CS 271
Computer Architecture & Assembly Language

Lecture 4
First MASM Program and Conditionals
1/13/22, Thursday
2019
Stay away from negative people.
Due Reminder

• Program #1
  • Due Sunday 11:59 pm on Canvas

• Weekly Summary Exercise 2
  • Due Sunday 11:59 pm on Canvas
Lecture Topics:

• Finish our first MASM Program
• Introduction to conditions and control structures
Example Problem Definition

Write a MASM program to perform the following tasks:
1. Introduce yourself to the user.
2. Get the user’s name and number of yards.
3. Greet the user, and report the yards in inches.
4. Say good-bye to the user.

Requirements:
1. The user’s name and yards must be entered by the user, and must be stored and accessed as data segment variables.
2. The “yard-to-inch factor” (36) must be defined as a constant.
Program Design

• Decide what the program should do
• Define algorithm(s)
• Decide what the output should show
• Determine what variables/constants are required
Implementing a MASM program

- Open project
- Start with template, “save as” <.asm file in the program directory>
  - This is the source code file
- Fill in identification block information
- Create comment outline for algorithms
- Define constants
  - Test/fix (syntax check, nothing happens)
- Declare variables (.data section)
  - Test/fix (syntax check, nothing happens)
- Enter the output code
  - Test/fix (no calculations, usually everything show 0)
- Enter the input code
  - Test/fix (no calculations, echo input)
- Enter the calculation code
  - Test/fix (logic check, verify)

*First try Debug, Start Without Debugging (more later on using the debug system)
Writing a MASM program

• Demo
Introduction to conditions and control structures
Branching Execution

• Sometimes it is necessary to interrupt sequential instruction execution
  • EIP is changed
    • But should not be changed directly
• Examples:
  • Skip ahead (e.g., skip the else block)
  • Jump backwards (e.g., repeat a section of code)
  • Call a procedure
• Conditional / Unconditional branching
• Label required
MASM Labels

• Same rules as other identifiers
• May not be any previously defined identifier
• Label definition ends with colon :
  • Don’t use colon when referencing the label
• Specifies the memory address of the associated instruction
  • ... just like a variable name
• Good practices:
  • Put labels on separate lines
  • Use meaningful label names
    • E.g., don’t use a label named label
Unconditional branching

• Instruction format is \texttt{jmp label}
  • Meaning is “Set EIP to label and continue execution”
    • Remember: label is a name that has been set equivalent to a memory address. I.E., label is a constant
  • \texttt{label}: should be inside the same procedure
    • MASM allows jumps to labels in other procedures, but execution will almost certainly get lost in space

• Examples later
Decision structures (alternation)

• We need a way to control branching by checking conditions
  • E.g., if a condition is true, do some task. Otherwise, do something else

• MASM provides a way to compare two operands. The result of the comparison is saved in the status register.
Conditional branching

- Used for:
  - if structures (decisions, alternation)
  - loop structure (repetition, iteration)

- In general, MASM requires you to build your own control structures

- Note: MASM provides some “advanced” conditional directives (.repeat, .if, .else, ... etc.) which we will NOT use in this course.
  - These directives don’t help you to understand how programs work.
CMP Instruction

- Compares the destination operand to the source operand
  - Non-destructive subtraction: source – destination (*destination is not changed*)
  - Set specific bits in the status register
  - Status bits indicate how source compares to destination
    - <, >, =, <=, >=, etc.
    - Other information in status register:
      - Overflow, zero, error, etc.
    - Program can conditionally jump to a label, based on status bits.

- Syntax: **CMP** destination, source
The Status (Flag) Register

Each bit is 0 or 1 to indicate “off” or “on”, “false” or “true”, etc.

<table>
<thead>
<tr>
<th>Bit abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>Overflow</td>
</tr>
<tr>
<td>D</td>
<td>Direction</td>
</tr>
<tr>
<td>I</td>
<td>Interrupt</td>
</tr>
<tr>
<td>T</td>
<td>Trap</td>
</tr>
<tr>
<td>S</td>
<td>Sign</td>
</tr>
<tr>
<td>Z</td>
<td>Zero</td>
</tr>
<tr>
<td>A</td>
<td>Auxiliary carry</td>
</tr>
<tr>
<td>P</td>
<td>Parity</td>
</tr>
<tr>
<td>C</td>
<td>Carry</td>
</tr>
</tbody>
</table>

• Notes:
  • This is a partial list
  • We usually do not access these bits directly
**Jcond Instruction**

- A *conditional jump* instruction checks the status register and branches (or not) to label depending on status of specific flags.
  - ... usually the next instruction after *cmp*

- Syntax: `Jcond label`
  - There are *many cond* forms that can be checked
  - *label* is defined by the programmer

- Example:
  ```
  cmp     eax, 100
  jle     notGreater ; if eax <=100, go to notGreater
  ```

  Meaning: if the value in *EAX* is less than or equal to 100, jump to the label *notGreater*. 
Common Jcond instructions

• JE       jump if destination = source
• JL       jump if destination < source
• JG       jump if destination > source
• JLE      jump if destination <= source
• JGE      jump if destination >= source
• JNE      jump if destination not = source

• NOTE: These conditions are for **signed** integers
  • OK to compare negative to non-negative, etc.
  • More later on this
Block-structured **IF** statements

- You can create assembly language control structures that are equivalent to statements written in C++/Java/etc...
- Example:

```assembly
if( op1 == op2 )
  x = 1;
else
  x = 2;
```

```assembly
mov eax, op1
cmp eax, op2
jne L1
jmp L2
L1:
  mov x, 1
L2:
```

Assembly Language Control Structures

• Extend the concept to create your own:
  • If-then
  • If-then-else
  • If-then-elseif-else
  • Compound conditions
  • While loop
  • Do-while loop
  • For loop
  • Nested structures, switch structures, etc.
If-then

- Check condition using \texttt{CMP}
- If condition is \texttt{false}, jump to endThen
  - code for TRUE block
- endThen

\[
\text{if } (a == b) \\
\text{ \quad } b = 3;
\]

```
l1: // ---

    mov eax, a
    cmp eax, b
    jne l1

    mov b, 3
```

```
l1:
    // ---
```
If-then-else (Method 1)

- Check condition using CMP
- If condition is false, jump to falseBlock
  - Code for TRUE block
  - Jump to endFalse
- falseBlock:
  - Code for FALSE block
- endFalse:
Convert pseudo-code to MASM

```assembly
if( op1 == op2 )
    x = 1;
else
    x = 2;
```

```assembly
mov eax,op1
cmp eax,op2 ; test condition
jne fBlock ; if op1 ≠ op2, jump to false block
mov x,1 ; true block
jmp done ; skip over false block

fBlock:
    mov x,2 ; false block
done: ; end of decision structure
```
If-then-else (Method 2)

- Check condition using CMP
- If condition is true, jump to trueBlock
  - Code for FALSE block
  - Jump to endTrue
- trueBlock:
  - Code for TRUE block
- endTrue:

```assembly
mov eax, op1
cmp eax, op2
je trueB
mov x, 2
jmp done

trueB:
mov x, 1

Done:
```

```assembly
if (op1 ! = op2)
x = 2;
else
x = 1;
```
If-then-elseif-else

- Check condition1 using **CMP**
- If condition1 is **true**, jump to trueBlock1
- Check condition2 using **CMP**
- If condition2 is **true**, jump to trueBlock2
  - Code for FALSE block
  - Jump to endBlock
- trueBlock1:
  - Code for TRUE block1
  - Jump to endBlock
- trueBlock2:
  - Code for TRUE block2
- endBlock:

```
if (op1 == op2)
    x = 1;
elseif (op1 == op3)
    x = 2;
else
    x = 3;
```

```
true1:    mov x, 1
          jmp endB
true2:    mov x, 2
          jmp endB
true1:    mov x, 1
          jmp endB
```
Compound conditions (AND)

- Check condition1 using **CMP**
- If condition1 is **false**, jump to falseBlock
- Check condition2 using **CMP**
- If condition2 is **false**, jump to falseBlock
  - Code for **TRUE** block
  - Jump to endBlock
- **falseBlock**:
  - Code for **FALSE** block
- **endBlock**:

  - Note: this structure implements **short-circuit evaluation**
Compound conditions (OR)

- Check condition1 using CMP
- If condition1 is true, jump to trueBlock
- Check condition2 using CMP
- If condition2 is true, jump to trueBlock
  - Code for FALSE block
  - Jump to endBlock
- trueBlock:
  - Code for TRUE block
- endBlock:
  - Note: this structure implements **short-circuit evaluation**
... and so on, and on ...

• Of course there is no end to the variety of decision structures in software systems

• Things can get complicated. As you construct your decision structures in MASM, be sure to
  • Jump to the correct block based on the result of the comparison
  • Jump over the other blocks when you are finished with the selected block