Machine-Level Programming I: Basics

CS 485: Systems Programming
Fall 2015

Instructor:
James Griffioen

Adapted from slides by R. Bryant and D. O’Hallaron (http://csapp.cs.cmu.edu/public/instructors.html)
Definitions

- **Architecture**: (also instruction set architecture: ISA) The parts of a processor design that one needs to understand to write assembly code.
  - Examples: instruction set specification, registers.

- **Microarchitecture**: Implementation of the architecture.
  - Examples: cache sizes and core frequency.

- Example ISAs (Intel): x86, IA, IPF
Assembly Programmer’s View

- **Programmer-Visible State**
  - PC: Program counter
    - Address of next instruction
    - Called “EIP” (IA32) or “RIP” (x86-64)
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code, user data, (some) OS data
  - Includes stack used to support procedures
Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc -O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - Put resulting binary in file `p`
Compiling Into Assembly

C Code

```c
int sum(int x, int y) {
  int t = x+y;
  return t;
}
```

Generated IA32 Assembly

```
sum:
  pushl %ebp
  movl %esp,%ebp
  movl 12(%ebp),%eax
  addl 8(%ebp),%eax
  popl %ebp
  ret
```

Some compilers use instruction “leave”

Obtain with command

```
/usr/local/bin/gcc -O1 -S code.c
```

Produces file `code.s`
Object Code

Code for sum

0x401040 <sum>:
  0x55
  0x89
  0xe5
  0x8b
  0x45
  0x0c
  0x03
  0x45
  0x45
  0x08
  0x5d
  0xc3

- Total of 11 bytes
- Each instruction 1, 2, or 3 bytes
- Starts at address 0x401040

Assembler

- Translates .s into .o
- Binary encoding of each instruction
- Nearly-complete image of executable code
- Missing linkages between code in different files

Linker

- Resolves references between files
- Combines with static run-time libraries
  - E.g., code for malloc, printf
- Some libraries are dynamically linked
  - Linking occurs when program begins execution
Machine Instruction Example

**C Code**
- Add two signed integers

**Assembly**
- Add 2 4-byte integers
  - “Long” words in GCC parlance
  - Same instruction whether signed or unsigned
- Operands:
  - \( x \): Register \( \%eax \)
  - \( y \): Memory \( M[\%ebp+8] \)
  - \( t \): Register \( \%eax \)
    - Return function value in \( \%eax \)

**Object Code**
- 3-byte instruction
- Stored at address \( 0x80483ca \)
Disassembling Object Code

Disassembled

080483c4 <sum>:
  80483c4:  55  push  %ebp
  80483c5:  89 e5  mov  %esp,%ebp
  80483c7:  8b 45 0c  mov  0xc(%ebp),%eax
  80483ca:  03 45 08  add  0x8(%ebp),%eax
  80483cd:  5d  pop  %ebp
  80483ce:  c3  ret

- Disassembler

  `objdump -d filename`

  - Useful tool for examining object code
  - Analyzes bit pattern of series of instructions
  - Produces approximate rendition of assembly code
  - Can be run on either `a.out` (complete executable) or `.o` file
Alternate Disassembly

<table>
<thead>
<tr>
<th>Object</th>
<th>Disassembled</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x401040:</td>
<td>Dump of assembler code for function sum:</td>
</tr>
<tr>
<td>0x55</td>
<td>push %ebp</td>
</tr>
<tr>
<td>0x89</td>
<td>mov %esp,%ebp</td>
</tr>
<tr>
<td>0xe5</td>
<td>mov 0xc(%ebp),%eax</td>
</tr>
<tr>
<td>0x8b</td>
<td>add 0x8(%ebp),%eax</td>
</tr>
<tr>
<td>0x45</td>
<td>pop %ebp</td>
</tr>
<tr>
<td>0x03</td>
<td>ret</td>
</tr>
</tbody>
</table>

Within gdb Debugger

- First run “gdb filename”
- Then inside gdb type “disassemble sum”
  - Disassemble procedure
- x/11xb sum
  - Examine the 11 bytes starting at sum
What Can be Disassembled?

