CMPT 295
Machine-Level Programming
Lecture 13 – Program Control – Conditional Statement - 2
Last Lecture

- In C, we can change the execution flow of a program
  1. Conditionaly
     - Conditional statements: if/else, switch
     - Iterative statements: loops
  2. Unconditionally
     - Functions calls
- In x86-64 assembly, we can also change the execution flow of a program
  - `cmp*` instruction (compare)
  - `jX` instructions (jump)
  - `call` and `ret` instructions
- How conditional statement `if/else` can be implemented in x86-64 assembly
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
  - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement – Condition Code + cmovX
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations
Example – alternative: `int abs(int x)`

in C:

```c
int abs(int x){
    if ( x < 0 )
        x = -x;
    return x;
}
```

in assembly:

```
# x in edi, result in eax
abs:
    movl %edi, %eax  # result = x

ret
```
Conditional Move \texttt{cmovX}

- Conditional Move Instructions
  - Example: \texttt{cmovle Src, Dest}  \# if (Test) Dest \leftarrow Src
  - \texttt{gcc} tries to use them, but only when safe

- Why?
  - Branches are very disruptive to instruction flow through pipelines
  - Since conditional moves do not require control transfer, less disruptive

\textbf{C Code}

```c
result = test ? val1 : val2;
```

\textbf{How \texttt{cmovX} behaves (in C)}:

```c
result = val1;
temp = val2;
if (!test) result = temp;
return result;
```
What do we mean by “safe”?

- Since both values are always computed ...

- Example of unsafe situations:
  1. Expensive computations
     - Only makes sense when computations are very simple
     
     ```
     val = Test(x) ? Hard1(x) : Hard2(x);
     ```
  2. Risky computations
     - Only makes sense when computations do not crash the application
     
     ```
     val = p ? *p : 0;
     ```
  3. Computations with side effects
     - Only makes sense when computations do not have side-effects
     
     ```
     val = x > 0 ? x*=7 : x+=3;
     ```
Condition Codes

- A part of what is visible to assembly programmers
- Single bit registers
  - ZF  Zero Flag
  - CF  Carry Flag (for unsigned)
  - SF  Sign Flag (for signed)
  - OF  Overflow Flag (for signed)
- Set by various instructions like \texttt{cmp} * \rightarrow \texttt{cmpq} b,a \# a - b
  - ZF set if \( a - b = 0 \) (\( a == b \))
  - CF set if carry out from most significant bit (used for unsigned comparisons)
  - SF set if \( a - b < 0 \) (for signed)
  - OF set if two’s-complement (signed) overflow
    \( (a > 0 \&\& b < 0 \&\& a - b < 0) \ | \ | (a < 0 \&\& b > 0 \&\& a - b > 0) \)
Using Condition Codes

- Once the condition codes are set, we can either alter
  - control flow -> branch using \textit{jX} Instructions (jump)
  
  OR

  - data flow -> move data using \textit{cmovX} instructions
    (conditionally transfer data)

- Once the condition codes are set, we can use
  - \textit{set} family of instructions to set 1 byte of a register
setX instruction

- **Syntax**: sete Dest (i.e., setz)
- **Meaning (Effect)**: Dest <- ZF
  - Set low-order byte of Dest to 0 or 1 based on condition (see Condition column)
  - Does not alter remaining 7 bytes

<table>
<thead>
<tr>
<th>SetX</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sete</td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>setne</td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>sets</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>setns</td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>setg</td>
<td>~(SF^OF) &amp;~ZF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>setge</td>
<td>~(SF^OF)</td>
<td>Greater or Equal (Signed)</td>
</tr>
<tr>
<td>setl</td>
<td>(SF^OF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>setle</td>
<td>(SF^OF)</td>
<td>ZF</td>
</tr>
<tr>
<td>seta</td>
<td>~CF&amp;~ZF</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>setb</td>
<td>CF</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>
Example – int max(int x, int y)

in C:
int max(int x, int y)
{
    int result = x;
    if (y > x)
        result = y;
    return result;
}

in assembly:
    # x in edi, y in esi, result in eax
max:
    movl %edi, %eax  # result = x
    ret

    endif:
    ret
Example: alternate \texttt{int max(int x, int y)}

\begin{itemize}
  \item \textbf{In C:}
  \begin{verbatim}
  int max(int x, int y) {
    int result = x;
    if (y > x) {
      result = y;
    }
    return result;
  }
  \end{verbatim}
  \end{itemize}

\begin{itemize}
  \item \textbf{in assembly:}
  \begin{verbatim}
  # x in edi, y in esi, result in eax
  max:
    movl %edi, %eax  # result = x
  ret
  \end{verbatim}
  \end{itemize}
Summary

- *if/else* can be implemented in x86-64 assembly using branching method:
  - `cmp*` instruction
  - `jX` instructions (jump)
  - `cmovX` instructions -> conditional move

- Condition codes: ZF, CF, SF, OF

- Examples – various ways of implementing ...
  - `int abs(int x)`
  - `int max(int x, int y)`
Next Lecture

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
    - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement – Condition Code + cmovX
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations