CMPT 295

Machine-Level Programming

Lecture 9 – Assembly language basics: Data and move operation
Review: von Neumann architecture
- Data and code are both stored in memory during program execution

**Question:** How does our C program end up being represented as a series of 0's and 1's?
- C program -> assembly code -> machine level code
- gcc: C preprocessor, C compiler, assembler, linker
- Snapshot of compiled code
  - Example of a C statement compiled into assembly code which is then assembled into a machine level instruction

**Question:** How does our C program (now represented as a series of 0's and 1's) end up in memory?
- When C program is executed

**Question:** How is our C program (now represented as a series of 0's and 1's stored in memory) executed by the CPU?
- CPU executes C program by looping through the fetch-execute cycle
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, move operation
- Operations -> Arithmetic & logical operations
- Conditional Statement
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations
Programming in C versus in x86-64 assembly language

When programming in C, we can ...
- Store/retrieve data into/from memory (variables)
- Perform calculations on data
  - e.g., arithmetic, logic, shift
- Transfer control: decide what part of the program to execute next based on some condition
  - e.g., if-else, loop, function call

When programming in assembly language, we can do the same things, but ...
Programming in x86-64 assembly

- … with assembly language (and machine level code), parts of the processor (CPU) state are visible to assembly programmers that normally are hidden from C programmers
- As assembly programmers, we now have access to …
x86-64 Assembly Language - Data

- Data (integral) not stored in variables but in registers
  - No distinction between signed and unsigned integers
  - Distinction between different integer size: 1, 2, 4 and 8 bytes
- Addresses not stored in pointer variables but in registers
  - Size: 8 bytes
  - Treated as integral numbers
- Data (floating point numbers) stored in different registers
  - Distinction between different floating point numbers: 4 and 8 bytes
- No aggregate types such as arrays or structures
## x86-64 Assembly Language - Integer Registers

<table>
<thead>
<tr>
<th>64-bit (quad)</th>
<th>32-bit (double)</th>
<th>16-bit (word)</th>
<th>8-bit (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63..0</td>
<td>31..0</td>
<td>15..0</td>
<td>7..0</td>
</tr>
<tr>
<td>rax</td>
<td>eax</td>
<td>ax</td>
<td>al</td>
</tr>
<tr>
<td>rbx</td>
<td>ebx</td>
<td>bx</td>
<td>bl</td>
</tr>
<tr>
<td>rcx</td>
<td>ecx</td>
<td>cx</td>
<td>cl</td>
</tr>
<tr>
<td>rdx</td>
<td>edx</td>
<td>dx</td>
<td>dl</td>
</tr>
<tr>
<td>rsi</td>
<td>esi</td>
<td>si</td>
<td>sil</td>
</tr>
<tr>
<td>rdi</td>
<td>edi</td>
<td>di</td>
<td>dil</td>
</tr>
<tr>
<td>rbp</td>
<td>ebp</td>
<td>bp</td>
<td>bpl</td>
</tr>
<tr>
<td>rsp</td>
<td>esp</td>
<td>sp</td>
<td>spl</td>
</tr>
<tr>
<td>r8</td>
<td>r8d</td>
<td>r8w</td>
<td>r8b</td>
</tr>
<tr>
<td>r9</td>
<td>r9d</td>
<td>r9w</td>
<td>r9b</td>
</tr>
<tr>
<td>r10</td>
<td>r10d</td>
<td>r10w</td>
<td>r10b</td>
</tr>
<tr>
<td>r11</td>
<td>r11d</td>
<td>r11w</td>
<td>r11b</td>
</tr>
<tr>
<td>r12</td>
<td>r12d</td>
<td>r12w</td>
<td>r12b</td>
</tr>
<tr>
<td>r13</td>
<td>r13d</td>
<td>r13w</td>
<td>r13b</td>
</tr>
<tr>
<td>r14</td>
<td>r14d</td>
<td>r14w</td>
<td>r14b</td>
</tr>
<tr>
<td>r15</td>
<td>r15d</td>
<td>r15w</td>
<td>r15b</td>
</tr>
</tbody>
</table>

- Storage locations in CPU -> fastest storage
- 16 registers are used explicitly – must name them in assembly code
- Some registers are used implicitly
  - e.g., PC, FLAGS
- Each register is 64 bits in size, but we can refer to its:
  - first byte LSB (8 bits),
  - first 2 bytes (16 bits),
  - first 4 bytes (32 bits),
  - or to all of its 8 bytes (64 bits)
x86-64 Assembly Language - Operations

- Transfer data between memory and registers
  - **Load** data from memory into register
  - **Store** register data into memory
  - **Move** data from one register to another

- Perform calculations on register data
  - e.g., arithmetic, logic, shift

- Transfer control
  - Unconditional jumps to/from functions
  - Conditional branches
Move data – \texttt{mov*}

- Transfer data between memory and registers
- Syntax: \texttt{mov* Source, Destination}
- Example: \texttt{movq %rdi, %rax}

- Allowed moves:
  - From register to register (\texttt{Move})
  - From memory to register (\texttt{Load})
  - From register to memory (\texttt{Store})

- Conditional move (\texttt{cmov*})
  - Same as above, but based on result of comparison

\* \rightarrow \text{Size designator}

- q \rightarrow \text{long 64}
- l \rightarrow \text{int 32}
- w \rightarrow \text{short 16}
- B \rightarrow \text{char 8}
void swap(long *xp, long *yp) {
    long L1 = *xp;
    long L2 = *yp;
    *xp = L2;
    *yp = L1;
    return;
}

swap:
    movq (%rdi), %rax  # L1 = *xp
    movq (%rsi), %rdx  # L2 = *yp
    movq %rdx, (%rdi)  # *xp = L2
    movq %rax, (%rsi)  # *yp = L1
    ret

<table>
<thead>
<tr>
<th>Registers</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>0x0020</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x0018</td>
</tr>
<tr>
<td>%rax</td>
<td>0x0010</td>
</tr>
<tr>
<td>%rdx</td>
<td>0x0008</td>
</tr>
<tr>
<td></td>
<td>0x0000</td>
</tr>
</tbody>
</table>
**Operand Combinations for mov\***

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
<th>Src, Dest</th>
<th>in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Register</td>
<td>movq $0x4,%rax</td>
<td>result = 0x4;</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>movq $-147,(%rax)</td>
<td>*result = -147;</td>
</tr>
<tr>
<td>Register</td>
<td>Register</td>
<td>movq %rax,%rdx</td>
<td>var1 = result;</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>movq %rax,(%rdx)</td>
<td>*var1 = result;</td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
<td>movq (%rax),%rdx</td>
<td>var1 = *result;</td>
</tr>
</tbody>
</table>

*Cannot do memory-memory transfer with a single mov\* instruction*
Summary

- As x86-64 assembly programmers, we now get to see more of the processor (CPU) state: PC, registers, condition codes

- x86-64 assembly language – Data
  - Integer registers (16) of 1, 2, 4 or 8 bytes + memory address of 8 bytes
  - Floating point registers of 4 or 8 bytes
  - No aggregate types such as arrays or structures

- x86-64 assembly language – Operations
  - `mov*` instruction family
  - From register to register
  - From memory to register
  - From register to memory
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
- Loop
- Function/procedure call – Stack – Recursion
- Array
- Floating-point operations