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

مجموعة التعليمات Instruction Set

1. تعليمات نقل البيانات

تستخدم تعليمات هذا النمط لنقل البيانات /العناوين من وإلى المسجلات ومن وإلى مواقع الذاكرة ومن وإلى بوابات I/O.
Instructions that are used to transfer data/ address in to registers, memory locations and I/O ports.

المستهدف بشكل عام هو حدين two operands: حدّي المصدر Source والهدف نبفس الحجم same size.

المصدر Source: مسجل أو موقع ذاكرة أو بيانات فورية immediate data.
الهدف Destination: مسجل أو موقع ذاكرة.

بحجم بايت أو كلمة byte or a word.

يمكننا نقل بيانات بحجم 8-bit إلى مسجلات بحجم ومواقع ذاكرة بنفس الحجم والبيانات بحجم 16-bit إلى مسجلات وماقع ذاكرة بنفس الحجم.

تعريف المعطيات يمكن تعريف المعطيات في لغة التجميع

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

تعليمات نقل البيانات
Data Transfer Instructions

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT …

MOV reg2/ mem, reg1/ mem	
MOV reg2, reg1	**(reg2) ← (reg1)**
MOV mem, reg1	**(mem) ← (reg1)**
MOV reg2, mem	**(reg2) ← (mem)**
MOV reg/ mem, data	
MOV reg, data	**(reg) ← data**
MOV mem, data	**(mem) ← data**
XCHG reg2/ mem, reg1	
XCHG reg2, reg1	**(reg2) ↔ (reg1)**
XCHG mem, reg1	**(mem) ↔ (reg1)**

تعليمة Mov
نقل من مسجل الى مسجل أخر
نقل من مسجل الى ذاكرة
نقل من ذاكرة الى مسجل

تعليمة Mov
نقل معطيات الى مسجل أخر
نقل معطيات الى موقع ذاكرة

تعليمة تبديل XCHG
تبديل محتوى مسجل بمحتوى مسجل أخر
تبديل محتوى مسجل بمحتوى موقع ذاكرة

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`

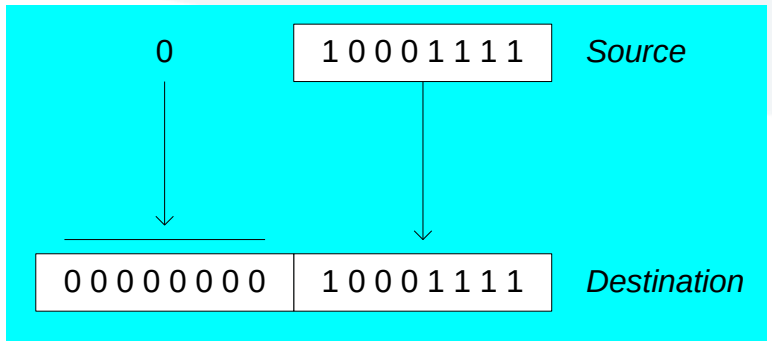
مثال :
حدد الأخطاء في البرنامج التالي

```
.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

Zero Extension

MOVZX instruction fills (extends) the upper half of the destination with zeros.



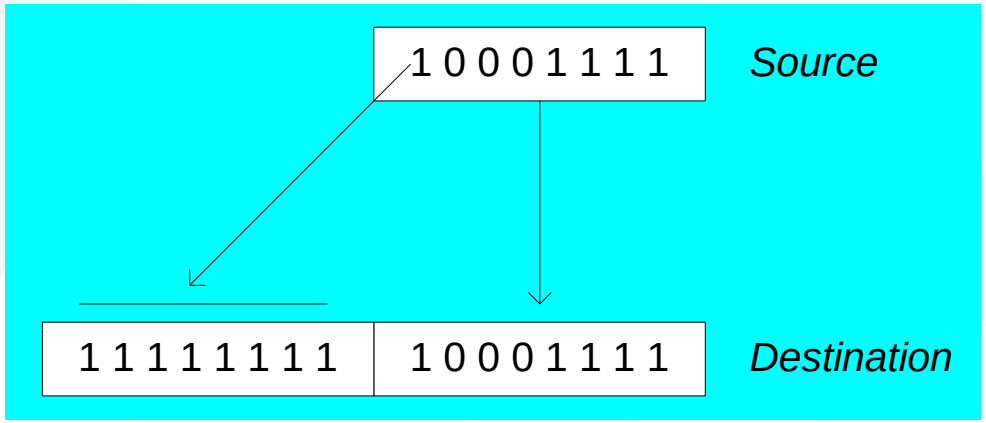

