Arrays and Addressing Modes

Module 10
COSC 2329
Sam Houston State University
Dr. Tim McGuire

Arrays

- A one-dimensional array is an ordered list of elements, all of the same type.
- To define an array in assembly language
  - `W dw 10,20,30,40,50,60`
- The address of the array variable is called the base address of the array
- If the offset address of the array is 0200h, the array looks like this in memory:

element	offset address	symbolic address	contents
W[0]	0200h	W	10
W[1]	0202h	W+2h	20
W[2]	0204h	W+4h	30
W[3]	0206h	W+6h	40

The TIMES Operator

- Arrays whose elements share a common initial value are defined using the TIMES pseudo-op
- It has the form:
  - ***TIMES repeat_count   directive***
- **`gamma  TIMES 100 dw 0`**
  - sets up an array of 100 words, with each entry initialized to 0
- For uninitialized data we use the **resb** directive:
  - **`delta resb 212`**
    - sets up an array of 212 uninitialized bytes

Location of Array Elements

- The address of an array element may be computed by adding a constant to the base address
- If A is an array and S denotes the number of bytes in an element, then the address of element A[i] is A + i*S (assuming zero-based arrays; for one-based arrays it would be A + (i-1)*S)
- To exchange W[9] and W[24] in an word array W:

```
mov  ax,[W+18]    ; ax has W[9]
xchg [W+48],ax    ; ax has W[24]
mov  [W+18],ax    ; complete exchange
```

Addressing Data in Memory

- Three forms:
  - Immediate data -- stored directly in machine code
    - example: `mov ax,5 ; 5 is an immediate value`
    - immediate operands are always source operands
  - Register data -- held in processor registers
    - example: `add ax,bx`
  - Memory data -- held in memory
    - processor calculates the 16-bit effective address

Memory-Addressing Modes

Direct	`mov ax, count`
Register-indirect	`mov ax, [bx]`
Base	`mov ax, [record + bp]`
Indexed	`mov ax, [array + si]`
Base-indexed	`mov ax, [recordArray + bx + si]`
String	`lodsw`
I/O Port	`in ax, dx`

Direct Addresses

- Direct address references are usually relative to **`ds`**
- To change this, use a ***segment override***:
  - **`mov    ch, es:OverByte`**
  - Other segment bases are possible:
  - **`mov    dh, cs:CodeByte`**
  - **`mov    dh, ss:StackByte`**
  - **`mov    dh, ds:DataByte`**
- An override occupies a byte of machine code which is inserted just before the affected instruction

Register-Indirect Mode

- The offset address of the operand is contained in a register
- The register acts as a *pointer* to the memory location
- The operand format is
  - **`[register]`**
  - The register is **`bx`**, **`si`**, **`di`**, or **`bp`**
- For **`bx`**, **`si`**, or **`di`**, the segment register is **`ds`**
- For **`bp`**, **`ss`** has the segment number

Example

- **si** = 0100h, [0100h] = 1234h
- To execute **mov ax,[si]** the CPU
  - examines **si** and obtains the offset address **0100h**
  - uses the address **ds**:**0100h** to obtain the value **1234h**
  - moves **1234h** into **ax**
- This is not the same as **mov ax**,**si** which simply moves the value of **si** (**0100h**) into **ax**

Another example

bx=1000h, **si**=2000h, **di**=3000h, [1000h]=1BACh, [2000h]=20FEh, [3000h]=031Dh

instruction		source offset	result
mov	**bx,[bx]**	**1000h**	**1BACh**
mov	**cx,[si]**	**2000h**	**20FEh**
mov	**bx,[ax]**	*illegal source register*	
add	**[si],[di]**	*illegal memory-memory* **add**	
inc	**word [di]**	**3000h**	**031Eh**

