EC-310 Microprocessor & Microcontroller based Design

Dr. Ahsan Shahzad

Department of Computer and Software Engineering (DC&SE)

College of Electrical and Mechanical Engineering (CEME)

National University of Science and Technology (NUST)

Week # 5

Arrays and Addressing Modes

- **Arrays:**
  - Single name to a group / collection of similar data items
  - Array name holds the address of the 1st element
  - Any element can be accessed by using an Index
- **2-Dimensional Array:**
  - Array whose each element is also an array (Array of arrays)
- **Addressing Modes:**
  - The way an operand is specified in the instruction

1-D Arrays

Declaration of Arrays

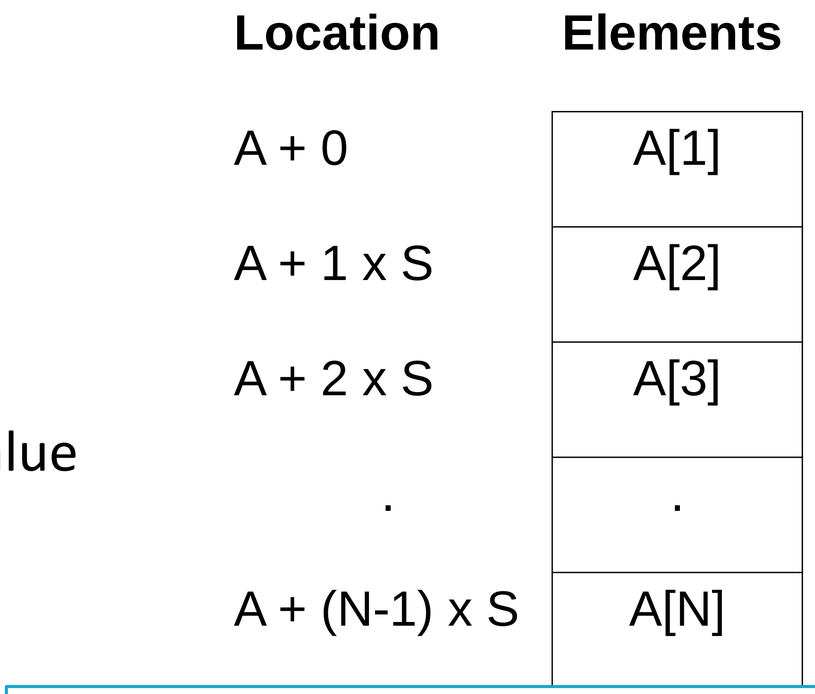
- 1-D Arrays: ordered list of elements, having same type
- The address of the array variable is called the **Base address of the array**

1) DB and DW pseudo-ops

- MSG **DB** ‘abcde’
- W **DW** 10,20,30,40,50,60

2) The DUP Operator

- To define arrays whose elements share a common initial value
  - Repeat_count DUP (Initial_Value)
  - Delta **DW** 212 **DUP**(?)
  - Line **DB** 5,4, 3 **DUP** (2, 3 **DUP** (0), 1)



S = 1 for byte & S = 2 for word array

Addressing Modes

1. **Register Mode** - Operand is a register
2. **Immediate Mode** - Operand is a constant
3. **Direct Mode** - Operand is a variable

Examples:

MOV AX, 0
; Destination AX is register mode,
; Source 0 is immediate mode

ADD ALPHA, AX
; Destination ALPHA is direct mode,
; AX is register mode

4. **Register Indirect Mode**
5. **Based Mode**
6. **Indexed Mode**
7. **Based Indexed Mode**

To address memory operands indirectly

Register Indirect Mode

- Register acts as a Pointer to memory location
- Operand Format is: **[Register]**
- Supported Registers: BX, BP, SI, DI
- Segment Address: For BX, SI, DI -> **DS** and For BP -> **SS**

- **Examples:**
  - MOV AX, [SI]
  - Write a code to sum the elements of the 10-element array

    W DW 1,2,3,4,5,6,7,8,9,10

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

        LOOP ADDNOS
```

Based and Indexed Addressing Modes

- The operand's offset address is obtained by adding a number called **Displacement**
- **Displacement:**
  - The offset address of a variable
  - A constant (+ve or -ve)
  - E.g. A , -2, A +4
- Supported Registers: BX, SI, DI -> **DS** and BP -> **SS**
- **Possible Formats:**
  - [Reg. + Displ.] -> MOV AX, [BX + W]
  - [Displ. + Reg.] -> MOV AX, [W + BX]
  - [Reg.] + Displ. -> MOV AX, [BX] + W
  - Displ. + [Reg.] -> MOV AX, W + [BX]
  - Displ.[Reg.] -> MOV AX, **W[BX]**

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

- Based Addressing Mode -> BX or BP AND Indexed Addressing Mode -> DI, SI used

Based and Indexed Addressing Modes

- Examples:

Example 10.6 Suppose that ALPHA is declared as

```
ALPHA   DW   0123h,0456h,0789h,0ABCDh
```

in the segment addressed by DS. Suppose also that

BX contains 2
SI contains 4
DI contains 1

Offset 0002 contains 1084h
Offset 0004 contains 28ACh

Tell which of the following instructions are legal. If legal, give the source offset address and the number moved.

