EECS 470 *Midterm Exam*

Winter 2008 – answers

Name: _____KEY_____ unique name: KEY

Sign the honor code:

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

_____

Scores:

#Page	Points
2	**/10**
3	**/12**
4	**/15**
5	**/13**
6	**/17**
7	**/13**
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 is a 3 point extra credit problem on page 6. It is fairly hard, so don't even start it until you are done with everything else and have checked it over.
- There are ***9*** pages including this one.
- Be sure to show work and explain what you've done when asked to do so.
- **There are two "answer areas" for problem 8 (pages 8&9). Clearly mark which one you want graded or we will grade the first one.**

1. **Written short answer (Your answers must be one or two sentences.) [10 points]**
   a. In the algorithm we're calling T3 (using an RRAT) explain exactly when a PRF entry is freed assuming there is no mis-speculation. **[3]**

      The PRF entry is freed when a retiring instruction overwrites that PRF entry in the RRAT.

   b. In the algorithm we're calling T3 (using an RRAT) explain exactly when a PRF entry is freed by a mis-predicted branch. **[3]**

      When the branch mispredicts, all values not in the RRAT are freed.

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

      When an instruction completes execution it will update the ARF if (and only if) the RAT is currently pointing to the RS entry of the retiring instruction.

2. **Short answer (*Clearly* show your work). [12 points]**

a. If you have a 32-bit address space (addresses are 32 bits) and you have a 4-way associative cache that uses 5 bits as the set index and 4 bits as the byte offset, how large is the data portion of the cache (in bytes)? **[4]**

There are 2^5 sets, and 4 lines per set. Each line is 2^4 bytes in size. So (2^5)*(2^2)*(2^4)=2^11 or 2048 bytes.

b. In T2 if you have 16 architected registers, 8 Reservation stations and 32 Reorder Buffer entries, how many bits of memory will you need to store the rename table? **[3]**

$16*\log_2(32)=80$ bits

c. 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). Say 20% of all instructions were loads, 30% were branches, and 10% were stores. Assume the program is quite long (millions and millions of instructions). In addition, 25% of all instructions are data dependent on the instruction in front of them, and branches are taken 25% of the time. What would you expect the CPI to be? **[5]**

1 + .3 * .25 * 3 + .2*.25*1 = 1.275

3. **Mr. Amdahl (*Clearly* show your work) [7 points]**
   a. Consider a workload where 50% of the execution time consists of multimedia processing for which a multi-media instruction set extension might be helpful. According to Amdahl's law, what is the maximum speedup that can be achieved by implementing these extensions? **[3]**

      If you speed up the 50% by an infinite amount, you get 2x performance.

   b. Now, say that you work at AMD and the designers there claim that multimedia code sequences will see a 3.5 times (3.5X) speedup by using special multi-media extensions. What is the fraction of the execution time that must be multi-media code in order to achieve an overall speedup of 1.5X? **[4]**

      1/((N/3.5)+1-N)=1.5. Solve for N. You get N=.46666 or 7/15.

4. **Physical Register File [8 points]**
   a. Consider the algorithm we've named "T3". What is the size of the physical register file you should use if you want to *guarantee* there will never be a stall because the free list for the physical register file is empty? Assume that there are 32 architected registers, 48 RoB entries, 12 reservation stations, and that we are using an RRAT. Clearly *explain* your work. **[4]**

      32+48=80.

   b. Explain why you should probably not make your PRF as big as your calculation above would suggest. **[4]**

      Lots of possible arguments here, but there are two parts that are important.
      #1 – We don't need that many PRF entries. Because of instructions that don't need PRF entries (stores, most branches) we don't need all those PRF entries.
      #2 – There is a cost to having too many entries. This could be a speed cost or a cost due to too many bits (we would need one less bit if we had 64 PRF entries for example.

      Skipping #1 is a problem; otherwise you'd think that the savings would come at the cost of not being able to use all of the RoB (which would mean you shouldn't have them!)

