

# AACS3064

# Computer Systems Architecture

## Chapter 8: Assembly Language Fundamental II

# Chapter Overview

## 1) Arithmetic

- Multiplication & Division
- Shifting

## 2) Multiple Initializers

## 3) Direct-Offset Operands

## 4) Data related operators & Directives

- (DUP, OFFSET, TYPE, LENGTHOF, ALIGN, SIZEOF)

## 5) Indirect Operands

## 6) Unconditional Jump and Loop

# 1.Arithmetic

# 1.Arithmetic

## MUL **instruction**

---

- ▶ Performs multiplication on unsigned data.
- ▶ Affect the carry and overflow flags.
- ▶ Format:

<code>MUL    <i>register</i> / <i>memory</i></code>
---

# 1.Arithmetic (Continued)

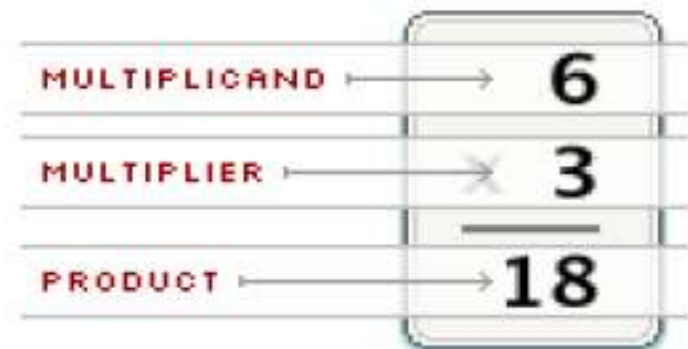
## MUL **instruction**

- Byte times byte (8-bit)

Before :

AH (.....)	AL (Multiplicand)
AX (Product)	

After:



# 1.Arithmetic (Continued)

## MUL **instruction**

- ▶ Word times word (16-bit)

Before :

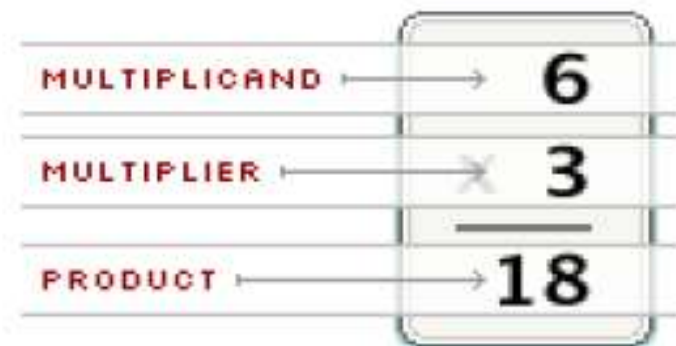
DX  
(Ignored)

AX  
(Multiplicand)

After:

DX  
(High product)

AX  
(Low product)



# 1.Arithmetic (Continued)

## MUL **instruction**

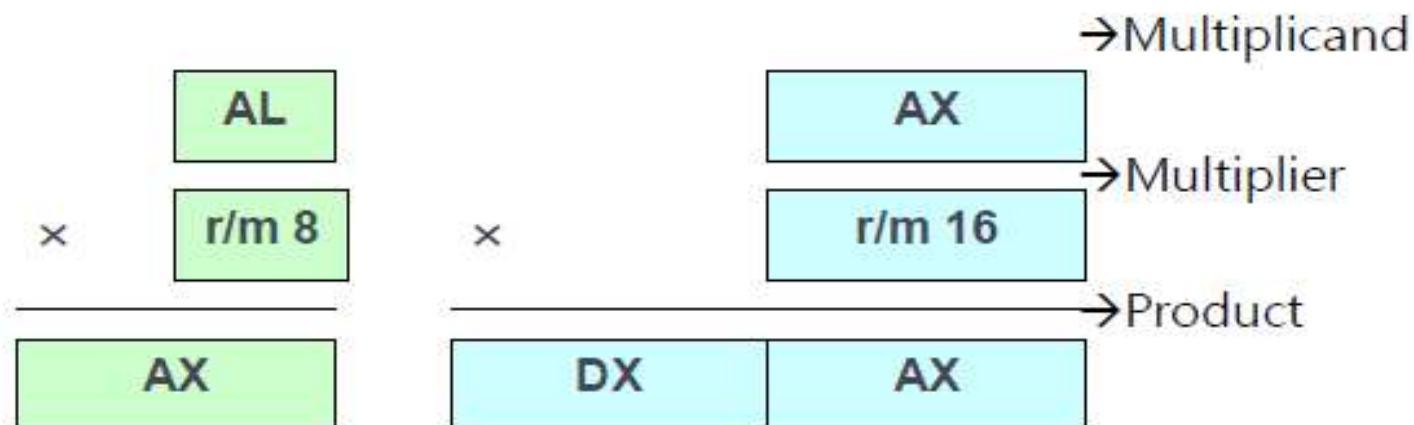
- ▶ When the multiplier is in register, the size of the register determines the type of operation :

