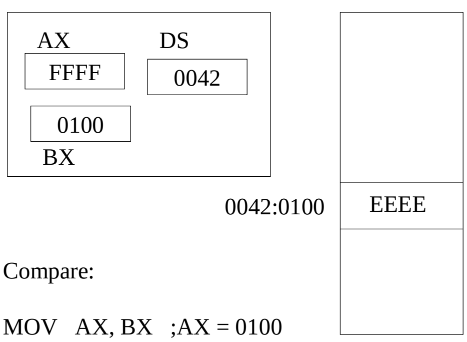
Lecture 12: Addressing Modes (Part 2)

- Review of Indirect Addressing
- Based and Indexed Operands
- Base-Index Operands
- Base-Index with Displacement

Indirect Addressing

- An indirect operand is a register that contains the offset of data in memory.
- When the offset of the variable is placed in a register, the register becomes a pointer to the label.
- You can use SI, DI, BX, and BP to hold indirect operands.
  - BX: base register
  - SI, DI: index registers
  - BP: base pointer (contains an offset from the SS register)



Compare:

MOV AX, BX ;AX = 0100
MOV AX, [BX] ; AX = EEEE

The [] around BX indicates that BX contains the address of the data you want.

Useful because we can change the value of BX at execution time to be able to pick up data from different places.

```
        TITLE Largest and Smallest Signed Numbers

        .model small
        .stack 100h
        .data
array   dw      -1, 2000, -4000, 32767, 500, 0
largest dw      ?
smallest dw ?
        .code
main proc
        mov AX, @data
        mov DS, AX
        mov     di, offset array
        mov     ax, [di]        ; get first element
        mov     largest, ax     ; initialize largest
        mov     smallest, ax    ; initialize smallest
        mov     cx, 6           ; loop counter
A1:     mov     ax, [di]        ; get array value
        cmp     ax, smallest    ; [DI] >= smallest?
        jge     A2              ; yes: skip
        mov     smallest, ax    ; no: move [DI] to smallest
A2:     cmp     ax, largest     ; [DI] <= largest?
        jle     A3              ; yes: skip
        mov     largest, ax     ; no: move [DI] to largest
A3:     add     di, 2           ; point to next number
        loop    A1              ; repeat loop until CX = 0
        mov     AX, 4C00h
        int     21h
done:   nop
        main endp
        end main
        end
```

Based and Indexed Operands

- Based operands and indexed operands are the same: A register, either base or index) is added to a displacement to generate an effective address.
- The displacement is a constant.
- BX and BP are base registers (used as based operands) and SI and DI are index registers (used as indexed operands).

Forms Allowed

```
.data
ROWVAL = 3
array dw        123, 549, 3403, 235
```

Register Added to an Offset	**Register Added to a Constant**
mov dx, array[bx]	mov ax, [bx+ROWVAL]
mov dx, [di + array]	mov dx, [bp+4]
mov dx, [array+si]	mov dx, 2[si]

Example

- array example, p. 110 in Irvine

Base-Index Operands

- A base-index operand adds the value of a base register to an index register to get a memory offset.
- One important restriction: you can not combine two base registers (i.e. BP with BX) or two index registers (SI with DI)
- Why is this useful?
  - You can set your displacement at execution time by storing the base address in one register (BX) and your offset in another (SI or DI).

Example: Two Dimensional Array

1050	10	20	30	40	50
1055	60	70	80	90	0a0
	0b0	0c0	0d0	0e0	0f0

- picture from p.111, Irvine

2D Array, continued

- Example 7, p. 111 Irvine

Base-Index with Displacement

- You can also create an operands effective address by combining a base register, an index, register and a displacement.
- Some formats are:

```
mov  dx, array[bx][si]
mov  ax, [bx+si+array]
add   dl, [bx+si+3]
sub   cx, array[bp+si]
```

Example: Two Dimensional Array

1050	10	20	30	40	50
1055	60	70	80	90	0a0
	0b0	0c0	0d0	0e0	0f0

- array is at offset 1050
- if bx = 5 (pointing to second row) and si = 2 (third column)
- array[bx][si] will get the value at offset 0157 -> 80

Two Dimensional Array, cont.

- Example from p. 112, Irvine

Be careful with arrays!

1050	10	20	30	40	50
1055	60	70	80	90	0a0
	0b0	0c0	0d0	0e0	0f0

- if you want the third column, second row, it is tempting to try to access it like this:
  - array[bx][si] ; where bx = 1 for the second row, si = 2 for the third column
  - (like array[1][2] in C)
- this will not work! This will actually point to 40, not to 80
  - 1050 +1 + 2 = 1053

Accessing arrays

1050	10	20	30	40	50
1055	60	70	80	90	0a0
	0b0	0c0	0d0	0e0	0f0

- to access row r, column c, you need to set your registers as follows:
  - row register = (r – 1) * rowlength
  - column register = (c – 1)
  - so, for row 2, col 3
    - row register = 5
    - column register = 2
    - 1050 + 5 + 2 = 1057 -> 80!
- Note: this assumes that you store your array by rows!

How does this affect machine code?

- r/m and mod definitions from Intel sheet

Addressing Modes We've Looked At

- mod = 11 (Register to Register)
- mod = 00 — r/m field tells you
  - 01 — how to calculate
  - 10 — the address
- Indirect addressing:

  Uses BX, SI, DI as the register to hold the address.

  This uses mod 00 (displacement = 0)

  r/m field = 100 [SI]
  101 [DI]
  111 [BX]

Other Possibilities (not counting BP)

- Use r/m field above (indicating register: 100, 101, 111) but allow a non-zero displacement (mod = 01 or 10)
- Or,

  r/m = 000 [BX][SI] (+displ)

  r/m = 001 [BX][DI] (+displ)