EECS 470 *Midterm Exam* Answers

Fall 2009

Name: _____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

#	Points
Page 2	/18
Page 3	/15
Page 4	/8
Page 5	/15
Page 6	/11
Page 7	/13
Page 8 & 9	/20
Total	***/100***

NOTES:

- Open book and Open notes
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam.
- There are ***9*** pages including this one.
- **Be sure to show work and explain what you've done when asked to do so.**
- **The last page has two "answer areas". Clearly mark which one you want graded or we will grade the first one.**

1. Multiple choice/fill in the blank. *Select the best answer*.
   [18 points, -2 per wrong or blank answer, minimum 0]

   a. In the algorithm we are calling T3, when using an RRAT we will free ***all PRF entries / no PRF entries / those PRF entries which aren't pointed to by the RRAT / those PRF entries in the RAT that are overwritten by the RRAT*** when a branch mispredict occurs.

   b. If, in the algorithm we are calling T2, the RAT had only one port for writing values, you could only have one instruction that writes a register ***issue / start execution / complete execution / retire*** per cycle.

   c. If, in the algorithm we are calling T3, the PRF had only one write port, you could only have one instruction that writes a register ***issue / start execution / complete execution / retire*** per cycle.

   d. In the algorithm we are calling T1, the RAT points to ***a reorder buffer entry / a reservation station / a physical register / an execution unit***.

   e. Say that a certain wire is 1 mm long, 50nm wide and 60nm high in one process. We then convert to a different (smaller) process and find that our wire is now 500µm long, 40nm wide and 60nm high. *If* the capacitance of the wire were the same in both cases, we'd expect the new wire to have ***more / less / almost the same*** delay as the old wire. (For the VLSI folks: assume no skin or other effects, we're just using a simple RC model for delay.)

   f. The branch target buffer is a(n) ***address / confidence / direction*** predictor that has particular problems with ***function calls / conditional branches / returns from functions / indirect branches****. To help with that case we might add a(n) ***Translation Lookaside Buffer (TLB) / Reorder Buffer (RoB) / Alias Table List Address Selector (ATLAS) / Return Address Stack (RAS)***.

   g. In the algorithm we are calling T3, if you have 32 RoB entries, 16 architected registers, 8 RS entries and a 4KB instruction-cache, you will not have a use for more than ***8 / 16 / 32 / 48 / 64*** PRF entries.

   h. Given a 4-KB four-way associative cache with 16-byte cache lines and a 32-bit address space there will be ___**6**___ bits used for the index. If that same cache were direct-mapped you'd need ___**20**___ bits to be used for the tag.

- (For 1f "indirect branches" is also acceptable)

2. In the algorithm we're calling T2, explain exactly when the RAT is updated by a committing instruction. **[4]**

   It is updated if the current RAT value for the associated Architected Register points to the ROB entry of the committing instruction.

   (Also could argue that when a mispredicted branch commits it will update the RAT)

3. In the algorithm we're calling T1, explain exactly when the ARF is updated. **[4]**

   It is updated if the current RAT value for the associated Architected Register points to the RS entry of the committing instruction.

4. We are working with a non-pipelined, in-order machine. 10% of all instructions executed are divide instructions. All non-divide instructions take one cycle. All divide instructions take 50 cycles. **[7]**
   a. What is the CPI of this program on this processor? Show your work.

      .9*1 (non-divide) +.1*5 (divide) = 5.9

   b. What percent of *time* is spent just doing divides? Show your work.

      5/5.9= 84.7%

   c. What would the *speedup* be if we sped up divide by a factor of 2? Show your work.

      Speedup=1/((1-f)+f/s))
      Speedup=1/((1-.847)+.847/2) = 1.735

5. Identify two problems with the following code that could/will result in "compiler" errors or potential synthesis issues. **[8]**

```
`define WIDTH 4
module updown_modulo (
clk       ,   // Clock input
reset     ,   // Reset input
enable    ,   // Enable input
up_down   ,   // Up Down input
count         // Count output
);

 input clk;
 input reset;
 input enable;
 input up_down;

 output [`WIDTH-1 : 0] count;

 reg [`WIDTH-1 : 0] next_count;

 always @*
 begin
    if(up_down && enable)
        next_count=count+1;
    else if (!up_down && enable)
        next_count=count-1;
 end

 always @(posedge clk)
 begin
    if (reset)
       count <= #1 `WIDTH'b0;
    else
       count <= #1 next_count;
 end
endmodule
```

Problem #1: **count needs to be declared as a reg.**

Problem #2: **next_count isn't assigned on *all* paths. (this will simulate correctly but likely won't synthesize correctly…)**

6. Consider the following programs:

Program A	**Program B**	**Program C**
R1=MEM[R2+0]	R1=MEM[R2+0]	R1=MEM[R2+0]
R0=R1+4	R2=R2+R2	R2=R3+4
MEM[R2+4]=R2	MEM[R2+4]=R1	R3=R0+R3
R2=R3+4	R0=R1+1	R1=R1+R1
MEM[R2+4]=R2	R3=R3+R2	R0=R0+R0
R3=R3+1	R2=R2+R0	R2=R2+1
R0=R1+R2	R2=R2+R1	R3=R0+19
R1=R1+4	MEM[R2+4]=R2	R0=R1+3

