Kingdom of Saudi Arabia
Royal Commission at Yanbu
University College – Yanbu
Department of ACS & AIT
Yanbu Al-Sinaiyah

المملكة العربية السعودية
الهيئة الملكية ينبع
الكلية الجامعية-ينبع
ينبع الصناعية

2ND Semester 2007-08

CS-203 HANDOUT (HO - 8)

Addressing Modes in 8086

Unlike high-level languages, a real assembly language has no notion of a data type. Data in memory are not self-identifying. The bits at a memory location have no inherent meaning. The meaning is determined by how an instruction uses them. A doubleword in memory might be interpreted as an unsigned integer, a floating-point number, a character, several characters, or part of an instruction.

The assembly language for a real computer exposes these details to the programmer. It is the responsibility of the programmer to assure, for example, that integer variables are added together using an integer add instruction, and that memory cells containing instructions are not interpreted as floating point operands.

8.1 Addressing Modes

An instruction acts on zero or more operands. The way an instruction accesses its operands is called its **Addressing modes**.

We can classify the different addressing modes into four groups:

- Immediate addressing
- Register addressing
- Memory addressing
- I/O port addressing

Operands may be implicit or explicit or both.

Implicit operands mean that the instruction by definition has some specific operands. The programmers do NOT select these operands.

Example: Implicit operands

```
XLAT    ; automatically takes AL and BX as operands
AAM     ; it operates on the contents of AX.
```

Explicit operands mean the instruction operates on the operands specified by the programmer.

Example: Explicit operands

```
MOV AX, BX;   it takes AX and BX as operands
XCHG SI, DI;  it takes SI and DI as operands
```

Implicit and explicit operands:

Example: Implicit/Explicit operands

```
MUL BX; automatically multiply BX explicitly times AX
      ; implicitly and save the result in DX:AX implicitly.
```

The location of an operand value in memory space is called the **effective address (EA)**

8.2 Immediate addressing mode

In this addressing mode, the operand is stored as part of the instruction. The immediate operand, which is stored along with the instruction, resides in the code segment -- not in the data segment. This addressing mode is also faster to execute an instruction because the operand is read with the instruction from memory. Here are some examples:

Example: Immediate Operands

```
MOV AL, 20     ; move the constant 20 into register AL
ADD AX, 5      ; add constant 5 to register EAX
MOV DX, offset msg    ; move the address of message to register DX
```

8.3 Register addressing mode

In this addressing mode, the operands may be:

- reg16: 16-bit general registers: AX, BX, CX, DX, SI, DI, SP or BP.
- reg8 : 8-bit general registers: AH, BH, CH, DH, AL, BL, CL, or DL.
- Sreg : segment registers: CS, DS, ES, or SS. There is an exception: CS cannot be a destination.

For register addressing modes, there is no need to compute the effective address. The operand is in a register and to get the operand there is no memory access involved.

Example: Register Operands

```
MOV AX, BX     ; mov reg16, reg16
ADD AX, SI     ; add reg16, reg16
MOV DS, AX     ; mov Sreg, reg16
```

8.3.1 Some restrictions in register addressing modes

1. You may not specify CS as the destination operand.

   Example: mov CS, 02h -> wrong

2. 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 ds,cs  -> wrong

mov ax, cs
mov ds, ax  -> the way we do it
```

You should never use the segment registers as data registers to hold arbitrary values. They should only contain segment addresses

8.4 Memory Addressing Modes

Memory (RAM) is the main component of a computer to store temporary data and machine instructions. In a program, programmers many times need to read from and write into memory locations.

The question is how to specify exactly which memory location we want to access? The simple answer is to give the label of the desired memory variable.

Example: Reading and writing memory locations

```
.
TempVar         DW      ?
NextVar         DW      ?
.
.
                mov     AX, TempVar
                sub     NextVar, AX
.
```

The above fragment code will read an operand value from memory location *TempVar* and copy it to the register AX. And then, read an operand value from memory location *NextVar*, and subtract the content of AX from the memory variable *NextVar*
However, the programming problem is much more complex. Look at the following problem.

> Suppose you have an array of byte named ByteArray. It consists of 10 elements:
>
> 12, 34, 2, 5, 1 ,7, 13, 45, 98, 4
>
> In memory, the location of ByteArray starts at offset 100 in the data segment.
>
> To read the eighth byte, 45, which is at address 107, in Java we have to write:
>
> byte d = ByteArray[7];
>
> How can we do the same in assemby?

Fortunately, MASM provides us many different ways to handle the addressing of memory location. The simplest solution of the above problem is:

Solution: Reading the eight byte of ByteArray

