80x86 ADDRESSING MODES

ROAD MAP

- Introduction
- Register Addressing Mode
- Immediate Addressing Mode
- Memory Addressing Mode
- Displacement Addressing Mode
- Register Indirect Mode
- Indexed Addressing Mode
- Base Indexed Addressing Mode
- Base Index Plus displacement addressing mode
- Program Memory addressing Modes
- Stack Addressing Modes

Data Memory Addressing Mode

80x86 Memory Addressing Modes

- The 8086 provides 17 different ways to access memory. This may seem like quite a bit at first, but fortunately most of the address modes are simple variants of one another so they're very easy to learn.
- The addressing modes provided by the 8086 family include displacement-only, base, displacement plus base, base plus indexed, and displacement plus base plus indexed.
- Variations on these five forms provide the 17 different addressing modes on the 8086. See, from 17 down to five. It's not so bad after all!

Register addressing Mode

1) 80x86 Register Addressing Modes

By specifying the name of the register as an operand to the instruction, you may access the contents of that register. Consider the 8086 mov (move) instruction:

```
mov        destination, source
```

- This instruction copies the data from the source operand to the destination operand. The only restriction is that both operands must be the same size. Now let's look at some actual 8086 mov instructions:

```
mov     ax, bx  ;Copies the value from BX into AX
mov     dl, al  ;Copies the value from AL into DL
mov     si, dx  ;Copies the value from DX into SI
mov     sp, bp  ;Copies the value from BP into SP
mov     dh, cl  ;Copies the value from CL into DH
mov     ax, ax  ;Yes, this is legal!
```

- Remember, the registers are the best place to keep often used variables.
- There are two restrictions on the use of the segment registers with the mov instruction. First of all, you may not specify cs as the destination operand, second, only one of the operands can be a segment register. You cannot move data from one segment register to another with a single mov instruction. To copy the value of cs to ds, you'd have to use some sequence like:

```
mov     ax, cs
mov     ds, ax
```

Immediate Addressing Mode

2) Immediate Addressing Mode:-

This addressing mode transfers the source –immediate byte or word of data into the destination register or memory location.

- Example:- MOV AL, 22H

This instruction copies a byte sized 22H into register AL.

In the 80386 and above, a double-word of immediate data can be transferred into a register or memory location.

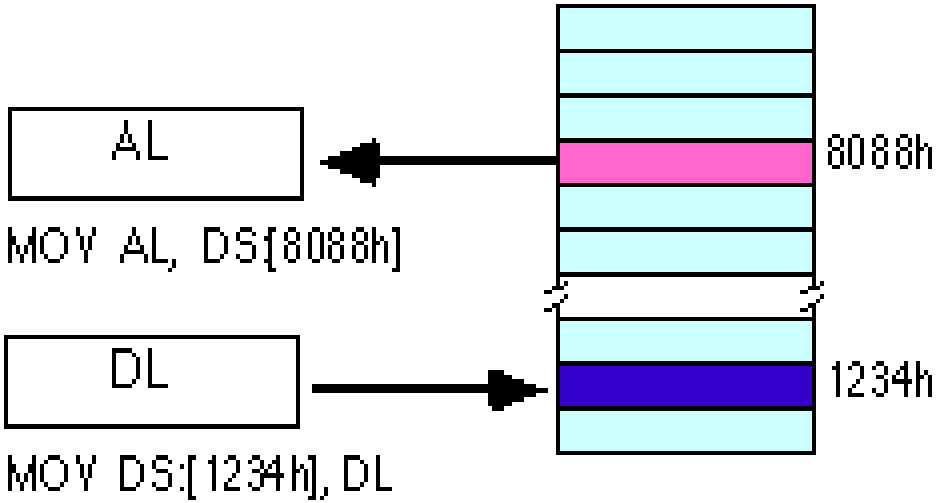
- MOV ESI,12345678H

This instruction copies a double-word sized 1234578H into register ESI.

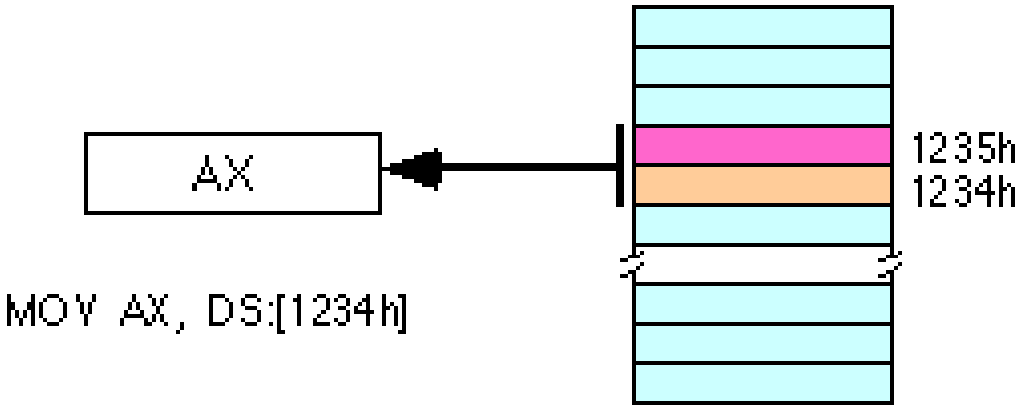
Displacement Mode

3) **The Displacement Only Addressing Mode**

4) The displacement-only addressing mode consists of a 16 bit constant that specifies the address of the target location. The instruction mov al,ds:[8088h] loads the al register with a copy of the byte at memory location 8088h. Likewise, the instruction mov ds:[1234h],dl stores the value in the dl register to memory location 1234h:



- The displacement-only addressing mode is perfect for accessing simple variables.
- Intel named this the displacement-only addressing mode because a 16 bit constant (displacement) follows the mov opcode in memory.
- In that respect it is quite similar to the direct addressing mode on the x86 processors . There are some minor differences, however.
- First of all, a displacement is exactly that- some distance from some other point. On the x86, a direct address can be thought of as a displacement from address zero. On the 80x86 processors, this displacement is an offset from the beginning of a segment (the data segment in this example).



