UNIVERSITY OF BAGHDAD

College of Education for Pure Science

(Ibn Al-Haitham)

Department of Computer Science

Microprocessor 8086

Second Stage

Prof. Dr. Alaa Abdul Hameed

Lecturer Sukaina Shokry

Lecturer Rasha Majid

Instructions set

Each instruction in the source program can be divided into two separate parts:

1- "opcode"

2- "Operand"

The opcode part identify the operation that is to be performed (such as add, subtract, move….). in assembly language, assign a unique one, two, or three letters combination for each operation such as (Add-add, Sub-subtraction, Mov-move, etc).

8086 has 117 instructions, these instructions divided into 6 groups:

1. Data transfer instructions
2. Arithmetic instructions
3. Logic instructions
4. Shift instructions
5. Rotate instructions
6. Advance instructions

1. Data Transfer Instructions

The microprocessor has a group of data transfer instructions that are provided to move data either between its internal registers or between an internal register and a storage location in memory. Some of these instructions are:

A- MOV use to transfer a byte or a word of data from a source operand to a destination operand. These operands can be internal registers and storage locations in memory. Notice that the MOV instruction cannot transfer data directly between a source and a destination that both reside in external memory.

MOV OP1,OP2

copy OP2 to OP1

ALGORITHM

OP1 = OP2

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM
SREG	MEM
MEM	SREG
SREG	REG

the move ins can't :-

*set the value of CS and IP REG

* Copy the value of CS and IP

*Copy the value of SREG to another S REG

(should copy to general REG first)

*copy immediate value to SREG

(should copy to general REG first)

EXAMPIE :

MOV AX,0F85

MOV DS,AX

MOV CL,’A’

MOV CH,01010011b

Ret ; return to operating system

B -XCHG OP1,OP2

ALGORITHM

OP1 <-------> OP2

EX :

ORG 100H

MOV AL,5

MOV AH,2

XCHG AL,AH

XCHG AL,AH

RET

OP1	OP2
REG	MEM
MEM	REG
REG	REG

2. Arithmetic instructions

a- ADD OP1,OP2

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM

ALGORITHM

OP1=OP1+OP2

EX:

```
ORG 100H
MOV AL,5
MOV BH ,3
ADD AL,2; AL=7
ADD AL,BH ;AL=0A =10d
ADD AX,BX ; AX= 030A
ADD AL,-3; AX=0307
RET
```

b- ADC Add with carry

ADC OP1,OP2

OP1	OP2
REG	MEM
MEM	REG
MEM	IMM
REG	IMM

ALGORITHM

OP1= OP1+OP2+CF

EX:

```
ORG 100H
```

STC ; SET CF=1

MOV AH,5

ADCAL,1 ;AL=7

RET

C- SUB SUBTRACT

OP1	OP2
REG	MEM
MEM	REG
MEM	IMM
REG	IMM

SUB OP1,OP2

Algorithm

OP1=OP1-OP2

EX:

ORG 100H

MOV AL,8; AL=8

SUB AL,1;AL=7

SUB AL,-2;AL=9

SUB AX,BX ;AX=0009

MOV DL,4

SUB AL,DL ;AL=5

D- SBB Subtract with borrow

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM

OP1
REG
MEM

SBB OP1,OP2

Algorithm

Op1= op1-op2-CF

EX :

STC

MOV AL,5

SBB AL,3 ; AL=5-3-1=1

RET

E- INC INCREMANT

INC OP1

Algorithm

OP1= OP1+1

EX:

ORG 100H

MOV AL,4

INC AL ;AL=5

RET

F- DEC DECREMANT

OP1
REG
MEM

DEC OP1

Algorithm

Op1=op1-1

EX:

MOV AL,255 ;AL=FF

DEC AL ; AL= FE

G- NEG NEGATIVE

NEG OP1

OP1
REG
MEM

Algorithm

1- Convert all bits to binary

2-find the first complement

2- add 1 to the converted operand

EX:

