + (SI)	(BP) + (SI) + d8	(BP) + (SI) + d16	DL	DX
011	(BP) + (DI)	(BP) + (DI) + d8	(BP) + (DI) + d16	BL	BX
100	(SI)	(SI) + d8	(SI) + d16	AH	SP
101	(DI)	(DI) + d8	(DI) + d16	CH	BP
110	d16 (direct address)	(BP) + d8	(BP) + d16	DH	SI
111	(BX)	(BX) + d8	(BX) + d16	BH	DI

MEMORY MODE REGISTER MODE

d8 = 8-bit displacement d16 = 16-bit displacement