```
.
.DATA
ByteArray      BYTE    12, 34, 2, 5, 1 ,7, 13, 45, 98, 4
.
.CODE
mov    AX, @Data
mov    DS, AX
mov    AL, [ByteArray + 7] ; or mov AL, ByteArray[7]
```

MASM evaluates the last statement above as,

mov AL, [100+7]

Since ByteArray starts at offset 100, the offset of ByteArray and 7 are added together and used as a memory address. The instruction becomes

mov AL, [107]

IMPORTANT

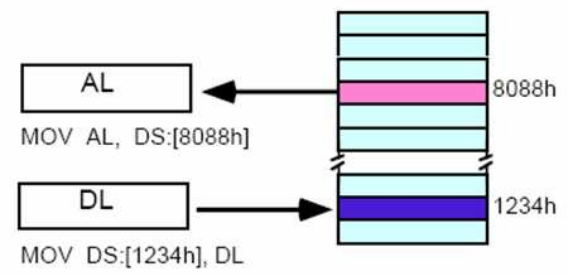
Everything between square brackets will be treated as an ADDRESS. The above instruction means copy the contents of memory location 107 into AL register.

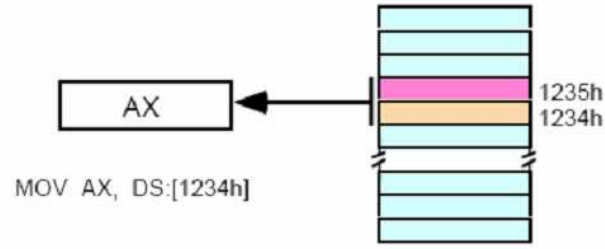
There are different forms of memory addressing modes

1. Direct Addressing
2. Register indirect addressing
3. Based addressing
4. Indexed addressing
5. Based indexed addressing
6. Based indexed with displacement

8.4.1 Direct Addressing Mode

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. 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.





the instruction mov al,ds:[8088h] is same as mov al, [8088h]. If not mentioned DS register is taken by default.

8.4.2 Register Indirect Addressing Mode

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]
```

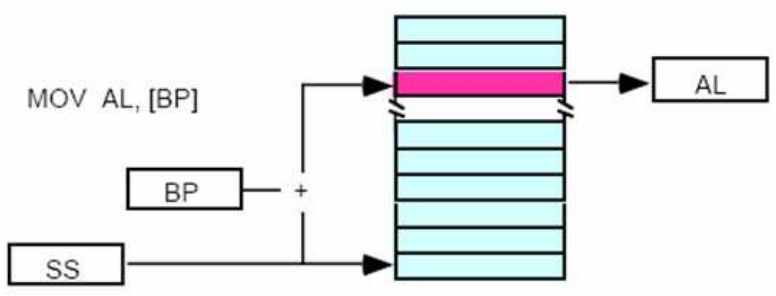
Code Example

```
MOV BX, 100H
MOV AL, [BX]
```

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). However, these addressing modes are functionally equivalent. This text will call these forms register indirect modes to be consistent.



8.4.3 Indexed Addressing Modes

The indexed addressing modes use the following syntax:

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

Code Example

```
MOV BX, 100H
MOV AL, [BX + 15]
MOV AL, [BX + 16]
```

If bx contains 1000h, then the instruction mov cl, [bx+20h] will load cl from memory location ds:1020h. Likewise, if bp contains 2020h, mov dh, [bp+1000h] 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 [bp+disp] 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:[bx+disp]
mov al, es:[bp+disp]
mov al, cs:[si+disp]
mov al, ss:[di+disp]
```

Example: MOV AX, [DI + 100]

8.4.4 Based Indexed Addressing Modes

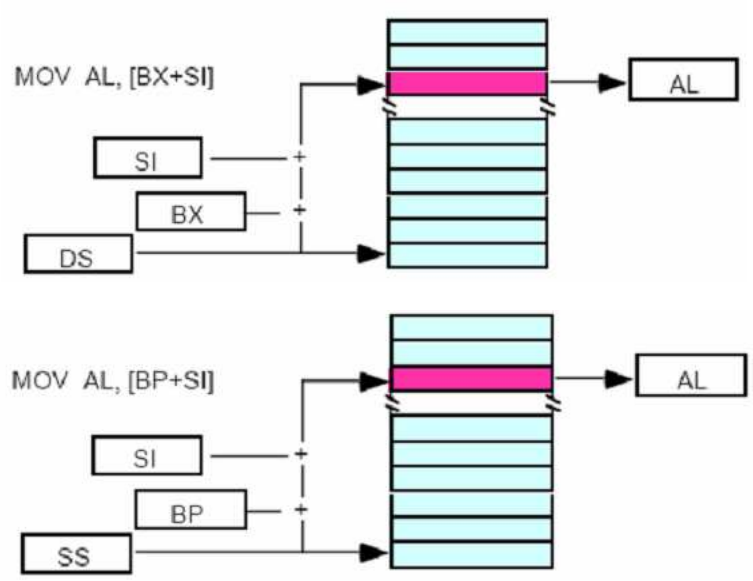
The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are

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

Code Example

