جَامعة
المَنارة
MANARA UNIVERSITY

https://manara.edu.sy/

جَامعة
المَنارة
MANARA UNIVERSITY

ADDRESSING MODES

Introduction

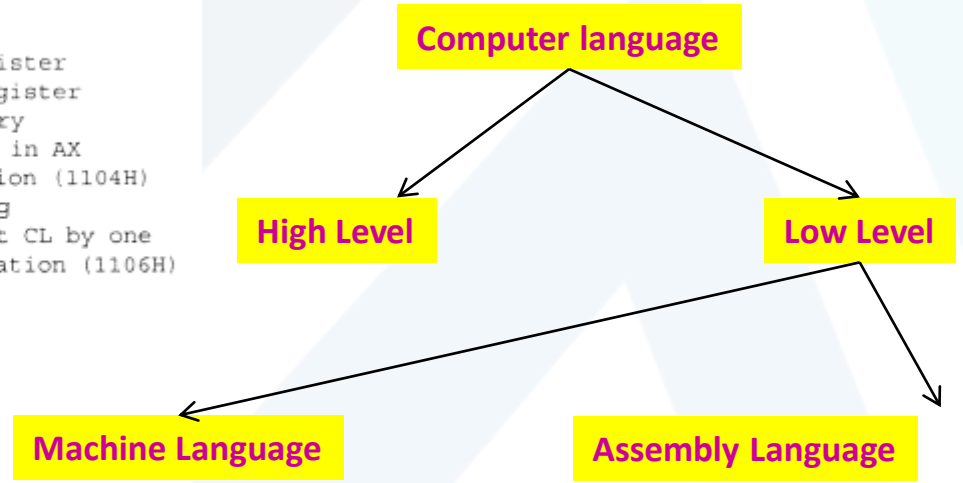
```
;PROGRAM TO ADD TWO 16-BIT DATA (METHOD-1)

DATA SEGMENT            ;Assembler directive

        ORG 1104H       ;Assembler directive
        SUM DW 0        ;Assembler directive
        CARRY DB 0      ;Assembler directive

DATA ENDS               ;Assembler directive

CODE SEGMENT            ;Assembler directive

        ASSUME CS:CODE  ;Assembler directive
        ASSUME DS:DATA  ;Assembler directive
        ORG 1000H       ;Assembler directive

        MOV AX,205AH    ;Load the first data in AX register
        MOV BX,40EDH    ;Load the second data in BX register
        MOV CL,00H      ;Clear the CL register for carry
        ADD AX,BX       ;Add the two data, sum will be in AX
        MOV SUM,AX      ;Store the sum in memory location (1104H)
        JNC AHEAD       ;Check the status of carry flag
        INC CL          ;If carry flag is set,increment CL by one
AHEAD:  MOV CARRY,CL    ;Store the carry in memory location (1106H)
        HLT

CODE ENDS               ;Assembler directive
END                     ;Assembler directive
```

Program

A set of instructions written to solve a problem.

Instruction

Directions which a microprocessor follows to execute a task or part of a task.



- **Binary bits**

- **English Alphabets**
- **'Mnemonics'**
- **Assembler** Mnemonics → Machine Language

جَامعة
المَنارة
MANARA UNIVERSITY

ADDRESSING MODES

Addressir

- **Every instruction of a program has to operate on a data.**
- **The different ways in which a source operand is denoted in an instruction are known as addressing modes.**

1. Register Addressing 2. Immediate Addressing	Group I : Addressing modes for register and immediate data
3. Direct Addressing 4. Register Indirect Addressing 5. Based Addressing 6. Indexed Addressing 7. Based Index Addressing 8. String Addressing	Group II : Addressing modes for memory data
9. Direct I/O port Addressing 10. Indirect I/O port Addressing	Group III : Addressing modes for I/O ports
11. Relative Addressing	Group IV : Relative Addressing mode
12. Implied Addressing	Group V : Implied Addressing mode

