CS 271
Computer Architecture & Assembly Language

Lecture 5

Repetition, Constants, and Data Validation

1/18/22, Tuesday

Oregon State University

Odds and Ends

- Due Sunday 1/23 midnight
  - Week 3 Summary
  - Program #2
  - Quiz 1

Recap: Conditional Structures

x, y, z, yes, no

- Ex. Convert the following to MASM assembly (assuming all variables have been defined):

```
if ((x < y) and (y < z))
    print yes
else
    print no
```

```
        mov   eax, x
        cmp   eax, y
        jge   false
        mov   ebx, y
        cmp   ebx, z
        jge   false
        mov   edx, offset yes
        call  writestring
        jmp   done
false:
        mov   edx, offset no
        call  writestring
done:
```

Recap: Conditional Structures

- Ex. Convert the following MASM assembly to high-level pseudocode (assuming all variables have been defined):

```
    mov     eax, a
    cmp     eax, b
    jl      true
    mov     edx, OFFSET no
    call    WriteString
    jmp     done
true:
    mov     edx, OFFSET yes
    call    WriteString
done:
```

if (a<b)
    print yes
else
    print no

if (a >= b)
    print no
else
    print yes

Lecture Topics:

- Repetition structures
- More about Constants
- Data Validation

Repetition Structures

Repetition Structures (iteration)

- Loops are really if (decision) statements
  - Repeat (jump backwards) if a condition is true
  - Otherwise, continue

Pre-test loop (while)

- Initialize loop control variable(s)
- top:
  - Check *condition* using CMP
  - If *condition* is false, jump to *endWhile*
    - Code for LOOP BODY (including loop control update)
    - Jump to top (unconditional jump)
- endWhile:

Example pre-test loop: Double x while x <= 1000

```
; initialize accumulator
            mov     eax, x
dblLoop:    ; Double x while x <= 1000
            cmp     eax, 1000
            jg      endLoop
            add     eax, eax
            jmp     dblLoop
endLoop:
            mov     x, eax
; ...
```

- Warning: Note what happens if x <= 0.
- More later about pre-conditions

Post-test loop (do-while) ← at least once

- top:
  - Code for LOOP BODY (including loop control update)
- Check *condition* using CMP ← after loop body
- If *condition* is true, jump to *top*

Example post-test loop: Double x until x > 1000

```
; initialize accumulator
            mov     eax, x
dblLoop:    ; Double x while x <= 1000
            add     eax, eax
            cmp     eax, 1000
            jle     dblLoop

            mov     x, eax
; ...
```

- Warning: Note what happens if x <= 0.

Counted loop (for)

- Initialize **ecx** to loop count
- top:
  - Code for LOOP BODY
  - loop statement decrements *ecx* and
    - Jump to *top* if *ecx* is not equal to 0
    - Continues to next statement if *ecx* = 0
- Warning: Note what happens if *ecx* is changed inside the loop body
- Warning: Note what happens if *ecx* starts at 0, or *ecx* becomes negative
- Exercise great care when constructing nested "loop" loops (nested for loops)
  - There is only one *ecx* register!!

Example counted loop (version 1) : Find sum of integers from 1 to 10

```
; initialize accumulator, first number, and loop control
        mov    eax, 0
        mov    ebx, 1
        mov    ecx, 10
sumLoop:       ; add integers from 1 to 10
        add    eax, ebx
        inc    ebx            ; add 1 to ebx
        loop   sumLoop        ; subtract 1 from ecx
                              ; if ecx ≠ 0, go to sumLoop
; Print result
        call   WriteDec       ; displays 55
; ...
```

Example counted loop (version 2) : Find sum of integers from 1 to 10

10 + 9 + - - - - + 1

```
; initialize accumulator, first number, and loop control
        mov     eax, 0
        mov     ecx, 10
sumLoop:        ; add integers from 10 to 1
        add     eax, ecx
        loop    sumLoop         ; subtract 1 from ecx
                                ; if ecx ≠ 0, go to sumLoop
; Print result
        call    WriteDec        ; displays 55
; ...
```

Various Solutions

- Any control structure may be implemented in a variety of ways.

- Learn the MASM instructions!
  - Make up a problem
  - Write code to solve it

- Experiment! Experiment!! Experiment!!!

Demo

5 10 5+6+7+...+10

- Problem Statement: gets two integers from the user, and calculates the summation of the integers from the first to the second.

- For example, if the user enters 1 and 10, the program calculates 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.

- Note: This program does not perform any data validation. If the user gives invalid input, the output will be meaningless.

Defining Constants

Symbolic Constants

- May appear in or before the .data segment
  - Usually before
- Two methods:
  - Equal-Sign (=) Directive
  - EQU Directive

Equal-Sign Directive

- *name* = expression
  - *name* is called a symbolic constant
  - *expression* is a 32-bit integer (expression or constant)
    - More later on this
  - Cannot be redefined in the same program
- Style note:
  - Use all CAPS for constant names

```
COUNT = 500
. . .
mov ecx,COUNT
```

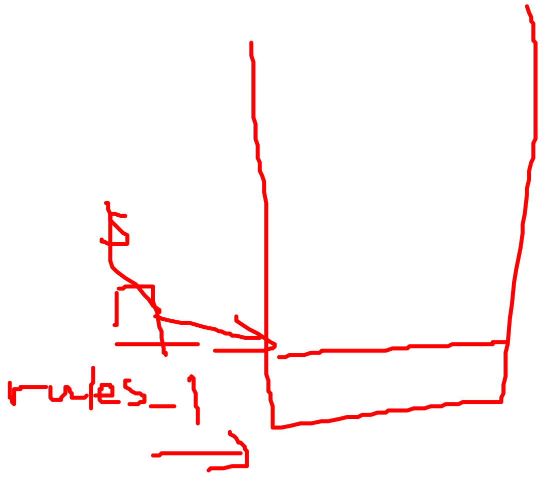
EQU Directive

- Define a symbol as numeric or text expression. (Note <…>)
- Cannot be redefined in the same program

```
PI EQU <3.1416>
PRESS_KEY EQU <"Press any key to continue...",0>

.data
prompt BYTE PRESS_KEY
```

Calculating the size of a string



- Current location in data segment is $
- Subtract address of string
  - Difference is the number of bytes

```
.data
rules_1 BYTE    "Enter the lower limit: ",0
SIZE_1 = ($ - rules_1)
            ;constant length of rules_1 (24)
```

Constants

- Constants are treated like labels (Labels **<u>are</u>** constants!!)
  - Literal value is substituted by assembler

- Q: Why is it a good idea to use constants instead of literals in your program code?

Boolean Constants ?

- MASM does not have a Boolean data type
  - OK to use literal integer values:
    - 0 for FALSE, 1 or -1 for TRUE
  - Traditionally, any value not equal to 0 means TRUE

Data Validation

Data Validation

- In most cases, programs require specific types of data within a specific range of values.
- Check input to verify that input data satisfies the specifications and preconditions.
- It is probably not possible to imagine every kind of input error.
- “Robust” programs …
  - Try to verify that user’s input can be handled by the program
  - Try to keep the program from crashing on invalid input
  - Try to inform the user if there is an input data error
  - Try to permit the user to correct input data errors

Data Validation

- Simple range checking
- One form of interactive data validation:
  - Repeat user-input until a valid value arrives
- Pseudo-code example:

```
repeat
    valid = true
    get value
    if value is not in range
        valid = false
        give error message
until valid
```