Machine Programming I: Basics

- What is an ISA (Instruction Set Architecture)
- A brief history of Intel processors and architectures
  - Intel processors [Wikipedia]
  - Intel microarchitectures
- C, assembly, machine code
- Assembly basics: registers, operands, move instructions

What should the HW/SW interface be?
The General ISA

General ISA Design Decisions

- Instructions
  - What instructions are available? What do they do?
  - How are they encoded?

- Registers
  - How many registers are there?
  - How wide are they?

- Memory
  - How do you specify a memory location?
HW/SW Interface: Code / Compile / Run Times

What makes programs run fast?
(deja-vu? 😊)

Executing Programs Fast!

- The time required to execute a program depends on:
  - The program (as written in C, for instance)
  - The compiler: what set of assembler instructions it translates the C program into
  - The ISA: what set of instructions it made available to the compiler
  - The hardware implementation: how much time it takes to execute an instruction

- There is a complex interaction among these
Intel x86 Processors

- Totally dominate the server/laptop market

- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on

- Complex instruction set computer (CISC)
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - Hard to match performance of Reduced Instruction Set Computers (RISC)
  - But, Intel has done just that!

---

Intel x86 Evolution: Milestones

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>29K</td>
<td>5-10</td>
</tr>
<tr>
<td>386</td>
<td>1985</td>
<td>275K</td>
<td>16-33</td>
</tr>
<tr>
<td>Pentium 4F</td>
<td>2005</td>
<td>230M</td>
<td>2800-3800</td>
</tr>
</tbody>
</table>

- First 16-bit processor. Basis for IBM PC & DOS
- 1MB address space
- First 32 bit processor, referred to as IA32
- Added “flat addressing”
- Capable of running Unix
- 32-bit Linux/gcc uses no instructions introduced in later models
- First 64-bit processor
- Meanwhile, Pentium 4s (Netburst arch.) phased out in favor of “Core” line
Intel x86 Processors

- **Machine Evolution**
  - 486 1989 1.9M
  - Pentium 1993 3.1M
  - Pentium/MMX 1997 4.5M
  - PentiumPro 1995 6.5M
  - Pentium III 1999 8.2M
  - Pentium 4 2001 42M
  - Core 2 Duo 2006 291M

- **Added Features**
  - Instructions to support multimedia operations
    - Parallel operations on 1, 2, and 4-byte data, both integer & FP
  - Instructions to enable more efficient conditional operations

- **Linux/GCC Evolution**
  - Very limited impact on performance --- mostly came from HW.

x86 Clones: Advanced Micro Devices (AMD)

- **Historically**
  - AMD has followed just behind Intel
  - A little bit slower, a lot cheaper

- **Then**
  - Recruited top circuit designers from Digital Equipment and other downward trending companies
  - Built Opteron: tough competitor to Pentium 4
  - Developed x86-64, their own extension to 64 bits
Intel’s 64-Bit

- Intel Attempted Radical Shift from IA32 to IA64
  - Totally different architecture (Itanium) and ISA
  - Executes IA32 code only as legacy
  - Performance disappointing
- AMD Stepped in with Evolutionary Solution
  - x86-64 (now called “AMD64”)
- Intel Felt Obligated to Focus on IA64
  - Hard to admit mistake or that AMD is better
- 2004: Intel Announces EM64T extension to IA32
  - Extended Memory 64-bit Technology
  - Almost identical to x86-64!
- Meanwhile: EM64T well introduced, however, still often not used by OS, programs

Our Coverage in 351

- IA32
  - The traditional x86
- x86-64/EM64T
  - The emerging standard – all Labs in 64 bits!
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 (we’ll come back to that)

Definitions

- **Architecture**: (also instruction set architecture or ISA)
  The parts of a processor design that one needs to understand to write assembly code (“what is directly visible to SW”)

- **Microarchitecture**: Implementation of the architecture

- How about CPU frequency?
- The number of registers?
- Is the cache size “architecture”?
Turning C into Object Code

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

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

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

Compiling Into Assembly

Obtain with command

```
gcc -O -S code.c
```

Produces file `code.s`
Three Basic Kinds of Instructions

- Perform arithmetic function on register or memory data

- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory

- Transfer control (control flow)
  - Unconditional jumps to/from procedures
  - Conditional branches

Assembly Characteristics: Data Types

- “Integer” data of 1, 2, or 4, 8 (x86-64) bytes
  - Data values
  - Addresses

- Floating point data of 4, 8, or 10 bytes

- What about aggregate types such as arrays or structures?
Assembly Characteristics: Data Types

- "Integer" data of 1, 2, or 4, 8 (x86-64) bytes
  - Data values
  - Addresses (untyped pointers)

- Floating point data of 4, 8, or 10 bytes

- No aggregate types such as arrays or structures
  - Just contiguously allocated bytes in memory

