Assembly Language for x86 Processors

- Arrray; Data-related Operators and Directives

CS 271 Computer Architecture

Purdue University Fort Wayne

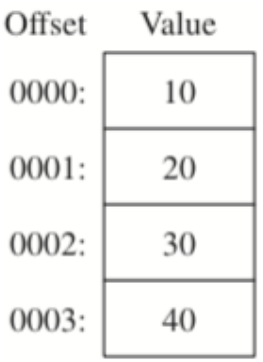
Outline

- Defining Arrays
- Data related directives
- Addressing

Defining Arrays

- Arrays use multiple initializers:

```
list1 BYTE 10,20,30,40
list2 BYTE 10,20,30,40
      BYTE 50,60,70,80
      BYTE 81,82,83,84
list3 BYTE ?,32,41h,00100010b
list4 BYTE 0Ah,20h,'A',22h
myList WORD  1,2,3,4,5 ; array of words
val4 SDWORD -3,-2,-1,0,1   ; signed array
```



Offset	Value
0000:	10
0001:	20
0002:	30
0003:	40

Using the DUP Operator

- Use DUP to allocate (create space for) an array or string.
- Syntax:
  - counter DUP (argument)
- Counter and argument must be constants or constant expressions

```
var1 BYTE 20 DUP(0)          ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)          ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")     ; 20 bytes: "STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20     ; 5 bytes
```

Defining Strings

- A string is implemented as an array of characters
  - For convenience, it is usually enclosed in quotation marks
  - It is often null-terminated
- Examples:

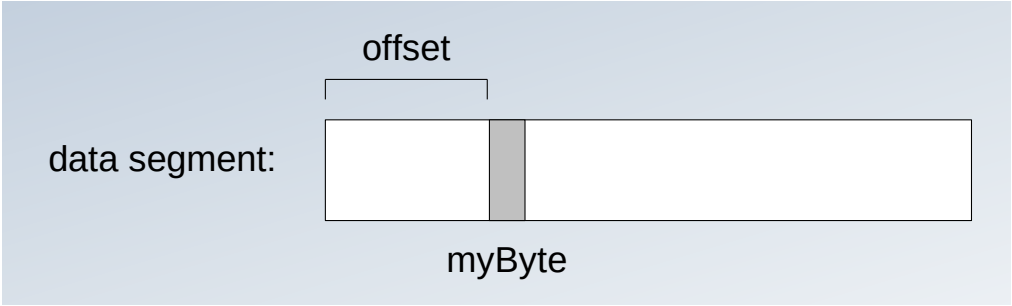
```
str1 BYTE "Enter your name",0
str2 BYTE 'Error: halting program',0
str3 BYTE 'A','E','I','O','U'
greeting  BYTE "Welcome to the Encryption Demo program "
          BYTE "created by Kip Irvine.",0
```

DATA-RELATED OPERATORS AND DIRECTIVES

- OFFSET Operator
- TYPE Operator
- LENGTHOF Operator
- SIZEOF Operator

OFFSET Operator

- OFFSET returns the distance in bytes, of a label from the beginning of its enclosing segment
- The value returned by OFFSET is a pointer.




```
// C++ version:

char array[1000];
char * p = array;
```

```
; Assembly language:

.data
array BYTE 1000 DUP(?)
.code
mov  esi,OFFSET array
```

Examples

- Let's assume that the data segment begins at 00404000h:

```
.data
bVal BYTE ?
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?
.code
mov esi,OFFSET bVal
                                ; ESI = 00404000
mov esi,OFFSET wVal
                                ; ESI = 00404001
mov esi,OFFSET dVal
                                ; ESI = 00404003
mov esi,OFFSET dVal2
                                ; ESI = 00404007
```

TYPE Operator

- The TYPE operator returns the size (in bytes) of a single element of a data declaration.

```
.data
var1 BYTE ?
var2 WORD ?
var3 DWORD ?
var4 QWORD ?

.code
mov eax,TYPE var1          ; 1
mov eax,TYPE var2          ; 2
mov eax,TYPE var3          ; 4
mov eax,TYPE var4          ; 8
```

LENGTHOF Operator

- The LENGTHOF operator counts the number of elements in a single data declaration.

```
.data                                          LENGTHOF
byte1  BYTE 10,20,30                             ; 3
array1 WORD 30 DUP(?),0,0                        ; 32
array2 WORD 5 DUP(3 DUP(?))                      ; 15
array3 DWORD 1,2,3,4                             ; 4
digitStr BYTE "12345678",0                       ; 9

.code
mov ecx,LENGTHOF array1                          ; 32
```

