CS152 Homework II, Fall 2005

Name: ____________________

SSID: ____________________

Homework II is due in class on Thursday December 1st at 11:10 AM. This class is the Mid-term I review session.

Late homeworks are NOT accepted. Thus, if you will not be attending the review session, you MUST make arrangements to hand off the homework to the instructor before class time.

Homework will be graded on effort (did you make an honest attempt to solve each problem?), not correctness. We will distribute the correct answers for the homework in the review session, but we will probably not return the homework you hand in until after the exam. So, you may wish to make a copy for reference before you hand it in.

This homework will count for approximately 2% of your final grade. The homework is based on the Mid-term I & II exams from Spring 05. You may discuss the homework problems with fellow students and the TAs, but what you write down must be your own work (no copying the answers from someone else's homework). Good luck! John Lazzaro

1 Set-associative Caches (10 points)

For an N-way set-associative cache, with K cache lines, and B bytes in a cache block (i.e. in a 4-way set-associative cache, each line has 4 blocks for a total of 4B cache data bytes), derive a general-purpose equation for the fraction of total cache RAM bits (defined as tag bits + block bits + valid bits) that is devoted to tag RAM bits.

The equation should take the form $\frac{1}{1+f(K,B)}$. Draw a box around your final equation.

2 Cache Debugging (15 points)

We fabricate a processor with separate instruction and data caches, and with no address translation hardware (thus, like your class CPU, programs run in physical address space). The data cache is direct-mapped, is not allocate-on-write. The data cache has 28-bit cache tags and 4 cache lines. Thus, each line stores a single word.

After the chip goes out to fab, we realize we made an error in the design, and the data cache valid bits are permanently stuck at "1". The processor has no special cache-control instructions, just normal LW and SW instructions. We must assume that on power-up, the cache tag and cache block RAM for each line will hold arbitrary values. However, the RAMs works correctly – whatever value it holds remains until the cache state machine changes it. (questions begin on next page).

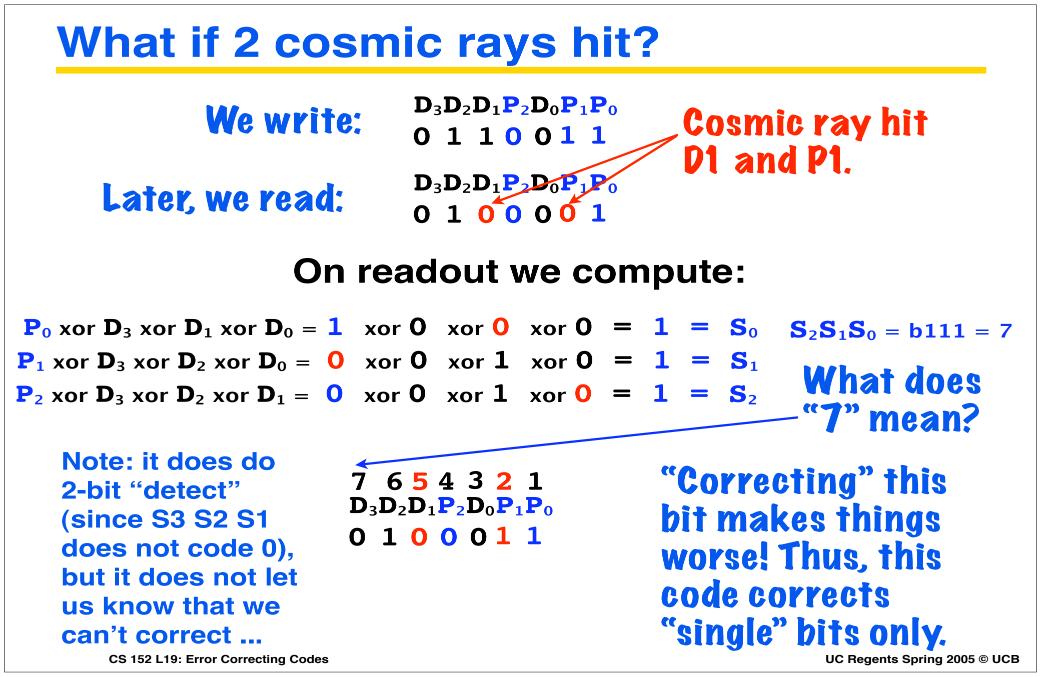
Question 2a (8 points). The processor starts reading instructions from location 0x000000, in which we can put a program to execute at power up. Write a program to place at address location 0x00000000 that is guaranteed to fill the data cache with data that matches the data held in (fully populated) main memory. Your program may only use LW instructions. A known solution uses 8 LW instructions.

