Processor Dissection Report

Group 9

140241C R.Jathushan
140246V L.D.S.A.Jayasekara
140247B M.H.N.H.P.Jayasekara
140257F M.H.V.Y.Jayasundara

This is submitted as a partial fulfillment for the module
EN2080: Fundamentals of Computer Organization and Design
Department of Electronic and Telecommunication Engineering
University Of Moratuwa

May 11, 2016

Contents

List of Figures

1 Abstract

This report is a study of the Intel 8086 16-BIT HMOS MICROPROCESSOR. It gave rise to the x86 architecture, which eventually became Intel's most successful line of processors. The report contains Details of this microprocessor in the following aspects,

- Set Architecture of the Processor
- Focus on Instruction Encoding
- Micro-Architecture (Data Path and the Controller)
- ALU
- Timing and Memory Interfacing

2 Introduction

between early 1976 and mid-1978 this 16-bit processor is produced by the Intel. At that time intel is a memory manufacturing company. The 8086 gave rise to the x86 architecture which eventually became Intel's most successful line of processors. principal architect of intel 8086 is Stephen P. Morse.

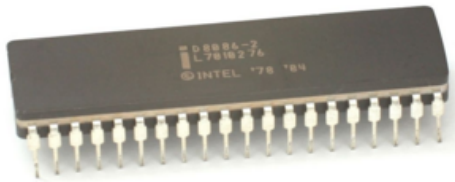

Figure 1: Intel 8086 Chip

8086 was designed to allow assembly language for the 8008, 8080, or 8085 to be automatically converted into equivalent (suboptimal) 8086 source code. However, the 8086 design was expanded to support full 16-bit processing, instead of the fairly basic 16-bit capabilities of the 8080/8085. New kinds of instructions were added as well; full support for signed integers, base+offset addressing, and self-repeating operations All internal registers, as well as internal and external data buses, are 16 bits wide. It gave better support stack-based high-level programming languages. It instructions can be 1 to 6 bytes in length.

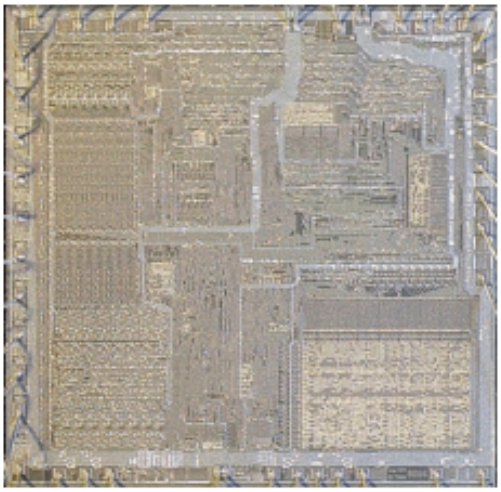

Figure 2: Intel 8086 Chip die

3 Instruction Set of the Intel 8086

An instruction is a binary pattern designed inside a microprocessor to perform a specific task. The set of all instructions that a microprocessor supports called instruction set. Intel 8086 has more than 20000 instructions.

Classification of ISA. according to the operation performed by the Instruction we can divide them into groups.

- Data transfer instruction
- Arithmetic instruction
- Bit manipulation instruction
- Program execution transfer instruction
- String instruction
- Processor control instruction

3.1 Data transfer instruction

These instructions are used to transfer data from source to destination. These instructions can have constants, memory locations, registers or I/O port address as operands.

MOV des,src

Src can be immediate, register or memory location while Des can be memory or register but both cannot be memory at same time.

Eg: MOV AX, 0A52 H
MOV AL, CL
MOV AX, [02C4 H]

PUSH Operand

It's pushes the src to the top of the stack.
Eg: PUSH AX

POP Des

It pops the operand form the top of the stack and put it in the destination. General purpose registers or segment registers or memory locations can be used as Des.
Eg: POP AX

XCHG Des, Src

Its interfaces the source and the destination. Two memory locations can be interchanged.
Algorithm: operand1 $< - >$ operand2
Example:

```
MOV AL, 5
MOV AH, 2
XCHG AL, AH ; AL = 2, AH = 5
XCHG AL, AH ; AL = 5, AH = 2
RET
```

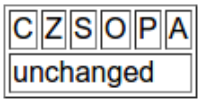

Figure 3: Flag Register

IN Accumulator, PortAddress

Its moves the operand for the I/O port address to the accumulator.
Eg: IN Ax, 012A H

OUT Port Address, Accumulator

It transfers accumulator value to the port with the given address.
Eg: OUT 124D H, AX

LEA Reg, Src

It loads the register with an offset given by the src.
Eg: LEA BX, (DI)

LDS Des, Src

This instruction loads 32-bit pointer to the destination register and Ds. the offset is placed in the destination register and the segment is placed in Ds.
Eg: LDS BX, [152A H]

LES Des, Src

This instruction also loads 32-bit pointer from memory to destination register and Es. the offset is placed in the destination register and the segment is placed is placed in Es. this is nearly same as LDS but it uses Es instead of Ds.
Eg: LES BX, [4A21 H]

LAHF - it copies a lower byte of flag register to AH
SAHF - it copies the values of AH to the lower byte of the flag register.
PUSHF - pushes the flag register to the top of the stack
POPF - pops out the top of the stack to the flag register.
There are many other instructions for the data transfer.

3.2 Arithmetic instructions

Arithmetic operations are used to do math on operands. these are the important instructions on a processor. they define the quality of a processor. for example supercomputers are defined by how many additions or floating point operation they can do in a second.

ADD Des, Src

It's adds a byte to byte or word to word. It can change the AF, CF, OF, PF, SF, ZF flags.
Addressing modes
REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate
Algorithm:
operand1 = operand1 + operand2
Example:
MOV AL, 5 ; AL = 5
ADD AL, -3 ; AL = 2
RET

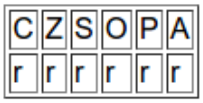

Figure 4: Flag Register

Eg: ADD AL, 74 H
ADD AX, BX
ADD AX, [BX]

ADC Dst, Src

Adds two operands with CF. as ADD it can changes the states of the flag register.
Eg: ADC AX, 41 H

ADC AX, DX
ADC AX, [BX]

SUB Des, Src

This instruction is used to subtract byte for byte or word form word. It can change the AF, CF, OF, PF, SF, ZF flags. For subtraction operation Carry Flag(CF) acts as a Borrow flag.

