
# Topic 4
# Lecture 4b
# Data Transfer

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

# What's Next (2 of 5)

- Data Transfer Instructions
- Addition and Subtraction
- **Data-Related Operators and Directives**
- Indirect Addressing
- JMP and LOOP Instructions

# Data-Related Operators and Directives

- Address of a label

# What's in a Label

- The label, when not surrounded by square brackets, returns the address of a data or code element.

# Label Examples

Let's assume that the data segment begins at 0x00404000:

```
section .data
bVal:   db 0xFF
wVal:  dw 0xFFFF
dVal:   dd 0xFFFFFFFF
dVal2: dd 0x1

section .text
mov ebx, bVal                   ; ebx = 0x00404000
mov ebx, wVal                   ; ebx = 0x00404001
mov ebx, dVal                   ; ebx = 0x00404003
mov ebx, dVal2                  ; ebx = 0x00404007
```

# Relating to C/C++

The value returned by the label is a pointer. Compare the following code written for both C++ and assembly language:

```
// C++ version:

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

```
; Assembly language:

section .data
    array: times 1000 db 0
section .text
    mov   esi, array
```

# Little Endian Order

- Little endian order refers to the way Intel stores integers in memory.
- Multi-byte integers are stored in reverse order, with the least significant byte stored at the lowest address
- For example, the dword 0x12345678 would be stored as:

| byte | offset |
|---|---|
| 78 | 0000 |
| 56 | 0001 |
| 34 | 0002 |
| 12 | 0003 |

When integers are loaded from memory into registers, the bytes are automatically re-reversed into their correct positions.

# What's Next (3 of 5)

- Data Transfer Instructions
- Addition and Subtraction
- Data-Related Operators and Directives
- **Indirect Addressing**
- JMP and LOOP Instructions

# Indirect Addressing

- Indirect Operands
- Array Sum Example
- Indexed Operands
- Pointers

# Indirect Operands (1 of 2)

An operand can hold the address of a label, usually an array or string. It can be dereferenced (just like a pointer).

```
.data
val1: db 10h,20h,30h
.text
mov ebx, val1
mov al, [ebx]                    ; dereference ebx (AL = 10h)

inc  ebx
mov al, [ebx]                    ; AL = 20h

inc   ebx
mov al, [ebx]                    ; AL = 30h
```

# Array Sum Example

Indirect operands are ideal for traversing an array. Note that the register in brackets must be incremented by a value that matches the array type.

```
section .data
arrayW: dw 1000h,2000h,3000h

section .text
        mov esi, 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
```

ToDo: Modify this example for an array of doublewords.

# Pointers

You can declare a pointer variable that contains the offset of another variable.

```
section .data
        arrayW: dw 1000h,2000h,3000h
        ptrW:     dd arrayW
section .text
        mov esi,  [ptrW]
        mov ax,   [esi]                         ; AX = 1000h
```

# What's Next (4 of 5)

- Data Transfer Instructions
- Addition and Subtraction
- Data-Related Operators and Directives
- Indirect Addressing
- **JMP and LOOP Instructions**

# JMP and LOOP Instructions

- JMP Instruction
- LOOP Instruction
- LOOP Example
- Summing an Integer Array
- Copying a String

# JMP Instruction

- JMP is an unconditional jump to a label that is usually within the same procedure.
- Syntax: JMP *target*
- Logic: EIP ← *target*
- Example:

```
top:
        .
        .
        jmp top
```

# LOOP Instruction

- The LOOP instruction creates a counting loop
- Syntax: *LOOP target*
- Logic:
  - ECX ← ECX – 1
  - if ECX != 0, jump to target
- Implementation:
  - The assembler calculates the distance, in bytes, between the offset of the following instruction and the offset of the target label. It is called the relative offset.
  - The relative offset is added to EIP.

# LOOP Example

The following loop calculates the sum of the integers 5 + 4 + 3 +2 + 1:

```
Offset    machine code                 source code
00000000  66 B8 0000                   mov ax,0
00000004  B9 00000005                  mov ecx,5