Processing Arrays using Register-Indirect Mode

⌘ Sum in ax the 10-element word array W

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

        xor     ax,ax           ; ax holds sum
        mov     si,W            ; si points to array W
        mov     cx,10           ; cx has number of elements
addnos:
        add     ax,[si]         ; sum = sum + element
        add     si,2            ; move pointer to the next
                                ;     element
        loop    addnos          ; loop until done
```


WORD and BYTE operators

- ⌘ Both operands of an instruction must be of the same type
  - **`mov ax,1`** is a word operation because **`ax`** is a 16-bit register
  - **`mov bh,5`** is a byte operation
  - **`mov [bx],1`** is illegal because the assembler can't tell whether the destination is a byte or a word
- ⌘ if you want the destination to be a byte, use

  `mov BYTE [bx],1`
- ⌘ and if you want it to be a word, use

  `mov WORD [bx],1`

Based and Indexed Addressing Modes

- The operands offset address is obtained by adding a number called a *displacement* to the contents of a register
- The displacement may be:
  - the offset address of a variable, *e.g.*, **`A`**
  - a constant, *e.g.*, **`-2`**
  - the offset address of a variable plus or minus a constant, *e.g.*, **`A + 4`**

Syntax of an operand

- Any of the following expressions are equivalent:
  - **`[register + displacement]`** ← ***preferred form***
  - **`[displacement + register]`**
  - **`[register] + displacement`**
  - **`displacement + [register]`**
  - **`displacement[register]`**
- The register must be **`bx`**, **`bp`**, **`si`**, or **`di`**.
- If **`bx`**, **`si`**, or **`di`** is used, **`ds`** contains the segment number
- If **`bp`** is used, **`ss`** has the segment number
- The addressing is called ***based*** if **`bx`** or **`bp`** is used; it is called ***indexed*** if **`si`** or **`di`** is used

Application of Index Mode

⌘Replace the lowercase letters in the string to uppercase using index addressing mode

```
msg     db   "this is a message"
        mov  cx,17         ; # chars in string
        xor  si,si         ; si indexes a char
top:    cmp  [si+msg],' '  ; blank?
        je   next          ; yes, skip over
        and [si+msg],0DFh  ; no, convert to upper
next:   inc  si            ; index next byte
        loop top           ; loop until done
```


Processing Arrays using Based Addressing Mode

⌘Sum in ax the 10-element word array W using based mode

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

       xor    ax,ax        ; ax holds sum
       xor    bx,bx        ; clear base register
       mov    cx,10        ; cx has number of elements
addnos:
       add    ax,[bx+W]    ; sum = sum + element
       add    bx,2         ; index next element
       loop   addnos       ; loop until done
```

Base-Indexed Addressing Mode

- In this mode the offset address is the sum of:
  - the contents of a base register (**bx** or **bp**)
  - the contents of an index register (**si** or **di**)
  - optionally, a variable's offset address
  - optionally, a positive or negative constant
- There are many valid ways to write the operand, some of them are:
  - **[base + index + variable + constant]** ← *preferred*
  - **variable[base + index + constant]**
  - **constant[base + index + variable]**

Use of Based, Indexed, and Base-Indexed Modes

- Based and indexed addressing mode is often used for array and string processing
- Based-indexed addressing mode can be used for two dimensional arrays
- We will discuss these in greater detail later

LEA vs. MOV

- lea ax,[data]

and

```
mov  ax, data
```

do the same thing, but the **mov** is more efficient

- However,

```
lea  bx,[A + si]
```

is more efficient than

```
mov  bx, A
add  bx, si
```


Two-Dimensional Arrays

- A 2D array is an array of arrays
- Usually view them as consisting of rows and columns

```
A[0,0]   A[0,1]   A[0,2]   A[0,3]
A[1,0]   A[1,1]   A[1,2]   A[1,3]
A[2,0]   A[2,1]   A[2,2]   A[2,3]
```

- Elements may be stored in row-major order or column-major order

Locating an Element in a 2D Array

- A is an *M* x *N* array in row-major order, where the size of the elements is S
- To find the location of A[i,j]
  - find where row i begins
  - find the location of the j^{th} element in that row
- Row 0 begins at location A -- row i begins at location A + i*N*S
- The j^{th} element is stored j*S bytes from the beginning of the row
- So, A[i,j] is in location A+(i*N + j)*S

2D Arrays and Based-Indexed Addressing Mode

- Suppose A is a 5x7 word array stored in row-major order. Write code to clear row 2.

```
       mov  bx,28            ; bx indexes row 2
       xor  si,si            ; si will index columns
       mov  cx,7             ; # elements in a row
clear: mov  [bx + si + A],0 ; clear A[2,j]
       add  si,2             ; go to next column
       loop clear            ; loop until done
```

Another Example:

⌘ Write code to clear column 3 -- Since A is a 7 column word array we need to add 2*7 = 14 to get to the next row

```
       mov  si,6            ; si indexes column 3
       xor  bx,bx           ; bx will index rows
       mov  cx,5            ; #elements in a column
clear: mov  [bx + si + A],0 ; clear A[i,3]
       add  bx,14           ; go to next row
       loop clear           ; loop until done
```