Eg: SUB AX, 41 H
SUB AX, DX
SUB AX, [BX]

SBB Des, Src

It subtracts Src and Barrow(CF) form the Des. It can change the AF, CF, OF, PF, SF, ZF flags.

Eg: SBB AX, 41 H
SBB AX, DX
SBB AX, [BX]

INC Src

It is a increment function, it increases src (byte or word) by one. It affects AF, OF, PF, SF, ZF flags. CF is not affected.

Algorithm:

operand = operand + 1
Example:
MOV AL, 4
INC AL ; AL = 5
RET

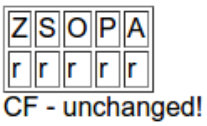

Figure 5: Flag Register

DEC Src

It is a decrement function, it decreases src (byte or word) by one. It affects AF, OF, PF, SF, ZF flags. CF is not affected.

Eg: DEC AX

AAA

This adjusts the results to ASCII format because data entered in the terminal is in ASCII format. 0- 9 is represented in 30 H - 39 H. using this instruction we can add ASCII codes and this instruction does not have any operands.Similar to AAA there are

AAS -ASCII adjust after subtraction
AAM- ASCII adjust after multiplication
AAD - ASCII adjust after division

DAA

It corrects decimal after addition of two BCD numbers. it works only on AL register. It also doesn't have any operands. Similarly, DAS - Decimal adjust after subtraction
Algorithm:
if low nibble of AL < 9 or AF $= 1$ then:
AL = AL + 6
AF = 1
if AL $>$ 9Fh or CF $= 1$ then:
AL = AL + 60h
CF = 1

```
Example:    MOV AL, 0Fh ; AL = 0Fh (15)
            DAA ; AL = 15h
            RET
```

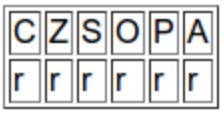

Figure 6: Flag Register

NEG Src

It creates a 2s compliment of a number. It means it creates a negative of a given number.

3.3 Bit manipulation instructions

These are the instruction which used in the instruction level. These instruction are used to shift the bits, zero test for a bit and set or reset a bit.

NOT Src

It creates a 1s compliment of the operand. It flips the each bits to the its compliment. Operand can be a register or a memory.

AND Des, Src

This instruction performs AND operation over Des and Src. immediate number or register or memory location can be used as src. Des should be memory location or register. Both can be memory location at the same time. CF and OF will become zero after this operation. PF, SF, ZF will be updated. Result is stored in operand1. Available addressing modes

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

Example:

```
MOV AL, 'a' ; AL = 01100001b
AND AL, 11011111b ; AL = 01000001b ('A')
RET
```

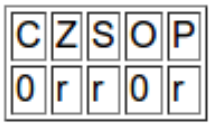

Figure 7: Flag Register

Eg: AND AX, BX
AND AX, [152E H]
AND AX, 41 H

OR Des, Src

This instruction performs OR operation over Des and Src. immediate number or register or memory location can be used as src. Des should be memory location or register. Both can be memory location at the same time. CF and OF will become zero after this operation. PF, SF, ZF will be updated.

Eg: OR AX, BX
OR AX, [152E H]
OR AX, 41 H

XOR Des, Src

This instruction performs XOR operation over Des and Src. immediate number or register or memory location can be used as src. Des should be memory location or register. Both can be memory location at the same time. CF and OF will become zero after this operation. PF, SF, ZF will be updated.

Eg: XOR AX, BX
XOR AX, [152E H]
XOR AX, 41 H

SHL Des, Count

This instruction is used to shift the bits of the byte or word to left by count. It changes the LSB to zero. MSB is given to carry flag. If count is one it can be put as immediate but when count is high it should be put in the CL register.

Eg: SHL AX,1
SHL AX, CL

SHR Des, Count

This instruction is used to shift the bits of the byte or word to right by count. It changes the MSB to zero. LSB is given to carry flag. If count is one it can be put as immediate but when count is high it should be put in the CL register.

Eg: SHR AX,1
SHR AX, CL

ROL Des, Count

It rotates the des in left direction by the count. MSB is given to LSB and to CF also. If count is one it can be put as immediate but when count is high it should be put in the CL register.

Eg: ROL AX,1
ROL AX, CL

ROR Des, Count

It rotates the des in left direction by the count. LSB is given to MSB and to CF also. If count is one it can be put as immediate but when count is high it should be put in the CL register.

Eg: ROR AX,1
ROR AX, CL

3.4 Program execution transfer instruction

These instructions changes the sequence of the instruction execution. They can be conditional or sometimes unconditional. Conditions are given form flag registers. CALL Des This instruction is used to call the functions or subroutines in a program. Address of the next instruction is pushed into the stack after the call instruction.

RET

Its returns the control form the subroutine to the main program. Every CALL instruction should have a RET instruction.

JMP

It used for the unconditional jumps from one place to another instruction.

Jxx Des

This is a conditional jump. Some conditions are from flag registers.

Conditional Jump Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

Figure 8: Jump conditions

LOOP Des

This is a loop instruction. Number of loops to be run is placed in the CX register. When each iteration is running the CX will be decreased by one. ZF is check for to run the loop again or not.

3.5 String instructions

In assembly language string is stored in sequentially as bytes or words. X86 ISA has a strong set for instructions for Strings. These string instructions are used to reduce the size of the program.

CMPS Des, Src

Compares the Des and the Src bytes or words.
Addressing modes:

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

Algorithm:

operand1 - operand2

result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

Example:
MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (so equal!)
RET

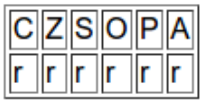

Figure 9: Flag Register

SCAS String

It scans the string. It compares the string byte with the AL or the string word with AX.