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

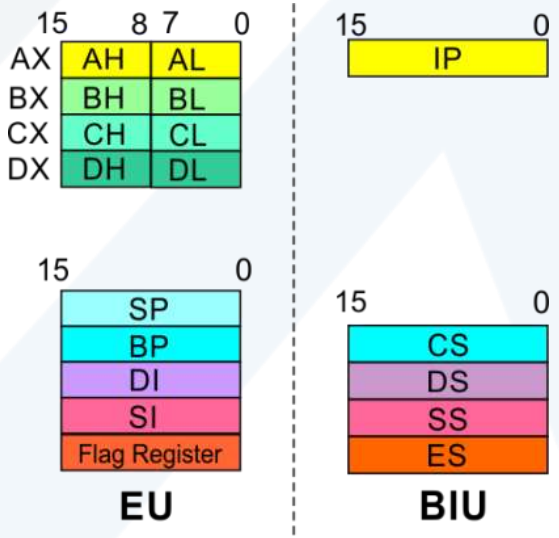
The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

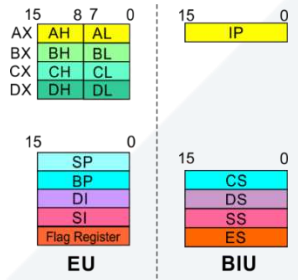
The content of 8-bit register DH is moved to another 8-bit register CL

(CL) ← (DH)



Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction

Example:

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

(DL) ← 08_H

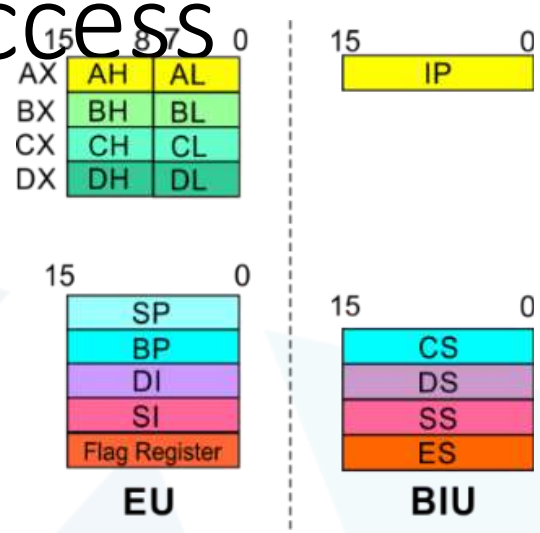
MOV AX, 0A9FH

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX register

(AX) ← $0A9F_H$

Addressing Modes : Memory Access

- 20 Address lines ⇒ 8086 can address up to 2^{20} = 1M bytes of memory
- However, the largest register is only 16 bits
- Physical Address will have to be calculated **Physical Address : Actual address of a byte in memory. i.e. the value which goes out onto the address bus.**
- Memory Address represented in the form – **Seg : Offset** (Eg - 89AB:F012)
- Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20- bit address

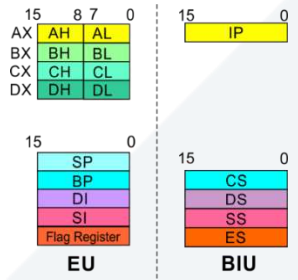


16 bytes of contiguous memory

89AB : F012 → 89AB → 89AB0 (Paragraph to byte → 89AB x 10 = 89AB0)
F012 → 0F012 (Offset is already in byte unit)
+ -------
98AC2 (The absolute address)

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

The effective address is just a 16-bit number written directly in the instruction.

Example:

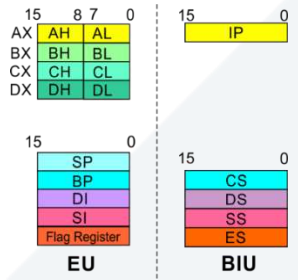
```
MOV   BX, [1354H]
MOV   BL, [0400H]
```

The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers:

BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example:

MOV CX, [BX]

Operations:

$EA = (BX)$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

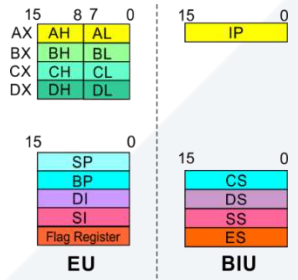
$(CX) \leftarrow (MA)$ or,

$(CL) \leftarrow (MA)$
$(CH) \leftarrow (MA + 1)$

Note : Register/ memory enclosed in brackets refer to content of register/ memory

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In Based Addressing, BX or BP is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.

When BP holds the base value of EA, BP and SS is used.

