Assembly and Bomblab

Steven Fackler (sfackler)

15-213 Fall 2011

9/19/11
Outline

1. Assembly
   - Basics
   - Operations

2. Bomblab
   - Tools
   - Walkthrough

3. Administrativa
Outline

1. **Assembly**
   - Basics
   - Operations

2. **Bomlab**
   - Tools
   - Walkthrough

3. **Administrativa**
Outline

1. Assembly
   - Basics
   - Operations

2. Bomblab
   - Tools
   - Walkthrough

3. Administrativa
x86 Architecture

- Program counter
  - Contains address of next instruction
  - \textit{eip} (x86), \textit{rip} (x86-64)

- Stack registers
  - Contain addresses of base and top of current stack frame
  - Covered tomorrow in lecture
  - \textit{esp} and \textit{ebp} (x86), \textit{rsp} and \textit{rbp} (x86-64)

- General purpose registers
  - \textit{eax}, \textit{ebx}, \textit{ecx}, \textit{edx}, \textit{esi}, \textit{edi} (x86)
  - and sometimes \textit{rbp} (x86-64)

- Condition codes

- Other stuff
  - Control registers, segment selectors, debug registers, SIMD registers, floating point registers, etc
Data Types

- Integer data
  - Data values (signed and unsigned)
    - 1, 2, or 4 bytes (or 8 on x86-64)
  - Addresses
    - 4 bytes (x86) or 8 bytes (x86-64)
- Floating point data
  - 4, 8 or 10 bytes
- No aggregate data types!
Outline

1. Assembly
   - Basics
   - Operations

2. Bomblab
   - Tools
   - Walkthrough

3. Administrativa
Operand Types

- **Immediate value**
  - Examples: $0x15213$, $-18213$
  - Like a C constant, prefixed with ‘$’
  - 1, 2 or 4 bytes (or 8 on x86-64)

- **Register**
  - Examples: `%esi`, `%eax`
  - Some instructions (e.g. `div`) use specific registers

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

- **movl src,dst**
  - Example: `movl $0x15213,%eax`
  - Moves data between registers and memory
  - Immediate value to register or memory
  - Register to other register or memory
  - Memory 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>addl src,dst</td>
<td>dst += src</td>
</tr>
<tr>
<td>subl src,dst</td>
<td>dst -= src</td>
</tr>
<tr>
<td>imull src,dst</td>
<td>dst *= src</td>
</tr>
<tr>
<td>sall src,dst</td>
<td>dst &lt;&lt;= src</td>
</tr>
<tr>
<td>sarl src,dst</td>
<td>dst &gt;&gt;= src</td>
</tr>
<tr>
<td>xorl src,dst</td>
<td>dst ^= src</td>
</tr>
<tr>
<td>andl src,dst</td>
<td>dst &amp;= src</td>
</tr>
<tr>
<td>orl src,dst</td>
<td>dst</td>
</tr>
</tbody>
</table>

### 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>

There are also 64 bit equivalents (e.g. addq).
Condition Codes

- 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 condition codes
- Use `set* reg` instructions to set register `reg` based on state of condition codes.
Change the instruction pointer with the \texttt{j*} operations

- \texttt{jmp dst} unconditionally jumps to the address \texttt{dst}
- Use other jump variants (e.g. \texttt{jne} or \texttt{jg}) to conditionally jump
  - Usually a \texttt{test} or \texttt{cmp} followed by a conditional jump

Conditional moves added in the x86 standard

- \texttt{cmov* src,dst}
- Significantly faster than a branch
- GCC does not use these by default for 32 bit code to maintain backwards compatibility
Outline

1  Assembly
   • Basics
   • Operations

2  Bomblab
   • Tools
   • Walkthrough

3  Administrativa
Overview

- Series of stages, all asking for a password
- Give the wrong password and the bomb explodes
  - You lose a half point every time your bomb explodes
  - The bomb should never explode if you’re careful
- We give you the binary, you have to find the passwords
- The binary *ONLY* runs on the shark machines
Outline

1 Assembly
   - Basics
   - Operations

2 Bomblab
   - Tools
     - Walkthrough

3 Administrativa
GDB - GNU Debugger

- Syntax: `$> gdb ./bomb`
- Useful commands
  - `run args` Runs the bomb with specified command line arguments
  - `break location` Will stop the bomb just before the instruction at the specified location is about to be run
  - `info functions` Will list the names of all functions in the bomb
  - `stepi` Steps the program one instruction. `nexti` will do the same, but skipping over function calls.
  - `print variable` Prints the contents of a variable
  - `x/format address` Prints contents of the memory area starting at the address in a specified format
  - `disassemble address` Displays the assembly instructions near the specified address
  - `layout type` Changes the layout of GDB. `layout asm` followed by `layout reg` is great

Steven Fackler (sfackler) | Assembly and Bomlab
strings
- Dumps all strings in the binary
- Function names, string literals, etc

objdump
- The -d option disassembles the bomb and outputs the assembly to the terminal
- The -t option dumps the symbol table (all function and global variable names) to the terminal
- You probably want to redirect the output into a file
  
  `objdump -d ./bomb > bomb_asm`
Outline

1. Assembly
   - Basics
   - Operations

2. Bomlab
   - Tools
   - Walkthrough

3. Administrativa
Example bomb walkthrough
Outline

1. Assembly
   - Basics
   - Operations

2. Bomblab
   - Tools
   - Walkthrough

3. Administrativa
If this is not the recitation you registered for, please come up so we can fix your autolab account.