MOVS / MOVSB / MOVSW

This helps to move a byte or a word form one string to another. For this instruction source is should be in data segment and destination string is in the extra segment. While SI and DI stores the offset values of the source and the destination index.

REP

This is an instruction prefix. It runs until CX became zero.
Eg: REP MOVSB Str1, Str2

It copies byte by byte contents until CX became zero.

3.6 Processor control instruction

Intel 8086 allows some flags to control the processor itself. It can be used to run the processor in one direction and processor synconation if more than one microprocessor is attached.

- HLT - Halt the System.
  Example:
  MOV AX, 5
  HLT

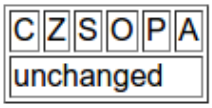

Figure 10: Flag Register

- STC - it sets the carry flag to 1
- CLC - it clears the carry flag to 0
- CMC - it complements the carry flag
- STD - it sets the direction flag to one, if its is one the string bytes are accessed from high memory location to lower memory locations
- CLD - it clears the direction flag to 0. The the memory is accessed from lower memory locations to higher memory locations
- CLD - it clears the direction flag to 0. The the memory is accessed from lower memory locations to higher memory locations
- CLD - it clears the direction flag to 0. The the memory is accessed from lower memory locations to higher memory locations
- CLI - Clear interrupt flag
- STI - Set interrupt flag
- WAIT - causes the processor to wait for the coprocessor.
- NOP - no operation. When this instruction is executed the it won't allow any other operation to operate except IP increment by one.

4 Instruction Encoding

8086 instructions are encoded as binary numbers. Instructions length vary from 1 to 6 bytes.

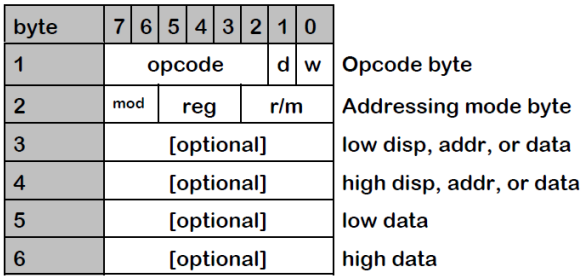

Figure 11: Instruction format

First and second bytes are divided into 6 fields.

Opcode
d direction
w word/byte
mod mode
reg register
r/m register/memory

Instruction may also be optionally preceded by one or more prefix bytes. Some instructions are one-byte instructions and lack the addressing mode byte. 16-bit values are stored in little-endian order.

$[Prefix]Opcode[Addrmode][Lowdisp][Highdisp][Lowdata][Highdata]$

4.1 Prefix Bytes

There are four types of prefix instructions:

Repetition
Segment Overrides
Lock
Address/Operand size overrides (for 32-bit machines)

Repetition

REP
REPE
REPZ F3H
REPNE
REPNZ F2H

REP and REPE and not distinct.Machine (microcode) interpretation of REP and REPE code depends on instruction currently being executed

Segment override

CS 2EH
DS 3EH
ES 26H
SS 36H

Lock - F0H

4.2 The Opcode Byte

The opcode field specifies the operation performed (mov, xchg, etc). Usually 6 bits.The d (direction) field specifies the direction of data movement.

if d = 1 destination is operand specified by REG field.
if d = 0 destination is operand specified by R/M field.

The d position MAY be interpreted as the "s" bit.

s = 1 means one byte of immediate data is present which must be sign-extended to produce a 16-bit operand.
s = 0 means two bytes of immediate are present.

The d position is interpreted as the "c" bit in Shift and Rotate instructions.

if C= 1, CL is used for shift count.
if C=0, shift/Rotate by 1 or by immediate value.

The w (word/byte) bit specifies operand size

W = 1 data is word
W = 0 data is byte

4.3 Addressing Mode Byte

Addressing mode byte contains three fields that specify operands.

Mod Bits 6-7 (mode; determines how R/M field is interpreted)
Reg Bits 3-5 (register) or SREG (Seg register)
R/M Bits 0-2 (register/memory)

```
MOD Field (determines how R/M operand is interpreted)
00      Use R/M Table 1 for R/M operand
01      Use R/M Table 2 with 8-bit signed displacement
10      Use R/M Table 2 with 16-bit unsigned displacement
11      Use REG table for R/M operand
REG Field                                   SegREG
   w=0  w=1                 w=0   w=1
000     AL AX          100   AH    SP          000   ES
001     CL CX          101   CH    BP          001   CS
010     DL DX          110   DH    SI          010   SS
011     BL BX          111   BH    DI          011   DS
R/M Table 1 (Mod = 00)
000     [BX+SI]  010   [BP+SI]          100   [SI]        110   Direct Addr
001     [BX+DI]  011   [BP+DI]          101   [DI]        111   [BX]
R/M Table 2 (Mod = 01 or 10)
000     [BX+SI+Disp]                                     101   [DI+Disp]
001     [BX+DI+Disp]  011   [BP+DI+Disp]                110   [BP+Disp]
010     [BP+SI+Disp]  100   [SI+Disp]                   111   [BX+Disp]
Direction Bit:  0 means data moves from REG operand to R/M operand
                1 means data moves from R/M operand to REG operand
(For some instructions with immediate operands, S-bit in place of D bit
means if s=1 data is sign extend 8-bit data for word operation)
Word Bit:       1 = word operands, 0 = byte operands
```

Figure 12: Addressing modes

If instruction has zero explicit operands, it is not present in general.

- For one-operand instructions the R/M field indicates where the operand is to be found
- For two-operand instructions (except those with an immediate operand) one is a register determined by REG (SREG) field and the other may be register or memory and is determined by R/M field.

The Direction bit has meaning only in two-operand instructions.Indicates whether "destination" is specified by REG or by R/M.This allows many instructions to be encoded in two different ways Swap R/M and REG operands and flip d bit

4.4 Addressing Mode 00

Specifies R/M Table 1 with NO displacement.

