4. 16-BIT MICROPROCESSOR & PROGRAMMING

(8086 µp)

2 theory. 2 numerical } for exam.

1 16-BIT µ-PROCESSOR (8086)

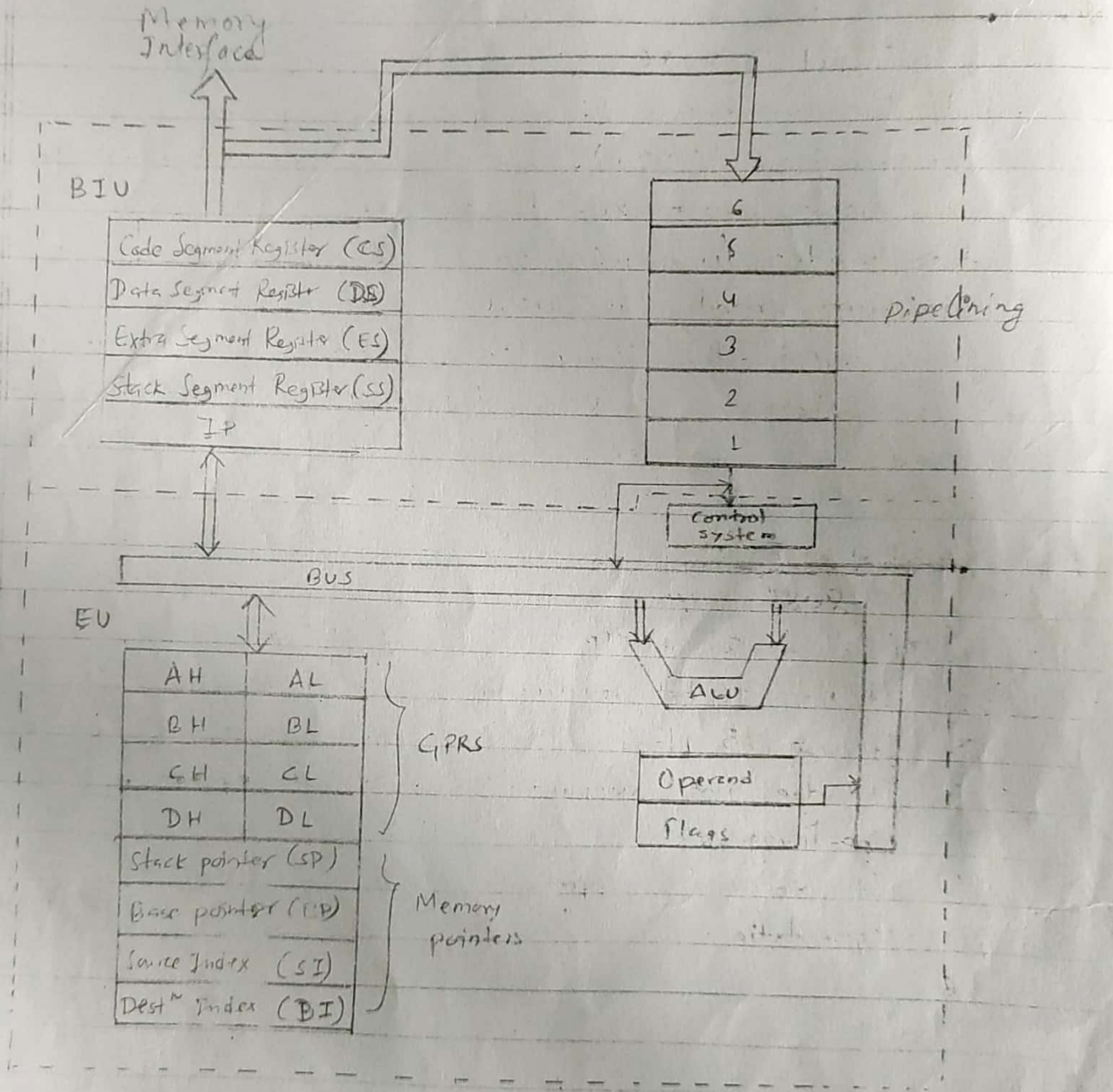
Intel's 8086 was the first 16-bit µ-processor to hit the market. It is packaged in a 40-pin dual-in line package (DIP). 8086 µ-processor has 16-bit data bus and uses 20-bit address lines. It can work on two modes (ie minimum mode and maximum mode). Minimum mode is used when only one 8086 µprocessor is used in the µcomputer, whereas, maximum mode is used when many 8086 µ-processor or multi-processors are used in µcomputer.