00000009  66 03 C1              .loop: add ax,cx
0000000C  E2 FB                        loop .loop
0000000E
```

When LOOP is assembled, the current location = 0000000E (offset of the next instruction). –5 (FBh) is added to the the current location, causing a jump to location 00000009:

00000009 ← 0000000E + FB

# Your turn . . . (9 of 12)

If the relative offset is encoded in a single signed byte,

a) what is the largest possible backward jump?

b) what is the largest possible forward jump?

(a) –128

(b) +127

# Your turn . . . (10 of 12)

What will be the final value of AX?

10

```
        mov ax,6
        mov ecx,4
.loop:
        inc ax
        loop .loop
```

How many times will the loop execute?

4,294,967,296

```
        mov ecx,0
.loop:
        inc  ax
        loop .loop
```

# Nested Loop

If you need to code a loop within a loop, you must save the outer loop counter's ECX value. In the following example, the outer loop executes 100 times, and the inner loop 20 times.

```
section .data
        count: dd 0
section .text
        mov ecx,100                ; set outer loop count
.loop1:
        mov [count], ecx           ; save outer loop count
        mov ecx, 20                ; set inner loop count
.loop2:
        loop .loop2                ; repeat the inner loop
        mov ecx, [count]           ; restore outer loop count
        loop .loop1                ; repeat the outer loop
```

# Summing an Integer Array

The following code calculates the sum of an array of 16-bit integers.

```
section .data
        intArray:       dw 100h,200h,300h,400h
        intArrayLen:    equ $-intArray
section .text
        mov edi, intArray                   ; address of intarray
        mov ecx, intArrayLen                ; loop counter
        mov eax, 0                          ; zero the accumulator
.loop:
        add ax, [edi]                       ; add an integer
        add edi, 2                          ; point to next integer
        loop .loop                          ; repeat until ECX = 0
```

# Copying a String

The following code copies a string from source to target:

```
section .data
        source:       db   "This is the source string",0
        sourceLen:    equ $ - source
        target:       TIMES sourceLen db 0

section .text
        mov ecx, sourceLen                  ; loop counter
        mov esi, source                     ; esi = src address
        mov edi, target                     ; edi = dst address
.loop:
        mov al, [esi]                       ; get char from source
        mov [edi], al                       ; store it in the target
        inc  esi                            ; inc src address
        inc edi                             ; inc dst address
        loop .loop                          ; repeat for entire string
```

# Your turn . . . (12 of 12)

Rewrite the program shown in the previous slide, using indirect addressing rather than indexed addressing.

# What's Next (5 of 5)

- Data Transfer Instructions
- Addition and Subtraction
- Data-Related Operators and Directives
- Indirect Addressing
- JMP and LOOP Instructions

# 64bit Programming

- MOV instruction in 64-bit mode accepts operands of 8, 16, 32, or 64 bits
- When you move a 8, 16, or 32-bit constant to a 64-bit register, the upper bits of the destination are cleared.
- When you move a memory operand into a 64-bit register, the results vary:
  - 32-bit move clears high bits in destination
  - 8-bit or 16-bit move does not affect high bits in destination

# More

- MOVSXD sign extends a 32-bit value into a 64-bit destination register
- LOOP uses the 64-bit RCX register as a counter
- RSI and RDI are the most common 64-bit index registers for accessing arrays.

# Other 64-Bit Notes

- ADD and SUB affect the flags in the same way as in 32-bit mode
- You can use scale factors with indexed operands.

# Summary

- Data Transfer
  - MOV – data transfer from source to destination
  - MOVSX, MOVZX, XCHG
- Operand types
  - direct, direct-offset, indirect, indexed
- Arithmetic
  - INC, DEC, ADD, SUB, NEG
  - Sign, Carry, Zero, Overflow flags
- Operators
  - OFFSET, PTR, TYPE, LENGTHOF, SIZEOF, TYPEDEF
- JMP and LOOP – branching instructions

43 6F 66 66 65 65 21