000 [BX+SI]	010[BP+SI]	100 [SI]	110 Drc't Add
001 [BX+DI]	011[BP+DI]	101 [DI]	111 [BX]

The 110 case (direct addressing) requires that the instruction be followed by two address bytes. There are then two possibilities:

Opcode Addressing Mode

Opcode Addressing Mode Offset-Low Offset-High

4.5 Addressing Mode 01

Specifies R/M Table 2 with 8-bit signed displacement.
R/M Table 2 (Mod = 01 or 10)

Add DISP to register specified

000 [BX+SI]	010[BP+SI]	100 [SI]	110 [BP]
001 [BX+DI]	011[BP+DI]	101 [DI]	111 [BX]

All instructions have the form

Opcode Addressing Mode Displacement

4.6 Addressing Mode 10

Specifies R/M Table 2 with 16-bit unsigned displacement.
R/M Table 2 (Mod = 01 or 10)
Add DISP to register specified

000 [BX+SI]	010[BP+SI]	100 [SI]	110 [BP]
001 [BX+DI]	011[BP+DI]	101 [DI]	111 [BX]

All instructions have the form

Opcode Addressing Mode Displacement-Low Displacement-High

4.7 Addressing Mode 11

Specifies that R/M bits refer to REG table.All two operand register-to-register instructions use addressing mode 11.

```
MOV AX,BX
MOV DX,CX
MOV AH,BL
```

Addressing Mode 11 is also used by immediate mode instructions where dest is a register

```
ADD BX,1
ADC CH,0
OR SI, 0F0F H
```

Selected Instruction Formats

Instruction	Opcode	Addr.Mode
ADC reg/mem with reg	000100dw	modregr/m [addr]
ADC immed to reg/mem	100000sw	mod010r/m data
ADD reg/mem with reg	000000dw	modregr/m [addr]
ADD immed to accumulator	0000010w	data
ADD immed to reg/mem	100000sw	mod000r/m [addr] data
OR reg/mem with reg	000010dw	modregr/m
OR immed to reg/mem	100000sw	mod001r/m [addr] data
OR immed to accumlator	0000110w	data
INC reg16	01000reg	
INC reg/mem	1111111w	mod000r/m [addr]
MOV reg/mem to/from reg	100010dw	modregr/m [addr]
MOV reg/mem to segreg	10001110	modsegr/m (seg = segreg)
MOV immed to reg/mem	1100011w	mod000r/m [addr] data
MOV immed to reg	1011wreg	data
MOV direct mem to/from acc	101000dw	addr
XCHG reg/mem with reg	1000011w	modregr/m [addr]
XCHG reg16 with accum.	10010reg	
CMP reg/mem with reg	001110dw	modregr/m [addr]
CMP immed to accumulator	0011110w	data
CMP immed to reg/mem	100000sw	mod111r/m [addr] data
POP reg	01011reg	
POP segreg	00reg111	
POP reg/mem	10001111	modxxxr/m (xxx = don't care)
RCL reg/mem,CL/immediate	110100cw	mod010r/m [addr] (if c=0 shift= 1,
RCR reg/mem,CL/immediate	110100cw	mod011r/m [addr] if c=1 shift = CL)
STOS	1010101w	
CMPS	1010011w	
MUL reg/mem	1111011w	mod100r/m [addr]

Figure 13: Instruction formats

5 Arithmetic and Logical Unit - ALU

ALU The arithmetic/logic unit (ALU) can perform various arithmetic and logical operation, if required, based on the instruction to be executed. A 16-bit ALU is used in the Execution Unit of Intel 8086. All registers and data paths in the EU are 16 bits wide for fast internal transfers.

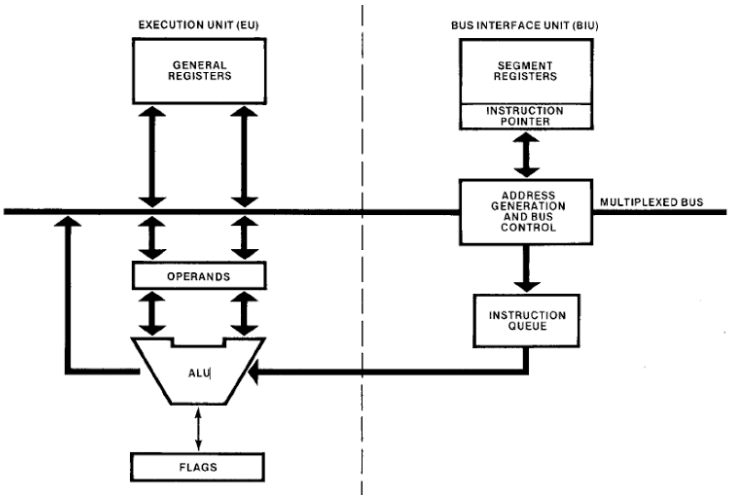

Figure 14: ALU

6 Microarchitecture

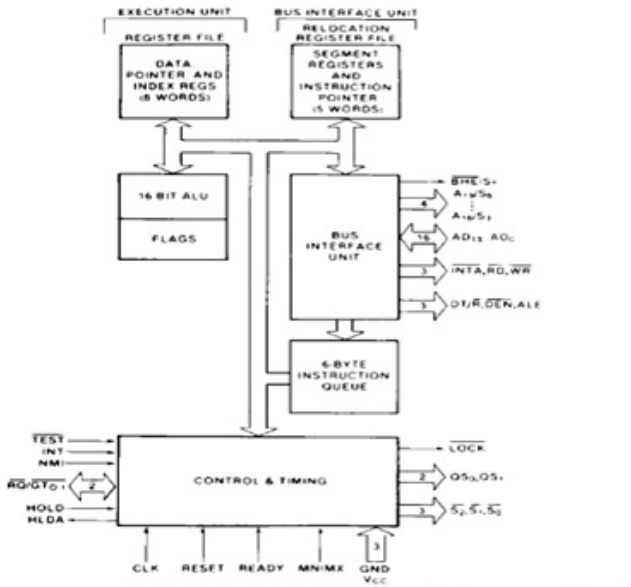

Figure 1. 8086 CPU Block Diagram

Figure 15: CPU block diagram

8086 microprocessor is logically divides into two segments.

- Bus Interface Unit(BIU)
- Execution Unit(EU)

These units can interact directly but for the most part perform as separate asynchronous operational processors.