By default, all displacement-only values provide offsets into the data segment. If you want to provide an offset into a different segment, you must use a segment override prefix before your address.

For example, to access location 1234h in the extra segment (es) you would use an instruction of the form mov ax,es:[1234h]. Likewise, to access this location in the code segment you would use the instruction mov ax, cs:[1234h]. The ds: prefix in the previous examples is not a segment override.

Register Indirect Mode

4) The Register Indirect Addressing Modes

- The 80x86 CPUs let you access memory indirectly through a register using the register indirect addressing modes. There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

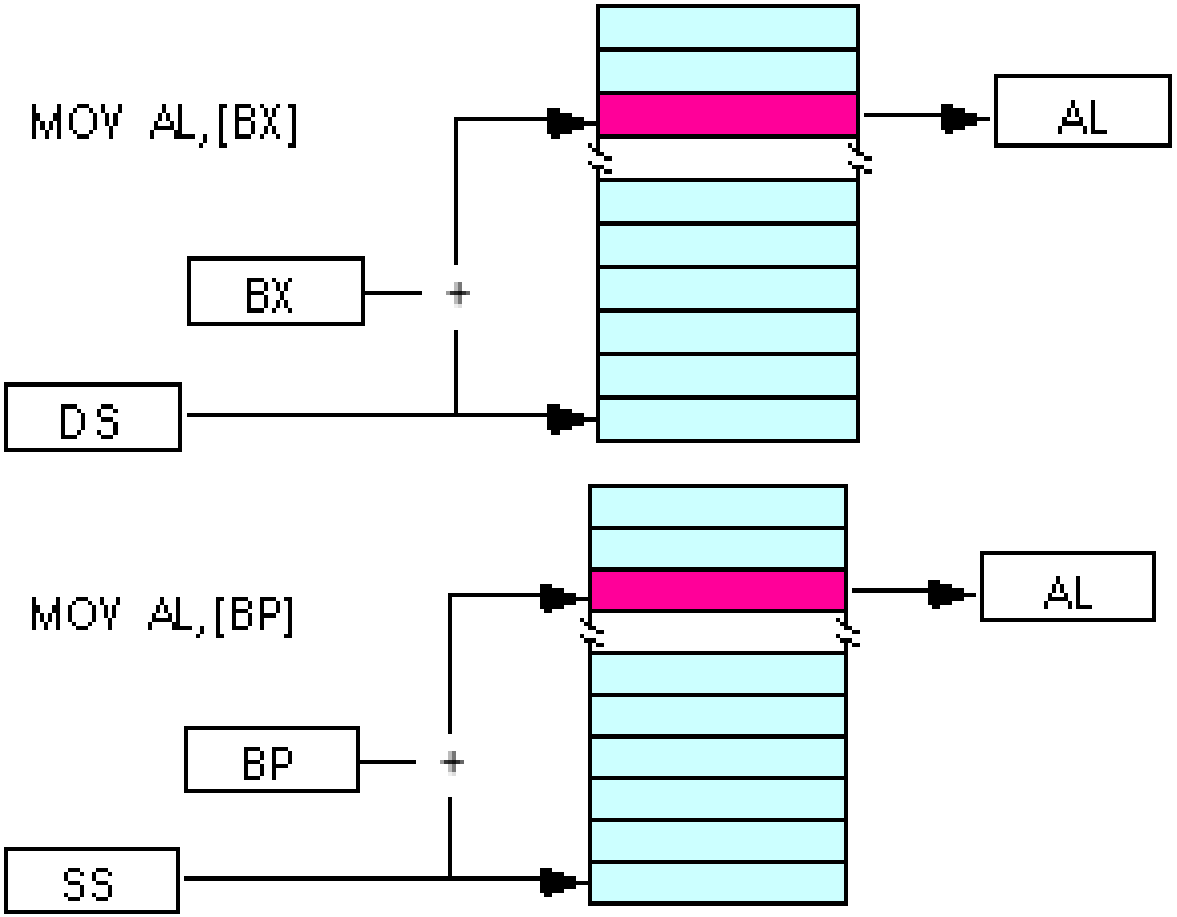
```
mov     al, [bx]
mov     al, [bp]
mov     al, [si]
mov     al, [di]
```