- Anything that can be interpreted as executable code
- Disassembler examines bytes and reconstructs assembly source

% objdump -d WINWORD.EXE

WINWORD.EXE: file format pei-i386

No symbols in "WINWORD.EXE".
Disassembly of section .text:

30001000 <.text>:
30001000: 55 push %ebp
30001001: 8b ec mov %esp,%ebp
30001003: 6a ff push $0xffffffff
30001005: 68 90 10 00 30 push $0x30001090
3000100a: 68 91 dc 4c 30 push $0x304cdc91
Recall: Assembly Programmer’s View

- **Programmer-Visible State**
  - PC: Program counter
    - Address of next instruction
    - Called “EIP” (IA32) or “RIP” (x86-64)
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code, user data, (some) OS data
  - Includes stack used to support procedures
# Integer Registers (IA32)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>16-bit Virtual Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>%ax %ah %al</td>
</tr>
<tr>
<td>%ecx</td>
<td>%cx %ch %cl</td>
</tr>
<tr>
<td>%edx</td>
<td>%dx %dh %dl</td>
</tr>
<tr>
<td>%ebx</td>
<td>%bx %bh %bl</td>
</tr>
<tr>
<td>%esi</td>
<td>%si</td>
</tr>
<tr>
<td>%edi</td>
<td>%di</td>
</tr>
<tr>
<td>%esp</td>
<td>%sp</td>
</tr>
<tr>
<td>%ebp</td>
<td>%bp</td>
</tr>
</tbody>
</table>

**Origin**
- accumulate
- counter
- data
- base
- source
- index
- destination
- index
- stack
- pointer
- base
- pointer

*Mostly obsolete*
Moving Data: IA32

Moving Data

movl Source, Dest:

Operand Types

- **Immediate:** Constant integer data
  - Example: $0x400, $–533
  - Like C constant, but prefixed with `$`
  - Encoded with 1, 2, or 4 bytes

- **Register:** One of 8 integer registers
  - Example: %eax, %edx
  - But %esp and %ebp reserved for special use
  - Others have special uses for particular instructions

- **Memory:** 4 consecutive bytes of memory at address given by register
  - Simplest example: (%eax)
  - Various other “address modes”
### movl Operand Combinations

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
<th>Src,Dest</th>
<th>C Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imm</td>
<td>Reg</td>
<td>movl $0x4,%eax</td>
<td>temp = 0x4;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movl $-147,(%eax)</td>
<td>*p = -147;</td>
</tr>
<tr>
<td>Reg</td>
<td>Reg</td>
<td>movl %eax,%edx</td>
<td>temp2 = temp1;</td>
</tr>
<tr>
<td></td>
<td>Mem</td>
<td>movl %eax,(%edx)</td>
<td>*p = temp;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movl (%eax),%edx</td>
<td>temp = *p;</td>
</tr>
</tbody>
</table>

*Cannot do memory-memory transfer with a single instruction*
Simple Memory Addressing Modes

- **Normal** (R) Mem[Reg[R]]
  - Register R specifies memory address
    
    ```
    movl (%ecx),%eax
    ```

- **Displacement** D(R) Mem[Reg[R]+D]
  - Register R specifies start of memory region
  - Constant displacement D specifies offset
    
    ```
    movl 8(%ebp),%edx
    ```
Using Simple Addressing Modes

```c
void swap(int *xp, int *yp) {
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

**swap:**

```
pushl %ebp
movl %esp,%ebp
pushl %ebx
movl 8(%ebp), %edx
movl 12(%ebp), %ecx
movl (%edx), %ebx
movl (%ecx), %eax
movl %eax, (%edx)
movl %ebx, (%ecx)
popl %ebx
popl %ebp
ret
```
Using Simple Addressing Modes

