Course Introduction

What this course is about

- Hardware/Software interface:
  - Compilers, assemblers, linkers, loaders: who does what in terms of getting my program to run?
  - What kind of instructions does the machine understand?
- Organization:
  - What are the basic pieces of the machine (registers, cache, ALU, busses)?
  - How are these pieces connected? How are they controlled?
- Performance:
  - What does it mean for one machine to be “faster” than another?
  - What are MFLOPS, MIPS, benchmark programs?

Levels of Abstraction

- We can describe a computer system as a set of layers:

```
Application program (e.g. C/C++, Java/etc)
Machine program (e.g. C/C++, Lisp, Java, etc)
OS architecture (system calls)
Machine architecture (ISA) (e.g. MIPS, x86, Alpha, etc)
```

Instruction Set Architecture

- ISA is an interface between the hardware and software.
- ISA is what is visible to the programmer (note that the OS and users might have different view)
- ISA consists of
  - instructions (operations, how are they encoded?)
  - information units (what is their size, how are they addressed)
  - registers (general or special purpose)
  - input-output control
- ISA is an abstract view of the machine: underlying details should be hidden from the programmer (although this is not always the case)
Computer Families

- Sequence of machines that have the same ISA (binary compatible). For example:
  1. IBM 360 Series (invented the notion of ISA in 1960s)
  2. DEC PDP-11, VAX [1970s]
  3. Intel x86 (80386, 80486, Pentium, PII, PIII, PIV)
  4. Motorola 680x0
  5. MIPS Rx000 [1980s to present]
  6. Sun SPARC [1980s to present]
  7. DEC Alpha (21x64) [1990s to present]

- With “portable” software, are “binary compatible” machines important?

Computer Generations

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Technology</td>
<td>Vacuum tubes</td>
<td>Transistors</td>
<td>Integrated circuits</td>
<td>LSI</td>
</tr>
<tr>
<td>Processor Structure</td>
<td>Single processor</td>
<td>Multiple functional units</td>
<td>Micros and minis</td>
<td>Workstations and PCs</td>
</tr>
<tr>
<td>Memory</td>
<td>Vacuum tubes</td>
<td>Magnetic core</td>
<td>Semi-conductors</td>
<td>Semi-cond. 64KB</td>
</tr>
</tbody>
</table>

Stored Program Computer

- Instructions and data are binary strings
- 5 basic building blocks: arithmetic (datapath), control, memory, input, output:

Computer Structure

- CPU
  - Control
  - ALU
  - Registers
  - PC
  - Status
- Memory
  - Memory Bus
  - Memory Hierarchy
- I/O
  - I/O Bus
  - I/O
The CPU - What does it do?

- The CPU “executes” the following program:
  ```plaintext
  while (TRUE) do
    fetch the next instruction
    decode it
    execute it
    calculate the address of the next instruction
  end while
  ```
- How does it know where to find the next instruction?
- Where does it “keep” the current instruction?
- Where do instructions come from?
- When does it stop?
- We’ll be refining this picture during the next few weeks....

Instructions

- An instruction tells the CPU:
  - The operation to be performed (the opcode)
  - The operands (zero or more)
- For a given instruction, the ISA specifies
  - the meaning (semantics) of the opcode
  - how many operands are required (and their types)
- Operands can be of the following type
  - registers
  - memory address
  - constant (immediate data)
- In MIPS, the operands are typically registers or small constants

Registers

- Registers are visible both to hardware and programmer
- High-speed storage of operands
- Easy to name
- Also used to address memory
- Most current computers have 32 or 64 registers
- Not all registers are “equal”
  - Some are special purpose (eg. in MIPS $0$ is hardwired to 0).
- Integer / Floating point
- Conventions (stack pointers)
- Why no more than 32 or 64? (at least 3 good reasons)

The Memory System

- Memory is a hierarchy of devices/components which get increasingly faster (and more expensive) as they get nearer to the CPU:

<table>
<thead>
<tr>
<th>Memory level</th>
<th>Capacity (bytes)</th>
<th>Speed</th>
<th>Relative Speed</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1000s</td>
<td>nanoseconds</td>
<td>1</td>
<td>??</td>
</tr>
<tr>
<td>Cache</td>
<td>16KB on-chip</td>
<td>nanoseconds</td>
<td>1-2</td>
<td>$100/MB</td>
</tr>
<tr>
<td></td>
<td>1MB off-chip</td>
<td>10s of ns</td>
<td>5-10</td>
<td>??</td>
</tr>
<tr>
<td>Primary memory</td>
<td>10-100MB</td>
<td>10s to 100s ns</td>
<td>10-100</td>
<td>$1/MB</td>
</tr>
<tr>
<td>Secondary mem.</td>
<td>1-10GB</td>
<td>10s of ms</td>
<td>1,000,000</td>
<td>$0.01/MB</td>
</tr>
</tbody>
</table>

- Library metaphor of memory hierarchy
Memory

- Memory is an array of information units
- Each unit has the same size
- Each unit has a unique address
- Address and contents are different

A memory of size N units

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>122</td>
</tr>
<tr>
<td>1</td>
<td>-4</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n-1</td>
<td></td>
</tr>
</tbody>
</table>

- A C variable is an abstraction for a memory location

Information Units

- Basic unit is the bit (stores a 0 or a 1)
- Bits are grouped together into larger units:
  - bytes = 8 bits
  - words = 4 bytes
  - double words = 2 words (8 bytes)

Binary Representation

- Computers represent all data (integers, floating point numbers, characters, instructions, etc.) in a binary representation. Interpretation depends on context.
- Know your (common) powers of two!

<table>
<thead>
<tr>
<th>Power</th>
<th>Value</th>
<th>Slang</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>256</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1024 or ~1000</td>
<td>1K</td>
</tr>
<tr>
<td>16</td>
<td>65536 or ~64000</td>
<td>64K</td>
</tr>
<tr>
<td>20</td>
<td>~1,000,000</td>
<td>1M</td>
</tr>
<tr>
<td>30</td>
<td>~1,000,000,000</td>
<td>1G</td>
</tr>
<tr>
<td>32</td>
<td>~4,000,000,000</td>
<td>4G</td>
</tr>
</tbody>
</table>

2s Complement

- Representing integers: What characteristics does our scheme need?
  - Easy test for positive/negative.
  - Equal number of positive and negative numbers
  - Easy check for overflow
- Different schemes: sign and magnitude, 1’s complement, 2’s complement
- 2’s complement tricks (sign bit extension, converting from positive to negative, addition/subtraction)
- Modern machines use 2s complement
- 2s complement numbers are easy to add and negate, giving us subtraction for “free”
- 2s complement tricks: sign extension, negation, addition/subtraction
- Hexidecimal notation
Addressing

- The *address space* is the set of all information units that a program can reference.
- Most machines today are *byte addressable*.
- Processor "size" impacts the size of the address space:
  - 16 bit processor: 64KB (too small nowadays)
  - 32 bit processor: 4GB (starting to be too small)
  - 64 bit processor: really big (should last for a while...)
- Rule of thumb: We're using up address space at a rate of around 1 bit per year...

Addressing Words

- On a byte addressable machine, every word starts at an address divisible by 4:

A memory of size N bytes

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>n-4</td>
</tr>
</tbody>
</table>

- Big vs. Little Endian: within a data unit (e.g., word), how are the individual bytes laid out?
- Little/Big: address of data unit is address of low/high order byte (DEC MIPS is Little; SGI MIPS, SPARC are Big)