Review

C program: foo.c

Assembly program: foo.s

Object (mach lang module): foo.o

Executable (mach lang pgm): a.out

Compiler

Assembler

Linker

Loader

Memory

Executable (mach lang pgm): a.out

Linker

Loader

Memory

C program: foo.c

Assembly program: foo.s

Object (mach lang module): foo.o
What are “Machine Structures”? 

Coordination of many levels of abstraction

Application (Netscape) 
Compiler 
Assembler 
Operating System (MacOS X) 
Processor 
Memory 
I/O system 
Datapath & Control 
Digital Design 
Circuit Design 
transistors 

Software 
Hardware 
Instruction Set Architecture
Below the Program

• High-level language program (in C)

```c
swap  int v[], int k){
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

• Assembly language program (for MIPS)

```assembly
swap:  sll  $2, $5, 2
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr    $31
```

• Machine (object) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000 . . .
```
Design Principles for Hardware

1. Simplicity favors regularity
   • Every instruction has 3 operands, opcode same place for EVERY instr

2. Smaller is faster
   • 32 registers, no more

3. Good design demands good compromise
   • MIPS Immediate format vs. R format

4. Make the common case fast
   • Support of constants via immediates
Computer Technology - Dramatic Change!

• Memory
  • DRAM capacity: 2x / 2 years (since ‘96);
    64x size improvement in last decade.

• Processor
  • Speed 2x / 1.5 years (since ‘85);
    100X performance in last decade.

• Disk
  • Capacity: 2x / 1 year (since ‘97);
    250X size in last decade.
Big Ideas so far

- 15 weeks to learn big ideas in CS&E
  - Principle of abstraction, used to build systems as layers
  - Pliable Data: a program determines what it is
  - Stored program concept: instructions just data
  - Compilation v. interpretation to move down layers of system
  - Principle of Locality, exploited via a memory hierarchy (cache)
  - Greater performance by exploiting parallelism (pipeline)
  - Principles/Pitfalls of Performance Measurement
What does your 61C future hold?

• We’ve reached the half-way mark. It’s time to shift gears a bit, but to what?

• Architecture! (aka “Systems”)
  • Caches
  • Virtual Memory
  • CPU Organization
  • Pipelining
  • I/O
  • Networks
  • Performance
The Big Picture

Computer

Processor (active)
Control ("brain")
Datapath ("brawn")

Memory (passive) (where programs, data live when running)

Devices
Input
Output

Keyboard, Mouse
Disk, Network
Display, Printer
Memory Hierarchy (1/3)

• Processor
  • executes instructions on order of nanoseconds to picoseconds
  • holds a small amount of code and data in registers

• Memory
  • More capacity than registers, still limited
  • Access time ~50-100 ns

• Disk
  • HUGE capacity (virtually limitless)
  • VERY slow: runs ~milliseconds
As we move to deeper levels the latency goes up and price per bit goes down.

Q: Can $/bit go up as move deeper?
Memory Hierarchy (3/3)

• If level closer to Processor, it must be:
  • smaller
  • faster
  • subset of lower levels (contains most recently used data)

• Lowest Level (usually disk) contains all available data

• Other levels?
Memory Caching

• We’ve discussed three levels in the hierarchy: processor, memory, disk

• Mismatch between processor and memory speeds leads us to add a new level: a memory cache

• Implemented with SRAM technology: faster but more expensive than DRAM memory.
Memory Hierarchy Analogy: Library (1/2)

• You’re writing a term paper (Processor) at a table in Doe

• Doe Library is equivalent to disk
  • essentially limitless capacity
  • very slow to retrieve a book

• Table is memory
  • smaller capacity: means you must return book when table fills up
  • easier and faster to find a book there once you’ve already retrieved it
Memory Hierarchy Analogy: Library (2/2)

- Open books on table are **cache**
  - smaller capacity: can have very few open books fit on table; again, when table fills up, you must close a book
  - much, much faster to retrieve data

- Illusion created: whole library open on the tabletop
  - Keep as many recently used books open on table as possible since likely to use again
  - Also keep as many books on table as possible, since faster than going to library
Memory Hierarchy Basis

- Disk contains everything.
- When Processor needs something, bring it into to all higher levels of memory.
- Cache contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Entire idea is based on **Temporal Locality**: if we use it now, we’ll want to use it again soon (a Big Idea)
Cache Design

• How do we organize cache?

• Where does each memory address map to?
  (Remember that cache is subset of memory, so multiple memory addresses map to the same cache location.)

• How do we know which elements are in cache?

• How do we quickly locate them?
Direct-Mapped Cache (1/2)

• In a **direct-mapped cache**, each memory address is associated with one possible **block** within the cache
  • Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  • Block is the unit of transfer between cache and memory
Direct-Mapped Cache (2/2)

- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks => any memory location that is multiple of 4
Issues with Direct-Mapped

• Since multiple memory addresses map to same cache index, how do we tell which one is in there?

• What if we have a block size > 1 byte?

• Result: divide memory address into three fields

| tttttttttttttttttttt | iiiiiiiiiiiiiiiiiiiii | ooooo |

  tag to check if have correct block | index to select block | byte offset within block |
Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- **Index**: specifies the cache index (which “row” of the cache we should look in)
- **Offset**: once we’ve found correct block, specifies which byte within the block we want
- **Tag**: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location
Administrivia

- Midterm exam in 5.5 hours!
  - 7pm @ 155 Dwinelle
  - 9 Questions (many short-answer)
  - 75 points

- Project 2
  - All grades entered.
  - Grades frozen 2004-03-15 (one week to request regrade, same as always).
A. Mem hierarchies were invented before 1950. (UNIVAC I wasn’t delivered ‘til 1951)

B. If you know your computer’s cache size, you can often make your code run faster.

C. Memory hierarchies take advantage of spatial locality by keeping the most recent data items closer to the processor.
A. I’m **ready** / **not-ready** for the exam

B. The exam is out of 75. The tens digit of my score will probably be: 7 / 6 / 5 / 4 / \( \leq 3 \)
And in conclusion...

• We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.

• So we create a memory hierarchy:
  • each successively lower level contains “most used” data from next higher level
  • exploits temporal locality
  • do the common case fast, worry less about the exceptions (design principle of MIPS)

• Locality of reference is a Big Idea