2 COMPARISION BETWEEN 8085 & 8086 µP

8085 µP	8086 µP
1. It is a 8-bit µp	1. It is 16-bit µp.
2. It has 16 bit address lines.	2. It has 20-bit address lines.
3. It uses 5 flags	3. It uses 9 flags.
4. It does not support pipe-lining	4. It supports.
5. It does not support memory segmentation.	5. It supports.

DMP.

#3) INTERNAL STRUCTURE (Architecture) of 8088 µP



EU → execution unit

BIU → Bus Interface Unit.

The 8086 CPU is divided into two functional parts namely: (i) Bus Interface Unit (BIU)
(ii) Execution Unit (EU)

It is divided into two parts for increasing the processing speed.

1) Bus Interface Unit

The BIU sends out address, read data from ports and memory address, and writes data to ports and memory. To speed up the program execution, BIU fetches as many as six instructions bytes ahead of the time from memory. The pre-fetched instruction bytes are held for the execution in a FIFO group of registers called as queue. This is done to increase the processing speed and also known as pipelining. The BIU contains 4 segment registers ie (a) Code segment Register (CS register)
(b) Stack segment Register (SS register)
(c) Extra segment Register (ES register)
(d) Data Segment Register (DS register)

2) Execution Unit

The EU works in parallel with BIU. The different parts of EU are:- (a) Control System
(b) ALU
(c) Flag register
(d) General Purpose Register (GPRs)
(e) Pointer and Index Registers.

(a) Control System

The control circuit directs all the internal operations of the processor.

(b) Arithmetic & Logic Unit (ALU)

It performs all the arithmetic and logical manipulation of data such as increment or decrement.

(c) Flag Register

A flag is a flipflop which indicates some conditions produced by the execution of an instruction. There are 9 (nine) flags in 8086 µp.

(d) General Purpose Registers (GPRs)

All the General Purpose Registers (GPRs) in 8086 µp are of 8-bit (in group, 16-bit) and are labelled as:

AH, AL, BH, BL, CH, CL, DH, DL

Here AL register has some special features than others and also known as accumulator.

(e) Pointer and Index register

Execution Unit contains 4 more 16-bit registers namely:

stack pointer (SP)
Base pointer (BP)
Source Index (SI)
Destination Index (DI)

These registers have the capacity to store the data temporarily such like GPRs.

~~These registers have the capacity to store the data temporarily such that like GPRs~~

Flags of 8086 µP :-

8085 ki flag esi

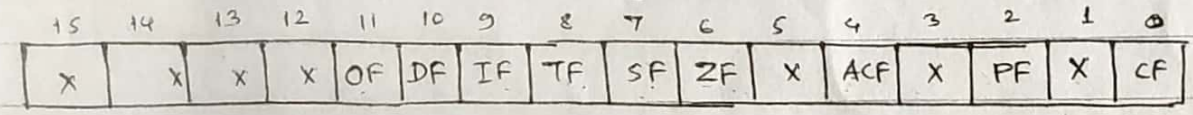


fig. 8086 flag registers

1. **Carry Flags (CF:)**

The carry flag is set to 1 if an arithmetic operation produces a carry or subtraction needs a borrow, otherwise reset.

2. **Parity Flag (PF)**

This flag is set if the result has ~~been~~ even number of 1's.

3. **Auxillary Carry Flag (ACF)**

The ACF is set to 1 if there is a carry out of 3rd bit during addition or borrow by a 3rd bit during subtraction.

4. **Zero flag**

It is set to 1, if an addition or subtraction produces zero result.

5. **Sign Flag**

This flag is set to 1, if MSB is 1 otherwise reset.

