# *CSC 222*: Computer Organization & Assembly Language

- ARRAYS
- ADDRESSING MODES

# One Dimensional Array

- A one dimensional array is ordered list of elements, all of same type.
- Sequence of memory bytes or words
- Base Address of an array or offset address assigned to an array
- **Example 1:**

  B_ARRAY DB 10h, 20h, 30h

| Symbol | Address | Contents |
|---|---|---|
| B_ARRAY | 0200h | 10h |
| B_ARRAY+1 | 0201h | 20h |
| B_ARRAY+2 | 0202h | 30h |

**If B_ARRAY is assigned offset address 0200h by assembler**

# Example 2

- W_ARRAY DW 1000, 40, 29887, 329

  ***If W_ARRAY is assigned offset address 0300h by assembler**

| Symbol | Address | Contents |
|---|---|---|
| W_ARRAY | 0300h | 1000d |
| W_ARRAY+ 2 | 0302h | 40d |
| W_ARRAY+ 4 | 0304h | 29887d |
| W_ARRAY+ 6 | 0306h | 329d |

# The DUP Operator

- Use to define array whose element share a common value
- Syntax:

  repeat_count DUP (value)

- This operator causes the value to be repeated the number of times specified by repeat_count
  - ARR DW 100 DUP (0)
  - ARR2 DB 212 DUP(?)

# Location Of Array Element

- The address of an array element may be specified by adding a constant to the base address.

| Position | Location |
|---|---|
| 1 | A |
| 2 | A + 1 X S |
| 3 | A + 2 X S |
| . | . |
| . | . |
| N | A + (N-1) X S |

- where A is an array and S is number of bytes

# Example

- Exchange the 10th and 25th elements in word array W.
- Solution:
- W[10] is located at address W + 9 X 2 = W + 18
- W[25] is located at address W + 24 X 2 = W + 48

```
MOV  AX ,W+18
XCHG  W+48 ,AX
MOV  W+18 ,AX
```

# Addressing Modes

# General Purpose/Data Registers

- **AX (Accumulator):** Used in arithmetic, logic and data transfer instructions. Also required in multiplication, division and input/output operations.
- **BX (Base):** It can hold a memory address that points to a variable.
- **CX (Counter):** Act as a counter for repeating or looping instructions. These instructions automatically repeat and decrement CX and quit when equals to 0.
- **DX (Data):** It has a special role in multiply and divide operations. Also used in input/output operations.

# Pointers and Index Registers

- **IP - instruction pointer**: Always points to next instruction to be executed. IP register always works together with CS segment register and it points to currently executing instruction.
- **SI - source index register**: Can be used for pointer addressing of data. Offset address relative to DS
- **DI - destination index register**: Can be used for pointer addressing of data . Offset address relative to ES
- **SP** and **BP** are used to access data inside the stack segment
- **BP - base pointer**: Primarily used to access parameters passed via the stack. Offset address relative to SS
- **SP – stack pointer**: Always points to top item on the stack. Offset address relative to SS

# Default Segment and Offset Registers

- CS: IP
- SS: SP or BP
- DS: BX, DI, SI

# Addressing Modes

- The way an operand specified is known as its addressing mode.
- The addressing modes we used so far are
  1. Register – where an operand is a register
  2. Immediate – where an operand is a constant
  3. Direct – where an operand is a variable

# Addressing Modes (Indirect)

- Other addressing modes are following:

1. Register Indirect
2. Based
3. Indexed
4. Based Indexed(used with 2D array)

# Register Mode

- **Operand = Register**
- Can be 8 or 16 bit register
- Efficient as no memory access required.
- **Example**

mov ax, bx

mov cl, al

mov si, ax

# Immediate Mode

- **Operand = Constant Expression**
- Constant Expression can be a number, a character, or string.
- **Example**

  mov ax, 5

  mov dl,'a'

**Important**: Destination operand cannot be in immediate mode

# Direct Mode

- Direct operand refers to the contents of memory at a location identified by the label in the data segment.