MOV AL ,6

NEG AL ; AL= FA , -6 d

AL
06
00000110
11111001
+ 1
11111010
F A

H.W :

1- Write a program to find the negative of CH which is equal to 12

2- Find the solution of AX= 0FC+ 3-BL Where bl =5

H – Un singed multiply

MUL OP1

OP1
REG
MEM

Algorithm

When operand is a byte

AX =AX* Operand

When operand is a word

(DX AX)= AX * Operand

EX :

20*2

ORG 100H

MOV AL , 20
MOV BL,2

MUL BL

RET

AL= 2O
BL = 2
AL = 14 H
AX = 28 H= 40 d

I MUL

Algorithm

When operand is a byte

AX =AX* Operand

When operand is a word

(DX AX)= AX * Operand

EX:

-2*-4

ORG 100H

MOV AL, -2; AL= FE

MOV BL,-4;BL=FC

IMUL BL ; AX= 8

Ret

Unsigned division

DIV OP

OP1
REG
EME

Algorithm

When operand is byte

AL=AX / OPERAND

AH= remainder (modulus)

When operand is a word

AX= (DX AX) /OPERAND

DX= remainder (modulus)

EX:

203 /4

```
ORG 100H
MOV AX,203 ; AX= 00CBH
MOV BL,4
DIV BL ;        AL = 50d
                AH= 3d
RET
```

H.W/ FIND 1- DX=(13/2)+6 2- CX=(12*4) /8 3- BX=(120-30) /5

I DIV

Algorithm

When operand is byte

AL=AX / OPERAND

AH= remainder (modulus)

When operand is a word

AX= (DX AX) /OPERAND

DX= remainder (modulus)

EX:

-203 /4

```
ORG 100H
MOV AX ,-203 ;AX =0FF35H
MOV BL,4
I DIV BL; AL = -50 (0CEH)
          AH = -3(0FDH)
RET
```

Variables

(Is a memory location) syntax for a Variables declaration

Name DB Value

Name DW Value

DB – Stays for Define Byte

DW – Stays for Define Word

Name can be any letter or digital combination through it should start with letter

Value can be any numerical value or ?

EX:

```
ORG 100H
MOVAL,VAR1
MOV BX,VAR2
MOV VAR3 ,AL
RET
VAR1 DB 7
VAR2 DW 1234H
VAR3 DB ?
```

CMP destination, source.

The CMP instruction compares the destination and source i.e., it subtracts the source from destination. The result is not stored anywhere. It neglects the results, but sets the flags accordingly. This instruction is usually used before a conditional jump instruction.

CMP OP1,OP2

Example:

```
Org 100h
MOV AL, 5
 MOV BL, 5
 CMP AL, BL ;      AL = 5, ZF = 1 (so equal!)
RET
```

JUMP INSTRUCTION

Jump Instructions are used for changing the flow of execution of instructions in the processor. ...

There are two kinds of jump instruction:-

a) Unconditional jump

JMP (address to jump to it)

b) Conditional jump

Jump only if condition is TRUE

1) JC : Stands for 'Jump if Carry'

2) JNC : Stands for 'Jump if Not Carry'

3) JE : Stands for 'Jump if Equal'

4) JZ : Stands for Jump if Zero'

5) JNE: Stands for 'Jump if Not Equal'

6) JNZ : Stands for Jump if Not Zero'

5) JA : Stands for Jump if above

Ex – add two numbers n & m if the result is greater or equal than 5 then ah=11h else ah=22H (n=5, m=7)

```
Org 100h

MOV Al,n

ADD m,AL

CMP m,5
```

JG AA

MOV AH,11H

JMP BB

AA:MOV Ah,00

BB:RET

n db5

m db 7

H.W

Check if AX is divisible by 2 then CX=0000 else CX=FFFF

How carry flag works

AX		11111111
		1
AX	1	00000000

Q2/ write a program to check if CX is divisible by (2)

then print ‘even ‘ Else print ‘odd’ (CX=08)

```
INCLUDE 'EMU8086.INC'
org 100h
mov cx,08
mov bl,2
mov ax,cx
div bl
cmp ah,0
je rr
print 'odd'
jmp ex
rr:print 'even'
ex:
ret
```

