8086 Addressing Modes

The different ways in which operands can be addressed are called the addressing modes. Addressing modes differ in the way the address information of operands is specified.

1. Immediate Addressing Mode

The value of the operand is (immediately) available in the instruction itself.

Immediate operand represents constant data (a byte or word). The number is stored in two's complement form.

```
mov al, 48 ;  load 30H in AL;
mov cx,2056H
xor si,1    ; invert LSB in SI register;
and al,80H ; highlight MSB of AL
or di, 8000H ; set to 1 MSB of DI
```

The advantage of immediate addressing is that no memory reference other than the instruction fetch is requiered to obtain the operand.

The disadvantages: the size of the number is restricted to the size of the address field;

2. Register Addressing Mode

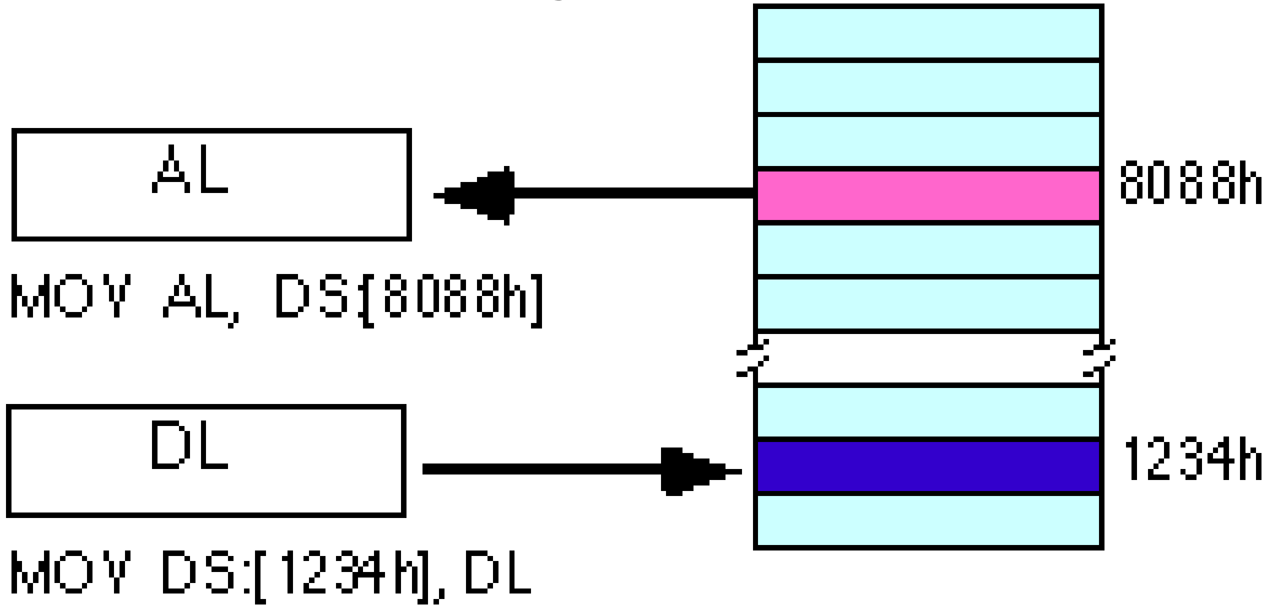
```
mov  ax, bx  ;Copies the value from BX into AX
mov     dl, al  ;Copies the value from AL into DL
mov     ax, ax  ;
add bx,di
sub cl,ah
```

Advantage: the registers are the best place to keep often used variables. Instructions using the registers are shorter and faster than those that access memory.

