

Name _____

Computer Architecture

EE 4720

Final Examination

Primary: 6 December 1999, 10:00–12:00 CST
Alternate: 7 December 1999, 15:00–17:00 CST

Problem 1 _____ (25 pts)

Problem 2 _____ (25 pts)

Problem 3 _____ (25 pts)

Problem 4 _____ (25 pts)

Alias _____

Exam Total _____ (100 pts)

Good Luck!

Problem 1: The code in the table on the next page executes on a dynamically scheduled machine using reorder buffer entry numbers to rename registers. The implementation performs branch and branch target prediction.

There are an unlimited number of reorder buffer entries, reservation stations, and functional units. Integer instructions use the EX functional unit, branches use the B unit, and loads and stores use the load/store unit, consisting of segments L1 and L2.

When the code in the table starts to execute all register values are available, as shown in the tables on the next page.

The `lw` instruction will suffer a miss; it will finish L2 five cycles after entering L2, loading a 100.

The `bneq` instruction is predicted not taken but is, in fact, taken.

(a) For this subproblem the register file is not backed up when branches are encountered. Using the tables provided on the next page show a pipeline execution diagram for the code and the changes to the register map and register file at the end of each cycle. Do not show reservation station numbers or reorder buffer entries in the pipeline execution diagram itself. The entry number for the next available reorder buffer entry is 1. Show when instructions commit or when they are squashed. (15 pts)

(b) Explain how execution would be different if the register file were backed up when branches are encountered. A second diagram is not necessary, just show where execution differs and how it differs. (5 pts)

There is one more part.

Problem 1, continued:

Pipeline Execution Diagram																	
Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw r4, 0(r5) add r1, r2, r3 or r2, r1, r3 bneq r4, SKIP sub r1, r2, r5 SKIP: add r2, r1, r2	IF																

Register Map																	
Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Arch. Reg.	Val. or ROB#																
r1	10																
r2	20																
r3	30																
r4	40																
r5	50																

Register File																	
Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Arch. Reg.	Val.																
r1	10																
r2	20																
r3	30																
r4	40																
r5	50																

Problem 1, continued:

(c) The DLX code in the table below executes on the dynamically scheduled machine described above. The machine uses a load/store queue and a nonblocking (lockup free) cache as described in class. For this part the cache miss latency is lower: If an instruction in L2 encounters a cache miss it returns to its reservation station for three cycles then returns to L2.

Show the execution of the code in the table below. Show when instructions commit but **do not** show reservation station or reorder buffer entry numbers.

The instructions at lines Line1 and Line3 miss the cache. The contents of each register is different. (5 pts)

Pipeline Execution Diagram															
Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Line1: sw 0(r1), r2	WF														
Line2: lw r3, 0(r2)															
Line3: lw r1, 0(r4)															
Line4: sw 0(r1), r2															
Line5: lw r5, 0(r2)															

Don't forget part (b)!!!!

This problem consists of four parts. For full credit do parts (a), (b), and (c) only. For reduced credit do parts (a) and (d). If you have time but lack confidence do all parts, the grade will be $\max\{a + b + c, a + d\}$.

Problem 2: The code below is run on systems having a 32-bit address space, 1-byte characters, and using 256-kibibyte (2^{18} -byte) caches as described below. Before the code is run the cache is cold (empty). Only consider accesses to the array, **a**.

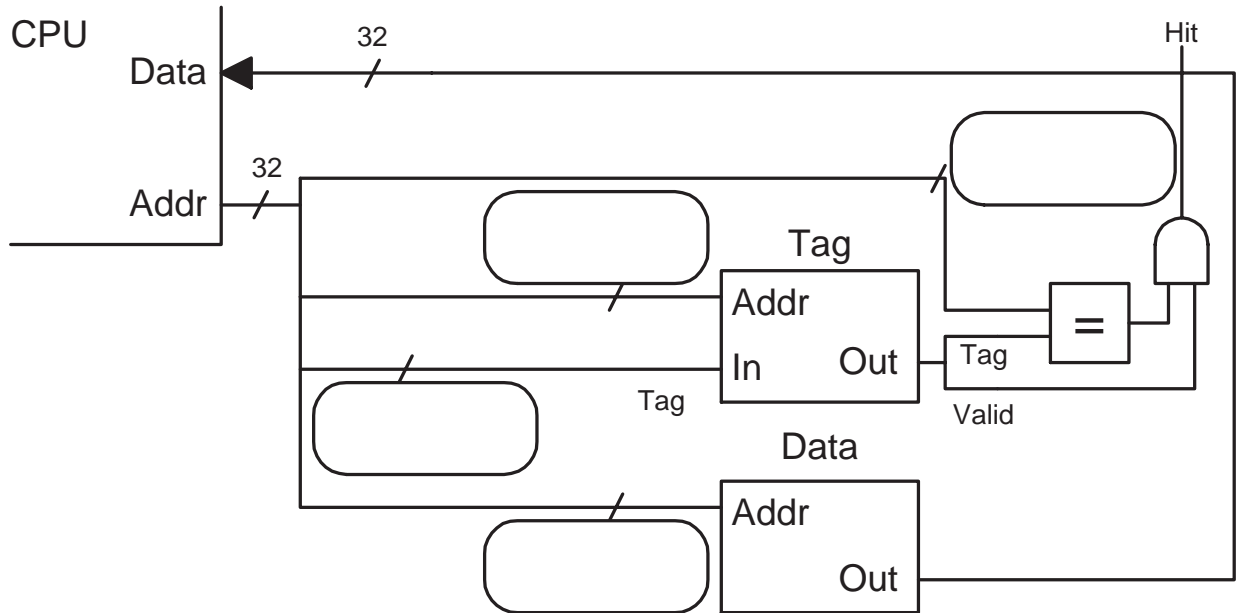
```
// sizeof(int) = 4 characters
int *a = 0x1000000; // Storage allocated elsewhere.
for(x=0; x<4; x++)
    for(i=0; i<512; i++)
        for(j=0; j<8; j++)
            sum += a[ i * 1024 + j ];
```