6.1 Bus Interface Unit

Functions related to instruction fetching and queuing, operand fetch and store, and address relocation are provided by this unit. Basic bus control is also provided by BIU. This unit increase processor performance by improving bus bandwidth utilization via overlapping of instruction pre-fetching. Since this architecture did not use modern pipeline architecture, BIU use 6 byte First-In-First-Out (FIFO) buffer to queue the instruction stream, waiting for decoding and execution. This will enables some sort of pipeline to the system. Buffer will be allowed the BIU to keep the memory utilized very efficiently and instruction bytes will be extracted by EU. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. If the queue is empty the first byte into the queue immediately becomes available to the EU.

6.2 Execution Unit

The execution unit receives pre-fetched instructions from the BIU queue. And it provides un-relocated operand addresses to the BIU by decoding the instructions. Memory operands are passed through the BIU for processing by the 16-bit ALU in the EU, which passes results to the BIU for storage.

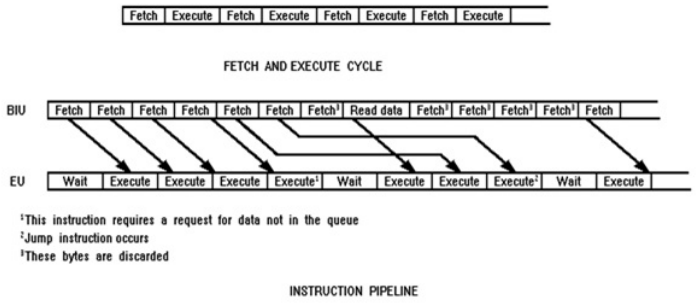

Figure 16: Instruction Pipeline

Three conditions can cause EU to enter the wait mode.

- When an instruction requires access to a memory location not in the queue.
- When the instruction to be executed is a "jump" instruction.
- When the instructions which are slow to execute is executing, it will cause BIU to suspend fetching instructions

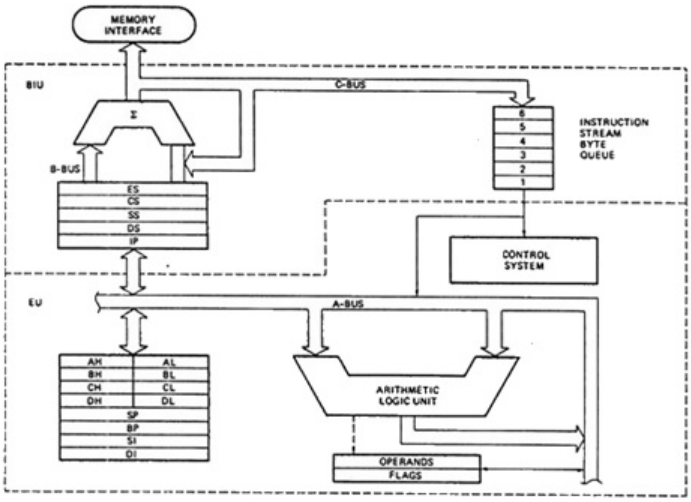

Figure 17: Pipeline of the processor

6.3 Registers

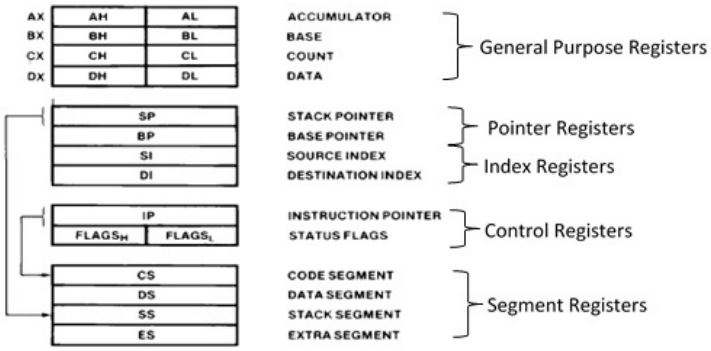

Figure 18: Registers

- Ax register stores operand for arithmetic operations.
- Bx register holds the starting base location of a memory region within a data segment.
- Cx register uses in loop instruction to store loop counter.
- Dx register uses to contain I/O port address for I/O instruction.
- SP register contains 16-bit offset from the start of the segment to the top of the stack.
- BP, SI and DI registers store data and acted as memory pointers.
- IP register holds the 16-bit address of the next code byte.
- Flag register has two type of registers,

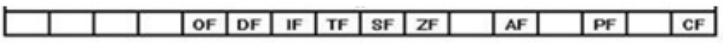

Figure 19: Flag Register

- Conditional Flags - These flags are status indicators or they store the results of last arithmetic and logic instructions.
  Carry Flag(CF)
  Auxiliary Flag(AF)
  Parity Flag(PF)
  Zero Flag(ZF)
  Sign Flag(SF)
  Overflow Flag(OF)

- Control Flags - These flags can be controlled by the user.
  Trap Flag(TF)
  Interrupt Flag(IF) Direction Flag(DF)

Segment registers are used to hold the upper 16 bits of the starting address for each of the segments of the memory. The part of a segment starting address stored in a segment register is often called the segment base.

- CS register uses for addressing a memory location in the Code Segment of the memory, where the executable program is stored.
- DS register contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.
- SS register defined a section of memory to store addresses and data while a subprogram executes.
- ES register is additional data segment that is used by some of the string to hold the extra destination data.