Question 2b (7 points). Assume this bug (stuck V bits) happened in your direct-mapped instruction cache. Like the data cache, the instruction cache has 28-bit cache tags and 4 cache lines. The processor starts reading instructions from location 0x000000, in which we can put a program to execute at powerup. However, all reads (including the first) go through the instruction cache (as the designers assumed the V bits would be cleared at startup, not set!).

If you are very unlucky, when your machine powers up, the instruction cache tag and cache data RAMs will hold values that will make it impossible for you to get control of the machine. Describe example values of the instruction cache RAMs that would produce this problem (there are many: just one example will suffice).

3 ECC (18 points)

Below, we show a slide from the Error Correcting Codes lecture.



What if 2 cosmic rays hit?

We write: $D_3D_2D_1P_2D_0P_1P_0$

0 1 1 0 0 1 1

Later, we read: $D_3D_2D_1P_2D_0P_1P_0$

0 1 0 0 0 0 1

Cosmic ray hit D1 and P1.

On readout we compute:

P_0 xor D_3 xor D_1 xor D_0 = 1 xor 0 xor 0 xor 0 = 1 = S_0

P_1 xor D_3 xor D_2 xor D_0 = 0 xor 0 xor 1 xor 0 = 1 = S_1

P_2 xor D_3 xor D_2 xor D_1 = 0 xor 0 xor 1 xor 0 = 1 = S_2

$S_2S_1S_0$ = b111 = 7

What does "7" mean?

7	6	5	4	3	2	1
D_3	D_2	D_1	P_2	D_0	P_1	P_0
0	1	0	0	0	1	1

Note: it does do 2-bit "detect" (since S3 S2 S1 does not code 0), but it does not let us know that we can't correct ...

"Correcting" this bit makes things worse! Thus, this code corrects "single" bits only.

CS 152 L19: Error Correcting Codes

UC Regents Spring 2005 © UCB

This slide shows 3 parity bits (P_o, P_1, P_2) protecting 4 data bits (D_o, D_1, D_2, D_3), permitting a single bit error to be corrected. However, as this slide also shows, if two bit errors occur, the attempt at correction will not work.

In this problem, we extend the code by adding a fourth parity bit, U, computed as:

$$U = P_o \text{ xor } P_1 \text{ xor } P_2 \text{ xor } D_o \text{ xor } D_1 \text{ xor } D_2 \text{ xor } D_3$$

In this problem, we assume that the U bit cannot be in error (what is written for U will always be read back correctly). Problem parts appear on the next page.

Question 3a. (10 points total) We can use the U bit to catch some (but not all) of the cases where the ECC fails because too many bits in the word flipped. We catch these cases by checking if the $S_2 S_1 S_o$ value is consistent with V, where

$$V = U \text{ xor } P_o \text{ xor } P_1 \text{ xor } P_2 \text{ xor } D_o \text{ xor } D_1 \text{ xor } D_2 \text{ xor } D_3$$