Object Code

Code for `sum`

```
0x401040 <sum>:
  0x55
  0x89
  0xe5
  0x8b
  0x45
  0xc0
  0x03
  0x45
  0x08
  0x89
  0xec
  0x5d
  0xc3
```

- **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
Example

```c
int t = x+y;
```

**C Code**
- Add two signed integers

**Assembly**
- Add 2 4-byte integers
  - “Long” words in GCC speak
  - 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 `0x401046`

Disassembling Object Code

```
0x401046: 03 45 08
```

**Disassembler**
- `objdump -d p`
  - 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>0x401040 &lt;sum&gt;: push %ebp</td>
</tr>
<tr>
<td>0x55</td>
<td>0x401041 &lt;sum+1&gt;: mov %esp,%ebp</td>
</tr>
<tr>
<td>0x89</td>
<td>0x401043 &lt;sum+3&gt;: mov 0xc(%ebp),%eax</td>
</tr>
<tr>
<td>0xe5</td>
<td>0x401046 &lt;sum+6&gt;: add 0x8(%ebp),%eax</td>
</tr>
<tr>
<td>0x8b</td>
<td>0x401049 &lt;sum+9&gt;: mov %ebp,%esp</td>
</tr>
<tr>
<td>0x45</td>
<td>0x40104b &lt;sum+11&gt;: pop %ebp</td>
</tr>
<tr>
<td>0x0c</td>
<td>0x40104c &lt;sum+12&gt;: ret</td>
</tr>
<tr>
<td>0x03</td>
<td>0x40104d &lt;sum+13&gt;: lea 0x0(%esi),%esi</td>
</tr>
</tbody>
</table>

- Within gdb Debugger
  - gdb p
  - disassemble sum
    - Disassemble procedure
  - x/13b sum
    - Examine the 13 bytes starting at sum

What Can be Disassembled?

% 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 $0x3001090
3000100a: 68 91 dc 4c 30 push $0x304cdc91

- Anything that can be interpreted as executable code
- Disassembler examines bytes and reconstructs assembly source
### Integer Registers (IA32)

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

#### General Purpose Registers

- %eax
- %ecx
- %edx
- %ebx
- %esi
- %edi
- %esp
- %ebp

#### Origin (mostly obsolete)

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

#### 16-bit Virtual Registers (backwards compatibility)
**x86-64 Integer Registers**

<table>
<thead>
<tr>
<th>%rax</th>
<th>%eax</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rbx</td>
<td>%ebx</td>
</tr>
<tr>
<td>%rcx</td>
<td>%ecx</td>
</tr>
<tr>
<td>%rdx</td>
<td>%edx</td>
</tr>
<tr>
<td>%rsi</td>
<td>%esi</td>
</tr>
<tr>
<td>%rdi</td>
<td>%edi</td>
</tr>
<tr>
<td>%rsp</td>
<td>%esp</td>
</tr>
<tr>
<td>%rbp</td>
<td>%ebp</td>
</tr>
<tr>
<td>%r8</td>
<td>%r8d</td>
</tr>
<tr>
<td>%r9</td>
<td>%r9d</td>
</tr>
<tr>
<td>%r10</td>
<td>%r10d</td>
</tr>
<tr>
<td>%r11</td>
<td>%r11d</td>
</tr>
<tr>
<td>%r12</td>
<td>%r12d</td>
</tr>
<tr>
<td>%r13</td>
<td>%r13d</td>
</tr>
<tr>
<td>%r14</td>
<td>%r14d</td>
</tr>
<tr>
<td>%r15</td>
<td>%r15d</td>
</tr>
</tbody>
</table>

- Twice the number of registers, accessible as 8, 16, 32, 64 bits

---

**x86-64 Integer Registers: Usage Conventions**

<table>
<thead>
<tr>
<th>%rax</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rbx</td>
<td>Callee saved</td>
</tr>
<tr>
<td>%rcx</td>
<td>Argument #4</td>
</tr>
<tr>
<td>%rdx</td>
<td>Argument #3</td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument #2</td>
</tr>
<tr>
<td>%rdi</td>
<td>Argument #1</td>
</tr>
<tr>
<td>%rsp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>%rbp</td>
<td>Callee saved</td>
</tr>
<tr>
<td>%r8</td>
<td>Argument #5</td>
</tr>
<tr>
<td>%r9</td>
<td>Argument #6</td>
</tr>
<tr>
<td>%r10</td>
<td>Caller saved</td>
</tr>
<tr>
<td>%r11</td>
<td>Caller saved</td>
</tr>
<tr>
<td>%r12</td>
<td>Callee saved</td>
</tr>
<tr>
<td>%r13</td>
<td>Callee saved</td>
</tr>
<tr>
<td>%r14</td>
<td>Callee saved</td>
</tr>
<tr>
<td>%r15</td>
<td>Callee saved</td>
</tr>
</tbody>
</table>