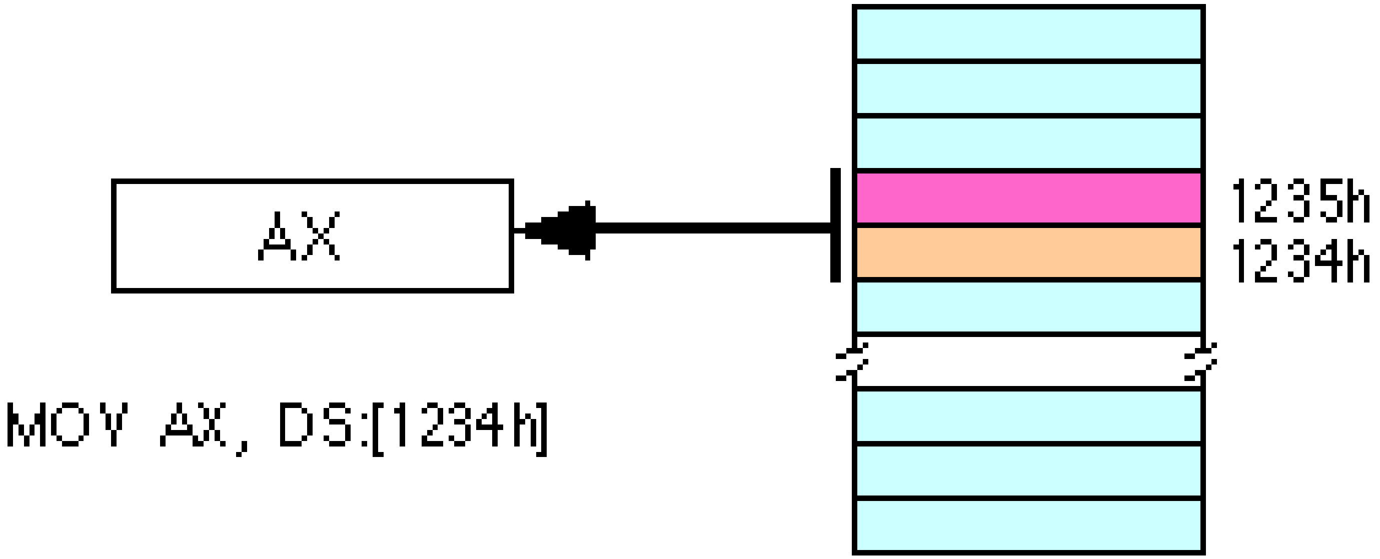
Disadvantage: limited address space and the limited number of general purpose registers.

3. Direct Addressing mode (displacement only)

- mov al, [8088h] loads the Al register with a copy of the byte at memory location 8088h.
- mov [1234h],dl stores the value in the Dl register to memory location 1234h:



- to access location 1234h in the extra segment (es) you would use an instruction of the form
- mov ax,es:[1234h].
- You can also access words on the 8086 processors :



- Other examples:

BETA dw 1234h

…………………………

MOV CX, BETA ;