Say we have an out-of-order machine (using Tomasulo's III) with 3 RSs, 6 ROB entries, 4 architected registers, and 9 PRF entries. Further, say the first load stalls at the head of the RoB for a **long** time. All three programs will come to a halt waiting for the load to finish. For each program ***circle*** the last instruction that will be placed in the ROB previous to the load finishing. You are to assume an instruction *stays* in its RS until it completes execution. You **must clearly** show/explain your work to receive credit. Specifically be sure to indicate why no other instruction can be fetched. **[15]**

In Program A the RoB fills up first. At that point the RS only has the first 2 instructions left in it and the PRF has only allocated 4 spaces (no destinations for the stores).

In Program B that instruction will fill up the RS. The first, 3rd and 4th need an RS.

Note: One could argue that the store would NOT take up an RS entry as the RS might only be used for address calculation and the store value might be handled in the load/store queue. If you *clearly* made that argument and argued for the final instruction being R2=R2+R0 that's also acceptable. But it needs to be really really clear.

In Program C we have no RS problem (only the load and R1=R1+R1 is in the RS) but the PRF fills up before the RoB does.

7. Consider the pipelined processor you did as part of project 3 *but with the structural hazards removed* (so you can fetch and load/store in the same cycle). For a given program it is the case that:
    - 35% of all instructions executed are loads,
    - 20% of all instructions executed are branches
        - 75% of the branches are conditional and PC-relative
        - The other 25% are unconditional and indirect
        - Branches are taken 60% of the time (total)
    - 40% of all instructions are dependent on the instruction in front of them.

   Assume the program is quite long (millions and millions of instructions).

    a. What would you expect the CPI to be for this program on this processor? Show your work. **[3]**

    CPI=1+branch penalty + data hazard penalty.
    CPI=1+(.2*.60*3)+(.35*.4*1) = 1.5

    b. Say that in an attempt to improve performance of the in-order machine you modify the program so you resolve branches in the decode stage.

        i. Accounting only for the changes in the control hazards, what will the CPI now be? **[2]**

        CPI=1+(.2*.60*1)+(.35*.4*1) = 1.26

        ii. This change also introduces new data hazards. Carefully explain what hazards are introduced and how they would need to be resolved. Estimate the new CPI including the impact it has on control and data hazards. Clearly show and explain your work. **[6]**

Because the load is resolving in decode, we need to have data ready for it by that point. Both the conditional and indirect branches can have data hazards (one for the condition, one for the location to branch to).
We are going to be adding a stall for each branch which is preceded by a dependent instruction. Furthermore, if we are dependent on a load in front of us we'll stall for 2 cycles rather than one. Lastly, we also have to account for those branches which are dependent on a load 2 cycles ahead of them but not dependent on the instruction immediately in front of them. We've got no numbers for that, so we'll just say that some fraction, Y, of all branches are data dependent on the instruction 2 in front of them and not data dependent on the instruction immediately in front of them.

CPI=1+branch penalty+ data hazard penalty.

- Branch penalty=.2*.6*1 or .12.
- Data hazard penalty= (A) load to EX stalls of non branches + (B) stalls due to branches dependent on non-load in front of them + (C) stalls due to branches dependent on loads in front of them + (D) stalls due to branches dependent two in front of them that aren't included in B or C. + (E) stalls due to branch squashes.

(.8*.35*.4*1)+(.2*.65*1*.4)+(.2*.35*2*.4) + (.2*.35*Y)+(.2*.6*1)=1.34+.07Y

8. Consider a local history predictor where the Branch History Table has 32 entries each 2 bits in length and the Pattern History Table consists of a 1-bit predictor. Neither is tagged. **[13]**
    a. How many bits of memory are used in the BHT? The PHT? **[3]**

    64, 4

    b. Say that a single branch is being taken 3 times in a row and then is not taken 3 times in a row and repeats this pattern forever (T,T,T,N,N,N,T,T,T,N,N,N, etc.) Assuming there is no aliasing of any kind, what will be the steady-state prediction rate for this branch? Justify your answer **[3]**

    That pattern results in TT twice, the first time the actual result is taken, the second time it's not taken. Same with NN. So those cases are all wrong. That leaves us with NT and TN both of which only happen once per 6 branches and so are right. 33.3% prediction rate.

    c. Say that a single branch is being taken 4 times in a row and then is not taken 4 times in a row and repeats this pattern forever. Assuming there is no aliasing of any kind, what will be the steady-state prediction rate for this branch? Justify your answer **[3]**

    Now we get TT three times, the first two the actual result is taken, the last time not taken. So we get the 2nd of those right. Same with NN. NT and TN again each appear once. So 50%.

    d. Identify a pattern of taken and not taken branches that when repeated forever will result in a 0% prediction rate (again, assuming no aliasing of any kind). Your answer should be of a form (T,N,T). **[4]**

    (N,N,N,T,T,T,N,T)

9. Consider the following state of a machine implementing what we've called Tomasulo's third algorithm, where we are using an RRAT rather than back-pointers.