<i>Instruction</i>		<i>Multiplier</i>	<i>Multiplicand</i>	<i>Product</i>
<b>MUL</b>	CL	Byte	AL	AX
<b>MUL</b>	BX	Word	AX	DX:AX

# 1.Arithmetic (Continued)

## MUL **instruction**

- ▶ Note that the product is stored in a register (or group of registers) twice the size of the operands.





# 1.Arithmetic (Continued)

## MUL instruction

E.g.

```
.DATA
    var1      DB      80H
    var2      DB      40H
    word1     DW      8000H
    word2     DW      2000H

.CODE

    MOV     AL, var1          ; byte x byte
    MUL     var2

    MOV     AX, word1        ; word x word
    MUL     word2

    MOV     AL, var1          ; byte x word
    SUB     AH, AH
    MUL     word1
```

# 1.Arithmetic (Continued)

## DIV **instruction**

---

- ▶ Performs division on unsigned data.
- ▶ Format:

**DIV**   *register / memory*

# 1.Arithmetic (Continued)

## DIV instruction

- Byte into word (8-bit divisor)

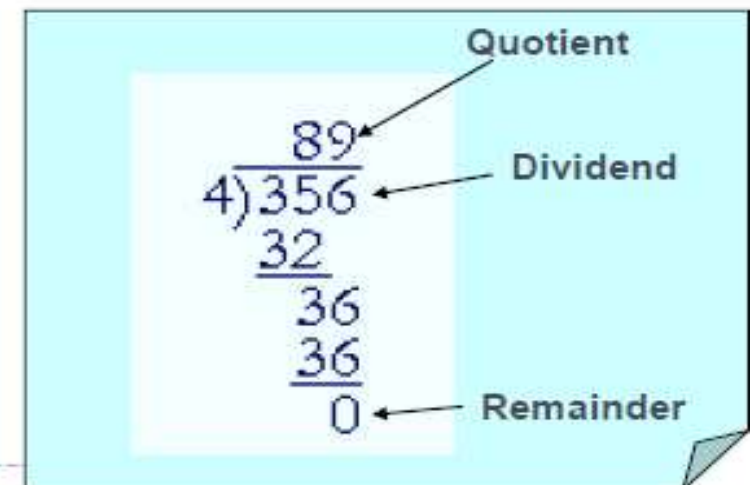
Before :

AX  
(Dividend)

After:

AH  
(Remainder)

AL  
(Quotient)



# 1.Arithmetic (Continued)

## DIV **instruction**

- ▶ Word into doubleword (16-bit divisor)

Before :

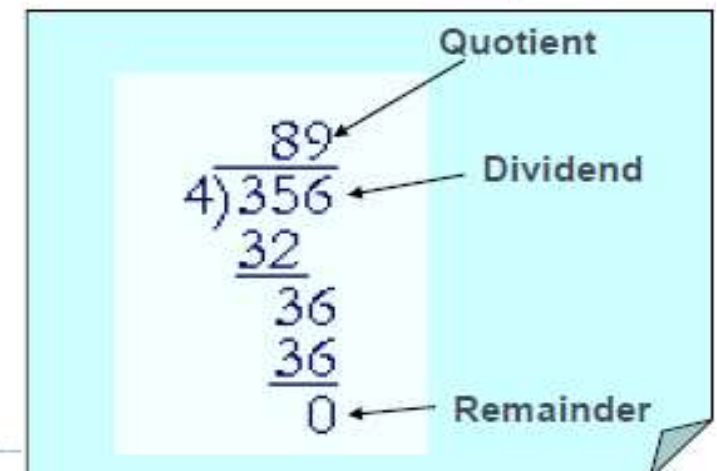
DX (High Dividend)
-----------------------

AX (Low dividend)
----------------------

After:

DX (Remainder)
-------------------

AX (Quotient)
------------------



# 1.Arithmetic (Continued)

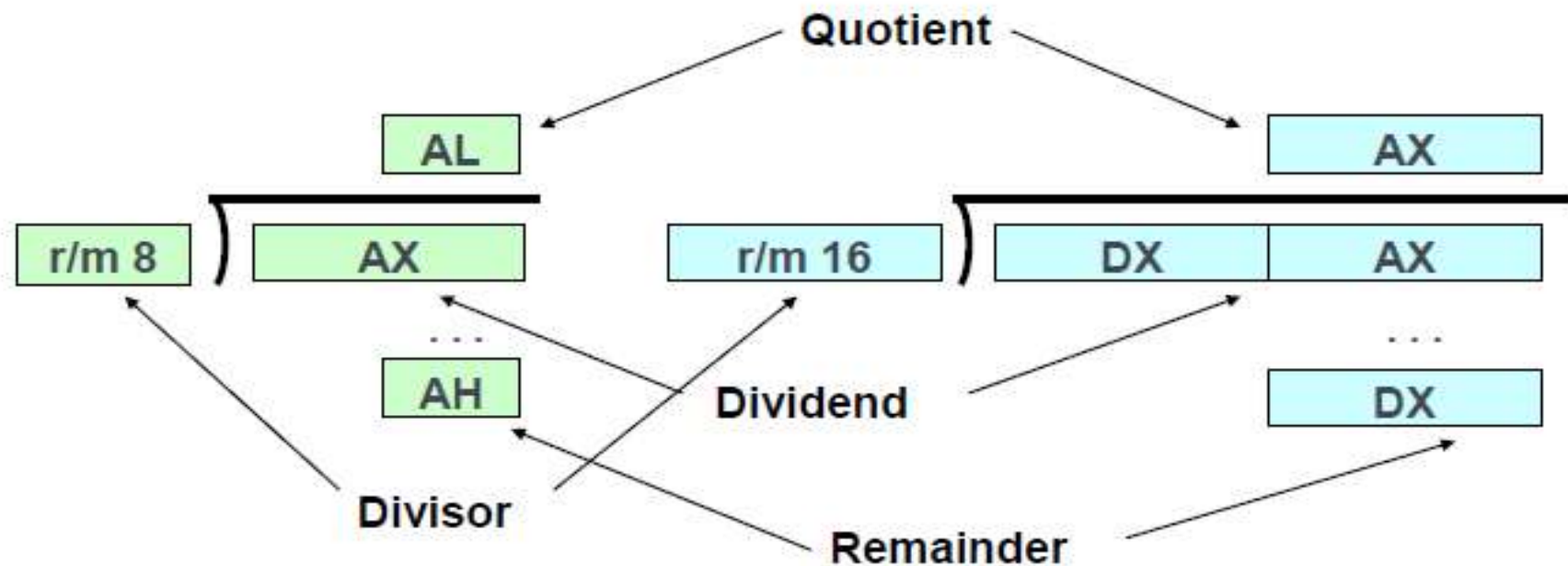
## DIV instruction

- ▶ When the divisor is in register, the size of the register determines the type of operation :

<i>Instruction</i>	<i>Divisor</i>	<i>Dividend</i>	<i>Quotient</i>	<i>Remainder</i>
<b>DIV CL</b>	Byte	AX	AL	AH
<b>MUL BX</b>	Word	DX:AX	AX	DX

# 1.Arithmetic (Continued)

## DIV instruction



# 1.Arithmetic (Continued)

## DIV instruction

E.g.

```
.CODE
    ; doubleword / word
    MOV     DX, 0
    MOV     AX, 8003H
    MOV     CX, 100H
    DIV     CX
```

AX =

DX =



# 1.Arithmetic (Continued)

## DIV **instruction**

E.g.

```
.DATA
    var1      DB      80H
    var2      DB      16H
    word1     DW      2000H
    word2     DW      0010H
    word3     DW      1000H

.CODE
    MOV     AX, word1      ; word / byte
    DIV     var1
    MOV     AL, var1       ; byte / byte
    SUB     AH, AH
    DIV     var2
    MOV     DX, word2      ; doubleword / word
    MOV     AX, word3
    DIV     word1
```



# 1.Arithmetic (Continued)

## CBW instruction

---

▶ E.g.