- **Example**

```
.data
count db 20
wordList dw 1000h,2000h
.code
mov al, count
mov bx, wordList
```

# Contd..

```
.data
  array db 10,20,30,40
.code
  mov al, array
  mov bl, array+1
  mov cl, array+2
  mov dl, array+3
```

# Offset Operator

- Used to move the offset of a label into a register or variable.
- **Example**
  - **Assume offset of aWord is 0200H**

```
.data
aWord dw 1234
.code
mov bx, offset aWord
```

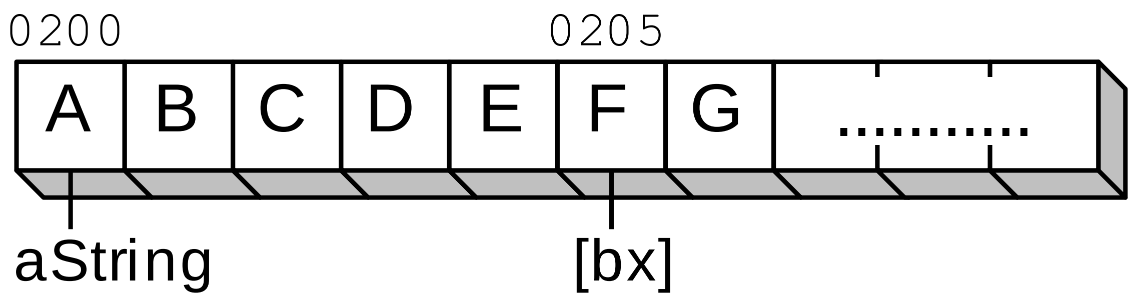
# Register Indirect Mode

- Register contains the offset of data in memory.
- The register become the pointer to the memory location.
- Format:

  [Register]
- The register is BX , SI , DI, or BP.
- For BX , SI , or DI, the operand's segment number is contained in DS.
- For BP, SS has the segment.

# Example

```
.data
aString db "ABCDEFG"
.code
mov bx,offset aString
add bx,5
mov dl,[bx]
```



# Example

- Suppose that

BX contains 1000h

SI contains 2000h

DI contains 3000h

Offset 1000h contains 1BACh

Offset 2000h contains 20FEh

Offset 3000h contains 031Dh

a) MOV BX , [BX]
b) MOV CX , [SI]
c) MOV BX , [AX]
d) ADD [SI] , [DI]
e) INC [DI]

# Adding 8-bit Integers

```
.data
aList db 10h,20h,30h
sum   db 0
.code
mov bx,offset aList
mov al,[bx]               ; AL = 10h
inc bx
add al,[bx]               ; AL = 30h
inc bx
add al,[bx]               ; AL = 60h
mov si,offset sum         ; get offset of sum
mov [si],al               ; store the sum
```

# Algorithm for addition of 10-element array

```
SUM = 0
N = 1
REPEAT
        SUM = SUM + A[N]
        N = N + 1
UNTIL N > 10
```

# Example

- Write some code to sum in AX the elements of the 10-element array W defined by

```
W   DW   10, 20, 30, 40, 50, 60, 70, 80, 90, 100
```

- Solution:

```
XOR AX, AX
LEA SI, W
MOV CX, 10
ADDNOS:
        ADD AX, [SI]
        ADD SI, 2
LOOP ADDNOS
```

# Based and Indexed Modes

- A register is added to a displacement to generate an effective address.
- Register may be: SI, DI, BX, or BP.
- Displacement can be a number or a label
- Based: If BX or BP used
- Indexed: IF SI or DI used
- Can be written as: **displacement [register]**

  **displacement + [register]**

  **[register] + displacement**

  **[displacement + register]**

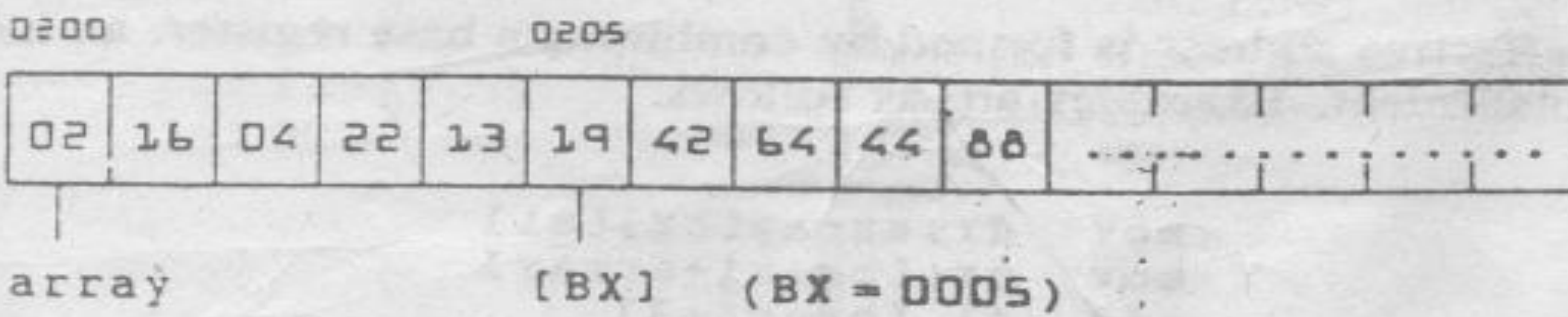
  **[register + displacement ]**

# Example

*Example.* If we create an array of byte values stored in memory at location 0200h and set BX to 5, BX will then point to the number at offset 5 into the array. This is shown by the following code and illustration:

```
array   db 2,16,4,22,13,19,42,64,44,88
.
.
mov bx,5
mov al,array[bx]   ; AL = 19
```