Example:

MOV AX, [BX + 08H]

Operations:

$0008_H \leftarrow 08_H$ (Sign extended)
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA

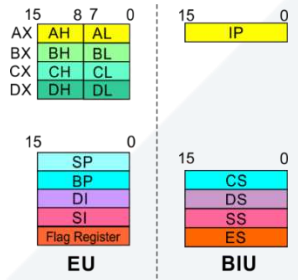
(AX) ← (MA) or,

(AL) ← (MA)
(AH) ← (MA + 1)

https://manara.edu.sy/

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:

MOV CX, [SI + 0A2H]

Operations:

$FFA2_H \leftarrow A2_H$ (Sign extended)

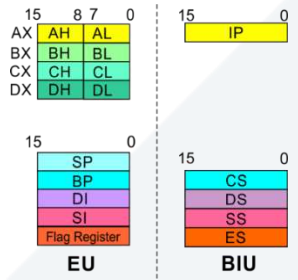
$EA = (SI) + FFA2_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

$(CX) \leftarrow (MA)$ or,

$(CL) \leftarrow (MA)$
$(CH) \leftarrow (MA + 1)$

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example:

MOV DX, [BX + SI + 0AH]

Operations:

$000A_H \leftarrow 0A_H$ (Sign extended)

$EA = (BX) + (SI) + 000A_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

$(DX) \leftarrow (MA)$ or,

$(DL) \leftarrow (MA)$
$(DH) \leftarrow (MA + 1)$

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

Note : Effective address of the Extra segment register

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.

Segment register for calculating base address of source data is DS and that of the destination data is ES

Example: MOVS BYTE

Operations:

Calculation of source memory location:

EA = (SI) BA = (DS) x 16_{10} MA = BA + EA

Calculation of destination memory location:

EA_E = (DI) BA_E = (ES) x 16_{10} MA_E = BA_E + EA_E

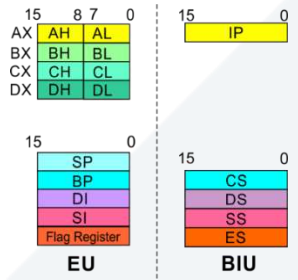
(MAE) ← (MA)

If DF = 1, then (SI) ← (SI) – 1 and (DI) = (DI) - 1

If DF = 0, then (SI) ← (SI) +1 and (DI) = (DI) + 1

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



These addressing modes are used to access data from standard I/O mapped devices or ports.

In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

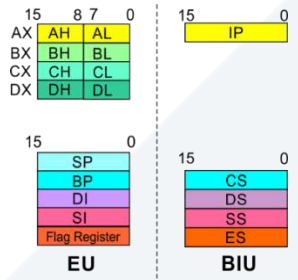
Example: **IN AL, [09H]**

Operations: $PORT_{addr} = 09_H$
$(AL) \leftarrow (PORT)$

Content of port with address 09_H is moved to AL register

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

Operations:

$000A_H \leftarrow 0A_H$ **(sign extend)**

If ZF = 1, then

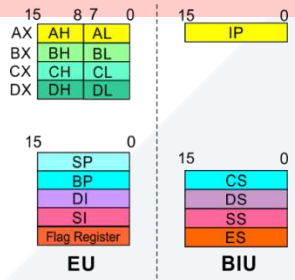
$EA = (IP) + 000A_H$
$BA = (CS) \times 16_{10}$
$MA = BA + EA$

If ZF = 1, then the program control jumps to new address calculated above.

If ZF = 0, then next instruction of the program is executed.

Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.

Example: **CLC**

This clears the carry flag to zero.

جَامعة
المَنارة
MANARA UNIVERSITY

Assembler directives

الموجهاتAssemble Directives

- متعلقة بالبرنامج قيد التنفيذ. Assemblerإرشادات للـ
- تتحكم بتوليد شيفرة الآلة وتنظيم البرامج؛ إلا أنه ليس لها أية شيفرة آلة.
- يطلق عليها أيضاً التعليمات الزائفة‘pseudo instructions’.
- تستخدم في:

تحديد بداية ونهاية البرنامج.
ربط قيمة مع المتغيرvariables.
تخصيص مواقع تخزين لبيانات الإدخال الإخراج.
تحديد بداية ونهاية القطاعات والإجرائيات والماكروات...إلخ..

الموجهاتAssemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- DB تعرّف بايتDefine Byte
- DW تعرّف متغير من نمط البايت(8-bit) variable.
- SEGMENT تحجز حيز محدد من مواقع الذاكرة لكل متغير
- unsigned value للقيم غير ذات الإشارة FF_H إلى 00_H المجال من
- positive value للقيم الموجبة $7F_H$ إلى 00_H المجال من و
- negative value للقيم السالبة FF_H إلى 80_H المجال من
- variable DB value/ values شكلها العام

مثال:

```
LIST DB 7FH, 42H, 35H
```

ثلاثة مواقع ذاكرة متتالية وتحمل المواقع الثلاث بالقيم الابتدائية المذكورة LISTتحجز للمتغير في المواقع المحجوزة.

الموجهاتAssemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- DB تعرّف كلمة Define Word
- DW تعرّف متغير من نمط الكلمة (16-bit) variable
- تحجز حيز محدد من موقعي ذاكرة متتاليين لكل متغير.
- للقيم غير ذات الإشارة unsigned value؛ المجال من 0000_H إلى $FFFF_H$
- و positive value للقيم الموجبة المجال من 0000_H إلى $7FFF_H$
- negative value للقيم السالبة المجال من 8000_H إلى $FFFF_H$
- شكلها العام variable DW value/ values

مثال:

```
ALIST DW 6512H, 0F251H, 0CDE2H
```

ALIST تحجز للمتغير ستة مواقع ذاكرة متتالية وتحمل المواقع الست بالقيم الابتدائية المذكورة مثنى مثنى في المواقع المحجوزة.

الموجهاتAssemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- SEGMENT تشير لبداية قطاعات الشيفرة/البيانات/الكدسة :code/ data/ stack segment
- ENDS تشير لنهاية قطاعات :code/ data/ stack segment.
- الشكل العام:

```
Segnam SEGMENT
...
...
...
...
...
...
Segnam ENDS
```

Program code
or
Data Defining Statements

اسم القطاع المعرّف من قبل المستخدم

الموجهاتAssemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- باسم قطاع البرنامج/البيانات الواجب استخدامه من أجل assemblerيخبر الـ قطاع محدد.
- الشكل العام:

ASSUME segreg : segnam, .. , segreg : segnam

الاسم المعرّف من قبل المستخدم للقطاع	مسجل القطاع

مثال:

ASSUME CS: ACODE, DS:ADATA	تخبر المترجم أن تعليمات البرنامج مختزنة في القطاع ACODE وأن البيانات مخزنة في القطاع ADATA.

Assemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- ORG (Origin) is used to assign the starting address (Effective address) for a program/ data segment
- END is used to terminate a program; statements after END will be ignored
- EVEN : Informs the assembler to store program/ data segment starting from an even address
- EQU (Equate) is used to attach a value to a variable

Examples:

ORG 1000H	Informs the assembler that the statements following ORG 1000H should be stored in memory starting with effective address 1000_H
LOOP EQU 10FEH	Value of variable LOOP is $10FE_H$
_SDATA SEGMENT ORG 1200H A DB 4CH EVEN B DW 1052H _SDATA ENDS	In this data segment, effective address of memory location assigned to A will be 1200_H and that of B will be 1202_H and 1203_H.

Assemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- PROC Indicates the beginning of a procedure
- ENDP End of procedure
- FAR Intersegment call
- NEAR Intrasegment call
- General form

```
procname PROC[NEAR/ FAR]
        ...
        ...
        ...
        RET
procname ENDP
```

Program statements of the procedure

Last statement of the procedure

User defined name of the procedure

Assemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

Examples:

ADD64 PROC NEAR … … … RET ADD64 ENDP	The subroutine/ procedure named ADD64 is declared as NEAR and so the assembler will code the CALL and RET instructions involved in this procedure as near call and return
CONVERT PROC FAR … … … RET CONVERT ENDP	The subroutine/ procedure named CONVERT is declared as FAR and so the assembler will code the CALL and RET instructions involved in this procedure as far call and return

Assemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- Reserves one memory location for 8-bit signed displacement in jump instructions

Example:

JMP SHORT AHEAD	The directive will reserve one memory location for 8-bit displacement named AHEAD

Assemble Directives

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- MACRO Indicate the beginning of a macro
- ENDM End of a macro
- General form:

```
macroname MACRO[Arg1, Arg2 ...]
        ...
        ...
        ...
macroname ENDM
```

Program statements in the macro

User defined name of the macro