EE 108B Review Session #5

Njuguna Njoroge
Friday, November 11, 2005
Skilling 193
Live on E3, 5:15-6:05 PM

Today's Agenda…

- Announcements
- PA2 Overview
- Motivation for studying memory systems
- Homework review
- Sample Problems

Announcements

- Lab 3 due next Tuesday (11/15)
- PA2, part A also due Tuesday (11/15)
- HW 4 due next Thursday (11/17)
- Homework regrade policy – please refer to "Homework" link on class webpage
- We generally do not accept e-mail submissions of homeworks
- TA evaluations

TA Evaluations Instructions

a) The TA evaluation form will be available Nov 10 to Nov 20

b) Go to http://eeclass.stanford.edu/ and click on "Online Forms"

c) Enter your name. Please note that all submissions are completely anonymous and your names will not be associated with your submissions.
   - You'll get **10 points** towards your quarter's homework score

d) Select the course

e) Evaluate all TAs and submit the forms.

PA2 Overview

- MIPS simulator in MIPS simulator
  - Part 1: Only support addi instruction
    - The idea is to get you started early
  - Part 2: Should support all following instructions
    - Loads and stores: lw, sw
    - ALU operations: add, addi, sub, and, andi, sll, slt, slti, lui
    - Branches and jumps: jr, jal, beq, bne

PA2 tips

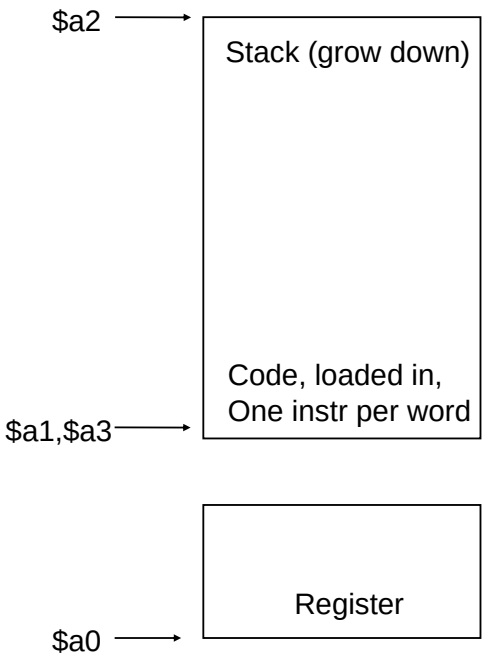
- Be sure to distinguish between the SPIM simulator and your own simulator
- Make sure to setup your system initially in the same way as a real machine would be initialized
- Look at appendix A of the book for diagrams of instruction formats.
- Please **READ** the FAQs before sending us e-mail…likely that most of your questions are answered there …

PA2: What you need to do

- Here is how it works
  - Read in a file of MIPS assembly instructions (in integer format)
  - Get yourself ready to do the operations by storing the $s registers and the $ra register
  - Fetch the current instruction from memory
  - Do some shifts to get the proper fields ready
    - Look at the opcode,
    - The register number, etc.
  - Once you have the opcode, jump to the proper point in your simulator to handle the given instruction
    - Execute the instruction
    - Save result in the proper register
  - The registers are in memory

PA2: Memory map

- $a0: first word of register space
  - Remember to multiply the register number by 4
- $a1: first word of code/static/stack space
  - All memory accesses are relative to this offset
- $a2: Last word of code/static/stack space
- $a3: first word of code



PA2 example: lw $t0, offset($s0)

- What you would need to do is (Recall that $t0 is $8 and $s0 is $16)

- Stores, branches, etc. work similarly. Just remember that you'll need to add in an extra $a1 offset in order to translate to the SPIM memory.

PA1 Pitfalls

- Not clearly differentiating the virtual memory map and where SPIM memory comes in play
- Confusing virtual registers and SPIM registers
- Not properly initializing the special registers

So what's the fuss with memory systems?

- Processors have become much faster than memories
- Memory system is now one of major bottlenecks
  - In today's CPU, memory accesses take 100s of CPU cycles…All that clever pipelining and forwarding down the drain !
  - So, in the spirit of Amdahl's law, architect focuses on the bottlenecks in improving performance
- Example from my research from a couple days ago
  - Xilinx's FPGA place and router (PAR) took 5.5 hours to complete on machine A and only 1 hour on machine B (same version of tools)
  - Relevant machine specs
    - Machine A: 3.06 GHz P4, 0.5 GB RAM, running Win XP Pro
    - Machine B: 2.79 GHz P4, 1.0 GB RAM, running Win XP Pro
    - Result: 2x more RAM, 5.5x speed-up in → This is what we call "Super-linear speed-up"
    - Moral of story: Spend extra $$ on getting on more $ (no pun intended ☺)

Homework Hints: Problem 1

- Use AMAT equations, but how…
- How do you deal with dirty data in write-back $L2?
- How do you compute CPIs given AMAT

HW Problem 2: More AMAT principles

- Part b hint

- Part c hints

HW Problem 2: More hints

- Part d and e hints

- Part f hints

HW Problems 3, 4 and 5

- Problem 3
- Problem 4
- Problem 5

Sample Problem 1

- Consider the following code from lecture 12

```
double a[8192], b[8192], c[8192];
void vector_sum()
{
        int i;
        for (i = 0; i < 8192; i++)
        {
                c[i] = a[i] + b[i];
        }
```

- What are the main issues of running this loop in a processor with relatively small caches?
- What were solutions proposed in lecture?

Sample Problem 1 cont'd

- Lecture solution is viable
- Any alternative solutions?

Sample Problem 2

- Claim: Miss rate of a fully associative cache is always lower than the miss rate of a direct mapped cache of the same size. True or false?

Sample Problem 2 cont'd

That's all folks

- No review session next weekend…enjoy the Turkey holiday!