- As with the x86 [bx] addressing mode, these four addressing modes reference the byte at the offset found in the bx, bp, si, or di register, respectively. The [bx], [si], and [di] modes use the ds segment by default. The [bp] addressing mode uses the stack segment (ss) by default.

- You can use the segment override prefix symbols if you wish to access data in different segments. The following instructions demonstrate the use of these overrides:

```
mov     al, cs:[bx]
mov     al, ds:[bp]
mov     al, ss:[si]
mov     al, es:[di]
```

- Intel refers to [bx] and [bp] as base addressing modes and bx and bp as base registers (in fact, bp stands for base pointer). Intel refers to the [si] and [di] addressing modes as indexed addressing modes (si stands for source index, di stands for destination index).


MOV AL,[BX]
AL
BX
+
DS
MOV AL,[BP]
AL
BP
+
SS

Indexed Addressing Mode

5) Indexed Addressing Modes

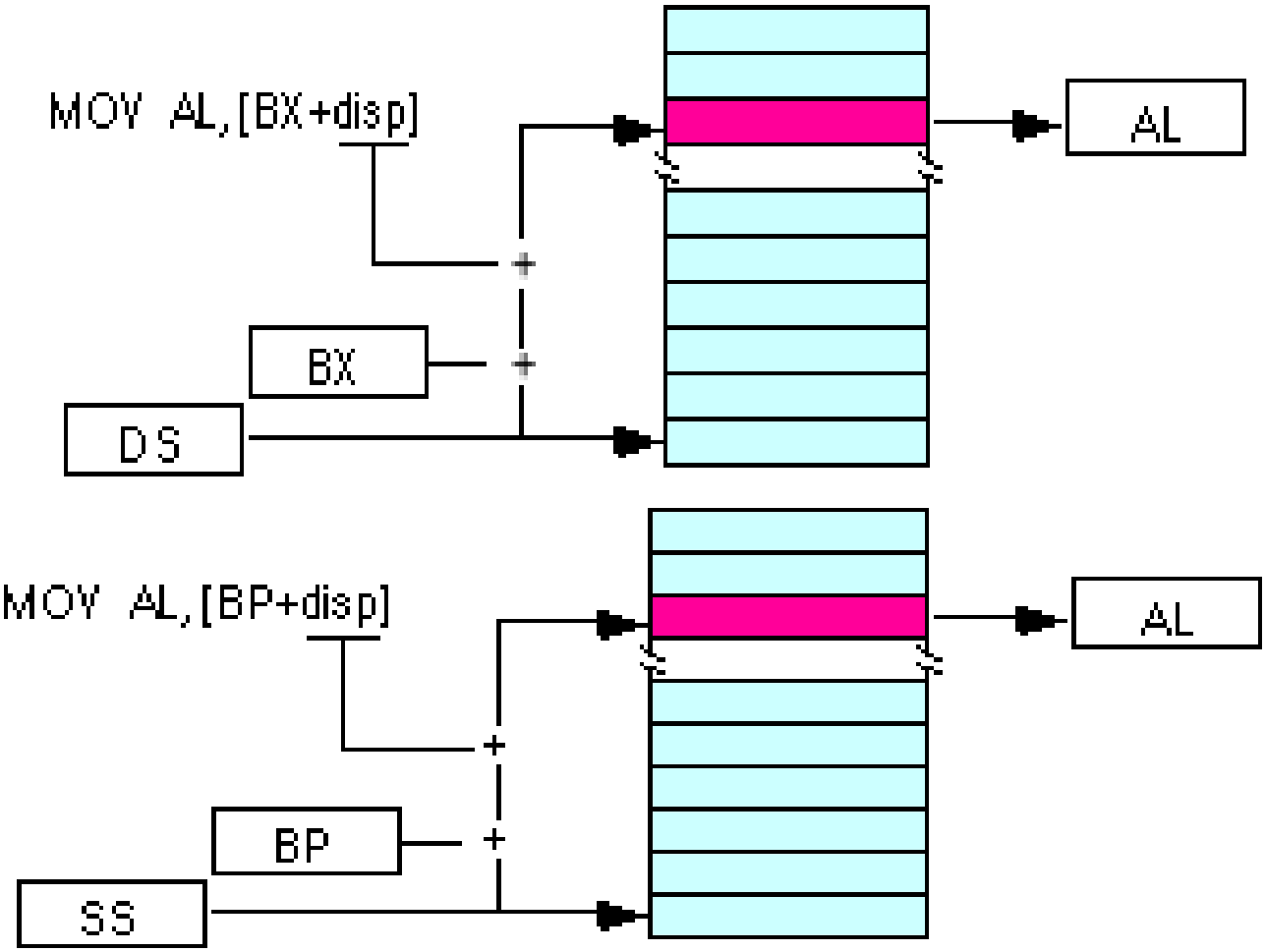
- The indexed addressing modes use the following syntax:

```
mov     al, disp[bx]
mov     al, disp[bp]
mov     al, disp[si]
mov     al, disp[di]
```

- If bx contains 1000h, then the instruction mov cl,20h[bx] will load cl from memory location ds:1020h. Likewise, if bp contains 2020h, mov dh,1000h[bp] will load dh from location ss:3020.

- The offsets generated by these addressing modes are the sum of the constant and the specified register. The addressing modes involving bx, si, and di all use the data segment, the disp[bp] addressing mode uses the stack segment by default. As with the register indirect addressing modes, you can use the segment override prefixes to specify a different segment:

```
mov     al, ss:disp[bx]
mov     al, es:disp[bp]
mov     al, cs:disp[si]
mov     al, ss:disp[di]
```


MOV AL,[BX+disp]
AL
BX
DS
MOV AL,[BP+disp]
AL
BP
SS

- You may substitute si or di in the figure above to obtain the [si+disp] and [di+disp] addressing modes.
- Note that Intel still refers to these addressing modes as based addressing and indexed addressing.