SIZEOF Operator

- The SIZEOF operator returns a value that is equivalent to multiplying LENGTHOF by TYPE.

```
.data                                          SIZEOF
byte1  BYTE 10,20,30                             ; 3
array1 WORD 30 DUP(?),0,0                        ; 64
array2 WORD 5 DUP(3 DUP(?))                      ; 30
array3 DWORD 1,2,3,4                             ; 16
digitStr BYTE "12345678",0                       ; 9

.code
mov ecx, SIZEOF array1                           ; 64
```

Spanning Multiple Lines

- A data declaration can span multiple lines if each line (except the last) ends with a comma.
- The LENGTHOF and SIZEOF operators include all lines belonging to the declaration:

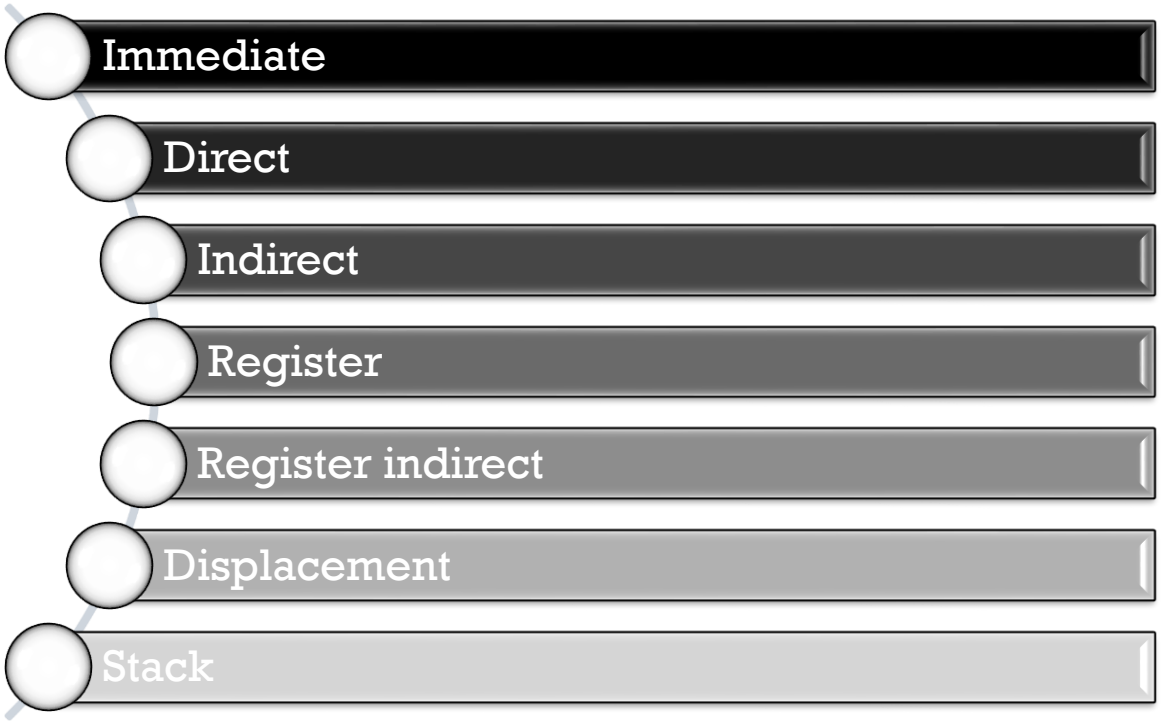
```
.data
array WORD 10,20,
           30,40,
           50,60

.code
mov eax,LENGTHOF array          ; 6
mov ebx,SIZEOF array            ; 12
```

ADDRESSING MODES

Addressing Modes

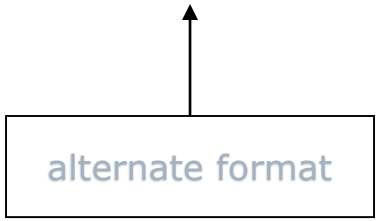
- The address field or fields in a typical instruction format are relatively small → various modes of addressing



Direct Memory Operands

- A direct memory operand is a named reference (variable) to storage in memory
- The variable is automatically dereferenced by the assembler
  - After dereferencing, its value can be obtained

```
.data
var1 BYTE 010h
.code
mov al, var1                ; After moving, AL = 010h
mov al,[var1]               ; After moving, AL = 010h
```



Direct-Offset Operands

- A constant offset is added to a data label to produce an effective address (EA).
  - The offset are 0, 1, 2, …..
- The address is dereferenced to get the value inside its memory location.

```
.data
arrayB BYTE 010h,020h,030h,040h
.code
mov al,arrayB+1                ; AL = 020h
mov al,[arrayB+1]              ; alternative notation
```

Q : Why doesn't arrayB+1 produce 11h?