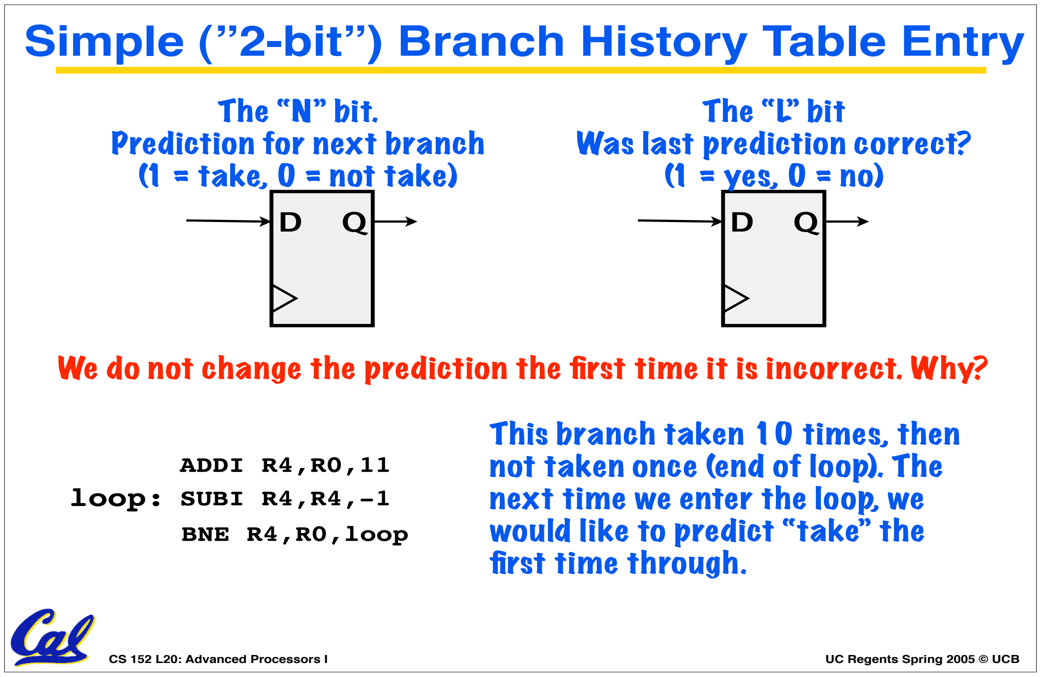
If $V = 1$, what subset of the 8 values possible for $S_2 S_1 S_o$ would be consistent (5 points)?

If $V = 0$, what subset of values of $S_2 S_1 S_0$ would be consistent (5 points)?

Question 3b (8 points). Given the algorithm implied by Question 1a, we wish to add a Halt (H) logic signal to the ECC readout. The H signal is set to 1 if the $S_2 S_1 S_o$ values cannot be trusted, based on the Question 1a algorithm. Write the logic equation for H. H is a function of S_o, S_1, S_2 and V.

4 Branch Prediction (15 points)

Below, we show a slide from the Advanced Processors lecture.



Simple ("2-bit") Branch History Table Entry

The "N" bit.
Prediction for next branch
(1 = take, 0 = not take)

The "L" bit
Was last prediction correct?
(1 = yes, 0 = no)

We do not change the prediction the first time it is incorrect. Why?

```
       ADDI R4,R0,11
loop:  SUBI R4,R4,-1
       BNE R4,R0,loop
```

This branch taken 10 times, then not taken once (end of loop). The next time we enter the loop, we would like to predict "take" the first time through.

CS 152 L20: Advanced Processors I

UC Regents Spring 2005 © UCB

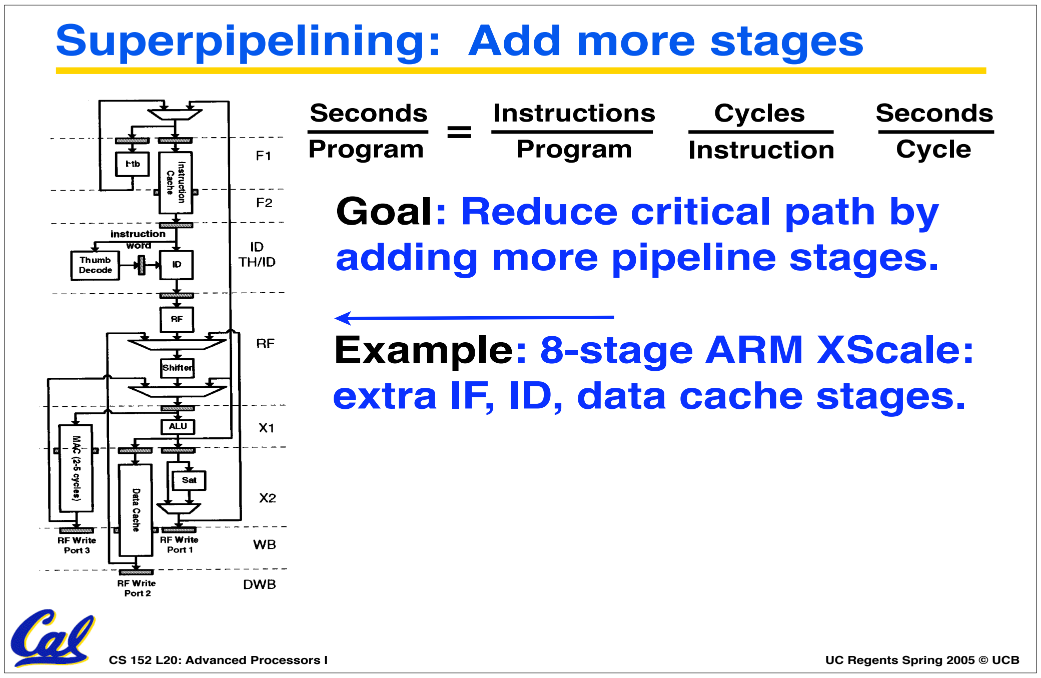
This slide shows the two bits (N, L) of a simple branch predictor. Each entry in the Branch History Table contains these two bits. When fetching a branch, the Branch Predictor uses these bits to predict "Taken" or "Not Taken". Once the processor knows whether the prediction was indeed accurate, it updates the N, L bits for the branch.