▶ Assume AL = 60H

```
CBW          ; Extend AL sign into AH
ADD    AX, 20H ; Add to AX
```

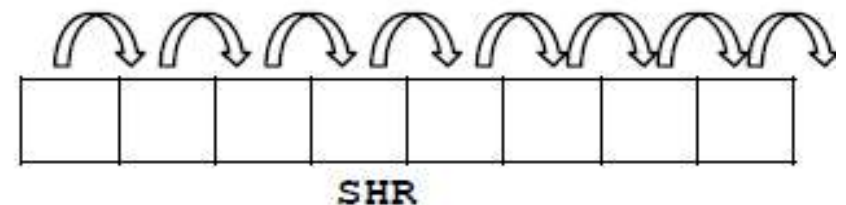
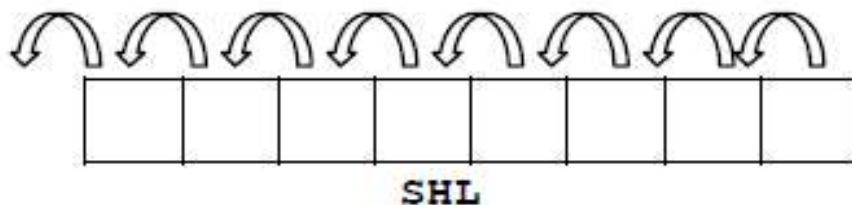
▶ AX =

# 1.Arithmetic (Continued)

## Shifting **instruction**

- ▶ SHL (Shift Left) : shift the bits to the left.
- ▶ SHR (Shift Right) : shift the bits to the right.
- ▶ Format:

SHL	<i>destination, 1/CL</i>
SHR	<i>destination, 1/CL</i>



# 1.Arithmetic (Continued)

## Shifting **instruction**

---

- ▶ E.g.
- ▶ A bit that is shifted off enters the carry flag.

SHR	AL, 1
-----	-------

Before	0000 0100B
After	0000 0010B

- ▶ A 0 is shifted into the msb position.
-

# 1.Arithmetic (Continued)

## Shifting **instruction**

---

► E.g.

```
MOV    CL, 03
MOV    AL, 10110111B
SHR    AL, 01           ; AL =
SHR    AL, CL           ; AL =
```

# 1.Arithmetic (Continued)

## Shifting **instruction**

---

- ▶ E.g.
- ▶ A bit that is shifted off enters the carry flag.

<code>SHL</code>	<code>AL, 1</code>
------------------	--------------------

Before	0000 0100B
After	0000 1000B

- ▶ A 0 is shifted into the rightmost bit position.

# 1.Arithmetic (Continued)

## Shifting **instruction**

---

► E.g.

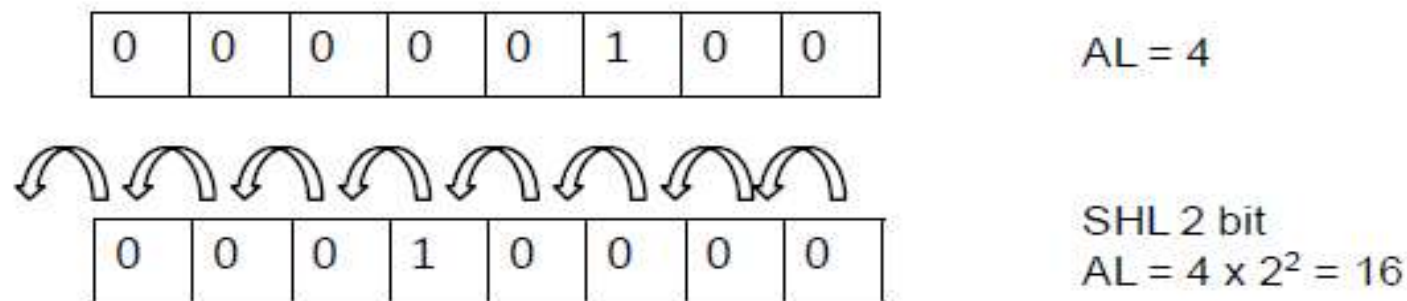
```
MOV    CL, 03
MOV    AL, 10110111B
SHL    AL, 01          ; AL =
SHL    AL, CL          ; AL =
```

# 1.Arithmetic (Continued)

## Shifting **instruction**

---

- ▶ To multiply or divide by a power of two.
  - ▶ A left shift will multiply  $2^n$ .
  - ▶ A right shift will divide by  $2^n$ .
  - ▶ where  $n$  is the number of bits to shift.