6. Trap Flag (TF)

If the trap flag is set to 1, an internal interrupt is executed after instruction. By this user can debug the program.

7. Interrupt Enable (IF)

If set, certain type of interrupt (maskable) can be recognized by CPU otherwise these interrupts are ignored.

8. Direction Flag (DF)

If set, indicates the left or right direction for moving or comparing the string.

9) Overflow flag (OF)

This flag is set if there is an arithmetic overflow ie if the size of result exceeds the capacity of destination location.

#4) Imp ADDRESSING MODES OF 8086

It is a method of specifying the address of the operands in memory. The various types of addressing modes are as follows:-

1. Register Addressing Mode
2. Immediate Addressing Mode
3. Direct Addressing Mode
4. Register Indirect Addressing Mode
5. Base plus Index Addressing
6. Register Relative Addressing
7. Base Relative plus Index Addressing

1. Register Addressing Mode:

It transfers a copy of a byte or word from source register or memory location to the destination register or memory location.

Eg. MOV AL, BL
MOV CH, CL
MOV CX, DX
MOV SI, DI

2. Immediate Addressing Mode:

It transfers the source immediate byte or word of data into destination register or memory.

Eg. MOV AX, 44H
MOV SI, 00H
MOV CH, 100

3. Direct Addressing Mode:

It moves a byte or word between a memory location and register.

Eg. MOV AL, (1234H)
MOV AH, (1000H)

4. Register Indirect Addressing Mode:

It transfers a byte or word between a register and memory location addressed by an index or base register

Eg. MOV (DI), BH
MOV AL, (DX)

5. Base plus Index Addressing

Transfers a byte or word between a register and memory location addressed by a base register plus ~~and~~ an indexed register.

Eg. MOV (BX + DI), CL
MOV CX, (BX + SI)
MOV CH, (BP + SI)

6. Register Relative Addressing

Moves a byte or word between register and memory location addressed by an ~~index~~ or base register plus a displacement.

Eg. MOV AX, (BX + 4)
MOV AX, (DI + 100)

7. Base Relative plus Index Addressing

It adds a displacement besides using index register ~~plus~~ base register.

Eg. MOV DH, (BX + DI + 100)

49 # INSTRUCTION SET

IP IS BAD / BAD SIPI /.

1. Data Transfer Group
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. String Instructions
5. Iteration Control Instructions
6. Process Control Instructions
7. Interrupt Instructions

8

INSTRUCTIONS SET

The different instructions set of 8086 MP are explained below:-

1. Data Transfer Group

These Group of instructions transfers data between registers and memory or registers and registers.

Data transfer may be 8 bit or of 16-bit. These instructions do not affect the flags.

Eg.

a) MOV Dest^n, source

b) PUSH source [decrement sp by 2] (sp → stack pointer)

c) POP dest^n [increment sp by 2]

d) XLATE [translate data from one code to another]

2. Arithmetic Instructions

These group of instructions perform arithmetic addition, subtraction, multiplication and division.

Eg.

a) ADD dest^n, source

b) ADC dest^n, source (Add with carry)

c) DAA (Decimal adjust after addition)

d) SUB dest^n, source

e) SBB dest^n, source (subtract with borrow)

f) MUL source

g) NEG dest^n (find 2's complement)

3. Bit Manipulation Instruction

These instructions perform logical operations like AND, OR, NOT, XOR, Compare, rotate & shift operations.

Eg.

a) NOT dest^n

b) AND dest^n, source

c) OR dest^n, source

d) NOR dest^n

e) ROL / ROR dest^n, source

4. String Instructions

It is a series of ASCII codes in sequence in memory, string instruction, does copy, compare, load and store functions with the repeat instructions.

Eg.

a) MOVS / MOVSB / MOVSW ; copy the data from DS:SI to ES:DI

b) CMPS / CMPSB / CMPSW ; compare the data from DS:SI to ES:DI

c) LODS / LODSB / LODSW ; load the string bytes from DS:SI to ES:DI

d) STODS / STODSB / STOSW ; store string byte/words from AL/AX into ES:DI

5. Iteration Control Instructions

These instructions control the repetition of the loop, functions or execution of instructions.