**Illustration:**



# Other possible formats

- MOV AL, [ARRAY + BX]
- MOV AL, [BX + ARRAY]
- MOV AL,  ARRAY + [BX]
- MOV AL, [BX] +ARRAY

# Example

Suppose SI contains the address of word array W.

- MOV AX, [SI + 2]
- MOV AX, [2 + SI]
- MOV AX, 2 + [SI]
- MOV AX, [SI] + 2
- MOV AX, 2[SI]

# Example

- Suppose that ALPHA DW 0123h, 0456h, 0789h, 0ABCDh

BX contains 2 offset 0002 contains 1084h

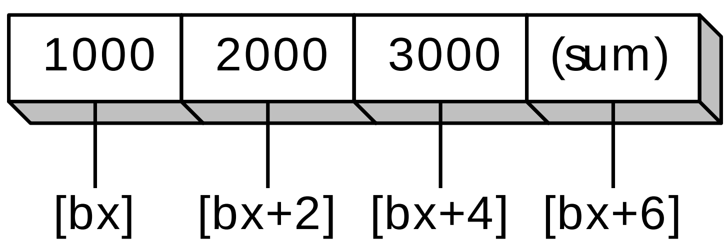
SI contains 4 offset 0004 contains 2BACh

DI contains 1

a) MOV AX, [ALPHA + BX]
b) MOV BX, [BX + 2]
c) MOV CX, ALPHA[SI]
d) MOV AX, -2[SI]
e) MOV BX, [ALPHA + 3 +DI]
f) MOV AX, [BX]2
g) ADD BX, [ALPHA + AX]

# Adding 16-bit Integers

```
.data
wordList dw 1000h,2000h,3000h, 0
.code
mov bx,offset wordList
mov ax,[bx]             ; first number
add ax,[bx+2]           ; second number
add ax,[bx+4]           ; third number
mov [bx+6],ax           ; store the sum
```



# Summing an Integer Array

```
.data
intarray dw 0100h,0200h,0300h,0400h
COUNT = ($ - intarray) / 2

.code
   mov   ax,0                     ; zero accumulator
   mov   di,offset intarray ; address of array
   mov   cx,COUNT                 ; loop counter
L1:
   add   ax,[di]                  ; add an integer
   add   di,2                     ; point to next integer
   Loop  L1                       ; repeat until CX = 0
```

# Summing an Integer Array of 10-Elments

- Write some code to sum in AX the elements of the 10-element array W defined by

  W DW 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

by using based mode.

- Solution:

```
    XOR AX,AX            ;AX holds sum
    XOR BX, BX           ;clear base register
    MOV CX, 10           ;CX has number of elements
    ADDNOS:
            ADD AX,W[BX]        ;sum=sum + element
            ADD BX, 2           ;index next element
    LOOP ADDNOS
```

# Displaying a String

```
.data
string db "This is a string."
COUNT = ($-string)   ; calculate string length

.code
   mov   cx,COUNT    ; loop counter
   mov   si,offset string
L1:
   mov   ah,2        ; DOS function: display char
   mov   dl,[si]     ; get character from array
   int   21h         ; display it now
   inc   si          ; point to next character
   Loop  L1          ; decrement CX, repeat until 0
```

# Example

Replace each lower case letter in string by its upper case equivalent using index addressing mode.

```
.DATA
MSG DB "this is a message"
COUNT=($-MSG)
.CODE
MOV CX,COUNT
TOP:
    CMP MSG[SI],' '
    JE NEXT
    AND MSG[SI],0DFh
NEXT:
    INC SI
    LOOP TOP
    MOV MSG[SI],'$'
    MOV AH,9
    LEA DX,MSG
    INT 21h
```

# Chapter # 8: Question # 6

- Write some code to:
  1. Place the top of the stack into AX, without changing the stack contents.
  2. Place the word that is below the stack top into CX, without changing the stack contents.
  3. Exchange the top two words on the stack without changing SP.

