مجموعة التعليمات Instruction Set

للـ 8086 أنماط للتعليمات6

1. تعليمات نقل البيانات Data Transfer Instructions
2. التعليمات الحسابية Arithmetic Instructions
3. التعليمات المنطقية Logical Instructions
4. تعليمات معالجة السلاسل String manipulation Instructions
5. تعليمات التحكم بالعملية Process Control Instructions
6. تعليمات التحكم بالنقل Control Transfer Instructions

تعريف المعطيات يمكن تعريف المعطيات في لغة التجميع

تعرف المعطيات في مقطع data segment
يمكن أن تعرف اما byte DB
أو كلمة من 16 bit Word
من 32 bit double Word

- Fivfor initialized data

```
DB   Define Byte  e define directives      ;allocates 1 byte
DW   Define Word            ;allocates 2 bytes
DD   Define Doubleword      ;allocates 4 bytes
DQ   Define Quadword        ;allocates 8 bytes
DT   Define Ten bytes       ;allocates 10 bytes
```

Examples

```
sorted     DB    'y'
value      DW    25159
Total      DD    542803535
float1     DD    1.234
```

تعريف مصفوفة أحادية البعد من 8 عناصر

- Multiple definitions can be cumbersome to initialize data structures such as arrays

Example

To declare and initialize an integer array of 8 elements

```
marks   DW    0,0,0,0,0,0,0,0
```

- What if we want to declare and initialize to zero an array of 200 elements?
  - There is a better way of doing this than repeating zero 200 times in the above statement
    - Assembler provides a directive to do this (DUP directive)

* Examples
  - Previous marks array

```
marks   DW    0,0,0,0,0,0,0,0
```

can be compactly declared as

```
marks   TIMES   8   DW   0
```

Symbol Table

* Assembler builds a symbol table so we can refer to the allocated storage space by the associated label

Example

```
.DATA
value     DW  0
sum       DD  0
marks     DW  10 DUP (?)
message   DB  'The grade is:',0
char1     DB  ?
```

name	offset
value	0
sum	2
marks	6
message	26
char1	40

- Directives for uninitialized data
- Five reserve directives

```
RESB   Reserve a Byte        ;allocates 1 byte
RESW   Reserve a Word        ;allocates 2 bytes
RESD   Reserve a Doubleword  ;allocates 4 bytes
RESQ   Reserve a Quadword    ;allocates 8 bytes
REST   Reserve a Ten bytes   ;allocates 10 bytes
```

Examples

```
response   resb   1
buffer     resw   100
Total      resd   1
```

- Multiple definitions can be abbreviated

Example

```
message   DB   'B'
          DB   'y'
          DB   'e'
          DB   0DH
          DB   0AH
```

can be written as

```
message   DB
'B','y','e',0DH,0AH
```

- **More compactly as**

```
message   DB 'Bye',0DH,0A
```

جَامعة المَنارة

The mov instruction

* Five types of operand combinations are allowed:

Instruction type	Example
`mov register,register`	`mov DX,CX`
`mov register,immediate`	`mov BL,100`
`mov register,memory`	`mov EBX,[count]`
`mov memory,register`	`mov [count],ESI`
`mov memory,immediate`	`mov [count],23`

جَامعة المَنارة
MANARA UNIVERSITY

مثال :
حدد الأخطاء في البرنامج التالي

```
.data
bVal   BYTE    100
bVal2  BYTE    ?
wVal   WORD    2
dVal   DWORD   5
.code
    mov ds,45
    mov esi,wVal
    mov eip,dVal
    mov 25,bVal
    mov bVal2,bVal
```

immediate move to DS not permitted
size mismatch

لا يمكن أن يكون المسجل هدف IP or eip

immediate value cannot be destination

memory-to-memory move not permitted

```
mov  al,48
mov  bl,4
imul bl              ; AX = 00C0h, OF=1
```

```
mov ax,8760h
mov bx,100h
imul bx
```