Eg.

a) LOOP label ; Repeat the function for cx times
b) LOOP E / LOOP Z ; conditional repeating of function.

6. Process Control Instructions:

They may set or reset the carry, direction and interrupt of processor.

Eg.

a) STC set carry flag
b) CLC clear carry flag
c) CMC complement carry
d) SIT set interrupt
e) CLD clear DF

7. Interrupt Instructions.

They enable / disable interrupts and executes ~~it~~ interrupt services routine.

Eg.

a) INT byte ; type 0 to 255
b) INTO ; interrupt on overflow
d) IRET ; return from ISR

#9 ASSEMBLY LANGUAGE PROGRAMMING in 8086 µP

In 8086 µP, there are three types of assembly language programming namely: ~~MASM~~,

(i) MASM (Macro Assembler) developed by Microsoft
(ii) TASM (Turbo Assembler) developed by Borland
(iii) AM-86 developed by Intel

These assembly language programming uses five different program development tools. Out of these three assembly language programming, MASM is widely used.

Program Development Tools

The different PDT are explained below:-

(i) Editor

It provides the facility for writing assembly program in text mode. This file is saved with .asm extension. The program may be written in DOS-editor or in notepad and saving the file with .asm extension.

(ii) Assembler

The assembler reads the source program and generates equivalent machine program. Here assembler generates OBJ project program and LST program.

In OBJ program, only object codes are present. Whereas in LST program, memory address, assembly code and object codes are given. This is helpful

for finding the fault in the program.

(iii) Linker

It is also one of the important tools for assembly language programming (ALP) such that it is used for linking multiple small programs into a large program. These small programs are OBJ programs and linker produces EXE program (ie executable program). Here while linking the program, it produces relative addresses for JUMP instructions.

(iv) Locater

It helps to locate the specific address to the program. Exe2Bin Exe in DOS works as a locater program. It converts the EXE program to BIN program.

(v) Debugger

It helps to provide the system memory of the ALP. Here debugging can be done over the program, and the breakpoint in the program can be inserted so that it makes easier to find the error if any.

ASSEMBLER DIRECTIVES

Directives are the commands or functions, directing the procedures while developing object programs from assembly language program. Various directives available for assembler are given as :-

i) SEGMENT and ENDS [PELE SAD PO]

(ii) DB, DW, DQ, DD, DT

(iii) EQU

(iv) ASSUME

(v) LABEL

(vi) END

(vii) PROC ... ENDP

(viii) PTR

(ix) ORG

(i) SEGMENT and ENDS

An .exe format of our ALP consists of one or more segment. SEGMENT directives define the start of segment and ENDS directive inidicates its end.

(ii) DB, DW, DQ, DT

These are data control directives

DB → define byte

DW → define coord

DD → define double coord

DQ → define Quad coord

DT → define ten bytes

(ii) EQU

It is used as a #define

for eg.

CONST EQU 3.14

(iv) ASSUME

It tells the assembler what addresses will be in the segment registers (CS, DS, SS and ES) at the execution time.

(v) LABEL

It is used to specify destination for jump or call instructions or to specify the reference to a data item

(vi) END

It is an assembler directive that is put at the end of program. Assembler will ignore any statement after it.

(vii) PROC ... ENDP

Procedures are defined by PROC. After this procedure name we use NEAR or FAR to specify procedure type. ENDP defines ends of procedure

(viii) PTR

(iii) ORG

. It is used to set the location counter which is used by a assembler to account for its relative position in DS or CS.

#8) DOS-BIOS INTERRUPTS

INT 21H

It is called DOS function calls. There are 87 different functions supported by this interrupt, identified by a function member placed in the AH register. Some of the important functions are as follows:

01: Console input with echo

- wait for input from standard input device. The character is stored in AL. The character is echoed ie. redisplayed console as soon as it is input.

```
ie. MOV AH, 01
    INT 21H
```

02: character output

- sends a character to standard output device

```
ie. MOV AH, 02
    INT 21H
```

05: printer output

- sends a character in parallel printer port.

ie. MOV AH, 05
INT 21 H

08: Console Input without echo

- The character is stored in AL but not displayed at console.

ie. MOV AH, 08
INT 21 H
MOV char, AL

09. string output

- sends a string character to a standard output device. DX must contain the offset address of string & string must be terminated by a dollar sign ($)

ie.