```c
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

```assembly
pushl %ebp
movl %esp,%ebp
pushl %ebx

movl 8(%ebp), %edx
movl 12(%ebp), %ecx
movl (%edx), %ebx
movl (%ecx), %eax
movl %eax, (%edx)
movl %ebx, (%ecx)

popl %ebx
popl %ebp
ret
```
Understanding Swap

```c
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
</tbody>
</table>
```

```
movl 8(%%ebp), %edx  # edx = xp
movl 12(%%ebp), %ecx # ecx = yp
movl (%edx), %ebx    # ebx = *xp (t0)
movl (%ecx), %eax    # eax = *yp (t1)
movl %eax, (%edx)    # *xp = t1
movl %ebx, (%ecx)    # *yp = t0
```
Understanding Swap

```
movl 8(%ebp), %edx  # edx = xp
movl 12(%ebp), %ecx  # ecx = yp
movl (%edx), %ebx  # ebx = *xp (t0)
movl (%ecx), %eax  # eax = *yp (t1)
movl %eax, (%edx)  # *xp = t1
movl %ebx, (%ecx)  # *yp = t0
```
Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td></td>
</tr>
<tr>
<td>%ebx</td>
<td></td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x104</td>
</tr>
</tbody>
</table>

```
movl 8(%ebp), %edx  # edx = xp
movl 12(%ebp), %ecx # ecx = yp
movl (%edx), %ebx   # ebx = *xp (t0)
movl (%ecx), %eax   # eax = *yp (t1)
movl %eax, (%edx)   # *xp = t1
movl %ebx, (%ecx)   # *yp = t0
```
Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x120</td>
</tr>
<tr>
<td>%ebx</td>
<td></td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x104</td>
</tr>
</tbody>
</table>

- `%edx` is set to 0x124
- `%ecx` is set to 0x120

Address Table:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0x120</td>
</tr>
<tr>
<td>8</td>
<td>0x124</td>
</tr>
<tr>
<td>4</td>
<td>Rtn adr</td>
</tr>
<tr>
<td>0</td>
<td>0x108</td>
</tr>
<tr>
<td>-4</td>
<td>0x100</td>
</tr>
</tbody>
</table>

Code:

```
movl 8(%ebp), %edx  # edx = xp
movl 12(%ebp), %ecx  # ecx = yp
movl (%edx), %ebx    # ebx = *xp (t0)
movl (%ecx), %eax    # eax = *yp (t1)
movl %eax, (%edx)    # *xp = t1
movl %ebx, (%ecx)    # *yp = t0
```
### Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x120</td>
</tr>
<tr>
<td>%ebx</td>
<td>123</td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x104</td>
</tr>
</tbody>
</table>

#### Instructions

- `movl 8(%ebp), %edx`  # edx = xp
- `movl 12(%ebp), %ecx` # ecx = yp
- `movl (%edx), %ebx`  # ebx = *xp (t0)
- `movl (%ecx), %eax`  # eax = *yp (t1)
- `movl %eax, (%edx)`  # *xp = t1
- `movl %ebx, (%ecx)`  # *yp = t0
Understanding Swap

%eax  456
%edx  0x124
%ecx  0x120
%ebx  123
%esi
%edi
%esp
%ebp  0x104

movl  8(%ebp), %edx          # edx = xp
movl  12(%ebp), %ecx         # ecx = yp
movl  (%edx), %ebx           # ebx = *xp (t0)
movl  (%ecx), %eax           # eax = *yp (t1)
movl  %eax, (%edx)           # *xp = t1
movl  %ebx, (%ecx)           # *yp = t0
Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>456</td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x120</td>
</tr>
<tr>
<td>%ebx</td>
<td>123</td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp</td>
<td>12</td>
</tr>
<tr>
<td>xp</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>%ebp</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-4</td>
</tr>
</tbody>
</table>

- `movl 8(%ebp), %edx`  # edx = xp
- `movl 12(%ebp), %ecx` # ecx = yp
- `movl (%edx), %ebx`   # ebx = *xp (t0)
- `movl (%ecx), %eax`   # eax = *yp (t1)
- `movl %eax, (%edx)`   # *xp = t1
- `movl %ebx, (%ecx)`   # *yp = t0
Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>456</td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x120</td>
</tr>
<tr>
<td>%ebx</td>
<td>123</td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x104</td>
</tr>
</tbody>
</table>

```
movl 8(%ebp), %edx  # edx = xp
movl 12(%ebp), %ecx # ecx = yp
movl (%edx), %ebx   # ebx = *xp (t0)
movl (%ecx), %eax   # eax = *yp (t1)
movl %eax, (%edx)   # *xp = t1
movl %ebx, (%ecx)   # *yp = t0
```