```
mov bl,10001111b
movzx ax,bl                    ; zero-extension
```

The destination must be a register.

ملاحظة يجب أن يكون الهدف مسجل

Sign Extension

The MOVSX instruction fills the upper half of the destination with a copy of the source operand's sign bit.




```
mov bl,10001111b
movsx ax,bl                    ; sign extension
```

XCHG Instruction

XCHG exchanges the values of two operands. At least one operand must be a register. No immediate operands are permitted.

```
.data
var1 WORD 1000h
var2 WORD 2000h
.code
xchg ax,bx                ; exchange 16-bit regs
xchg ah,al                ; exchange 8-bit regs
xchg var1,bx              ; exchange mem, reg
xchg eax,ebx              ; exchange 32-bit regs
```

تعليمة XCHG

تسمح بتبديل محتوى سجل بسجل آخر ولا تسمح بتبديل قيمة فورية

ملاحظة هامة لاتسمح بتبديل محتوى ذاكرة بمحتوى موق ذاكر آخر

مثال

```
xchg var1,var2    ;
error: two memory operands
```

تعليمات نقل البيانات

1. Data Transfer Instructions

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT …

PUSH reg16/ mem	
PUSH reg16	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (reg16)$
PUSH mem	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (mem)$

POP reg16/ mem	
POP reg16	$MA_S = (SS) \times 16_{10} + SP$ $(reg16) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$
POP mem	$MA_S = (SS) \times 16_{10} + SP$ $(mem) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$

مثال :

بفرض AX =3FC4

والمسجلات SS = 0050 , SP = 1234

PUSH AX;

الحل : push 21ABH

نحسب العنوان الفزيائي لذاكرة STACK

SS:SP

00500

1234 +

01734

دفع القيمة 21AB في ذاكرة المسجل تحت العنوان

01734

1- دفع مسجل الأعلام في ذاكرة STACK

- PUSHFD and POPFD
- **pushfw** and **popfw** for 16-bit flags (FLAGS)

```
pushfd (push 32-bit flags)
popfd  (pop 32-bit flags)
```

2- تعليمة دفع 8 مسجلات 16 bit في Stack

PUSHA and POPA

AX, CX, DX, BX, SP, BP, SI, DI

3- تعليمة دفع 8 مسجلات 32 bit في ذاكرة Stack

PUSHAD pushes the 32-bit general-purpose registers on the stack

order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI

POPAD pops the same registers off the stack in reverse order

Instruction Set مجموعة التعليمات

Mnemonics: **MOV, XCHG, PUSH, POP, IN, OUT …**

IN A, [DX]	
IN AL, [DX]	$PORT_{addr}$ = (DX) (AL) ← (PORT)
IN AX, [DX]	$PORT_{addr}$ = (DX) (AX) ← (PORT)
IN A, addr8	
IN AL, addr8	(AL) ← (addr8)
IN AX, addr8	(AX) ← (addr8)

OUT [DX], A	
OUT [DX], AL	$PORT_{addr}$ = (DX) (PORT) ← (AL)
OUT [DX], AX	$PORT_{addr}$ = (DX) (PORT) ← (AX)
OUT addr8, A	
OUT addr8, AL	(addr8) ← (AL)
OUT addr8, AX	(addr8) ← (AX)

2. Arithmetic Instructions التعليمات الحسابية .2

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP

ADD reg/mem, data
ADD reg, data
ADD mem, data
	(reg) ← (reg)+ data
	(mem) ← (mem)+data

ADD A, data
 	
ADD AL, data8
ADD AX, data16
	
	(AL) ← (AL) + data8
	(AX) ← (AX) +data16

تعليمة الجمع ADD

مثال

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code
    mov eax,var1 ;        EAX = 10000h
    add eax,var2 ;        EAX = 30000h
```