DX = FF87h, AX = 6000h, OF = 1

```
mov eax,00128765h
mov ecx,10000h
mul ecx
```

EDX = 00000012h, EAX = 87650000h, CF = 1

DIV Examples

Divide 8003h by 100h, using 16-bit operands:

```
mov dx,0                    ; clear dividend, high
mov ax,8003h                ; dividend, low
mov cx,100h                 ; divisor
div cx                      ; AX = 0080h, DX = 3
```

مثال : ماهو مجتوى المسجلين Ax , dx بعد بنقيذ عملية القسمة

```
mov dx,0087h
mov ax,6000h
mov bx,100h
div bx
```

DX = 0000h, AX = 8760h

CBW, CWD, CDQ Instructions

- **The CBW, CWD, and CDQ instructions provide important sign-extension operations**:
  - CBW (convert byte to word) extends AL into AH
  - CWD (convert word to doubleword) extends AX into DX
  - CDQ (convert doubleword to quadword) extends EAX into EDX
- Example:

```
mov eax,0FFFFFF9Bh        ; (-101)
cdq                ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

Example: 16-bit division of –48 by 5

```
mov  ax,-48
cwd                ; extend AX into DX
mov  bx,5
idiv bx            ; AX = -9,  DX = -3
```

Example: 8-bit division of –48 by 5

```
mov  al,-48
cbw                ; extend AL into AH
mov  bl,5
idiv bl            ; AL = -9,  AH = -3
```

Example: `eax = (-var1 * var2) + var3`

```
mov   eax,var1
neg   eax
imul  var2
jo    TooBig            ; check for overflow
add   eax,var3
jo    TooBig            ; check for overflow
```

Example: `var4 = (var1 + var2) * var3`

```
; Assume unsigned operands
mov   eax,var1
add   eax,var2          ; EAX = var1 + var2
mul   var3              ; EAX = EAX * var3
jc    TooBig            ; check for carry
mov   var4,eax          ; save product
```

Example: `var4 = (var1 * 5) / (var2 - 3)`

```
mov   eax,var1          ; left side
mov   ebx,5
imul  ebx               ; EDX:EAX = product
mov   ebx,var2          ; right side
sub   ebx,3
idiv  ebx               ; EAX = quotient
mov   var4,eax
```

مثال اكتب مقطع برمجي لتحقيق المعادلة التالية لأعداد ب اشارة من 32 bit

```
eax = (ebx * 20) / ecx
```

```
mov eax,20
imul ebx
idiv ecx
```

Example: `var4 = (var1 * -5) / (-var2 % var3);`

```
mov   eax,var2          ; begin right side
neg   eax
cdq                     ; sign-extend dividend
idiv  var3              ; EDX = remainder
mov   ebx,edx           ; EBX = right side
mov   eax,-5            ; begin left side
imul  var1              ; EDX:EAX = left side
idiv  ebx               ; final division
mov   var4,eax          ; quotient
```

العمليات المنطقية

AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16

AND reg/mem, data AND reg, data	(reg) ← (reg) & data
AND mem, data	(mem) ← (mem) & data

```
mov al,'a' ;                AL = 01100001b
and al,11011111b;           AL = 01000001b
```

Logical Instructions التعليمات المنطقية

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …

OR reg2/mem, reg1/mem OR reg2, reg1	(reg2) ← (reg2) | (reg1)
OR reg2, mem	(reg2) ← (reg2) | (mem)
OR mem, reg1	(mem) ← (mem) | (reg1)