```
MOV BX, 100H
MOV SI,   200H
MOV AL, [BX + SI]
INC BX
INC SI
```

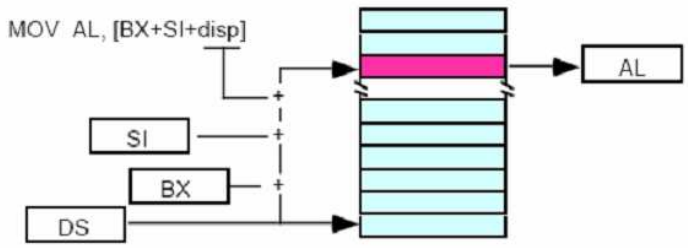
Suppose that bx contains 1000h and si contains 880h. Then the instruction mov al,[bx][si] would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax,[bp+di] will load the 16 bits in ax from locations SS:259C and SS:259D. The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.

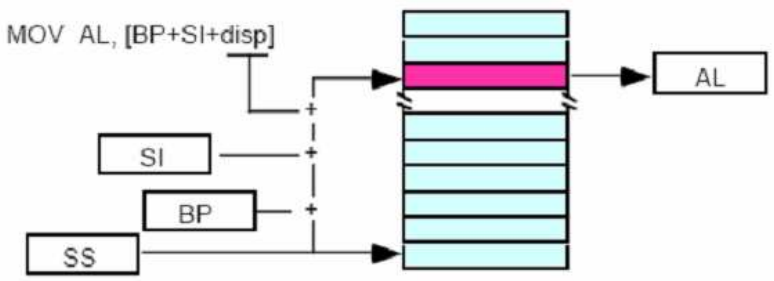


8.4.5 Based Indexed Plus Displacement Addressing Mode

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]
```





Code Example

```
MOV BX, 100H
MOV SI,   200H
MOV AL, [BX + SI +100H]
INC BX
INC SI
```

8.4.6 Memory Addressing Modes Classification

There are 17 different ways to specify a memory address using 16-bit memory addressing modes:

Operand Type	Pattern
direct	[displacement] Example: MOV AX, [76]
register indirect	[BX] [SI] [DI] [BP] Example: MOV AX, [SI]
based	[BP + displacement] [BX + displacement] MOV AX, [BX + 100]
indexed	[SI + displacement] [DI + displacement] MOV AX, [DI + 100]
based-indexed	[BX + SI] [BX + DI] [BP + SI] [BP + DI] MOV AX, [BP + DI]
based-indexed with displacement	[BX + SI + displacement] [BX + DI + displacement] [BP + SI + displacement]

	■ [BP + DI + displacement]

where *displacement* means any expression that produces a 16-bit constant value.

16-bit memory addressing modes

There are many different styles having the same meaning to write the above addressing modes:

[BP + 2], 2[BP], [BP][2], [BP]+2, 2+[BP] are the same

ByteArray[BX][SI]+1, [ByteArray + BX + SI + 1], [ByteArray][BX][SI][1], etc. are the same.

All the following instructions refer exactly to the same memory location, [ByteArray+7].

To load the eighth byte of ByteArray into AL

```
.
.DATA
ByteArray      BYTE    12, 34, 2, 5, 1 ,7, 13, 45, 98, 4
.
.CODE
mov    AX, @Data
mov    DS, AX
mov    AL, [ByteArray + 7]
.
mov    SI, OFFSET ByteArray+7
mov    AL, [SI]
.
mov    BX, 7
mov    AL, [ByteArray + BX]
.
mov    BX, OFFSET ByteArray
mov    AL, [BX + 7]
.
mov    SI, 7
mov    AL, [ByteArray + SI]
.
mov    BX, OFFSET ByteArray
mov    DI, 7
mov    AL, [BX + DI]
.
mov    SI, OFFSET ByteArray
mov    BX, 7
mov    AL, [SI + BX]
.
mov    BX, OFFSET ByteArray
mov    SI, 6
mov    AL, [BX + SI + 1]
.
mov    BX, 3
mov    SI, 4
mov    AL, [BX + ByteArray + SI]
.
```

Example Test Questions

Fill in the blanks

1. MOV AX, BX is example of _____addressing mode
2. MOV AX, [BX+SI] is example of _____ addressing mode.

Mention True or False

1. MOV DS, CS is a valid instruction (True/False)
2. MOV AX, [BX][SI] is a valid instruction. (True/False)

Choose the best answer.

1. In the based and indexed addressing mode, which of the following segment register is used by default?

1. Data Segment
2. Code Segment
3. Stack Segment
4. Extra Segment

2. MOV AX, [ByteArray][BX][SI][1] belongs to which addressing mode.

1. Register indirect addressing mode
2. Based indexed with displacement addressing mode.
3. Immediate addressing
4. Based Indexed addressing