Array In 8086

In assembly language we use two types of data

1- 'DB' Data Byte. (8 bit)

2- 'DW' Data Word. (16 bit)

the very common method to declare an array in emu 8086 is

Array_Name Data_Type Values

a DB 10,20,30,40,50

b DW 10,20,30,40,50

when an array contain the same elements declare as

x db 02,02,02,02

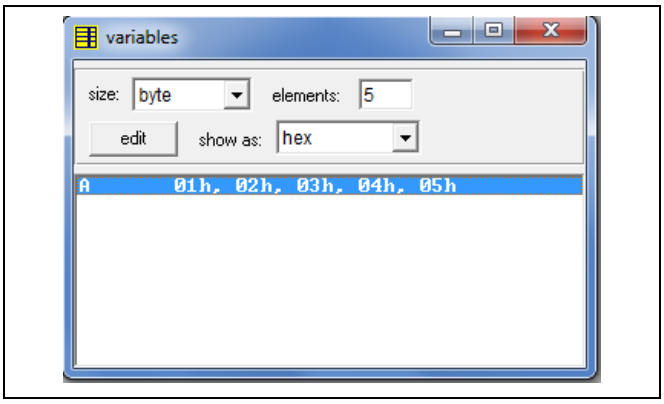
or

x db 4 dup (2)

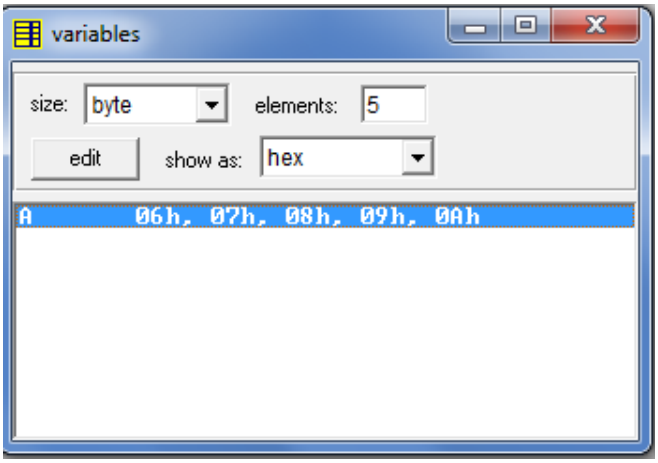
SI is the source index the index for array

EX: suppose array a(01,02,03,04,05) add 5 to each element in array a

```
Org 100h
MOV si,0
hhh:add  a[si],5
INC si
CMP si,5
JNE hhh
RET
a db 01,02,03,04,05
```



array before execution



array After execution

EX : add two arrays(a,b) and put the result in third array (c)

```
Org 100 h
Mov si,0
Mov cx,0
hhh: mov al,b[si]
Add  al,a[si]
Mov c[si],al
Inc si
Inc cx
Cmp cx,6
JNE hhh
Ret
a db 01h,02h,03h,04h,06h,09h
b db  01,02,02,03,01,03
c db 6 dup (0)
```

Find the biggest value in an array a{2,8,10,5,4}

```
Org 100h
Mov  si,0
Mov  al,a[si]
Inc si
bbb: cmp a[si],al
Ja aaa
Jmp ccc
aaa: mov al,a[si]
ccc: inc si
cmp si,5
jb bbb
ret
a db 2,8,10,5,4
```

Si	A[si]	al
0	2	2/8/10
1	8	
2	10	
3	5	
4	4	

Loop label

Decrease cx, jump to label if cx not zero

Algorithm

Cx=cx-1

If cx<>0 then

Jump

Else

No jump

Continue

EX write program to print ‘stage two ‘ 5 times on screen

Include ‘emu8086.inc’

Org 100h

Mov cx,5

Lable :

Print ’stage two’