Inc COUNT

Mul X ; multiply ax with variable X

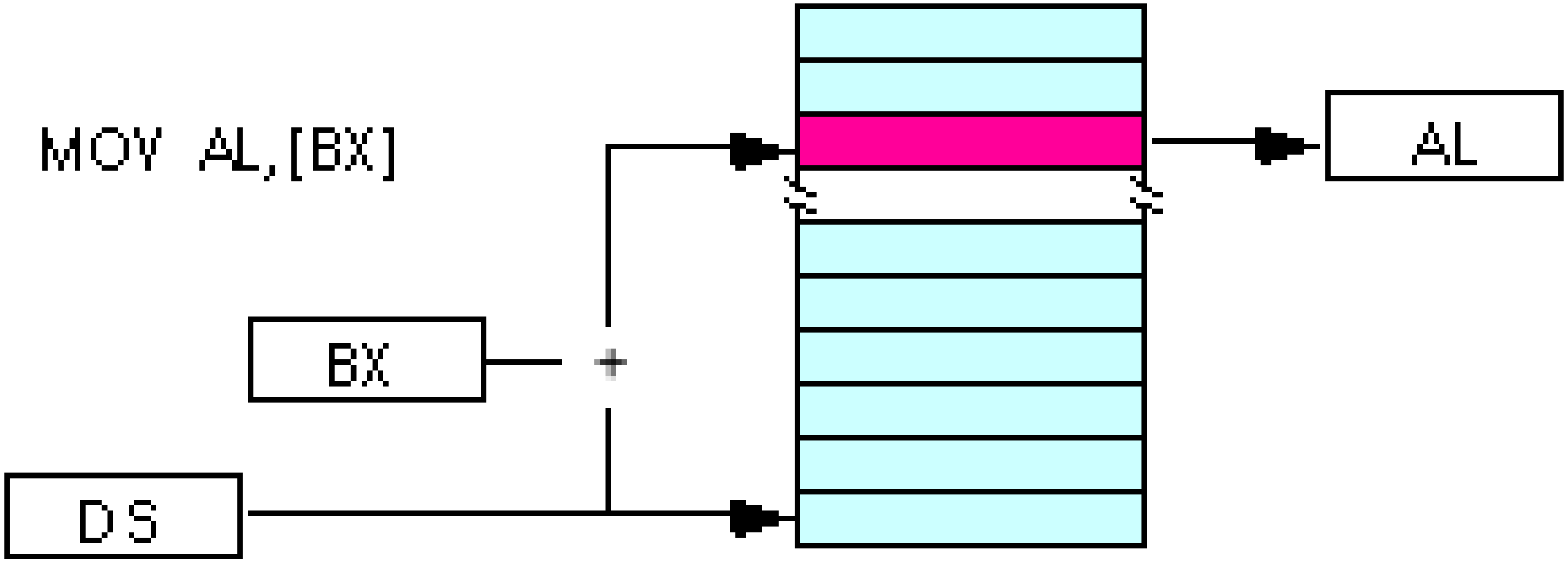
Ror TEMP ; shift right variable TEMP

- In inc, mul, ror instructions it is impossible to determine the size of a variable
- Inc word ptr COUNT
- Ror byte ptr TEMP

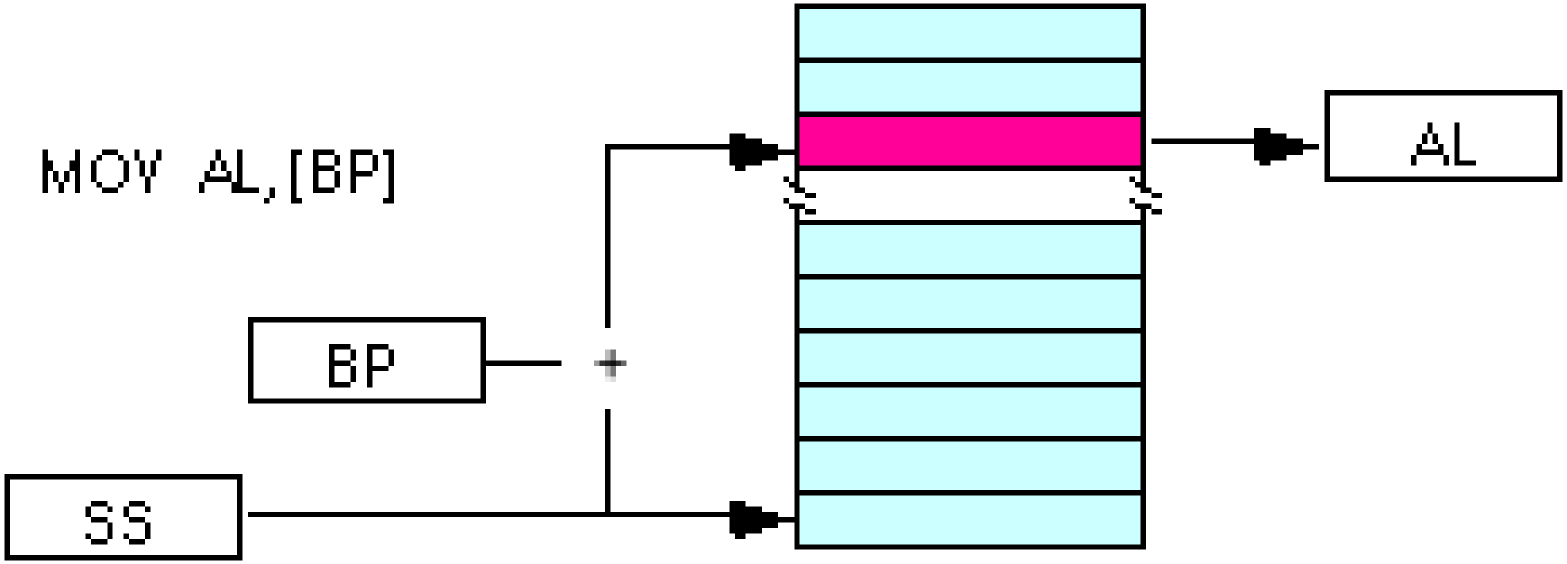
4. Register Indirect Addressing mode:

-
- mov al, [bx]
- mov al, [bp]
- mov al, [si]
- mov al, [di]
- The [bx], [si], and [di] modes use the ds segment by default. The [bp] mode uses the stack segment (ss) by default.