```
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]
```

PTR Operator

- MOV AX, 5 Vs. MOV AL, 5

Legal: Assembler infers type from other operand

Word type Vs. Byte type instruction

- MOV [BX], 1 **;illegal**
- **Sol:** MOV **BYTE PTR** [BX], 1 ; for destination is to be a byte
- OR: MOV **WORD PTR** [BX], 1 ; for word type destination

Using PTR to Override a Type

- Type PTR address_expression
- Type: BYTE, WORD, DWORD
  - DOLLARS DB 1Ah
  - CENTS DB 52h
  - MOV AX, DOLLARS ; **illegal**
  - MOV AX, **WORD PTR** DOLLARS ; Legal, AH = cents, AL = Dollars , AX = 521Ah

Example: Replace 't' by 'T'

```
'MSG  DB  'this is a msg'

; Register Indirect
LEA SI, MSG
MOV BYTE PTR [SI], 'T'

;Using Index mode
XOR SI,SI
MOV  MSG[SI], 'T'
```

LABEL Pseudo-op

- Can be used to avoid type-conflict problems
  - MONEY LABEL WORD ; MONEY declared as word variable
  - DOLLARS DB 1Ah
  - CENTS DB 52h
- No memory is going to reserve for MONEY, instead MONEY and DOLLARS being assigned the same address by the assembler
  - MOV AX, MONEY ; Legal, AH = cents, AL = Dollars
  - MOV AL, DOLLARS ; Legal
  - MOV AH, CENTS ; Legal

Segment Override

- BX, SI, DI -> DS BP, SP-> SS
- Default segment Register can be override
- Format: **Segment_register: [pointer_register]**
- Example: MOV AX, **ES:[SI]**
- Use Case: When ES represents 2nd data segment.

2-D Arrays

- An array of arrays
- How 2-D arrays stored in 1-D memory?
- Most High-level lang. compilers used row major order
- Location of an Element
  - Find where row ith begins
  - Find the location of the jth element

- Row Major order A(M x N): A[i,j] -> A + ((i-1) x N + (j-1)) x S
- Col. Major order A (M x N): A[i,j] -> A + ((i-1) + (j-1)x M) x S

- For A is an M x N word array stored in a row-major order

1. Where does row i begin?
2. Where does column j begin?

  - A[i, 1]: A + (i-1) x N x 2
  - A[1, j]: A + (j-1) x 2

Based Indexed Addressing Modes

- Both Base and index registers are used
- The offset address is the sum of **base + index + displacement** (optional)
- MOV AX, W[SI][BX]
- MOV AX, [W+BX+SI]

Overview of Addressing Modes

- Assume **DS:** 1000h, **BX:** 0200h, **SI:** 0300h

Addressing Mode	Example	Source Operands		
		Working	Address Generation	Address
Register	MOV AX, **BX**	Copy BX's contents to AX	--	--
Immediate	MOV AX, **0F7h**	Copy constant 0F7h to AX	--	--
Direct	MOV AX, **[1234h]**	Copy contents at memory location 11234h to AX	**DS** x 10h + 1234h	11234h
Reg. Indirect	MOV AX, **[BX]**	Copy contents at mem. Loc. Pointed by BX (i.e. 10200h) to AX	**DS** x 10h + 0200h	10200h
Based	MOV AX**, [BX+6]**	Copy contents at mem. Loc. Pointed by BX+6 (10206) to AX	**DS** x 10h + 0200h + 0006h	10206h
Indexed	MOV AX, **[SI+6]**	Copy contents at mem. Loc. pointed by SI+6 (10306h) to AX	**DS** x 10h + 0300h + 0006h	10306h
Based-Indexed	MOV AX, **[BX+SI+6]**	Copy contents at mem. Loc. Pointed by BX+SI+6 (10506h) to AX	**DS** x 10h + 0200h + 0300h + 0006h	10506h

XLAT instruction

- Convert one value into another value based on pre-defined table
- No operand instruction
- Computes [BX] + AL to find the complete offset address of look-up entry in table (Stored in memory)
- Used to translate a byte in AL using a table in the memory.
- Return the result in AL

```
TABLE   DB 030h,031h,032h,033h,034h,035,036h,037h,038h,039h
        DB 041h,042h,043h,044h,045h,046h
```

For instance, to convert 0Ch to "C", we do the following:

```
MOV  AL,0Ch          ;number to convert
LEA  BX,TABLE        ;BX has table offset
XLAT                 ;AL has 'C'
```

INCLUDE Pseudo-op

- To insert procedures into the program
- Format: **INCLUDE filepath**

Program Listing PGM9_2.ASM