RAT

Arch Reg #	Phy. Reg #
0	~~4~~ 3
1	~~8~~7
2	~~95~~2
3	1
4	0

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
0	~~0~~	~~NY~~	~~6~~	~~1~~	← HEAD
1	~~4~~	~~Y~~	~~5~~	~~2~~	
2	~~8~~	~~NY~~	~~7~~	~~1~~	
3	~~12~~	~~Y~~	—	—	
4	~~16~~	~~Y~~	~~9~~	~~2~~	
5	~~20~~	~~N~~	~~8~~	~~1~~	
6	40	Y	2	2	← TAIL
7	44	N	3	0	
8					

RRAT

Arch Reg #	Phy. Reg #
0	4
1	~~36~~7
2	~~2~~5
3	1
4	0

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	~~Add~~	~~NY~~	~~6,4~~	~~NY~~	~~6,4~~	~~7~~	~~2~~
1	~~Add~~	~~N~~	~~7~~	~~Y~~	~~9~~	~~8~~	~~5~~
2	~~Add~~	~~Y~~	~~3~~	~~Y~~	~~1~~	~~6~~	~~0~~
3	Add	Y	8	Y	10	3	7
4							

PRF

Phy Reg #	Value	Free	Valid
0	6	N	Y
1	5	N	Y
2	~~4,~~10	~~N~~YN	~~YN~~Y
3	3	~~NY~~N	~~Y~~N
4	2	N	Y
5	9	N	Y
6	~~11,4~~	~~N~~Y	~~NY~~N
7	~~1,~~8	N	~~N~~Y
8	0	~~N~~Y	N
9	7	~~N~~Y	~~Y~~N
10	12	Y	N
11	5	Y	N
12	-1	Y	N

KEY:

- **Op1 PRN/value** is the value of the first argument if "Op1 ready?" is yes; otherwise it is the Physical Register Number that is being waited upon.
- **Op2 PRN/value** is the same as above but for the second argument.
- **Dest. PRN** is the destination Physical Register Number.
- **Dest. ARN** is the destination Architectural Register Number.
- **ROB** is the associated ROB entry for this instruction.
- **Free/Valid** indicates if the PRF entry is currently available for allocation and if the valid in it is valid. A free entry should be marked as invalid.

Say that the instruction in ROB #3 is a branch and it was mis-predicted: the next PC should have been 40. Say that the instruction in memory location 40 is R2=R1+R0 and in 44 is R0=R1+R2. Update the machine to the state where the branch has left the RoB, and the instructions at memory 40 has issued and executed, but not committed while the one at location 44 has issued but not executed. When faced with an arbitrary decision, just be sure to make a legal choice. ***Be sure to update the head and tail pointers!*** **[20]**

On the following page is an extra copy of this state. You may use this one or the one on the next page but be sure to cross out (with a BIG X) the one you don't want graded.

Consider the following state of a machine implementing what we've called Tomasulo's third algorithm, where we are using an RRAT rather than back-pointers.

RAT

Arch Reg #	Phy. Reg #
0	4
1	8
2	9
3	1
4	0

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
0	0	N	6	1	← HEAD
1	4	Y	5	2	
2	8	N	7	1	
3	12	Y	--	--	
4	16	Y	9	2	
5	20	N	8	1	
6					← TAIL
7					
8					

RRAT

Arch Reg #	Phy. Reg #
0	4
1	3
2	2
3	1
4	0

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	Add	N	6	N	6	7	2
1	Add	N	7	Y	9	8	5
2	Add	Y	3	Y	1	6	0
3							
4							

PRF

Phy Reg #	Value	Free	Valid
0	6	N	Y
1	5	N	Y
2	4	N	Y
3	3	N	Y
4	2	N	Y
5	9	N	Y
6	11	N	N
7	1	N	N
8	0	N	N
9	7	N	Y
10	12	Y	N
11	5	Y	N
12	-1	Y	N

KEY:

- **Op1 PRN/value** is the value of the first argument if "Op1 ready?" is yes; otherwise it is the Physical Register Number that is being waited upon.
- **Op2 PRN/value** is the same as above but for the second argument.
- **Dest. PRN** is the destination Physical Register Number.
- **Dest. ARN** is the destination Architectural Register Number.
- **ROB** is the associated ROB entry for this instruction.
- **Free/Valid** indicates if the PRF entry is currently available for allocation and if the valid in it is valid. A free entry should be marked as invalid.

Say that the instruction in ROB #3 is a branch and it was mis-predicted: the next PC should have been 40. Say that the instruction in memory location 40 is R2=R1+R0 and in 44 is R0=R1+R2. Update the machine to the state where the branch has left the RoB, and the instructions at memory 40 has issued and executed, but not committed while the one at location 44 has issued but not executed. When faced with an arbitrary decision, just be sure to make a legal choice. ***Be sure to update the head and tail pointers!*** **[20]**

You may use this one or the one on the previous page but be sure to cross out (with a BIG X) the one you don't want graded.