Question 4a. (3 points) Write the logic equation for the "Taken" signal during a branch fetch, for the two-bit predictor. This equation is a function of N and/or L.

Question 4b. (12 points) Draw a state machine that shows how the two branch predictor bits are updated after the processor knows if the branch was indeed taken or not. This state machine should have four states, corresponding to the 4 possible values of N and L. State transitions occur after the fate of a branch is known; thus arcs from one state to another are labeled with "$taken == 0$" or "$taken == 1$". Be sure to label every state and every arc, and show which state the machine resets to on powerup. The reset state should be chosen so that the first BNE execution for the loop shown on the slide will be predicted correctly.

5 Superpipelining (16 points)

Below, we show a slide from lecture, to refresh your memory about superpipelining. Note that no arithmetic is necessary to deduce the answers in this section.


F1
F2
ID
TH/ID
RF
X1
X2
WB
DWB
Itb
Instruction Cache
instruction word
Thumb Decode
ID
RF
Shifter
ALU
MAC (2-5 cycles)
Data Cache
Sat
RF Write Port 3
RF Write Port 1
RF Write Port 2

Superpipelining: Add more stages

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \quad \frac{\text{Cycles}}{\text{Instruction}} \quad \frac{\text{Seconds}}{\text{Cycle}}$$

Goal: Reduce critical path by adding more pipeline stages.

Example: 8-stage ARM XScale: extra IF, ID, data cache stages.

CS 152 L20: Advanced Processors I

UC Regents Spring 2005 © UCB

A CPU design team begins with a 5-stage pipelined MIPS (IF:ID:EX:MEM:WR), and improves it to be an 8-stage pipeline, by adding one pipeline stage to split the IF, EX, and MEM stages. The branch decision occurs in the second EX stage. No dynamic branch predictor was added; the architecture is optimized on the assumption that branches are always taken.

Unfortunately, Berkeley engineers were not hired to do the circuit design, and the clock rate of the 8-stage design was exactly the same as the 5-stage design!

Question 5a (4 points). A program has an instruction mix of 40% ALU, 30% memory ops, and 30% branches. The branches are not taken 25% of the time. **True or False?** For this program, the 8-stage design will be just as fast (in seconds/program CPU time terms) as the 5-stage design (no faster, no slower). If you answer False, state if the 8-stage design will be faster or slower.

Questions 5b-d (12 points total).

The circuit designers were promoted to upper management (welcome to the real world), and Cal engineers took their place (the Cal hires did get a good stock option package, though).

The 8-stage pipelined was redesigned, and the clock rate is now 30% faster than the original 5-stage design. The number of clock cycles it takes for an L1 or L2 cache hit is unchanged from the 5-stage design. A good branch predictor was also added to the design (for the questions below, assume it is perfect for the programs under test). However, the DRAM memory system was not changed, and so the number of nanoseconds a DRAM read access takes is unchanged.

Questions 5b (4 points). True or False? For programs where essentially all LWs hit the cache, and where the branch predictor works very well, the cycles per instruction (CPI) for LW instructions for the 8-stage pipeline is lower (i.e. better CPI) than for the 5-stage pipeline.

Questions 5c (4 points). True or False? For programs that do not have good locality, and thus many LWs require DRAM access, the CPI for LW instructions for the 8-stage pipeline is higher (i.e. worse CPI) than for the 5-stage pipeline.

Questions 5d (4 points). A program has 20% ALU, 80% memory ops, 0% branches. The memory operations do not have good locality, and thus many LWs require DRAM accesses.
True or False? The performance of this program (in seconds/program CPU time terms) will be significantly worse (more seconds/program) for the 8-stage pipeline than for the 5-stage pipeline.