```
TITLE  PGM9_2:  DECIMAL  OUTPUT
.MODEL   SMALL
.STACK   100H
.CODE
MAIN   PROC
       CALL   OUTDEC
       MOV    AH,4CH
       INT    21H          ;DOS exit
MAIN   ENDP
INCLUDE A:PGM9_1.ASM
       END    MAIN
```

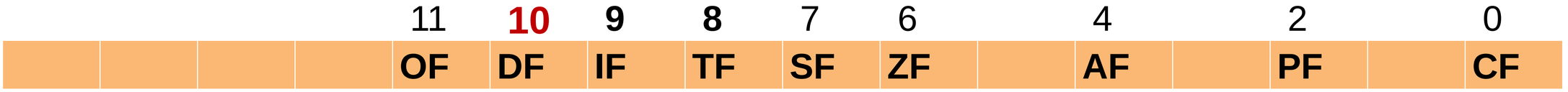
Program Listing PGM9_1.ASM

```
1:   OUTDEC  PROC
2:   ;prints AX as a signed decimal :
3:   ;input: AX
4:   ;output: none
5:            PUSH  AX          ;save
6:            PUSH  BX
7:            PUSH  CX
8:            PUSH  DX
9:   ;if AX < 0
```

Flags: The Direction Flag

STRING1 DB 'ABC' ; Assume string going to save at offset 0200h		
Offset Address	**Content**	**ASCII Character**
0200h	041h	A
0201h	042h	B
0202h	043h	C

- **6 Status Flags:** reflect the result of an operation
- **3 Control Flags:** are used to control the processor's operation
- **Direction FLAG:**
  - determine the direction in which the string operations will proceed
- String operations used two index registers SI and DI
  - If **DF = 0, SI and DI** proceed in the direction of **increasing** memory address(from left to right across the string)
  - If **DF=1, SI and DI** will proceed in the direction of **decreasing** memory addresses (from right to left)



The Flags Register

Flags: Interrupt and Trap Flags

- **Interrupt Flag (IF):** This flag is for **interrupts**
  - If **IF = 1**, the microprocessor will recognize interrupt requests from the peripherals.
  - If **IF = 0**, the microprocessor will ignore all interrupt requests.

- **TRAP FLAG (TF):** This flag is used for **on-chip debugging**
  - Setting trap flag puts the microprocessor into single step mode for debugging. In single stepping, the microprocessor executes a instruction and enters into single step ISR.
  - If trap flag is **set (1),** the CPU automatically generates **an internal interrupt after each instruction,** allowing a program to be inspected as it executes instruction by instruction.
  - If trap flag is **reset (0)**, no function is performed.

				11	10	9	8	7	6		4		2		0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values

- **STC –** Used to set carry flag CF to 1
- **CLC –** Used to clear/reset carry flag CF to 0
- **CMC –** Used to put complement at the state of carry flag CF.
- **STD –** Used to set the direction flag DF to 1
- **CLD –** Used to clear/reset the direction flag DF to 0
- **STI –** Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI –** Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

- **INT –** Used to interrupt the program during execution and calling service specified.
- **INTO –** Used to interrupt the program during execution if OF = 1
- **IRET –** Used to return from interrupt service to the main program.

The STRING Instructions

- In 8086 assembly language, string is simply a byte or word array
- Following Operations supported:
  - Copy / Move a string into another string
  - Search a string for a particular byte or word
  - Store characters in a string
  - Compare strings of characters alphabetically
- String Instructions have built-in advantages:
  - Automatic updating of pointer registers
  - Allow memory-memory operations

Copy / Moving a String (MOVSB, MOVSW)

- Move the contents of string1 (source) pointed by DS:SI to destination string addressed by ES:DI
- Auto update SI and DI. (increments if DF = 0 and decrements if DF = 1)
- Format:
  - MOVSB ; byte transfer,
  - MOVSW ; word transfer
- REP prefix: Repeat until counter CX become zero
- **Example:**

1. Write instructions to copy String1 into String 2 in reverse order
2. Arr Dw 10,20,40,50,60,?
   Insert 30 b/w 20 and 40

```
LEA   SI,STRING1+4
LEA   DI,STRING2
STD
MOV   CX,5
MOVE:
MOVSB
ADD   DI,2
LOOP MOVE
```

```
STD
LEA SI,ARR+8h
LEA DI,ARR+Ah
MOV CX,3
REP MOVSW
MOV WORD PTR [DI],30
```

Storing a String

- STOSB
- STOSW
- move the contents of AL or AX into byte word addressed by ES:DI

- NO effect on flags (Copy, Storing and Loading instructions)
- INT 21H function 0Ah can be implemented by using STOS + INT 21h function 1

LOAD a String

- LODSB
- LODSW
- Byte or word addressed by DS:SI to AL or AX
- No effect on flags

SCAN a String

- SCASB
- SCASW
- Target value in AL or AX is subtracted from byte or word addressed by ES:DI
- will effect flags

- **REPNE** Prefix: Repeat until not zero/equal and CX!= 0

Summary

- COMPARE String Homework reading

Instruction	*Destination*	*Source*	*Byte form*	*Word form*
Move string	ES:DI	DS:SI	MOVSB	MOVSW
Compare string*	ES:DI	DS:SI	CMPSB	CMPSW
Store string	ES:DI	AL or AX	STOSB	STOSW
Load string	AL or AX	DS:SI	LODSB	LODSW
Scan string	ES:DI	AL or AX	SCASB	SCASW

* Result not stored.