Loop label

ret

1- Find the number of even element in array

Z= 2,3,8,6,9,1,5

```
org 100h
 mov si,0
 mov cx,0
 mov bl,2
 aa: mov al,z[si]
 div bl
 cmp ah,0
 je bb
 jmp cc
 bb:inc cx
 cc:inc si
 cmp si,6
  jbe aa

ret
 z db 02,03,01,06,04,09,05
```

2- Find the number of element greater than 5 in array

Z= 2,3,8,6,9,1,5

```
Org 100h
MOV SI,0
MOV CX,0
CC:CMP Z[SI],5
JE BB
JMP AA
BB:INC CX
AA: INC SI
CMP SI,6
JBE CC
RET
```

Example :-

Suppose array a={11,FC,2A,24,36,98}

Execute the following:

1- Shift arithmetic right the first element CL=1 .

2- Rotate the second element 2 times to left .

3-Add one to third element .

4- Divide the fourth element by 3 store the result in bx .

5- XOR the last element with 7D .

1- sol :-

```
org 100h
mov si,0
mov cl,1
sar a[si],1
ret

a db 11h,0fch,2ah,24h,36h,98h
```

A[si]	1	1
Binary	0001	0001
Result	0000	1000

2- sol:-

org 100h

A[si]	f	c
Binary	1111	1100
Result	0011	1111

mov si,1

mov cl,2

ror a[si],cl

ret

a db 11h,0fch,2ah,24h,36h,98h

3- sol :-

A[si]	2	a
Binary	0010	1010
Result	0010	1011

Org100h

Mov si,2

Inc a[si]

Ret

a db 11h,0fch,2ah,24h,36h,98h

4- sol :-

```
org 100h

mov si,3
mov cl,3
mov al,a[si]
div cl
mov bx,ax

ret

a db 11h,0fch,2ah,24h,36h,98h
```

5- sol :-

```
org 100h
mov si,5
```

A[si]	9	8
Binary	1001	1000
Xor	7	d

Binary	0111	1101
Result	1110	0101

```
mov cl,7d
xor a[si],cl
ret
a db 11h,0fch,2ah,24h,36h,98h
```

Addressing mode

1- Register addressing mode

EX: MOV AL,O5H

MOV BL,O7H

MOV BL,AL

2- Immediate addressing mode

EX:

MOV AL,O5H

MOV BL,O7H

3- Direct addressing mode

ORG 100H

MOV AL, 05

MOV [0010AH],AL

RET

DS=700

07000	
0010A	+
0710A	

Physical address = DS *10 +0010A

0710A ←---05H

4- Register Indirect Addressing mode

المسجلات المسموح بها REG {SI,DI,BX,BP}

ORG 100H

MOV AX, 5544H

MOV SI,0010AH

MOV [SI],AX

RET

DS=0700

SI= 0010A

07000	
0010A	+
0710A	

ADDRESS	VALUE

PA= DS *10+SI

PA =0710A

0710A	44
0710B	55

5- Base Addressing mode

المسجلات المسموح بها

BX, BP work with segment registers DS, SS respectively

ORG 100H

MOV AX,02211H

MOV BX, 00107H

MOV [BX]+4,AX

RET

DS=0700

BX=00107

Disp = 4

PA= DS*10 + BX+ Disp 0710B =11H

07000
00107
4 +
0710B =

0710C =22H

6- Indexed Addressing mode

```
ORG 100H
MOV SI, 00107H
MOV [SI]+4,77H
MOV AL,[SI+4]
RET
```

PA= DS*10+SI+ Disp

= 0710B --→ 77H --→ AL

7-Based Indexed Addressing mode

```
ORG 100H
MOV SI,00107H
MOV BX,00005
MOV [SI+BX]+3, 6677H
RET
```

DS	700(0)
SI	00107
BX	0005
Disp	3
PA	0710F

0710F <---77

07110 ←-- 66

Logical instructions

1-AND OP1,OP2

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM

Logical AND between all bits

Of operands .result stored in op1

EX:

