Assembly and Bomb Lab

15-213: Introduction to Computer Systems
Recitation 4, Sept. 17, 2012

Instructor: Abraham Levkoy (alevkoy)
Section J, 11:30a – 12:20p WEH5310
Outline

- Assembly
  - Basics
  - Operations
- Bomblab
  - Tools
  - Demo
Registers

• Program counter
  ▪ eip (x86) or rip (x86-64)
  ▪ Contains address of next instruction to execute

• General purpose registers
  ▪ You can put stuff in them
  ▪ eax, ebx, ecx, edx, esi, edi (x86)
  ▪ rax, rbx, rcx, rdx, rsi, rdi, r 8, r 9, r 10, r 11, r 12, r 13, r14, r 15, rbp (x86-64)
Data Types

• Integer data
  ▫ Data values (signed and unsigned)
  ▫ 1, 2, or 4 bytes (and 8 in x86-64)
  ▫ Addresses
    ▫ 4 bytes (x86) or 8 bytes (x86-64)
• Floating point data
  ▫ 4 or 8 bytes
  ▫ Special 10-byte type on Intel CPUs
  ▫ No aggregate data types!
Operands

• Immediate value
  ▪ Examples: $0x15213, $-18213
  ▪ Like a C constant, prefixed with “$”
  ▪ 1, 2, or 4 bytes (or 8 on x86-64) depending on instruction

• Register
  ▪ Examples: %esi, %rax
  ▪ Some instructions (e.g. div) use specific registers implicitly

• Memory locations
  ▪ Examples: (%esi), 12(%eax, %ebx, 4)
  ▪ Format is X(Rb, Ri, S)
  ▪ Rb is the base address register
  ▪ Ri is the index register
  ▪ S is the index scale (1, 2, 4, or 8)
  ▪ X is a constant offset
  ▪ Equivalent to C style Rb[Ri*S + X]
Operands

- `movl src, dst`
  - Example: `movl $0x15213, %eax`
  - Moves data between registers and memory
  - Immediate value to register

- `leal src, dst`
  - Example: `leal (%eax, %eax, 2), %eax`
  - Computes an address specified by `src` and saves it in `dst`
  - Does not actually dereference `src`!
  - Sometimes used by compilers as a fast alternative to `imul`
  - Example above triples `%eax`
# Arithmetic Operations

## Two-operand commands

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addl src, dst</code></td>
<td>dst += src</td>
</tr>
<tr>
<td><code>subl src, dst</code></td>
<td>dst -= src</td>
</tr>
<tr>
<td><code>imull src, dst</code></td>
<td>dst *= src</td>
</tr>
<tr>
<td><code>sall src, dst</code></td>
<td>dst &lt;&lt;= src</td>
</tr>
<tr>
<td><code>sarl src, dst</code></td>
<td>dst &gt;&gt;= src</td>
</tr>
<tr>
<td><code>xorl src, dst</code></td>
<td>dst ^= src</td>
</tr>
<tr>
<td><code>andl src, dst</code></td>
<td>dst &amp;= src</td>
</tr>
<tr>
<td><code>orl src, dst</code></td>
<td>dst</td>
</tr>
</tbody>
</table>

## One-operand commands

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>incl dst</code></td>
<td>dst++</td>
</tr>
<tr>
<td><code>decl dst</code></td>
<td>dst--</td>
</tr>
<tr>
<td><code>negl dst</code></td>
<td>dst = -dst</td>
</tr>
<tr>
<td><code>notl dst</code></td>
<td>dst = ~dst</td>
</tr>
</tbody>
</table>

These instructions have variants for different operand sizes, including 8 bytes for x86-64, e.g. `addb, addw, addq`
# Arithmetic Operations

## One-operand commands

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>incl dst</td>
<td>dst++</td>
</tr>
<tr>
<td>decl dst</td>
<td>dst--</td>
</tr>
<tr>
<td>negl dst</td>
<td>dst = -dst</td>
</tr>
<tr>
<td>notl dst</td>
<td>dst = ~dst</td>
</tr>
</tbody>
</table>
Condition Flags

• Set as side effect of arithmetic operations in the eflags register
  • CF set on unsigned integer overflow
  • ZF set if result was 0
  • SF set if result was negative
  • OF set on signed integer overflow
• testl a, b and cmpl a, b are similar to andl a, b and subl a, b but only set conditions codes
• Use set* reg instructions to set reg based on state of condition codes
Condition Flags

• Change the instruction pointer with j* instructions
  • jmp dst unconditionally jumps to address dst
  • Other jump variants (e.g. jne or jg) conditionally jump
  • Use a test or cmp followed by a conditional jump
• Conditional moves added in x686 standard
  • cmov* src, dst
  • Significantly faster than branch
  • GCC does not use them by default to maintain backwards compatibility
Bomblab Overview

• Series of stages, all asking for a password
• Bomb explodes if you give the wrong password
  • Half point deducted for every explosion
  • The bomb should never explode if you're careful
• You only get the binary, not the source
• Have to find the passwords and input them
• Can only run the binary on the shark machines
GDB – GNU Debugger

- $ gdb ./bomb
- Useful gdb commands
  - `run <args>` – run bomb with specified command line arguments
  - `break <location>` - Stop the bomb just before the instruction at the specified address or location is about to run
  - `stepi` – Run the next instruction (just one); `nexti` will do the same but skip over function calls
  - `print <expression>` - Prints the result of an expression (maybe just a variable)
  - `display <expression>` - Same as print but updates the output every time you stop the program
  - `x/<format> <address>` - Print contents of the memory area starting at the address in the specified format
  - `disassemble [address]` – Displays the assembly instructions near the specified address
  - `layout <type>` - Changes GDB layout; try `layout asm` followed by `layout reg`
GDB – GNU Debugger

• **strings**
  • Dumps all strings in the binary
  • Function names, string literals, etc.

• **objdump**
  • `-d` flag disassembles the bomb and outputs the assembly
  • `-t` outputs all of the function and global variable names
  • Output is long, so pipe it to `less` or redirect it to a file
    • `objdump -d bomb | less` or
    • `objdump -d bomb > bomb_asm`
Bomb Walkthrough

Example bomb walkthrough