Based Indexed Addressing Mode

6) Based Indexed Addressing Modes

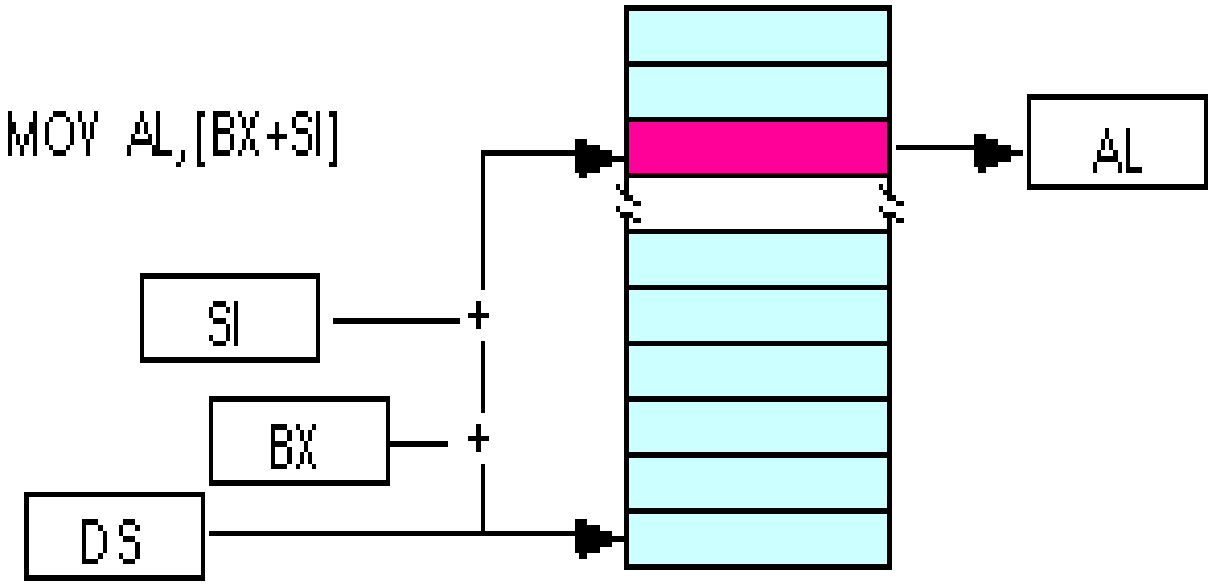
- The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are

```
mov     al, [bx][si]
mov     al, [bx][di]
mov     al, [bp][si]
mov     al, [bp][di]
```

- Suppose that bx contains 1000h and si contains 880h. Then the instruction

  ```
  mov     al,[bx][si]
  ```

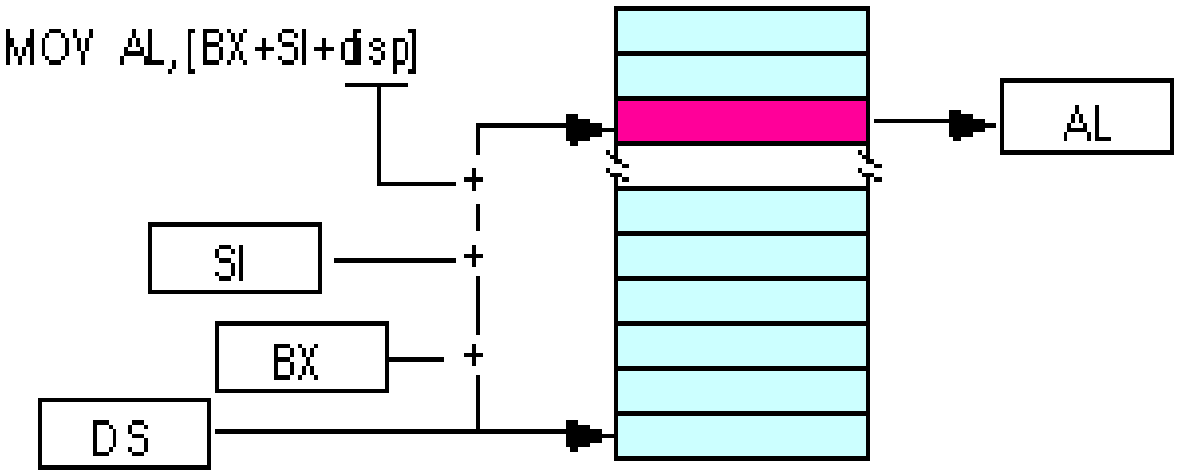
  would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax,[bp+di] will load the 16 bits in ax from locations SS:259C and SS:259D.
- The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.