5. **CMOV [13 points]**

a. Consider the following pseudo-assembly code segment that computes the absolute value of a number in R1 and puts it into R2.

```
        If(R1>0) goto DONE
        R1=(-1)*R1
DONE:   R2=R1
```

Re-write this code using a conditional move so that does the same thing but without using a branch. Your conditional move instruction should be of the form: `If(Rx==0) Ry=Rz` where the "==" can be replaced by ">" or "<" as you desire (but the 0 must be a 0) and the registers can be specified to be any given register. After this code segment, all registers other than R1 are potentially live (meaning they may be read before they are written to). Your solution should be no more than 6 lines of assembly. **[5]**

There's a lot of ways to do this. One is:
R2=(-1*R1)
IF(R1>0) R2=R1.

b. Now re-write the following code using a conditional move to again eliminate a branch while preserving functionality (including faults). You must follow the restrictions described above on the form of CMOV you can use.
In this case, all registers other than R3, R5, R6 and R7 are potentially live after this code segment. Your solution should have no more than 12 lines of assembly for full credit. **[8]**

```
        R4=0
TOP:    R3=R3+1
        If(R4>2000) goto BOT
        R5=R2/R3
        R4=R4+R5
BOT:    IF(R3<50) goto TOP
```

Maintaining the exact same fault behavior is hard. If the above code faults, this should. If it doesn't this shouldn't!

```
        R4=0
TOP:    R3=R3+1
        R6=R4-2000          // if R6>0 should skip
        R7=1
        IF(R6>0) R3=R7      // R7!=0
        R5=R2/R3
        R7=R6+R5
        R6=R4-2001          // Need to compute "should do" case. ☹
        IF(R6<0) R4=R7
        IF(R3<50) goto TOP.
```

6. **Predictors [17 points]**

Consider the following types of predictors:

A) A PC indexed branch-history table with eight 1-bit entries
B) A PC indexed branch-history table with four 2-bit entries (done as a standard counter)
C) A local history predictor where the branch-history table has eight 3-bit entries and points to a pattern-history table that has 1-bit entries.

a. *Assuming the entire program had only one branch,* what would be the expected hit rate of each of these branch predictors on a branch that has a repeating pattern of NNNNT (it does that pattern forever, so NNNNTNNNNT etc.).
[5 points, -2 for each wrong or blank answer, min 0]

A) 60%	B) 80%	C) 60%

b. Now consider the following code segment that is 5 assembly instructions in length:

```
X:    If(R1>0) goto BRANCH XY
      ALU or MEMORY OP
Y:    if(R1%2==0) goto Z
      ALU or MEMORY OP
Z:    if(R1!=1000000) goto X
```

The branch at X is taken 50% of the time (coin-flip each time). The branch at Y follows the pattern NNNNT repeating forever and the branch at Z is always taken. For each given predictor, compute the expected rate of *correctly* predicting the branch. (The one with an "X" is extra credit, if you get it right *and* have a correct detailed explanation you get +3 points. It's harder than it's worth so don't waste time on it until you have everything else done and checked over.)
[12 points, -2 per box wrong or blank, min 0]

	A)	**B)**	**C)**
Branch X	50%	50%	50%
Branch Y	60%	80%	
Branch Z	100%	100%	93.5%

Extra credit (with detailed explanation)**[+3]**.

7. **Performance of Tomasulo's [13 points]**

Consider the following pseudo-assembly which *loops forever*:

```
TOP:  R1=MEM[R2]
      R2=R2+4
      R3=R3+1
      R4=R1+R4
      R4=R4-2
      R5=R1+R5
      R5=R5/R3
      If(R5<100000) goto TOP
```

Say you have a machine which can issue one instruction per cycle, finish execution of one instruction per cycle, and retire one instruction per cycle. Branch mis-predictions are resolved when the branch hits the head of the RoB *and nothing is done about mis-predicted branches before that.* This machine implements what we have called "Tomasulo's 3", has a RS size of 3 and a RoB size of 5. Loads take 10 cycles to execute and all other instructions take 1 cycle to execute. There is no value prediction or other ways to break RAW hazards, but you can assume perfect branch prediction. Instructions don't release an RS until they finish execution.

a. What is the best CPI that could be achieved? To receive credit you must clearly explain your answer and any assumptions made. **[6]**

"Simple" argument: RS limit isn't actually important (just barely). RoB hits size 5 and stays full the whole time. Thus the load is in the RoB for 5 cycles before it causes a stall. So 5 stalls per 8 instructions or 13 cycles/8 instructions.

Instruction	Issue	EX done	Complete
TOP: R1=MEM[R2]	1	11	11
R2=R2+4	2	3	12
R3=R3+1	3	4	13
R4=R1+R4	4	12	14
R4=R4-2	5	13	15
R5=R1+R5	11	14	16
R5=R5/R3	12	15	17
If(R5<100000) goto TOP	13	16	18
TOP: R1=MEM[R2]	14	24	24

It will take 13 cycles from complete of one load to complete of the next. We assume that EX done means it is out of the RS that cycle and Complete means we are out of the RoB in that cycle (so new things can come in that cycle).

b. What is the minimum size of the RS and RoB to get this program to get a CPI of one (ignoring warm-up issues). To receive credit, you must clearly explain your answer and any assumptions made. **[7]**

A load would have to be "in system" for 10 cycles when it was at the head of the RoB. So the RoB must be at least size 10. In order to be able to issue instructions at will, you need enough RS entries. The worst case number of instructions in the RS over any 10 instructions is 8 (the two independent instructions will be out of the RS quite quickly.