```
mov al,6 ;              AL = 00000110b
or  al,00110000b;       AL = 00110110b
```

OR reg/mem, data OR reg, data	(reg) ← (reg) | data
OR mem, data	(mem) ← (mem) | data

OR A, data OR AL, data8	(AL) ← (AL) | data8
OR AX, data16	(AX) ← (AX) | data16

XOR reg2/mem, reg1/mem	
XOR reg2, reg1	(reg2) ← (reg2) ^ (reg1)
XOR reg2, mem	(reg2) ← (reg2) ^ (mem)
XOR mem, reg1	(mem) ← (mem) ^ (reg1)

XOR reg/mem, data	
XOR reg, data	(reg) ← (reg) ^ data
XOR mem, data	(mem) ← (mem) ^ data

XOR A, data	
XOR AL, data8	(AL) ← (AL) ^ data8
XOR AX, data16	(AX) ← (AX) ^ data16

مجموعة التعليمات Instruction Set

تعليمات التحكم بالعملية Processor Control Instructions

Mnemonics	Explanation
STC	Set CF ← 1
CLC	Clear CF ← 0
CMC	Complement carry CF ← CF/
STD	Set direction flag DF ← 1
CLD	Clear direction flag DF ← 0
STI	Set interrupt enable flag IF ← 1
CLI	Clear interrupt enable flag IF ← 0
NOP	No operation
HLT	Halt after interrupt is set
WAIT	Wait for TEST pin active
ESC opcode mem/ reg	Used to pass instruction to a coprocessor which shares the address and data bus with the 8086
LOCK	Lock bus during next instruction

Control Transfer Instructions تعليمات نقل التحكم

- Transfer the control to a specific destination or target instruction تنقل التحكم إلى تعليمة هدف محدد أو مصدر محدد
- Do not affect flags. لا تؤثر على الأعلام

❑ 8086 Unconditional transfers نقل غير مشروط

Mnemonics	Explanation
CALL reg/ mem/ disp16	Call subroutine
RET	Return from subroutine
JMP reg/ mem/ disp8/ disp16	Unconditional jump

❑ تعليمات التفريع المشروط مع الإشارة
❑ 8086 signed conditional branch instructions

❑ تعليمات التفريع المشروط دون الإشارة
❑ 8086 signed conditional branch instructions

- اختبار الأعلام Checks flags
- إذا كان الشروط محققة يجري نقل التحكم بالبرنامج إلى موقع جديد في الذاكرة في نفس القطاع من خلال تعديل محتوى IP.
- If conditions are true, the program control is transferred to the new memory location in the same segment by modifying the content of IP

TEST Instruction

تعليمات القفز المشرط conditional JUMP

تعمل عمل بوابة AND ولكن لا تعيد نتيجة تؤثر على علم الصفر ZF

- A conditional jump instruction branches to a label when specific register or flag conditions are met
- Examples:
  - JB, JC jump to a label if the Carry flag is set
  - JE, JZ jump to a label if the Zero flag is set
  - JS jumps to a label if the Sign flag is set
  - JNE, JNZ jump to a label if the Zero flag is clear
  - JECXZ jumps to a label if ECX equals 0

```
test al,00000011b
jnz  ValueFound
```

❑ تعليمات التفريع المشروط دون الإشارة

❑ 8086 signed conditional branch instructions

Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JA disp8 Jump if above	JNBE disp8 Jump if not below or equal
JAE disp8 Jump if above or equal	JNB disp8 Jump if not below
JB disp8 Jump if below	JNAE disp8 Jump if not above or equal
JBE disp8 Jump if below or equal	JNA disp8 Jump if not above

❑ تعليمات التفريع المشروط مع الإشارة

❑ 8086 signed conditional branch instructions

Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JG disp8 Jump if greater	JNLE disp8 Jump if not less or equal
JGE disp8 Jump if greater than or equal	JNL disp8 Jump if not less
JL disp8 Jump if less than	JNGE disp8 Jump if not greater than or equal
JLE disp8 Jump if less than or equal	JNG disp8 Jump if not greater

القفز المشروط يتم في التعليمات اختبار بعض الأعلام

Jump Instructions

Conditional Jump

* Conditional jump instructions can also test values of the individual flags

jz	**jump if zero** (i.e., if ZF = 1)
jnz	**jump if not zero** (i.e., if ZF = 0)
jc	**jump if carry** (i.e., if CF = 1)
jnc	**jump if not carry** (i.e., if CF = 0)

* **jz** is synonymous for **je**
* **jnz** is synonymous for **jne**

جَامعة المَنارة
MANARA UNIVERSITY

Jump Instructions

Conditional Jump

* Some conditional jump instructions
  - Treats operands of the CMP instruction as signed numbers

```
je      jump if equal
jg      jump if greater
jl      jump if less
jge     jump if greater or equal
jle     jump if less or equal
jne     jump if not equal
```

LOOP Instruction

What will be the final value of AX?

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
```

Ax =10

- The LOOP instruction creates a counting loop
- Syntax: LOOP *target*
- Logic:
  - ECX ← ECX – 1
  - if ECX != 0, jump to *target*
- Implementation:
  - The assembler calculates the distance, in bytes, between the offset of the following instruction and the offset of the target label. It is called the relative offset.
  - The relative offset is added to EIP.