## 2. Multiple Initializers

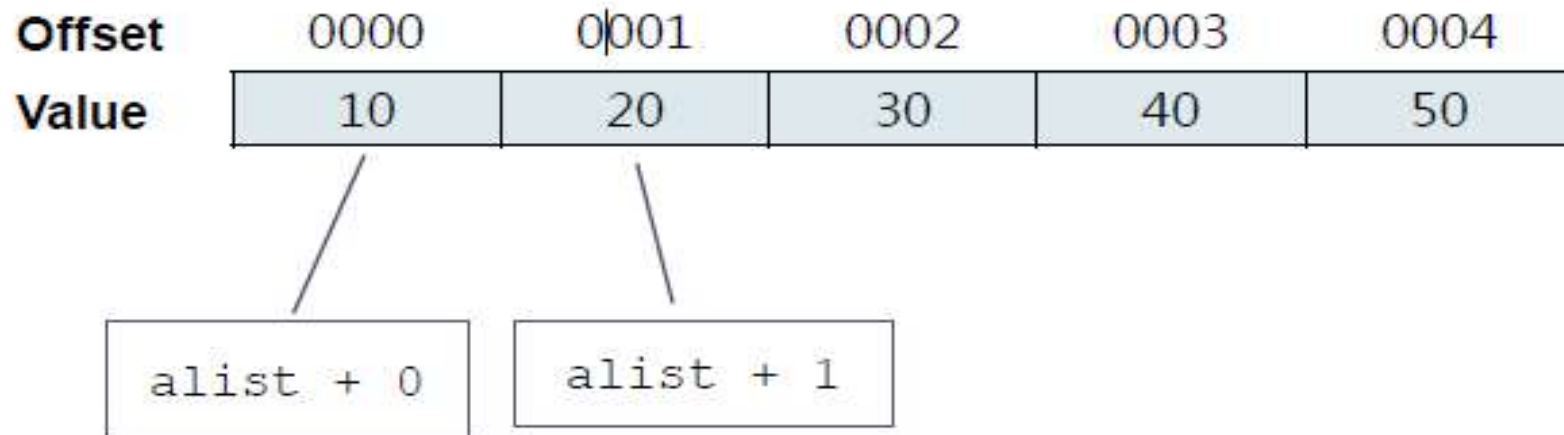


## 2. Multiple Initializers

### Multiple Initializers

- If a definition has multiple initializers, the label is the offset for the first data item

```
alist    DB    10, 20, 30, 40, 50
```



## 3. Direct-Offset Operands

# 3. Direct-Offset Operands

## Direct-Offset Operands

- ▶ Add a displacement to the name of a variable, creating a direct-offset operand.
- ▶ Enable access to memory locations that may not have explicit labels.

```
.DATA
    arrayB    DB    10H, 20H, 30H, 40H, 50H
.CODE
    MOV    AL, arrayB           ; AL =
    MOV    AL, [arrayB + 1]     ; AL =
    MOV    AL, [arrayB + 2]     ; AL =
```

# 3. Direct-Offset Operands

## Direct-Offset Operands

- ▶ Add a displacement to the name of a variable, creating a direct-offset operand.
- ▶ Enable access to memory locations that may not have explicit labels.

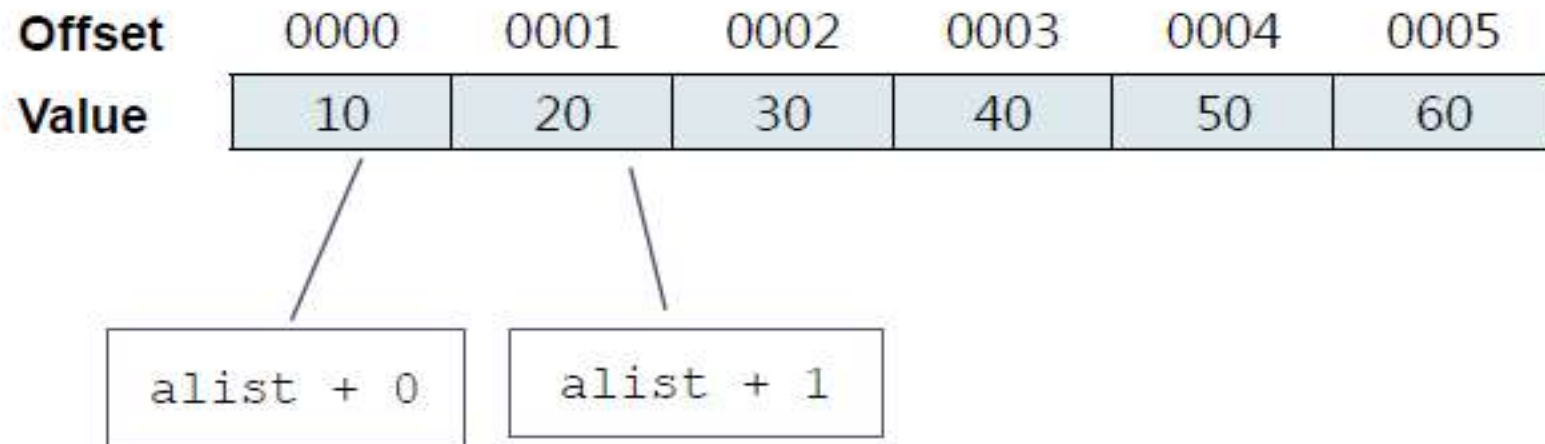
```
.DATA
    arrayB    DB    10H, 20H, 30H, 40H, 50H
.CODE
    MOV    AL, arrayB           ; AL =
    MOV    AL, [arrayB + 1]     ; AL =
    MOV    AL, [arrayB + 2]     ; AL =
```