Instruction	Issue	EX done	Complete
TOP: R1=MEM[R2]	1	11	11
R2=R2+4	2	3	12
R3=R3+1	3	4	13
R4=R1+R4	4	12	14
R4=R4-2	5	13	15
R5=R1+R5	6	14	16
R5=R5/R3	7	15	17
If(R5<100000) goto TOP	8	16	18
TOP: R1=MEM[R2]	9	19	19
R2=R2+4	10	17	20
R3=R3+1	11	18	21
R4=R1+R4	12	20	22
R4=R4-2	13	21	23
R5=R1+R5	14	22	24
R5=R5/R3	15	23	25
If(R5<100000) goto TOP	16	24	26
TOP: R1=MEM[R2]	17	27	27

8. Consider the following state of a machine implementing what we've called Tomasulo's third algorithm.

RAT

Arch Reg #	Phy. Reg #
0	7
1	~~9~~ 0
2	~~2~~ 1
3	3
4	~~8~~ 4

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
0	~~12~~	~~Y~~	~~5~~	~~1~~	~~← HEAD~~
1	~~16~~	~~N~~	~~6~~	~~1~~	
2	~~20~~	~~N~~	~~7~~	~~0~~	
3	~~24~~	~~Y~~	—	—	
4	~~28~~	~~N~~	~~8~~	~~4~~	
5	~~32~~	~~Y~~	~~9~~	~~1~~	~~← TAIL~~
6	100	N	0	1	← HEAD
7	104	N	1	2	← TAIL
8					

RRAT

Arch Reg #	Phy. Reg #
0	~~0~~ 7
1	~~1~~ 6
2	2
3	3
4	4

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	~~ADD~~	~~N~~	~~6~~	~~Y~~	~~8~~	~~7~~	~~2~~
1	~~LD~~	~~Y~~	~~13~~	~~Y~~	~~8~~	~~6~~	~~1~~
2	~~ADD~~	~~N~~	~~7~~	~~Y~~	~~6~~	~~8~~	~~4~~
3	ADD	Y	30	Y	8	0	6
4	ADD	Y	7	N	0	1	7

PRF

Phy Reg #	Value	Free	Valid
0	4	N	~~Y~~ N
1	5	N	~~Y~~ N
2	6	N	Y
3	7	N	Y
4	8	N	Y
5	13	~~N~~ Y	~~Y~~ N
6	~~10~~ 30	N	~~N~~ Y
7	~~12~~ 38	N	~~N~~ Y
8	13	~~N~~ Y	N
9	12	~~N~~ Y	~~Y~~ N
10	15	Y	N
11	16	Y	N
12	17	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 100. Also, assume the load in RS1 returns a value of 30. Now, say that the instruction in memory location 100 is R1=R1+R4 and in location 104 is R2=R3+R1. Update the machine to the state where the branch has left the RoB, and the instructions at location 100 and 104 have 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.

EXTRA COPY. BE SURE TO CROSS OUT EITHER THIS ONE OR THE ONE ON THE PREVIOUS PAGE.

RAT

Arch Reg #	Phy. Reg #
0	7
1	9
2	2
3	3
4	8

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
0	12	Y	5	1	← HEAD
1	16	N	6	1	
2	20	N	7	0	
3	24	Y	--	--	
4	28	N	8	4	
5	32	Y	9	1	← TAIL
6					
7					
8					

RRAT

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

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	ADD	N	6	Y	8	7	2
1	LD	Y	13	Y	8	6	1
2	ADD	N	7	Y	6	8	4
3							
4							

PRF

Phy Reg #	Value	Free	Valid
0	4	N	Y
1	5	N	Y
2	6	N	Y
3	7	N	Y
4	8	N	Y
5	13	N	Y
6	10	N	N
7	12	N	N
8	13	N	N
9	12	N	Y
10	15	Y	N
11	16	Y	N
12	17	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 100. Also, assume the load in RS1 returns a value of 30. Now, say that the instruction in memory location 100 is R1=R1+R4 and in location 104 is R2=R3+R1. Update the machine to the state where the branch has left the RoB, and the instructions at location 100 and 104 have 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!***