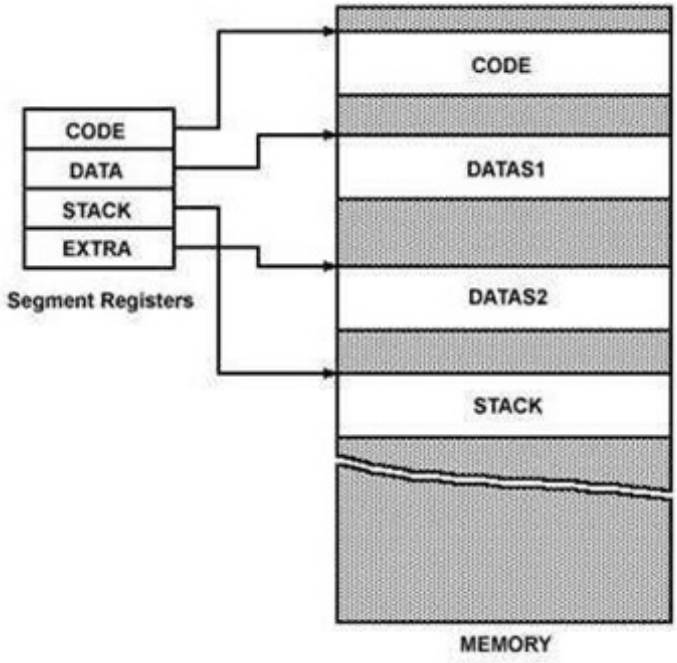

Figure 20: Segment Registers

6.4 Data path and controllers

6.4.1 Min and Max Modes

MIN and MAX modes are controlled by the MN/MX pin. If this pin strapped to ground, it enables the MAX mode. Otherwise it enables the MIN mode. According to the mode selected, pin configuration from 24 to 31 changed as shown in figure. In MIN mode all the control signals for memory and O/I are generated internally. But for Max mode can be used only when coprocessor is present in the system. This requires an external bus controller.

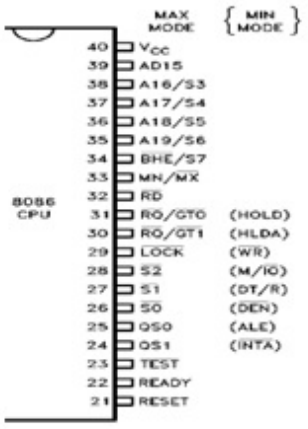

Figure 21: Pin diagram for MinMax modes

6.4.2 Bus Controller

8086 use 8288 bus controller.8288 bus controller uses Separate signals are used for I/O (IORC and IOWC) and memory (MRDC and MWTC). And Also provided are advanced memory (AIOWC) and I/O (AIOWC) write strobes plus INTA.

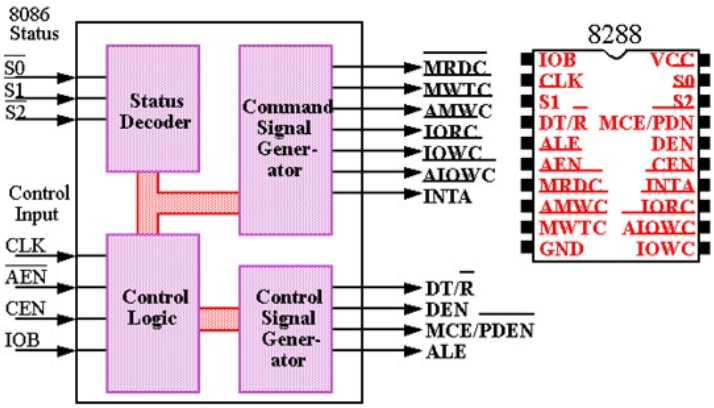

Figure 22: 8288 Bus controller

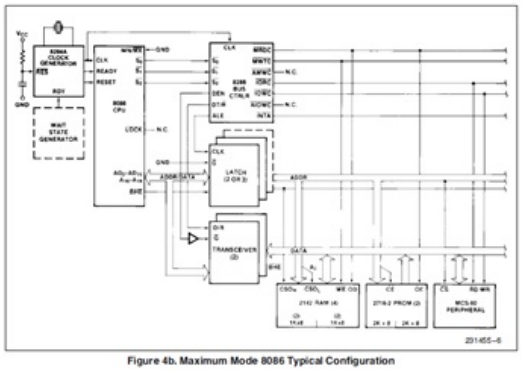

Figure 23: FMax mode

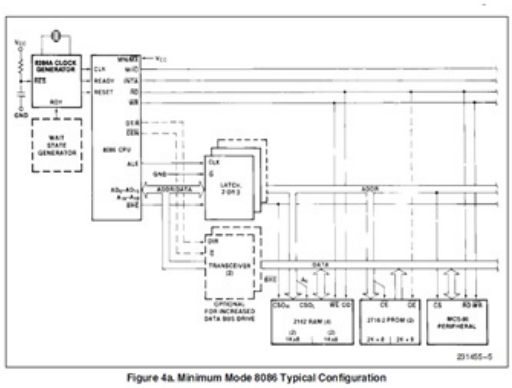

Figure 24: Min mode

6.4.3 Interrupt Controller

8086 use 8259A programmable interrupt controller for identify the high priority interrupt request. It adds eight vectored priority encoded interrupts to the microprocessor.There are two types of hardware interrupts,

- Maskable Interrupts
- Non Maskable Interrupts

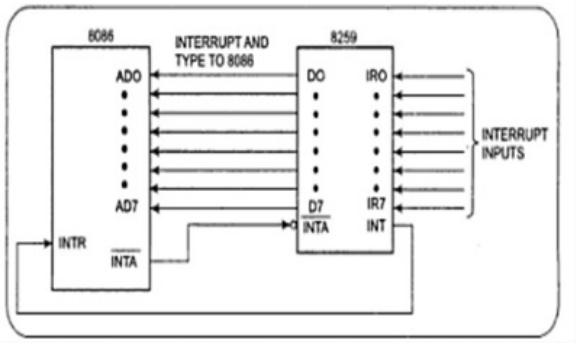

Fig:- Interface 8259 PIC with 8086 Microprocessor

Figure 25: 8259A Interrupt controller

7 System Timing

8086 use an 8284A Clock Generator for following purposes.

- Clock generation
- RESET synchronization.
- READY synchronization.
- Peripheral clock signal.

7.1 Clock generation

An Intel 8254 external clock generator the 8086 divides the external clock connected at the CLK pin internally by three. Therefore for 5 MHz internal clock, the 8284 must generate 15 MHz at its output, which will be given to the CLK pin of 8086.

7.2 Reset Synchronization

Negative edge-triggered flip flop applies the RESET signal to the 8086 on the falling edge. The 8086 samples the RESET pin on the rising edge.