LOOPZ and LOOPE

A jump outside the current procedure must be to a special type of label called a global label

JMP Instruction

- JMP is an unconditional jump to a label that is usually within the same procedure.
- Syntax: JMP *target*
- Logic: IP ← *target*
- Example:

```
top:
    .
    .
    jmp top
```

Loop Instruction

LOOP Instruction

* Format:

```
loop  target
```

* Semantics:
  - » Decrements ECX and jumps to target if ECX ≠ 0
    - – ECX should be loaded with a loop count value

- **Example:** Executes loop body 50 times

```
        mov    ECX,50
    repeat:
        <loop body>
        loop   repeat
           ...
```

LOOPNZ and LOOPNE

- LOOPNZ (LOOPNE) is a conditional loop instruction
- Syntax:
  - LOOPNZ *destination*
  - LOOPNE *destination*
- Logic:
  - ECX ← ECX – 1;
  - if ECX > 0 and ZF=0, jump to *destination*
- Useful when scanning an array for the first element that matches a given value.

LOOPZ and LOOPE

- Syntax:
  - LOOPE *destination*
  - LOOPZ *destination*
- Logic:
  - ECX ← ECX – 1
  - if ECX > 0 and ZF=1, jump to *destination*
- Useful when scanning an array for the first element that does not match a given value.

مثال اكتب المقطع البرمجي الموافق للعلاقة التالية

مثال

```
if( op1 == op2 )
  X = 1;
else
  X = 2;
```

```
mov eax,op1
cmp eax,op2
jne L1
mov X,1
jmp L2
L1:mov X,2
L2:
```

مثال

```
if( ebx <= ecx )
{
  eax = 5;
  edx = 6;
```

المقطع البرمجي الموافق للعلاقة

الحل

```
cmp ebx,ecx
ja  next
mov eax,5
mov edx,6
next:
```

جَامعة المَنارة
MANARA UNIVERSITY

مثال اكتب برنامج لإيجاد موقع أول عنصر موجب في مصفوفة أحادية البعد

```
.data
array SWORD -3,-6,-1,-10,10,30,40,4
sentinel SWORD 0
.code
mov esi,OFFSET array
mov ecx,LENGTHOF array
next:
test WORD PTR [esi],8000h      ; test sign bit
pushfd                         ; push flags on stack
add esi,TYPE array
popfd                          ; pop flags from stack
loopnz next                    ; continue loop
jnz quit                       ; none found
sub esi,TYPE array             ; ESI points to value
quit:
```

قيمة تتم اختبار كل عنصر من عناصر المصفوفة مع القيمةH 8000في الخانة العليا تدل على العد د سالب

```
if (a1 > b1) AND (b1 > c1)
  X = 1;
```

```
    cmp a1,b1                ; first expression...
    ja  L1
    jmp next
L1:
    cmp b1,c1                ; second expression...
    ja  L2
    jmp next
L2:                          ; both are true
    mov X,1                  ; set X to 1
next:
```

```
if( var1 <= var2 )
  var3 = 10;
else
{
  var3 = 6;
  var4 = 7;
}
```

```
mov eax,var1
cmp eax,var2
jle L1
mov var3,6
mov var4,7
jmp L2
L1: mov var3,10
L2:
```

```
if (al > bl) AND (bl > cl)
   X = 1;
```

```
    cmp al,bl                ; first expression...
    jbe next                 ; quit if false
    cmp bl,cl                ; second expression...
    jbe next                 ; quit if false
    mov X,1                  ; both are true
next:
```

```
if( ebx <= ecx
    && ecx > edx )
{
   eax = 5;
   edx = 6;
}
```

```
while( ebx <= val1)
{
    ebx = ebx + 5;
    val1 = val1 - 1
}
```

```
top: cmp ebx,val1;     check loop condition
     ja  next;         false? exit loop
     add ebx,5;        body of loop
     dec val1
     jmp top;        repeat the loop
next:
```

```
cmp ebx,ecx
ja  next
cmp ecx,edx
jbe next
mov eax,5
mov edx,6
next:
```