### 3. Direct-Offset Operands (Continued)

#### Multiple Initializers

---

<code>alist</code>	<code>DB</code>	<code>10, 20, 30,</code>
	<code>DB</code>	<code>40, 50, 60</code>



## 3. Direct-Offset Operands (Continued)

### Multiple Initializers

---

- Different initializers can use different radices

<code>alist</code>	<code>DB</code>	<code>10, 25, 41H, 00100010B</code>
<code>blist</code>	<code>DB</code>	<code>0BH, 'A', 60</code>

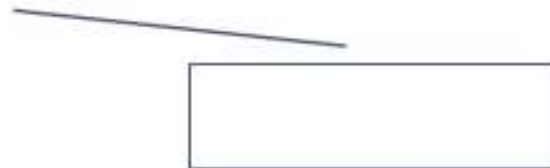
### 3. Direct-Offset Operands (Continued)

#### Defining Data – ( 16-bit )

value1	DW	65535	; unsigned word
value2	DW	-32768	; signed word

```
myList  DW  1, 2, 3, 4, 5
```

Offset	0000	0002	0004	0006	0008
Value	1	2	3	4	5





# 3. Direct-Offset Operands (Continued)

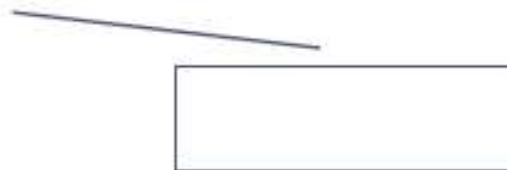
## Defining Data – ( 32-bit )

value1	DD	12345678H	; unsigned doubleword
value2	DD	-21474836648	; signed doubleword

► E.g.

```
myList DD 1, 2, 3, 4, 5
```

Offset	0000	0004	0008	000C	0010
Value	1	2	3	4	5





## 4. Data related operators & Directives

## 4. Data related operators & Directives

### DUP **Operator**

---

- ▶ Allocates storage for multiple data items, using a constant expression as a counter.
- ▶ It is useful when allocating space for a string or array, and can be used with initialized or uninitialized data.

```
array1 DB 20 DUP(0)           ; 20 bytes with zero   array2 DB  
20 DUP(?)                     ; 20 bytes uninitialized  
array3 DB 2 DUP("STACK")      ; "STACKSTACK"  
array4 DB 5, 4, 3 DUP (2, 3 DUP(0), 1)
```

## 4. Data related operators & Directives (Continued)

### OFFSET **Operator**

- ▶ The offset operator returns the number of bytes between the label and the beginning of its segment.
- ▶ It produce a 16-bit immediate value. Therefore, the destination must be a 16-bit operand.

```
.DATA
    bListDB    10H, 20H, 30H, 40H
    wList DW    1000H, 2000H, 3000H
.CODE
    MOV  DI, OFFSET bList           ; DI = 0000
    MOV  BX, OFFSET bList + 1       ; BX = 0001
    MOV  SI, OFFSET wList + 2       ; SI = 0008
```

## 4. Data related operators & Directives (Continued)

### LENGTHOF **Operator**

- ▶ Counts the number of elements in an array, defined by the values appearing on the same line as its label.

```
.DATA
    byte1      DB      10, 20, 30          ; LENGTHOF =
    array1     DW      30 DUP(?), 0, 0      ; LENGTHOF =
    array2     DW      5 DUP(3 DUP(?))      ; LENGTHOF =
    array3     DW      1, 2, 3, 4           ; LENGTHOF =
    digitStr   DB      "12345678$"         ; LENGTHOF =
```

## 4. Data related operators & Directives (Continued)

### LENGTHOF **Operator**

- ▶ Counts the number of elements in an array, defined by the values appearing on the same line as its label.