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code                          ; ---EAX---
    mov eax,var1               ; 00010000h
    add eax,var2               ; 00030000h
    mov ax,0FFFFh              ; 0003FFFFh
    add eax,1                  ; 00040000h
    sub ax,1                   ; 0003FFFFh
```

Arithmetic Instructions التعليمات الحسابية

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

ADC reg2/ mem, reg1/mem	
ADC reg2, reg1	(reg2) ← (reg1) + (reg2)+CF
ADC reg2, mem	(reg2) ← (reg2) + (mem)+CF
ADC mem, reg1	(mem) ← (mem)+(reg1)+CF
ADC reg/mem, data	
ADC reg, data	(reg) ← (reg)+ data+CF
ADC mem, data	(mem) ← (mem)+data+CF
ADDC A, data	
ADD AL, data8	(AL) ← (AL) + data8+CF
ADD AX, data16	(AX) ← (AX) +data16+CF

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

SUB reg2/ mem, reg1/mem	

SUB reg2, reg1
SUB reg2, mem
SUB mem, reg1
	(reg2) ← (reg1) - (reg2)
	(reg2) ← (reg2) - (mem)
	(mem) ← (mem) - (reg1)

SUB reg/mem, data	
SUB reg, data SUB mem, data	(reg) ← (reg) - data (mem) ← (mem) - data
SUB A, data	
SUB AL, data8 SUB AX, data16	(AL) ← (AL) - data8 (AX) ← (AX) - data16

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

SBB reg2/ mem, reg1/mem	
SBB reg2, reg1 SBB reg2, mem SBB mem, reg1	(reg2) ← (reg1) - (reg2) - CF (reg2) ← (reg2) - (mem)- CF (mem) ← (mem) - (reg1) –CF
SBB reg/mem, data	
SBB reg, data SBB mem, data	(reg) ← (reg) – data - CF (mem) ← (mem) - data - CF
SBB A, data	
SBB AL, data8 SBB AX, data16	(AL) ← (AL) - data8 - CF (AX) ← (AX) - data16 - CF

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

INC reg/ mem	
INC reg8	(reg8) ← (reg8) + 1
INC reg16	(reg16) ← (reg16) + 1
INC mem	(mem) ← (mem) + 1

DEC reg/ mem	
DEC reg8	(reg8) ← (reg8) - 1
DEC reg16	(reg16) ← (reg16) - 1
DEC mem	(mem) ← (mem) - 1

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
    inc myWord              ; 1001h
    dec myWord              ; 1000h
    inc myDword             ; 10000001h

    mov ax,00FFh
    inc ax                  ; AX = 0100h
    mov ax,00FFh
    inc al                  ; AX = 0000h
```

```
.data
myByte BYTE 0FFh, 0
.code
    mov al,myByte           ; AL = FFh
    mov ah,[myByte+1]       ; AH = 00h
    dec ah                  ; AH = FFh
    inc al                  ; AL = 00h
    dec ah                  ; Ah = FE
```

NEG (negate) Instruction

The NEG (negate) instruction reverses the sign of a number by converting the number to its two's complement

```
.data
valB BYTE -1
valW WORD +32767
.code
    mov al,valB                 ; AL = -1
    neg al                      ; AL = +1
    neg valW                    ; valW = -32767
```

Example

mov al,-128 ; AL = 10000000b

neg al ; AL = 10000000b, OF = 1

مثال اكتب برنامج بلغة Assembly لحل المعادلة

```
Rval = -Xval + (Yval - Zval)
```

حيث أن المتغييرات التالية : 32 bit

```
Rval ,Xval ,Yval , Zval
```

```
Rval DWORD ?
Xval DWORD 26
Yval DWORD 30
Zval DWORD 40
.code
    mov eax,Xval
    neg eax                     ; EAX = -26
    mov ebx,Yval
    sub ebx,Zval                ; EBX = -10
    add eax,ebx
    mov Rval,eax                ; -36
```