Shift Instructions التعليمات الازاحة

Mnemonics:, SHR, SHL, RCR, RCL …

SHR reg/mem

SHR reg

i) SHR reg, 1

ii) SHR reg, CL

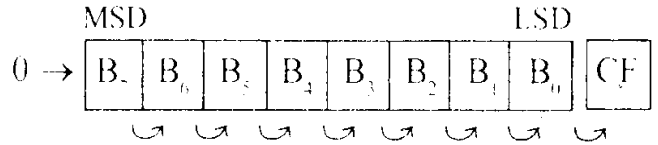
SHR mem

i) SHR mem, 1

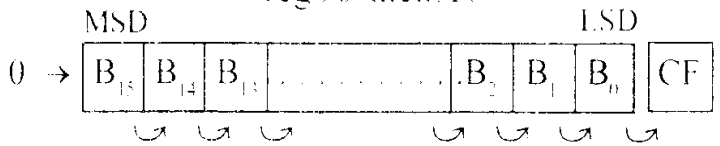
ii) SHR mem, CL

$$CF \leftarrow B_{LSD} ; B_n \leftarrow B_{n+1} ; B_{MSD} \leftarrow 0$$

reg 8 / mem 8



reg 16 / mem 16



تعليمات الازاحة
تقسم تعلمات الازاحة الى:

- SHR تعليمات ازاحة منطقية الى اليمين

Shift Instructions تعليمات الازاحة

Mnemonics: SHR, SHL, RCR, RCL …

SHL تعليمات ازاحة منطقية الى اليسار

SHL reg/mem or SAL reg/mem

SHL reg or SAL reg

i) SHL reg, 1 or SAL reg, 1

ii) SHL reg, CL or SAL reg, CL

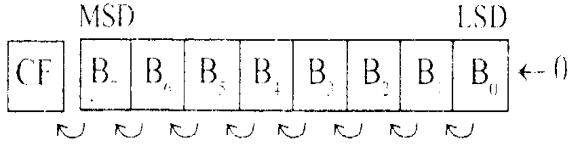
SHL mem or SAL mem

i) SHL mem, 1 or SAL mem, 1

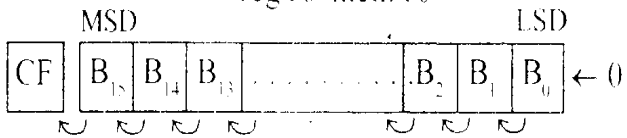
ii) SHL mem, CL or SAL mem, CL

$CF \leftarrow B_{MSD} ; B_{n+1} \leftarrow B_n ; B_{LSD} \leftarrow 0$

reg 8 / mem 8



reg 16 / mem 16



Instruction	Before shift AL or AX	After shift AL or AX	After shift CF
shl AL,1	1010 1110	0101 1100	1
shr AL,1	1010 1110	0101 0111	0
mov CL,3 shl AL,CL	0110 1101	0110 1000	1
mov CL,5 shr AX,CL	1011 1101 0101 1001	0000 0101 1110 1010	1

