EE 108B Review Session # 2

Njuguna Njoroge

Friday, October 14, 2005

5:15 PM – 6:05 PM

Room Skilling 193

Agenda

- Introducing myself…
- Announcements
- HW 2 Topics Overview + Hints
- Basic Computer Arithmetic
- Sample Problems

A little about myself

- My nickname is Nju, pronounced as “Joe” (name is Kenyan)
- PhD student in Electrical Engineering, also doing MS&E Masters
  - Research interest mainly parallel architectures
  - Working on Prof. Olukuton’s TCC (Transactional Coherence and Consistency) project (http://tcc.stanford.edu)
    - Implementing the TCC cache model in Verilog targeted to the Xilinx XUP FPGA boards (same boards as in ee108b lab)
- Took ee182+ee183 (now ee108b) as undergrad here and this is my 5th time TA-ing ee108b and its one of my favorite classes to TA ☺

Announcements

- PA1 due next Thurs (10/20)…Try to save your late day for PA2.b
- HW 2 due Tues. 10/25 @ 5 PM
  - No late day for midterm (need to post solutions on-line that night.
  - No credit (i.e. 0 points) for late homework after solutions are posted
- Lab 1 due on Tuesday at 5 PM (save late days for lab 3 or lab 4)
- Midterm in 2 weeks (10/27/05) – practice midterms coming out next week

HW 2 Coverage + Hints

- Prob. 1: Compiler Optimizations
  - Use Lecture 6 (10/13), slides 17-31
  - For each compiler optimization, try to look through code to see if it was used…
- Prob. 2: Memory Access frequencies
  - Typo in book -- use figure 2.48 on page 146

HW 2 Coverage + Hints continued…

- Prob. 3, 4, 7 – Basics of performance equations
  - Execution Time = Total Instructions x CPI x Clock Cycle Time
  - Execution Time = Total Cycles x Clock Cycle Time
  - Total Cycles = $\sum_{i=1}^{n} (CPI_i \times IC_i)$
  - CPI = $\sum_{i=1}^{n} (CPI_i \times \frac{Instruction\_Count_i}{Total\_Instruction\_Count})$

HW 2 Coverage + Hints continued…

- Prob 5 – Material will be covered in next week's review session
- Prob 6 – Material will be covered in next week's review session
  - Execution Time = Total Cycles x Clock Cycle Time

Basic Computer Arithmetic

- unsigned binary notation

- 1’s complement notation

- 2’s complement notation
  - derivation
  - unsigned versus signed arithmetic (addu vs. add)

1. How do we test for Overflow in MIPS addition?
    a. For unsigned numbers?

b. For signed numbers?

2. For some multimedia applications, a useful operation is *saturating addition:* if the result of an add exceeds some threshold value, we "pin" the result to that value. In other words, whereas regular unsigned Add could be defined as Add(a,b)=a+b, an unsigned Add that "saturates at N" is defined as: AddSat(a,b) = Min(a+b, N). We want to implement addsat32 to an existing ALU, which we've depicted building a 32-bit ALU out of 1-bit ALU slices. ALUop[2] is the Subtract signal, and ALUop[1:0] select one of four results to send through the ALU output mux. You decide to modify the ALU to perform 32-bit saturating add if ALUop[2:0]=**011,** which is currently unused. (See table.)

ALUop [2:0]	**Function**
000	and
001	or
010	add
011	**addsat32 (new)**
110	sub
111	slt

a. Why are we using the unused encoding **011** for **addsat32**, rather than one of the other unused encodings **100** or **101**?

b. Modify the 1-bit ALU slice so that setting ALUop to **011** will cause the ALU to compute the 32-bit saturating add of A and B. Existing ALU behaviors should remain the same.