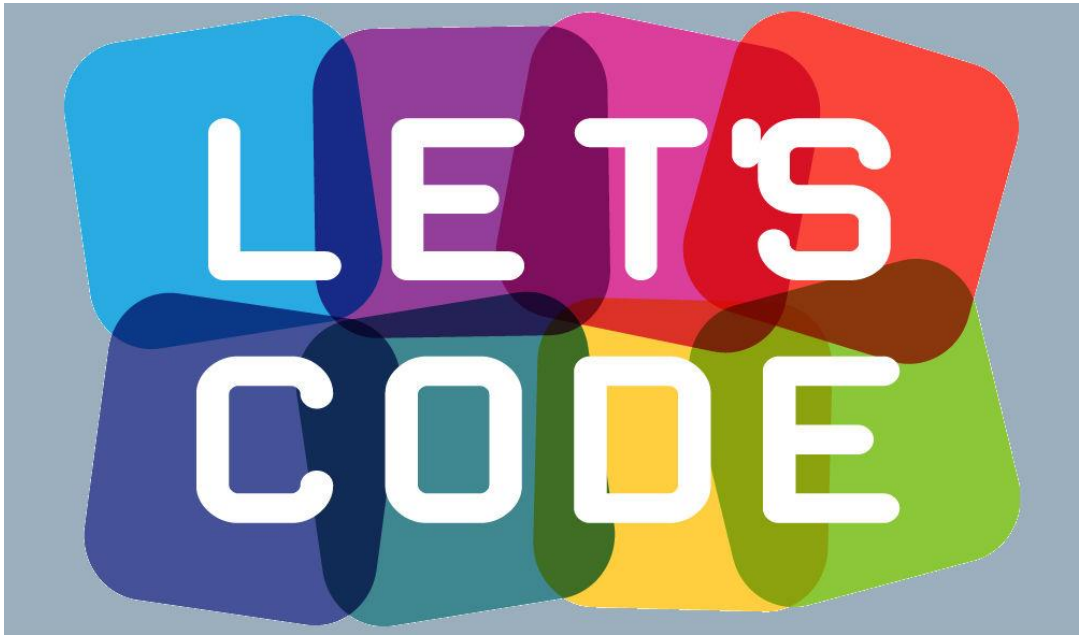
Your turn . . .

```
.data
arrayW  WORD 01000h,02000h,03000h
arrayD  DWORD 1,2,3,4
.code
mov ax, arrayW                  ;
mov ax,[arrayW+2]               ;
mov ax,[arrayW+4]               ;
mov eax,[arrayD+4]              ; EAX = 00000002h
```

What will happen when they run?

Write a program that sums the elements of a WORD array that is initialized with 080h,066h,0A5h

Use base addressing



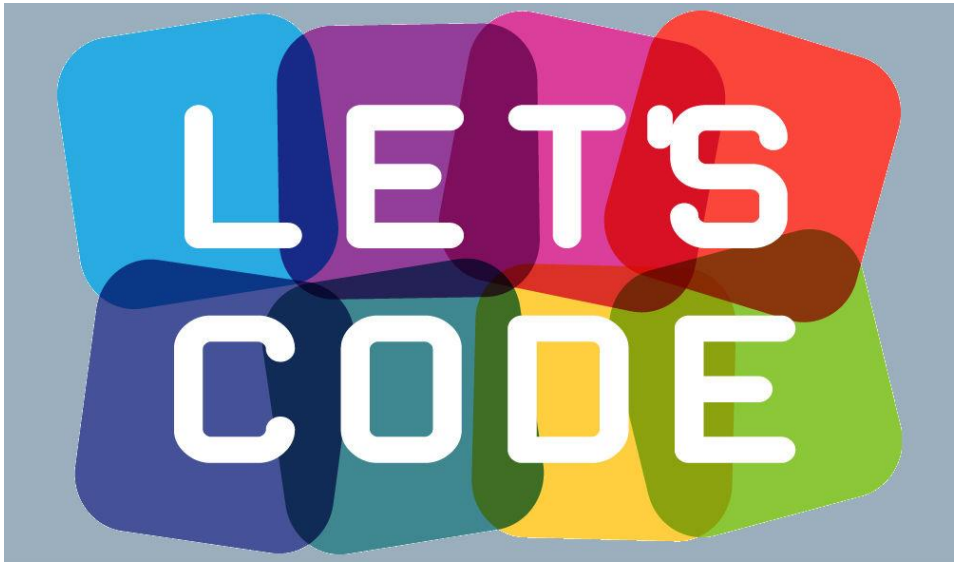
Solution(s)

```
.data
myBytes BYTE 080h,066h,0A5h
```

```
mov al, myBytes         ;al=080h
add al,[myBytes+1]      ;al=0E6h
add al,[myBytes+2]      ;al=018bh
```

Any other possibilities?