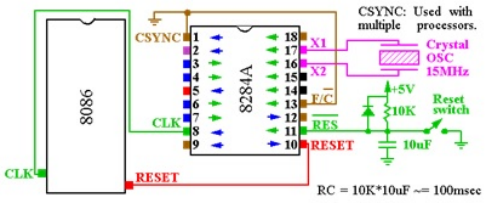

Figure 26: Falling edge RESET timing

7.2.1 Bus Timing

In 8086, address and data bus has been combined together and referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor. This local bus can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. Also this bus can be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Reading

Put the memory address into the address bus.
Put the data into the data bus.

Enable WRITE and set M/ IO to 1

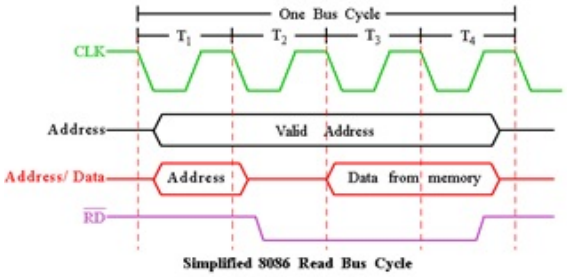

Figure 27: Read bus Cycle

Writing

Put the memory address into the address bus.
Enable READ and set M/ IO to 1
Wait for the memory access cycle.

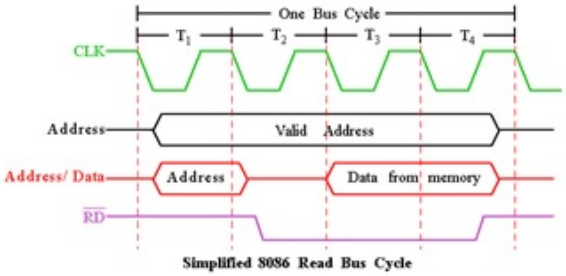

Figure 28: Write bus cycle

This one bus cycle contains 4 clock cycles. Therefore 8086 would take 800nS to complete a bus cycle. Also each read or write operation take one bus cycle.

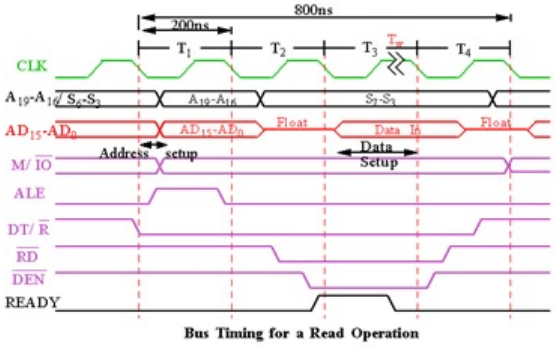

Figure 29: Read operation bus timing

1. During T1, address is placed on the address bus and set the direction of data transfer on data bus.
2. During T2, microprocessor issues the READ or WRITE signal and enables the memory or I/O device to receive the data for writes and the 8086 to receive the data for reads.
3.During T3, microprocessor allows memory to access data.
4.During T4, system prepares for next cycle by deactivating the bus signals.

7.3 Tw (Waiting Time)

This wait states are inserted in the bus cycle between T3 and T4, when NOT READY indication is given by addressed devices. Each Tw is equal to a clock cycle. Microprocessor use this time slots for internal housekeeping, therefore Tw is called idle states or inactivated clock cycles.

7.4 READY Synchronization

This is an input to the 8086 that causes wait states for slower memory and I/O components. Following figure shows the basic system timing in MIN mode including READY synchronization and bus timing.

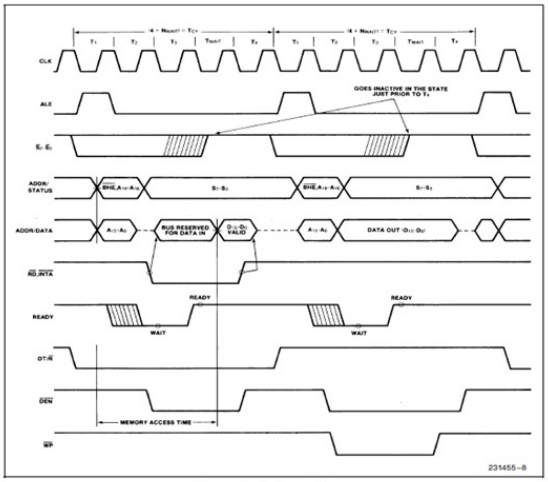

Figure 5. Basic System Timing

Figure 30: System timing

8 Memory Interfacing

8086 microprocessor has a 16 bit Data bus. The microprocessor must be able to access 16 bit or 8 bit memory locations. Therefore 16 bit memory is divided into two parts.

Low Bank

Contains even numbered byte locations.
Ex : 2,4,6,.
Low 8 bits of the data bus (D0:D7) or LS bytes indicates low bank address.

High Bank

Contains odd numbered byte locations.
Ex : 1,3,5,.
High 8 bits of the data bus (D8:D16) or MS bytes indicates high bank address.

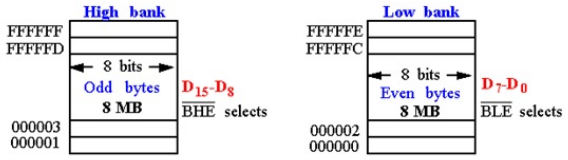

Figure 31: Memory bank

These banks are selected by the 8086 according to the BANK Enable signals.There are two bank enable signals.

BLE/ (A0) for select Low Bank.
BHE for select High Bank.

BHE	BLE	Function
0	0	Both banks enabled for 16-bit transfer
0	1	High bank enabled for an 8-bit transfer
1	0	Low bank enabled for an 8-bit transfer
1	1	No banks selected

Figure 32: Selecting memory banks

Bank selection can be done in two ways.

- By dedicating separate write decoders for each bank.
- By using a separate write signal (strobe) to each bank.

8.0.1 By separate decoders

This is the least effective way to decode memory address. In this method, high bank and lower bank will be selected by separate decoders. Advantage of this method is to conserve energy because only the bank or banks are enabled.