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …

RCR تعليمات ازاحة منطقية دورانية الى اليمين

RCR reg/mem

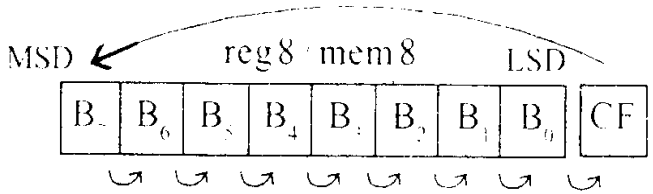
RCR reg

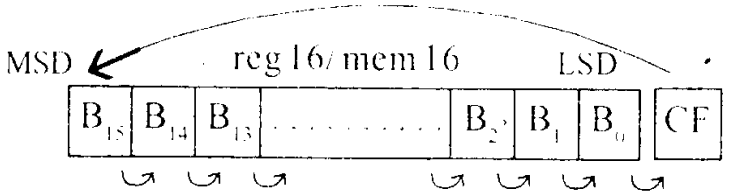
i) RCR reg, 1
ii) RCR reg, CL

RCR mem

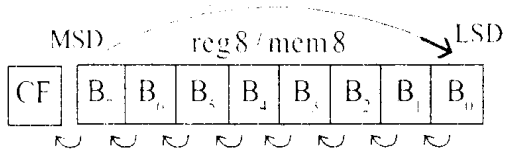
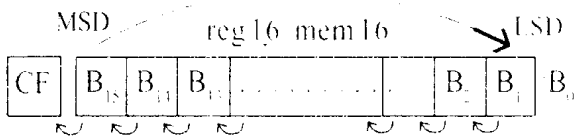
i) RCR mem, 1
ii) RCR mem, CL

$$B_n \leftarrow B_{n-1} ; B_{MSD} \leftarrow CF ; CF \leftarrow B_{LSD}$$





ROL تعليمات ازاحة منطقية دورانية الى اليسار

ROL reg/mem ROL reg i) ROL reg, 1 ii) ROL reg, CL	$B_{n+1} \leftarrow B_n : CF \leftarrow B_{MSD} ; B_{LSD} \leftarrow B_{MSD}$ 
ROL mem i) ROL mem, 1 ii) ROL mem, CL	

Instruction	Before execution	After execution	
	AL or AX	AL or AX	CF
rol AL,1	1010 1110	0101 1101	1
ror AL,1	1010 1110	0101 0111	0
mov CL,3 rol AL,CL	0110 1101	0110 1011	1
mov CL,5 ror AX,CL	1011 1101 0101 1001	1100 1101 1110 1010	1

SIZEOF Operator

معامل يعد عدد عناصر المصفوفة × نوع البيانات المصرح عنها

The SIZEOF operator returns a value that is equivalent to multiplying LENGTHOF by TYPE.

```
.data                                          SIZEOF
byte1  BYTE 10,20,30                           ; 3
array1 WORD 30 DUP(?),0,0                      ; 64
array2 WORD 5 DUP(3 DUP(?))                    ; 30
array3 DWORD 1,2,3,4                           ; 16
digitStr BYTE "12345678",0                     ; 9

.code
mov ecx,SIZEOF array1                          ; 64
```

Summing an Integer Array

The following code calculates the sum of an array of 16-bit integers.

مثال

اكتب برنامج لحساب مجموع عناصر مصفوفةالأعداد الصحيحة من16 bit

```
.data
intarray WORD 100h,200h,300h,400h
.code
mov edi,OFFSET intarray          ; address of intarray
mov ecx,LENGTHOF intarray        ; loop counter
mov ax,0                         ; zero the accumulator
L1:
add ax,[edi]                     ; add an integer
add edi,TYPE intarray            ; point to next integer
   loop L1                       ; repeat until ECX = 0
```

مثال

LENGTHOF Operator

معامل يعد عدد عناصر المصفوفة المصرح عنها

```
.data
array WORD 10,2,30,40,50,60
.code
mov eax,LENGTHOF array                    ; 6
mov ebx,SIZEOF array                      ; 12
```

The LENGTHOF operator counts the number of elements in a single data declaration.

```
.data                                              LENGTHOF
byte1   BYTE 10,20,30                              ; 3
array1  WORD 30 DUP(?),0,0                         ; 32
array2  WORD 5 DUP(3 DUP(?))                       ; 15
array3  DWORD 1,2,3,4                              ; 4
digitStr BYTE "12345678",0                         ; 9

.code
mov ecx,LENGTHOF array1                            ; 32
```