MOV AL, [BX] ; This instruction moves the contents of the memory location DS:BX to the AL register.



MOV AL, [BP] ; This instruction moves the contents of the memory location SS:BP to the AL register.



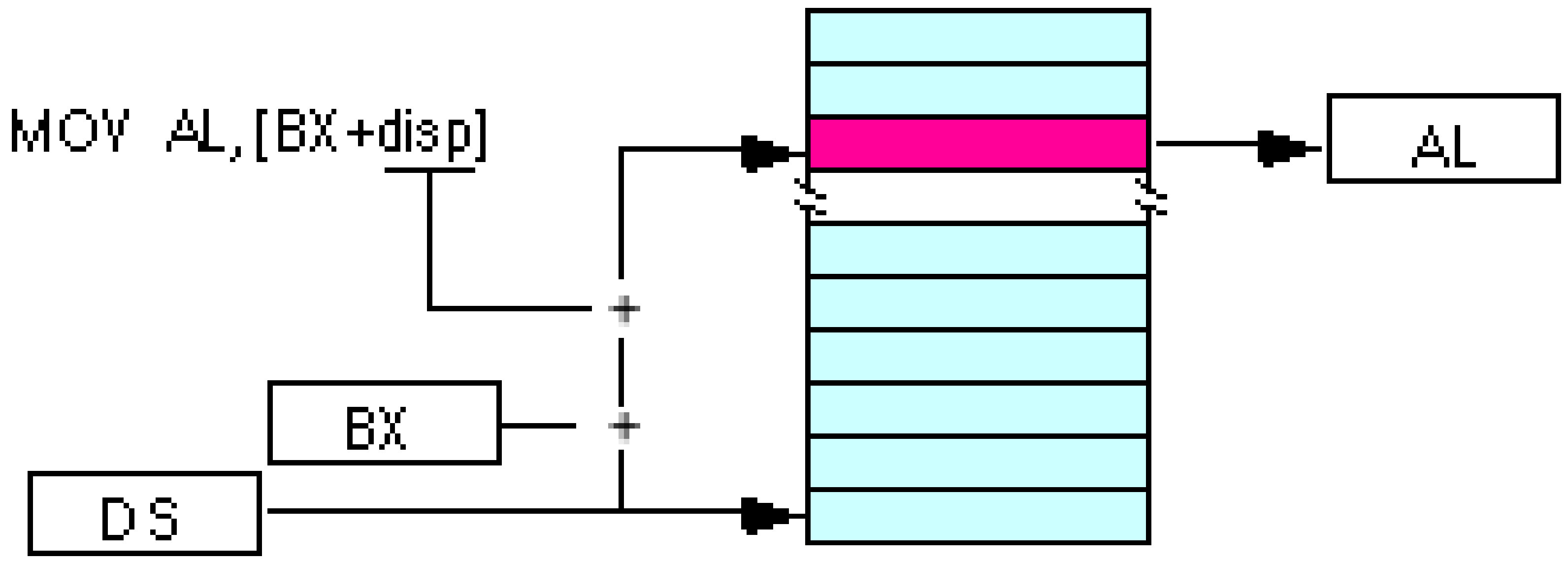
5. Based Addressing mode:

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

The displacement field can be a signed eight bit constant or a signed 16 bit constant.