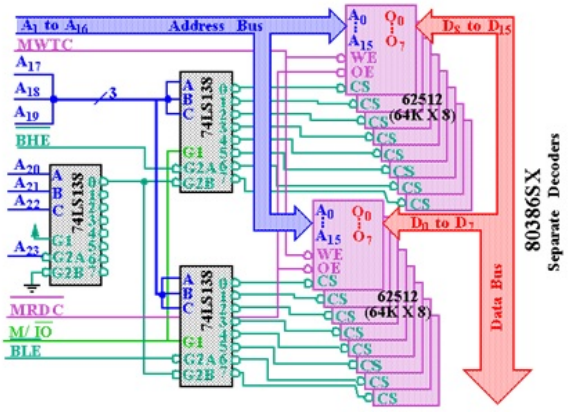

Figure 33: Memory bank selection by decoder

8.0.2 By separate write signals

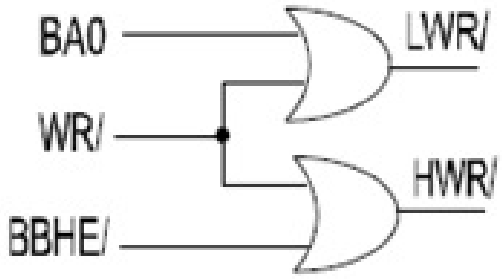

Figure 34: Memory bank selection by write signal

This is the most effective way to decode memory address. This method requires only one decoder. Therefore the number of components required is less than previous method. For generating separate bank write strobes for memory banks the A0 is combined with WR for the low bank selection signal (LWR) and BHE is combined with WR for the high bank selection signal (HWR). Here 8-bit read requests in this scheme are handled by the microprocessor.

8.1 DRAM Interfacing

DRAM need to be refreshed in every 2ms to 4ms. This refreshing is called transparent refresh or cycle stealing because internal circuitry is refreshing cells that are not accessed over a read or write cycle. Only cycle strobes a row address into the DRAM, obtained by 7- or 8-bit binary counter. The capacitors are recharged for the selected row by reading the bits out internally and then writing them back. For a 256K X 1 DRAM with 256 rows, a refresh must be occurred in every 15.6us (4ms/256). In 8086, one instruction cycle is 800 ns .This allows 19 memory reads/writes per refresh or 5 of the time.

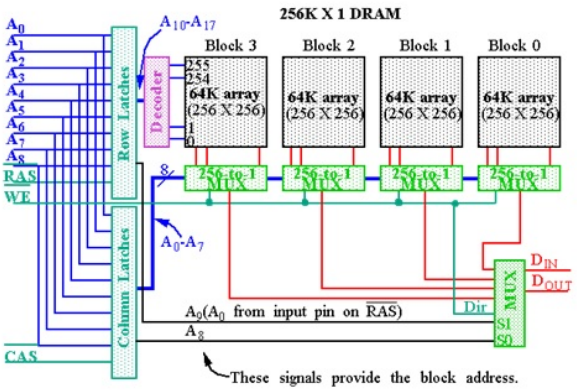

Figure 35: DRAM

Different is in the access time. Access time is varying from 10ns to 8 ns. This improves performance, particularly for reads into cache block sizes of 256 bits.

8.1.1 Extended Data Output Memory Interfacing

Any memory access in an EDO memory (including a refresh) stores the 256 bits in a set of latches. Any subsequent access to bytes in this set is immediately available in the processor.

8.1.2 DRAM Memory Controllers

8086 use Intel 82C08 as the DRAM controller for address multiplexing and generating the DRAM control signals. Microprocessor bits A 1 through A 18 (18 bits) drive the 9 Address Low (AL) and 9 Address High (AH) bits of the 82C08.

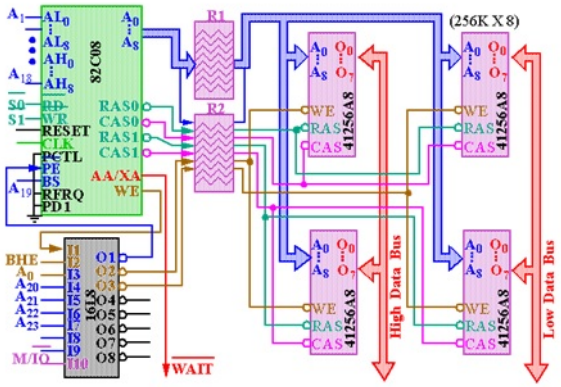

Figure 36: DRAM path flow

References

[1] 2016. [Online]. Available: http://www.cosc.brocku.ca/bockusd/3p92/LocalPages/8086achitecture.h [Accessed: 11- May- 2016].

[2] "8086 and 80386SX 16 bit Memory Interface", Eceresearch.unm.edu, 2016. [Online]. Available: http://eceresearch.unm.edu/jimp/310/slides/8086memory3.html. [Accessed: 11- May- 2016].

[3] 2016. [Online]. Available: http://www.ece.cmu.edu/ece740/f11/lib/exe/fetch.php?media=wiki:8086 [Accessed: 11- May- 2016].

[4] "8086/88 Device Specifications", Ece Research.unm.edu, 2016. [Online]. Available: http://eceresearch.unm.edu/jimp/310/slides/8086chipset.html. [Accessed: 11- May- 2016].

[5] "Instruction set of 8086", Slideshare.net, 2016. [Online]. Available: http://www.slideshare.net/9840596838/instruction-set-of-8086-16837268. [Accessed: 11- May- 2016].

[6] "8086 instructions", Electronics.dit.ie, 2016. [Online]. Available: http://www.electronics.dit.ie/staff/tscarff/8086instructionset/8086instructionset.html. [Accessed: 11- May- 2016].

[7] "8086 Instruction Set - Instruction Set of 8086 Microprocessor - MHE - Microprocessors and Microcontrollers", GradeStack Courses, 2016. [Online]. Available: http://gradestack.com/Microprocessors-and/Instruction-Set-of-8086/8086-Instruction-Set/19318-3912-38179-study-wtw. [Accessed: 11- May- 2016].

[8] "Intel 8086", Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Intel8086. [Accessed: 11- May- 2016].