**.DATA**

byte1	DB	10, 20, 30	; LENGTHOF =
array1	DW	30 DUP(?), 0, 0	; LENGTHOF =
array2	DW	5 DUP(3 DUP(?))	; LENGTHOF =
array3	DW	1, 2, 3, 4	; LENGTHOF =
digitStr	DB	"12345678\$"	; LENGTHOF =



## 4. Data related operators & Directives (Continued)

### **ALIGN Operator**

---

- Aligns a variable on a byte, word, doubleword, or paragraph boundary.

bVal	BYTE	?	; 00404000
ALIGN	2		
wVal	WORD	?	;
bVal2	BYTE	?	;
ALIGN	4		
dVal	DWORD	?	;
dVal2	DWORD	?	;

## 4. Data related operators & Directives (Continued)

### **ALIGN Operator**

---

- Aligns a variable on a byte, word, doubleword, or paragraph boundary.

bVal	BYTE	?	; 00404000
ALIGN	2		
wVal	WORD	?	;
bVal2	BYTE	?	;
ALIGN	4		
dVal	DWORD	?	;
dVal2	DWORD	?	;

## 4. Data related operators & Directives (Continued)

### **SIZEOF Operator**

---

- ▶ Returns the number of bytes an array takes up.
- ▶ It is equivalent to multiplying LENGTHOF by TYPE.

```
.DATA
    intArray  DW    32 DUP(0)
.CODE
    MOV  AX,  SIZEOF intArray ; AX = 64
```



## 5. Indirect Operands

# 5. Indirect Operands

## Indirect Operand

- ▶ An indirect operand is a register containing the offset for data in the memory location.
- ▶ If the register is used as an indirect operand, it may only be SI, DI, BX, or BP. Avoid BP unless you are using it to index into the stack.

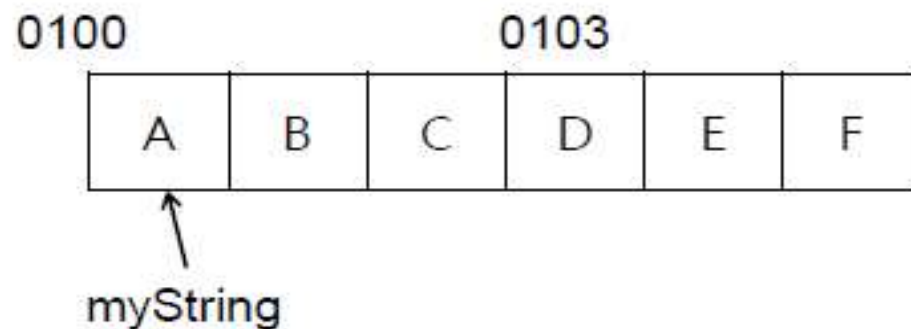
```
.DATA
    byteVal    DB    10h
.CODE
    MOV    SI, OFFSET byteVal
    MOV    AL, [SI]           ; AL = 10h
```

# 5. Indirect Operands (Continued)

## Indirect Operand

```
.DATA
    myString DB "ABCDEF"

.CODE
    MOV     BX, OFFSET myString      ; BX = 0100
    ADD     BX, 3                    ; BX = 0103
    MOV     DL, [BX]                 ; DL = 'D'
```



## 5. Indirect Operands (Continued)

### **Indirect Operand – Array of 8-bit**

---

- ▶ Useful in manipulating arrays.

```
.DATA
    arrayA      DB      10H, 20H, 30H

.CODE
    MOV     SI, OFFSET arrayA
    MOV     AL, [SI]           ; AL = 10H
    INC     SI
    MOV     AL, [SI]           ; AL = 20H
    INC     SI
    MOV     AL, [SI]           ; AL = 30H
    INC     SI
```