EA=disp+[BP] or [BX].

mov AX, [BP+10] ; load in AX the 6th word of the array


MOV AL,[BX+disp]
AL
BX
DS

6. Indexed Addressing mode:

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

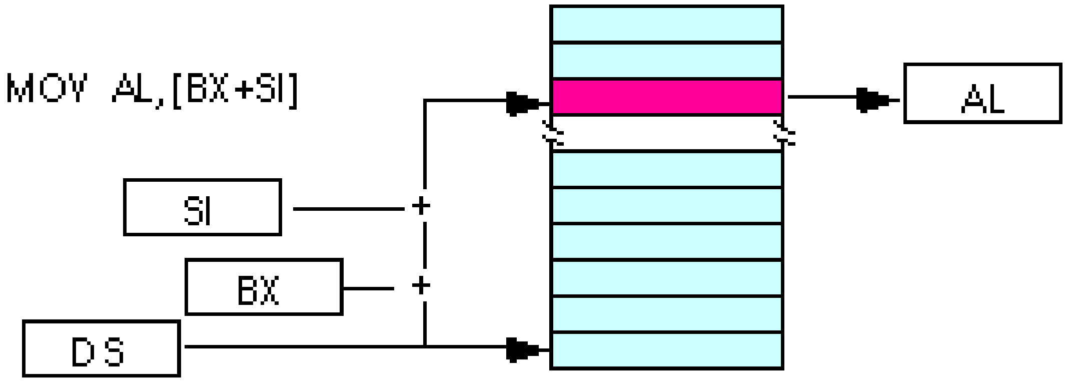
The displacement field can be a signed eight bit constant or a signed 16 bit constant.

EA=disp+[SI] or [DI].

- Inc DI
- ……
- Mov Z[DI], AX; move the content of AX to array element
- add AX, ARRAY[SI] ; add AX with the element of ARRAY,

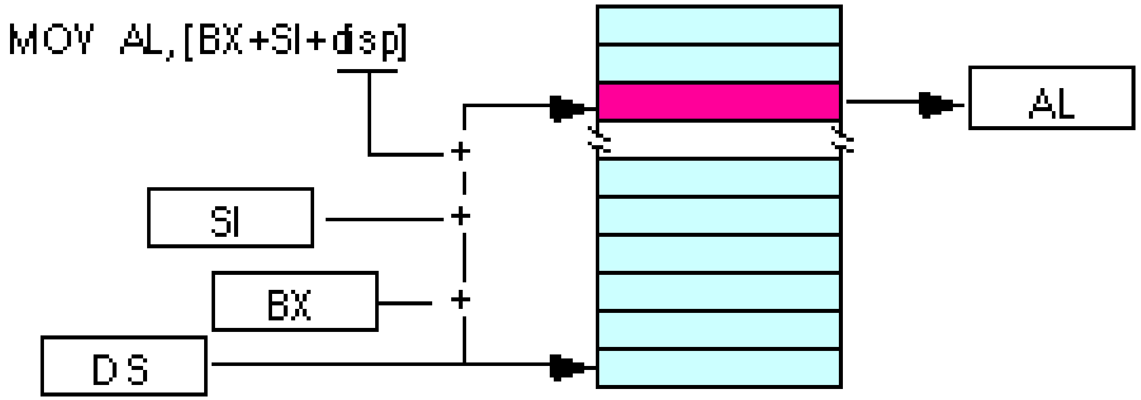
7. Based Indexed Addressing Mode:

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



8. Based Indexed Plus Displacement Addressing Mode

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



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

8088/8086 Effective Address (EA) Calculation

Description	**Clock Cycles**
Displacement	6
Base or Index (BX,BP,SI,DI)	5
Displacement+(Base or Index)	9
Base+Index (BP+DI,BX+SI)	7
Base+Index (BP+SI,BX+DI)	8
Base+Index+Displacement (BP+DI,BX+SI)	11
Base+Index+Displacement (BP+SI+disp,BX+DI+disp)	12

- add 4 cycles for word operands at odd addresses
- add 2 cycles for segment override

- The displacement field in all addressing modes except displacement-only can be a signed eight bit constant or a signed 16 bit constant.
- If the offset is in the range -128...+127 the instruction will be shorter (and therefore faster) than an instruction with a displacement outside that range.
- The size of the value in the register does not affect the execution time or size.
- So if you can arrange to put a large number in the register(s) and use a small displacement, that is preferable over a large constant and small values in the register(s).

- If the effective address calculation produces a value greater than 0FFFFh, the CPU ignores the overflow and the result wraps around back to zero.
- For example, if bx contains 10h, then the instruction

```
mov al,0FFFFh[bx]
```

will load the al register from location ds:0Fh, not from location ds:1000Fh.