MOV AL,01100001B

AND AL,11011111B; AL = 01000001

2- OR OP1,OP2

EX:

MOV AL,01000001B

OR AL,00100000B ; AL= 01100001B

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM

NOT OP1

Algorithm

OP1
REG
MEM

IF bit is 1 turn to 0

IF bit is 0 turn to 1

EX:

MOV AL, 00011011B

NOT AL ; AL= 11100100B

XOR OP1,OP2

Logical XOR (Exclusive OR)

Between all bits of two operands

OP1	OP2
REG	MEM
MEM	REG
REG	REG
MEM	IMM
REG	IMM

Result is stored in op1

These rules apply

1 xor 1= 0

1 xor 0 =1

0 xor 1 =1

0 xor 0= 0

EX :

MOV AL,00000111B

XOR AL,00000010B ; AL=00000101

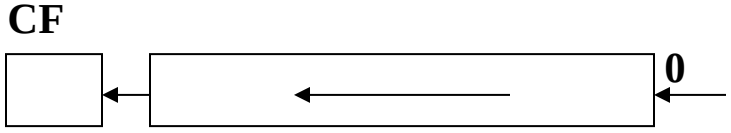
- Shift Instructions

The four types of shift instructions can perform two basic types of shift operations. They are the logical shift and arithmetic shift. Each of these operations can be performed to the right or to the left.

Shift logical

1- SHL Shift logical left

SHL D, COUNT
D= MEM, REG
COUNT =CL



EX:-

```
ORG 100H
MOV AL,11100000B ; AL= E0H
SHL AL,1 ;AL=C0H
RET
```
Flag affected OF ,CF

2- SHR Shift logical Right

SHR D, COUNT
D= MEM, REG
COUNT= CL

EX:-

```
ORG 100H
MOV AL,00000111B ;AL=07H
SHR AL,1 ;AL=03
RET
```
Flag affected OF,CF

- SHIFT ARITHMATIC

1- SAL Shift Arithmetic Left

SAL D,COUNT
Same work as SHL

2- SAR Shift Arithmetic Right

SAR D,COUN

D= MEM,REG
COUNT =CL

FLAG affected OF,SF,ZF,AF,PF,CF

EX:-

```
ORG 100H
MOV AL,E0H; AL= 11100000B
SAR AL,1 ; AL= 11110000
MOV BL,4C ; BL=01001100B
SAR BL,1 ;BL= 00100110B
RET
```

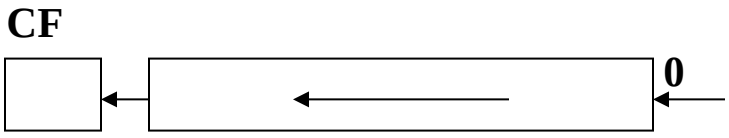
- Shift Instructions

The four types of shift instructions can perform two basic types of shift operations. They are the logical shift and arithmetic shift. Each of these operations can be performed to the right or to the left.