## 5. Indirect Operands (Continued)

### Indirect Operand – Array of 16-bit

```
.DATA
    arrayB      DW      1000H, 2000H, 3000H

.CODE
    MOV     SI, OFFSET arrayB
    MOV     AX, [SI]                ; AL = 1000H
    ADD     SI, 2
    MOV     AX, [SI]                ; AL = 2000H
    ADD     SI, 2
    MOV     AX, [SI]                ; AL = 3000H
    ADD     SI, 2
```



## 5. Indirect Operands (Continued)

### LEA **instruction**

---

- ▶ Stands for “Load Effective Address” .
- ▶ Initializes a register with an offset address.

```
.DATA
    dataTable    DB    25    DUP    (?)                value1
    DB    ?

.CODE
    LEA BX, dataTable ; equivalent to
                        ; MOV BX, OFFSET dataTable
    MOV    BYTEFLD, [BX]
```

## 6. Unconditional Jump and Loop



## 6. Unconditional Jump and Loop

### JMP **instruction**

---

- ▶ unconditional jump:
  - ▶ transfers control under all circumstances.
- ▶ Allows transfer of control to the target address (any instruction that has a label).
- ▶ Format:

```
JMP  short / near / far address
```



## 6. Unconditional Jump and Loop (Continued)

### JMP **instruction**

#### ► Format

Distance	Short	Near	Far
Instructions	-128 to 127 Same segment	-32,768 to 32,767 Same segment	Another segment
JMP	Yes	Yes	Yes
Jnnn	Yes	80386 / 486	No
LOOP	yes	No	No

## 6. Unconditional Jump and Loop (Continued)

### JMP **instruction**

---

- ▶ Instruction Labels

- ▶ The JMP, and LOOP instructions require an operand that refers to the label of an instruction.

- ▶ E.g.

```
                JMP  A90
                ...
A90:           MOV  AH, 00
                ...
```

## 6. Unconditional Jump and Loop (Continued)

### **JMP instruction**

---

#### ▶ Backward jump

A50 :

...

JMP A50

#### ▶ Forward jump

JMP A90

...

A90 :

## 6. Unconditional Jump and Loop (Continued)

### LOOP **instruction**

---

- ▶ Repeats a block of statements a specific number.
- ▶ CX is automatically used as a counter and is decremented each time the loop repeats.
- ▶ Format:

`LOOP shortAddress`

- ▶ Execution Steps:

`CX --`

`If (CX != 0)`

`jump to the target address.`

---

## 6. Unconditional Jump and Loop (Continued)

### LOOP instruction

---

► E.g.

Calculates the sum of the integers 1+2+3+4+5.

```
MOV  AX, 00H    ; AX = 0
MOV  CX, 05      ; CX = 5

L1:
INC  AX          ; Add 1 to AX
LOOP L1
```

## 6. Unconditional Jump and Loop (Continued)

### LOOP **instruction**

---

► E.g.

```
MOV AX,00
MOV BX,00
MOV CX,8    ;Initialize for 8 loops

A20:
    INC AX
    ADD BX,AX

    LOOP A20        ;Decrement CX
                    ;Repeat if nonzero
```



## 6. Unconditional Jump and Loop (Continued)

### Nested LOOP

- ▶ When creating a loop inside another loop, special consideration must be given to the outer loop counter in CX.

```
.DATA
    count DW    ?

.CODE
    MOV     CX, 100

L1:
    MOV     count, CX    ; save outer loop count
    MOV     CX, 20       ; set inner loop count
L2:
    ...
    LOOP    L2           ; repeat the inner loop
    MOV     CX, count    ; restore outer loop count
    LOOP    L1           ; repeat outer loop
```

## 6. Unconditional Jump and Loop (Continued)

### Summing an integer Array

- E.g. calculates the sum of an array of 8-bit integers.

```
.DATA
    arrayNum    DB    10H, 20H, 30H, 40H

.CODE
    MOV    DI, OFFSET    arrayNum    ; address
    MOV    CX, LENGTHOF arrayNum    ; loop counter
    MOV    AX, 0          ; zero the accumulator

L1 :
    ADD    AL, [DI]        ; add an integer
    INC    DI              ; point to next
    LOOP   L1
```



# Chapter Review

## 1) Arithmetic

- Multiplication & Division
- Shifting

## 2) Multiple Initializers

## 3) Direct-Offset Operands

## 4) Data related operators & Directives

- (DUP, OFFSET, TYPE, LENGTHOF, ALIGN, SIZEOF)

## 5) Indirect Operands

## 6) Unconditional Jump and Loop