MOV AL,[BX+SI]
SI
BX
DS
+
+
AL

Based Index + Displacement Mode

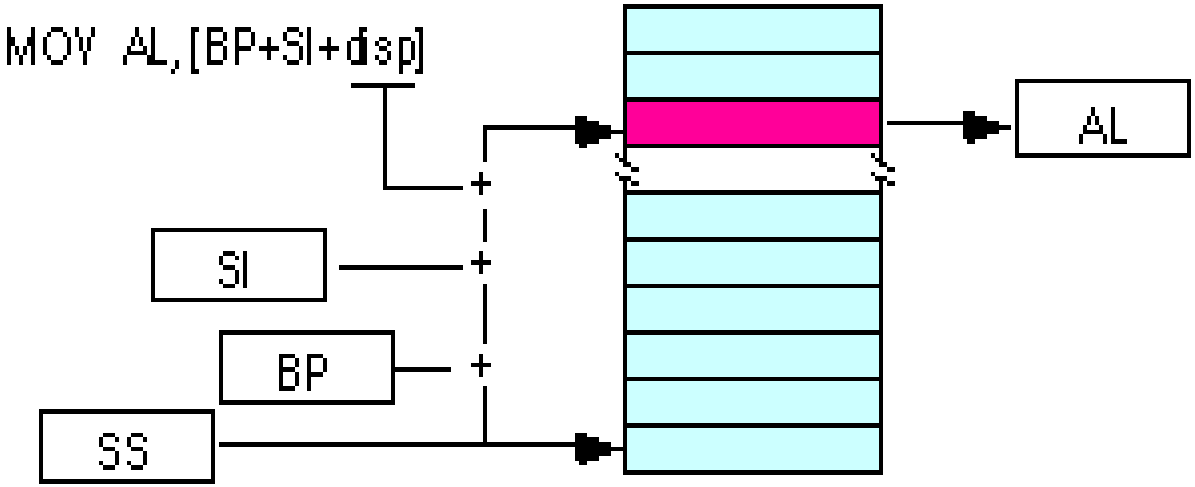
7) Based Indexed Plus Displacement Addressing Mode

- These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant. The following are some examples of these addressing modes:

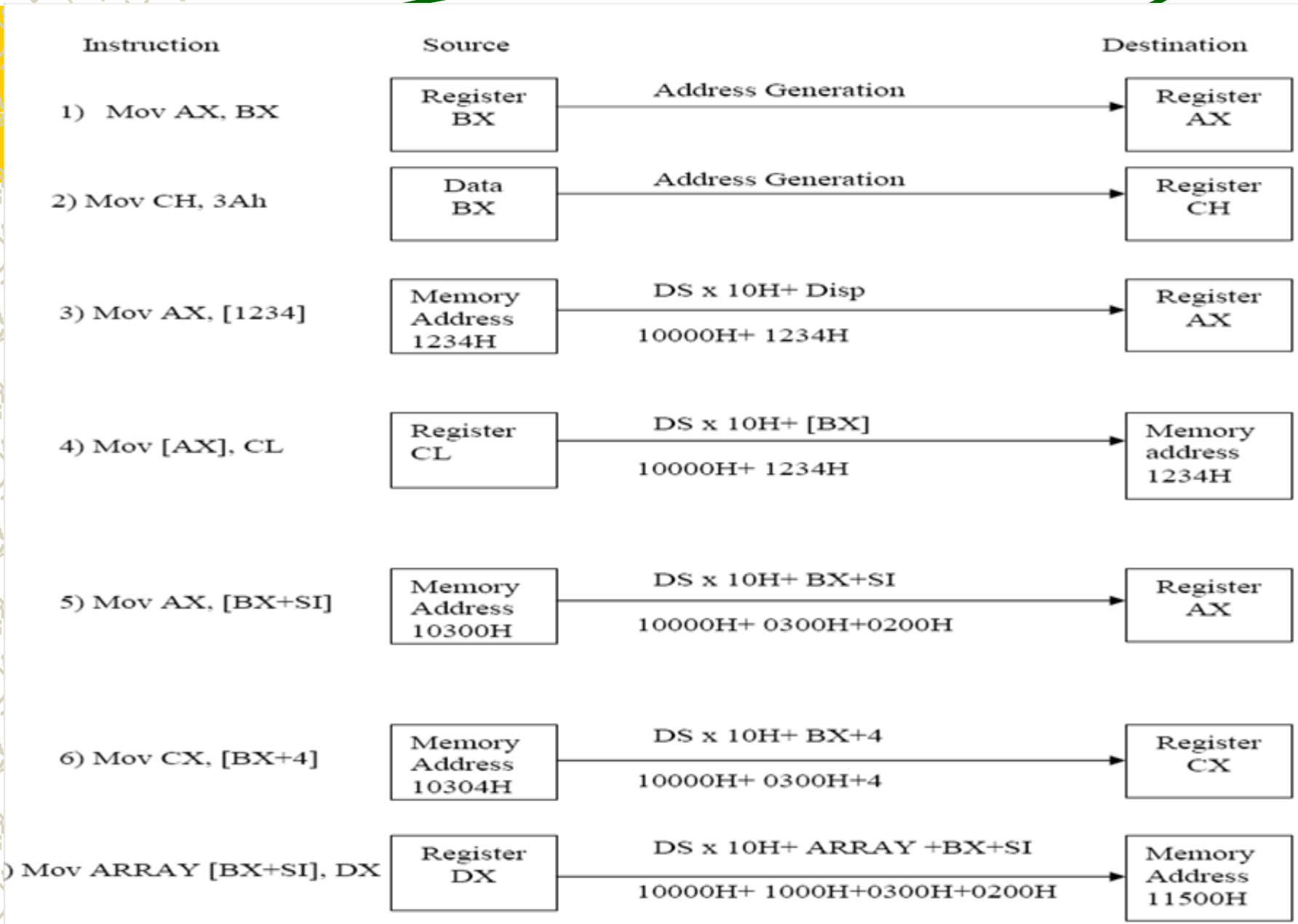


```
mov     al, disp[bx][si]
mov     al, disp[bx+di]
mov     al, [bp+si+disp]
mov     al, [bp][di][disp]
```

- You may substitute di in the figure above to produce the [bx+di+disp] addressing mode



- You may substitute di in the figure above to produce the [bp+di+disp] addressing mode.
- Suppose bp contains 1000h, bx contains 2000h, si contains 120h, and di contains 5. Then mov al,10h[bx+si] loads al from address DS:2130; mov ch,125h[bp+di] loads ch from location SS:112A; and mov bx,cs:2[bx][di] loads bx from location CS:2007.


Instruction
Source
Destination
1) Mov AX, BX
Register BX
Address Generation
Register AX
2) Mov CH, 3Ah
Data BX
Address Generation
Register CH
3) Mov AX, [1234]
Memory Address 1234H
DS x 10H+ Disp
10000H+ 1234H
Register AX
4) Mov [AX], CL
Register CL
DS x 10H+ [BX]
10000H+ 1234H
Memory address 1234H
5) Mov AX, [BX+SI]
Memory Address 10300H
DS x 10H+ BX+SI
10000H+ 0300H+0200H
Register AX
6) Mov CX, [BX+4]
Memory Address 10304H
DS x 10H+ BX+4
10000H+ 0300H+4
Register CX
) Mov ARRAY [BX+SI], DX
Register DX
DS x 10H+ ARRAY +BX+SI
10000H+ 1000H+0300H+0200H
Memory Address 11500H

Program Memory Addressing modes

NEAR

Intrasegment
(CS does not change)

JUMPS and CALLS

Direct -- IP relative displacement
new IP = old IP + displacement
Allows program relocation with no change in code.

Indirect -- new IP is in memory or a register.
All addressing modes apply.

FAR

Intersegment
(CS changes)

Direct -- new CS and IP are encoded in the instruction.

Indirect -- new CS and IP are in memory.
All addressing modes apply except immediate and register.

Stack Memory Addressing Mode

- Push
- Pop

Method to Remember all The Modes

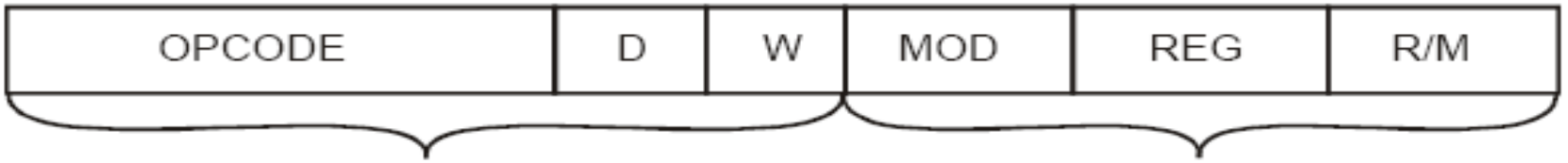
An Easy Way to Remember the 8086 Memory Addressing Modes

- There are a total of 17 different legal memory addressing modes on the 8086: disp, [bx], [bp], [si], [di], disp[bx], disp[bp], disp[si], disp[di], [bx][si], [bx][di], [bp][si], [bp][di], disp[bx][si], disp [bx][di], disp[bp][si], and disp[bp][di]. You could memorize all these forms so that you know which are valid (and, by omission, which forms are invalid). However, there is an easier way besides memorizing these 17 forms. Consider the chart:

DISP	[BX]	[SI]
	[BP]	[DI]

- If you choose zero or one items from each of the columns and wind up with at least one item, you've got a valid 8086 memory addressing mode. Some examples:
- Choose disp from column one, nothing from column two, [di] from column 3, you get disp[di].
- Choose disp, [bx], and [di]. You get disp[bx][di].
- Skip column one & two, choose [si]. You get [si]
- Skip column one, choose [bx], then choose [di]. You get [bx][di]
- Likewise, if you have an addressing mode that you cannot construct from this table, then it is not legal. For example, disp[dx][si] is illegal because you cannot obtain [dx] from any of the columns above.

8086 Instruction Format



- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information:
  - Opcode field (6 bits) specifies the operation such as add, subtract, or move
  - Register Direction Bit (D bit)
- Tells the register operand in REG field in byte 2 is source or destination operand
  - 1: destination
  - 0: source
  - Data Size Bit (W bit)
- Specifies whether the operation will be performed on 8-bit or 16-bit data
  - 0: 8 bits
  - 1: 16 bits
- Byte 2 has two fields:
  - Mode field (MOD)
  - Register field (REG)
  - Register/memory field (R/M field)

Table For Register Field

- REG field is used to identify the register for the first operand

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Table for Mod Field

- 2-bit MOD field and 3-bit R/M field together specify the second operand

Table 4.4(a)

Code	Explanation
00	Memory mode, no displacement follows*
01	Memory mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

* Except when R/M = 110, then 16-bit displacement follows

MOD = 11				Effective Address Calculation		
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	Direct Address	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

Examples for Decoded Instructions

Examples :

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
  - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

OPCODE	D	W	MOD	REG	R/M
000000	0	1	10	000	001

MOV BL,AL => 10001000 11000011 = 88 C3h

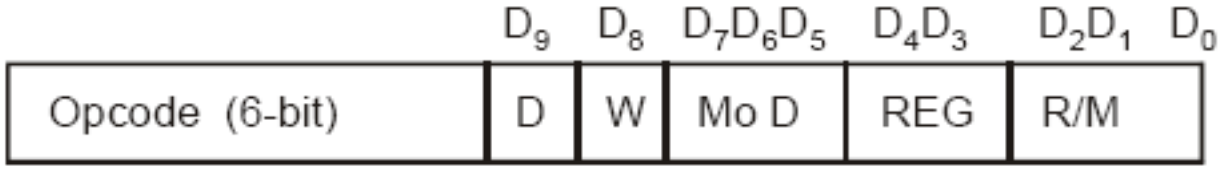
Example : MOV BL, CL

Process to decode the instruction :

(1) It is an 8-bit operation.

(2) It fits in the register/memory to/from register format.

(3) BL is the designation register and CL is the source reg.



- ➤ It is an 8-bit operation, hence 'w' bit is 0. If it would have been a 16-bit operation than 'w' bit would have been 1.
- ➤ If d = 1, then transfer of data is to the register shown by the REG field, i.e. the destination is a register. If d = 0, the source is a register shown by the REG field.

If we refer to the previous tables to search the REG to REG addressing in it, i.e. the last column with MOD 11.

According to the data sheet when MOD is 11, the R/M field is treated as REG field.

The REG field is used for source register and R/M field is used for the destination register if 'd' is 0.

If D is 1 the REG field is used for destination and the R/M field is used to indicate source. Hence the code can be written in decoded form as :

	Opcode	D	W	MoD	REG	R/M
MOV BL, CL ⇒	1 0 0 0 1 0	0	0	1 1	0 0 1	0 1 1

⇓

88 CB

Example : ADD AX, BX

Opcode	D	W	Mo	D	REG	R/M
0 0 0 0 0 0	1	1	1	1	0 0 0	0 1 1

⇓

Machine code = 03C3

Conclusion

We can Now conclude to say that we have studied all the different addressing modes present in 80x86 family of processors. The various addressing modes that we have seen are as follows:-

Introduction, Register Addressing Mode, Memory Addressing Mode, Displacement Addressing Mode, Register Indirect Mode, Indexed Addressing Mode, Base Indexed Addressing Mode, Base Index Plus displacement addressing mode, Program memory addressing Modes

THANKS!

For more Notes Follow http://www.edutechlearners.com