Shift logical

1- SHL Shift logical left

SHL D, COUNT
D= MEM, REG
COUNT =CL

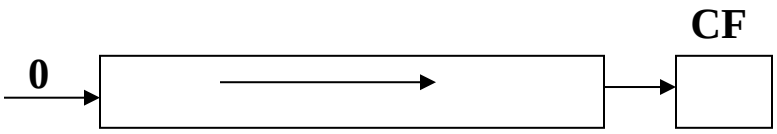


EX:-

ORG 100H
MOV AL,11100000B ; AL= E0H
SHL AL,1 ;AL=C0H
RET
Flag affected OF ,CF

2- SHR Shift logical Right

SHR D, COUNT
D= MEM, REG
COUNT= CL



EX:-
ORG 100H
MOV AL,00000111B ;AL=07H
SHR AL,1 ;AL=03
RET
Flag affected OF,CF

- **SHIFT ARITHMATIC**

1- SAL Shift Arithmetic Left

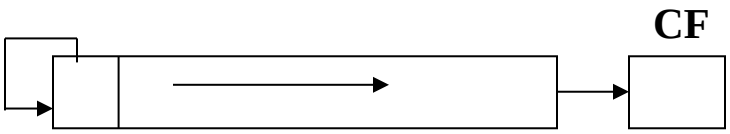
SAL D,COUNT
Same work as SHL

2- SAR Shift Arithmetic Right

SAR D,COUN

D= MEM,REG
COUNT =CL

FLAG affected OF,SF,ZF,AF,PF,CF



EX:-
ORG 100H
MOV AL,E0H; AL= 11100000B
SAR AL,1 ; AL= 11110000
MOV BL,4C ; BL=01001100B
SAR BL,1 ;BL= 00100110B
RET

- PUSH instruction

The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be .a general purpose register, segment register or a memory location

The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2. Push instruction does not .affect any flags

:Example

PUSH CX; Decrements SP by 2, copy content of CX to the stack

PUSH DS; Decrement SP by 2 and copy DS to stack

Push REG

SREG

MEM

IMM

- POP instruction

The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a

General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically .incremented by two

.The execution pattern is similar to that of the PUSH instruction

:Example

Copy a word from the top of the stack to CX and ; POP CX .increment SP by 2

Pushf no operand

Store flag register in stack

Algorithm

Sp = sp-2

Ss [sp] top of stack

Popf no operand

Get flag register from stack

Ex :

```
org 100h
mov ax,0ffffh
push ax
popf
pushf
pop ax
ret
```

PUSHF and POPF

are most used in writing interrupt service routines, where you must be able to save and restore the environment,

LAHF

LAHF no operand

Load ah from 8 low bits of flag register

ALGORITHM

AH = flag register

AH BIT

7	6	5	4	3	2	1	0
SF	ZF	0	AF	0	PF	1	CF

BITS 1,3,5 are reserved

SAHF

Store ah register in to 8 bit of flag register

SAHF no operand

Ex:-

MOV AH,0FFH; AH=FF

SAHF; CF,ZF,PF,AF,SF=1

LAHF ; AH=11010111=D7

REP INSTRUCTION

Repeat the following

MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW

ALGORITHM

Check CX

If CX<> 0

Do the following

CX=CX-1

GO to check CX

ELSE

Exit from rep cycle

Loop label Instruction

Decrease cx, jump to label if cx not zero

Algorithm

Cx=cx-1

If cx<>0 then

Jump

Else

No jump

Continue

EX write program to print ‘stage 2 ‘ 5 times on screen

```
Include ‘emu8086.inc’
Org 100h
Mov cx,5
Lable :
Print ’stage two’
Loop label
Ret
```

LEA Instruction – Load Effective Address

Examples of some interrupt instructions

1- INT 10h / AH=0Eh teletype output (print Al value)

Ex:

```
MOV AL, 'A'

MOV AH, 0Eh

INT 10h
```

2- INT 21h / AH=01h

Read character from standard input with echo, result is stored in AL.

EX:

```
MOV AH, 01h

INT 21h
```

3- INT 21h / AH=02h

Write character to standard output entry DL= character to write, after execution AL=DL

EX:

```
MOV AH, 02h

MOV DL, 'a'

INT 21h
```

Ex: Write assembly program to read five value from keyboard and store it in array

```
Org 100h
Mov ah, 01h
Mov cx, 5
aa: int 21h
Mov a[si], al
Inc si
Loop aa
Ret
a db 5 dup (0)
```

Ex: Write assembly program to print the value of array a where a= 1, 2, 3, 4, 5

```
Org 100h
Mov ah, 2
Mov cx, 5
aa: Mov Dl, a[si]
Int 21h
Inc si
Loop aa
Ret
a db 1, 2, 3, 4, 5
```

4- INT 33h (mouse driver interrupt)

- INT 33h / AX= 0001h (show mouse pointer)

Ex:

```
Mov ax, 1
Int 33h
```

-INT 33h / AX=0002h (Hide visible mouse pointer)

Ex:

```
Mov ax, 2
Int 33h
```