Write a program that rearranges the values of three double-word values in an array initialized with 1,2,3 as: 3, 1, 2.



Solution

- **Step1:** copy the 1st element into EAX and exchange it with the element in the 2nd position.
- **Step 2:** Exchange EAX with the 3rd element and copy the element in EAX to the first array position.

```
.data
   arrayD DWORD 1,2,3
.code
       mov eax,arrayD
       xchg eax,[arrayD+4]
       xchg eax,[arrayD+8]
       mov  arrayD,eax
```

Indirect Operands

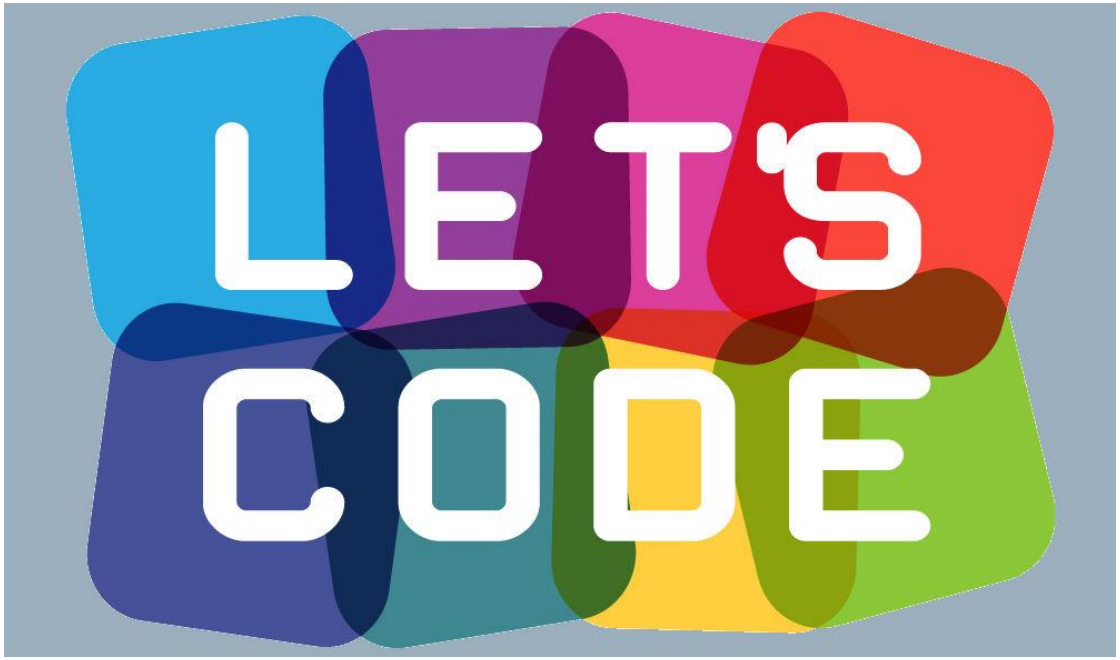
- An indirect operand holds the address of a variable, usually an array or string.
- It can be dereferenced by the assembler (just like a pointer).

```
.data
val1 BYTE 010h,020h,030h
.code
mov esi,OFFSET val1
mov al,[esi]                    ; dereference ESI (AL = 10h)

inc esi
mov al,[esi]
                                ; AL = 020h
inc esi
mov al,[esi]
                                ; AL = 030h
```

Write a program that sums the elements of a WORD array that is initialized with 01000h,02000h,03000h

Use indirect addressing



Solution

```
.data
   arrayW WORD 01000h,02000h,03000h
.code
   mov esi,OFFSET arrayW
   mov ax,[esi]
   add esi,2                  ; or: add esi,TYPE arrayW
   add ax,[esi]
   add esi,2
   add ax,[esi]               ; AX = sum of the array
```

The register in brackets must be incremented by a value that matches the array type

Indexed Operands

- An indexed operand adds a constant to a register to generate an effective address.
- There are two notational forms:

```
[label + reg]
label[reg]
```

- example

```
.data
arrayW WORD 01000h,02000h,03000h
.code
    mov esi,0
    mov ax,[arrayW + esi]           ; AX = 1000h
    mov ax, arrayW[esi]             ; alternate format
    add esi,2
    add ax,[arrayW + esi]
    etc.
```