# Indirect Addressing – Summary

- Indirect Operands

  `[si]. [di], [bx], [bp]`

- Based and Indexed Operands

  `array[si], array[di], array[bx]`

- Base-Index Operands

  `[bx+si], [bx+di]`

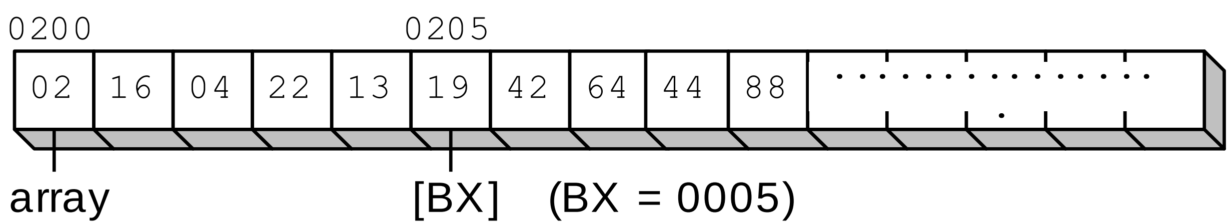
- Base-Index with Displacement

  `array[bx+si], array[bx+di]`

# Two-Dimensional Array Example

- Each row of this table contains five bytes. BX points to the beginning of the second row:

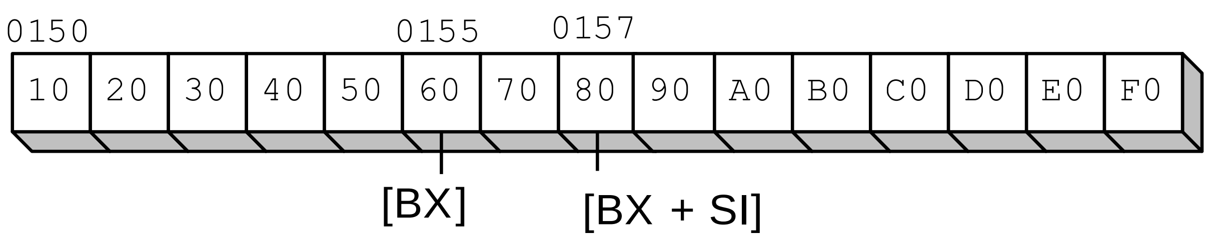
```
.data
ROWSIZE = 5
array  db  2h, 16h,  4h, 22h, 13h
       db 19h, 42h, 64h, 44h, 88h
.code
mov bx,ROWSIZE
mov al,array[bx]          ; AL = 19h
```



# Based-Index Operands

- Add the value of a base register to an index register, producing an effective address of 0157:

**`BX = 0155, SI = 0002`**



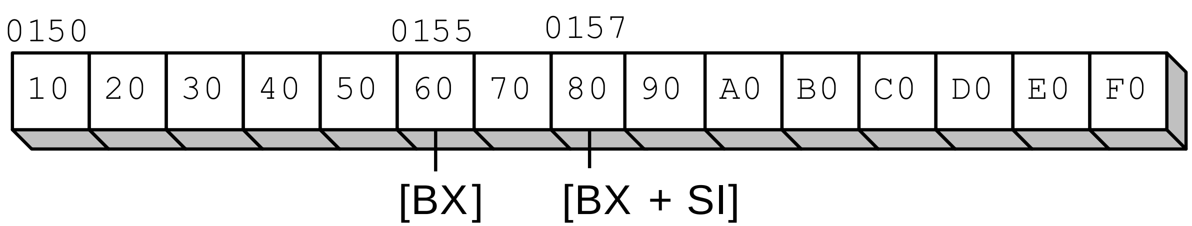
# Base-Index Example

```
.data
ROWSIZE = 5
array   db  10h, 20h, 30h, 40h, 50h
        db  60h, 70h, 80h, 90h,0A0h
        db 0B0h,0C0h,0D0h,0E0h,0F0h

.code
mov  bx,offset array     ; point to the array at 0150
add  bx,ROWSIZE          ; choose second row
mov  si,2                ; choose third column
mov  al,[bx + si]        ; get the value at 0157
```

# Base-Index with Displacement

```
.data
ROWSIZE = 5
array db  10h, 20h, 30h, 40h, 50h
      db  60h, 70h, 80h, 90h,0A0h
      db 0B0h,0C0h,0D0h,0E0h,0F0h
.code
mov bx,ROWSIZE                    ; row 1
mov si,2                          ; column 2
mov dl,array[bx + si]             ; DL = 80h
```



# Chapter Reading

- Chapter # 10: Arrays And Addressing Modes
  - Topic 10.1(One Dimensional Array)
  - Topic 10.2(10.2.1, 10.2.2, 10.2.5)(Addressing Modes)
  - Topic 10.3 (Sorting of Array)
  - Topic10.4, 10.5 (2D Array )