(a) Find the hit ratio encountered executing the code above on a direct-mapped 2^{18} -byte cache with a line size of 8 characters. How much of the cache is filled? (9 pts)

(b) Suppose the associativity of the cache could be increased while fixing the cache capacity at 2^{18} -byte and the line size at 8. What would the hit ratio be if the associativity were 2? What is the smallest associativity needed to achieve the maximum hit ratio on the code above? Is that associativity practical? Explain. (8 pts)

Problem 2, continued:

(c) Suppose **any** address bits could be used to form the index (set number, address used in the tag store) and any line size could be used. (Lines must still be contiguous.) In the diagram below show which address bits should be used (using the rounded boxes) to maximize the hit ratio of the code above. The capacity of the cache must still be 2^{18} -byte. (8 pts)



(d) **Optional:** Only solve this part if you cannot solve parts (b) and (c). Total credit will be lower.

This part is unrelated to the previous parts. Show how $2^{34} \times 32$ b memory devices should be connected to implement a 40-bit address space on a system with four-bit (one-nibble) characters and a bus width of 64 bits. Include the alignment network. (8 pts)

Problem 3: The code below is compiled and run on two machines, one using a one-level branch predictor and the other using a two-level gselect branch predictor. Both predictors use a 4096-entry BHT, the gselect predictor uses a 3-branch global history. Each entry holds a 2-bit saturating counter.

```
for(i=0; i<100000; i++)
{
Line1:  if( a == 1 ) aa++;
Line2:  if( b == 1 ) bb++;
Line3:  if( C == 1 ) cc++;
Line4:  if( i & 0x2 ) { x++; } /* N N T T N N T T N N T T N N T T ... */
Line5:  if( i & 0x4 ) { y++; } /* N N N N T T T T N N N N T T T T ... */
Line6:  if( i & 0x2 ) { z++; }
}
```

(a) The ISA has a 32-bit address space, all instructions are 32 bits (four characters) and must be aligned. How is the address for the BHT in the gselect predictor obtained? Be sure to specify bit positions. (9 pts)

(b) Assume that exactly one branch instruction is generated for each `if` statement and that the compiler does no optimizing. What is the prediction accuracy for each of the last three `if` statements using the one-level predictor after a large number of iterations? (8 pts)

(c) As above, assume that exactly one branch instruction is generated for each `if` statement and that the compiler does not do any optimizing. What is the prediction accuracy for each of the last three `if` statements using the gshare predictor after a large number of iterations? *Hint: The solution to this part does not require tedious computation or the construction of lengthy tables.* (8 pts)

Problem 4: Answer each question below.

(a) Synthetic instruction `clr 12(r2)` writes a zero to the memory location at address $12 + r2$. How could it be added to DLX? (5 pts)

(b) What would be the disadvantage of a RISC ISA that had 1048576 (2^{20}) integer (general-purpose) registers? (5 pts)

(c) Why would it be inappropriate to add a memory indirect load instruction to DLX. (For example, `lw r1,@(r2)`.) Justify your answer for the statically scheduled DLX implementation. Weigh the complexity of changes needed to the implementation against expected benefit over a software-only solution. Be brief. (5 pts)

(d) The code below runs on a dynamically scheduled 4-way superscalar machine with perfect branch target prediction and a cache that never misses. There are an unlimited number of reorder buffer entries, reservation stations, and functional units. This machine is not 100% perfect: its fetch mechanism is the type described in class. What is the minimum and maximum CPI executing the code below for a large number of iterations? Explain the conditions under which minimum and maximum CPI are encountered. (5 pts)

LOOP:

```
lb  r1, 0(r2)
addi r2, r2, #1
bneq r1, LOOP
```

(e) What is the difference between a write-through cache and a write-back cache? Which one needs a dirty bit and why? (5 pts)