MOV DX, offset string
INT 21 H
data
string dB 'Microprocessor $'

INT 10 H (video function)

Video display is controlled by using INT 10H

#9) TYPES OF ASSEMBLER

The two types of assembler are:-

(i) One pass assembler

(ii) Two pass assembler

(i) One pass assembler

This assembler goes through the assembly language programming once and translate the assembly language program. This assembler has the problem of defining forward references. This means that a JUMP instruction using an address that appear later in the program must be defined by the program after it is assembled.

(ii) Two pass assembler

This assembler scans the assembly language program twice. In the first pass, the assembler generates the table of symbols. A symbol table consists of LABELs with address assigned to them. In this way, LABEL can be used for JUMP statements and no address calculation is to be done by the user.
On the second pass, the assembler translates the assembly language program into machine language. The two pass assembler is hence more efficient and easier to use.

8086 PROGRAMMING

The general format is:-

```
. model small
. stack
. data
Data ---- (dB)
  }
Result dB (?)
. code
main proc
      :
      :
INT 21H
main endp
END
```

```
. model small
. stack
. data
------ ; initialize data variables
Result dB (?)
. code
main proc
 - -
 - -
 - -
 - - ; instrn sets
INT 21H
main endp
End main
```

1. Write a 8086 programming for addition of two variables.

```
.model small
.stack
.data
Data 1 dB
Data 2 dB
Result dw (?)
.code
main proc
    MOV AX, data 1
    MOV DS, AX
    MOV BX, data 2
    ADD AX, BX
    MOV DI, offset result
    MOV [DI], AX
    INT 21 H
main endp
END
```

2. WAP to multiply 08H and 07 H

```
.model
.stack
.data
INPUT 08H (dB)
INPUT 07 H (dB)
Result dw (?)
.code
```

```
.main proc
        MOV AX, 08 H
        MOV DS, AX
        MOV BX, 07 H
        MUL AX, BX
        MOV DI, offset result
        MOV [DI], AX
        INT 21 H
main endp
END
```

3. WAP to display "Hello World" on screen.

```
.model small
.stack
.data
message dB "Hello World $"
.code
main proc
        MOV DX, offset message
        MOV AX, segment message
        MOV DS, AX
        MOV AH, 09   [function display string]
        INT 21 H     [Dos call]
        MOV AH, 4 C0 H   [Return from DOS]
        INT 21 H
main endp
END
```

4. WAP to find square of given number

```
.model small
.stack
.data
Data  (dB)
Result dw (?)
.code
main proc
    MOV AX, Data
    MOV DS, AX
    MOV DX, 01
    MOV CX, Data
    MOV AX, 0
Again:  ADD AX, DX
        ADD AX, 02
        Loop Again
        MOV Result, AX
        INT 21H   (DOS call function)
main endp
END
```

Algorithm:

00 + 01 = 01
01 + 03 = 04
04 + 05 = 09
09 + 07 = 16
16 + 09 = 25

and so on

WAP using DOS / BIOS interrupts to accept a string from a keyboard.

```
.model small
.stack
.data
Getchar dB
.code
main proc far
    MOV AX, data
    MOV DS, AX
    MOV AH, 10H
    INT 16H        (wait for key)
    MOV Getchar , AL
main endp
end main
```

2. WAP to print A to Z with space in one line.

```
.model small
.stack
.data
data A to Z
.code
main proc
    MOV AX, data
    MOV DS, AX
    MOV CX, 26
    MOV DH, 'A'
    MOV AH, 02h

Loop: MOV DL, DH
      INT 21H
      MOV DL, " "
      INT 21H
      INC DH
      Loop
    MOV AX, 4c00H
    INT 21H
main endp
End main
```

o/p A, B, C, ... Z