EECS 470 *Midterm Exam Answers*

Fall 2006

Name: _____KEY_____ 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	**/13**
Page 4	**/15**
Page 5	**/10**
Page 6	**/12**
Page 7	**/10**
Page 8-9	**/22**
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.
- *A few questions have limits on the number or words you can use. We will only grade your answer until you hit that limit.*
- **Be sure to show work and explain what you've done when asked to do so.**
- **The last page has two "answer areas" for the Part III question. Clearly mark which one you want graded or we will grade the first one.**

Part 1 – Short answer – 46 points

1. Fill-in-the-blank or circle the best answer **[18 points, -2 per wrong/blank, minimum 0]**

   a. When using the algorithm we've named Tomasulo's 3, if you have **N** reorder-buffer entries, **M** reservation stations and **K** architected registers you can be sure that you will not have a use for more than ***N+K*** **or N+M or M+K** physical registers.

   b. Given a 16-KB two-way associative cache with 32-byte cache lines and a 64-bit address space there will be __*8*__ bits used for the index. If that same cache were fully-associative you'd need __*0*___ bits to be used for the index.

   c. A cylindrical metal wire which is 1m long and as a diameter of 10nm will have the ***higher*** **or lower or the same** resistance as a wire made of the same material which is 2m long and has a diameter of 20nm.

   d. Adding more reservation stations will reduce the amount of time that the processor stalls due to ***structural hazards*** **or data hazards or control hazards** but may decrease the **number of branch delay slots or clock period or clock frequency**.

   e. In Tomasulo's 2 an instruction might find its data in one of three places. Those three places are ___*ROB*_____, ____*CDB*____, or _____*ARF*____. In T3 there are ***two*** **or three or four** places an instruction might find its data.

   f. In the paper "Combining Branch Predictors" the Gshare predictor involved __*XORing*__ the global history with the __*selected bits of the PC of the branch*__ If those two things were instead concatenated the predictor is called a "_*gselect*_____ predictor" in the paper.

2. Consider the pipelined processor you did as part of project 3. Say 20% of all instructions were loads, 25% were branches, and 20% were stores and that the program was quite long (millions and millions of instructions). 20% of all instructions are data dependent on the instruction in front of them, and branches are taken 75% of the time. What would you expect the CPI to be? Show your work. **[5]**

   for 100 instructions we get:

   4 Data hazard stalls: 20 (#loads)*.2(% dep)*1(stall cycles)
   56.25 Control hazard stalls: 25 (# branches)*.75 (%taken)*3(# squashed)
   40 #stalls due to structural hazard
   ====
   100.25.

(100+100.25)/100=2.0025

3. In T3 if you have 32 architected registers, 64 physical registers, 8 RS and 32 RoB entries, how many bits of memory will you need to store the rename table? *Clearly* show your work. **[4]**

   32 entries each $\log_2(64)$ bits = 32*6=192 bits.

4. Write a short program for a stack machine which computes (A+B)*(C+D). **[4]**

   Push A
   Push B
   Add
   Push C
   Push D
   Add
   Mult

5. Consider a branch predictor which uses a branch history table (BHT) and a pattern history table (PHT). Say we use 10 bits of the PC to select the BHT entry, our histories cover the last 8 branches and we use a 2-bit predictor. How many bits of storage will be required for the BHT? The PHT? *Clearly* show your work. **[4]**

   BHT has 2^{10} entries, each with 8 bits. 8192 bits
   PHT has 2^{8} entries each 2 bits wide. 512 bits.

6. Provide a taken/not-taken pattern consisting of exactly 4 branches where a two-bit saturating up-down predictor will do better than the predictor found on page 198 of the book. Assume both are initially strongly not-taken. Your answer should be of the form (T, NT, T, T).**[5]**

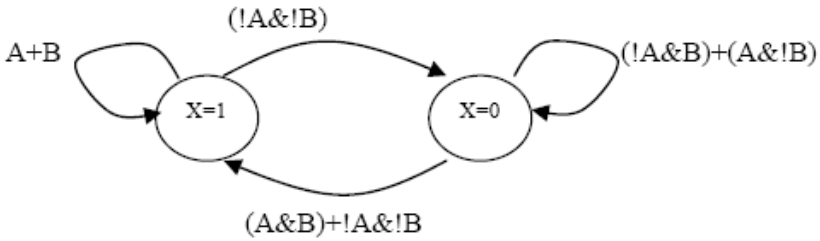
   T,T,NT,NT

7. Answer each of the following questions in 40 words or less.
   a. Why don't we let stores write to memory as soon as their address is known? **[3]**
      This question was thrown out for being poorly worded.
      The answer we wanted was that stores shouldn't write their data until they are non-speculative.

   b. In the algorithm we call T2, when does a committing instruction update the RAT? Be precise. **[3]**
      When the committing instruction's RoB entry is being pointed to by the RAT.

Part II – Longer Answer – 54 points

1. Write a Verilog module named FSA which implements the following state machine. You are to follow the Verilog style guidelines we've provided you for your assignments. You are to implement a synchronous reset which causes the state machine to reset to the state on the left. **[10]**



No answer provided.

2. Consider a set of code where there are three classes of instructions.
   - *Independent* instructions are not dependent on any other instruction and can execute in 3 cycles.
   - *Dependent* instructions are dependent on the instruction in front of them and take 3 cycles to execute.
   - *Branch* instructions are not dependent on any other instruction and take one cycle to execute. However, they are always *mispredictions*.

   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 16 and a RoB size of 64. Show your work!

   a. What is the best CPI this machine could achieve if the program being run consisted of groups of 10 instructions, where the first 9 were "independent" and the last was "branch". Assume there are a large number of these groups. (So the code is 9 independent, 1 branch, 9 independent, 1 branch, etc.) **[4]**
   Note: There are a number of assumptions one could make which would change this answer…
   The first independent instruction takes 3 cycles to finish (perhaps up to 2 more cycles due to fetch and commit times). Each one after that takes 1 cycle, including the branch. After the branch you restart. So 12 cycles/10 instructions =CPI of 1.2

   b. What is the best CPI this machine could achieve if the program being run consisted of groups of 10 instructions, where the first 9 were "dependent" and the last was "branch". Assume there are a large number of these groups. (So the code is 9 dependent, 1 branch, 9 dependent, 1 branch, etc.) **[4]**
   Again, note that there are a number of assumptions that could be made. If you made different assumptions than you did for part a) we took off points.
   This time each dependent instruction takes 3 cycles and the branch 1. The first dep instruction may add a cycle for fetching and/or the commit time of the branch. But without that you have 27 (dep instructions) +1 (branch) or 28/10 or 2.8.

   c. What is the best CPI this machine could achieve if the program being run consisted of groups of 100 instructions, where the first 99 were "dependent" and the last was "independent". Assume there are a large number of these groups. (So the code is 99 dependent, 1 independent, 99 dependent, 1 independent, etc.) **[4]**
   Each group of 100 will be able to start as the group in front of it finishes. That "head start" will last from when an RS opens up for the independent instruction and will last until that ind. instruction hits the head of the RoB. So you are looking at 45 cycles of overlap in execution. So 300 cycles-45-255/100 is 2.55

3. Consider a Tournament predictor made up of the following components:
   - Local
     - 8-entry 3-bit local history table indexed by PC
     - 8-entry 2-bit up/down counter indexed by local history
   - Global
     - 8-entry 2-bit up/down counter indexed by global history
   - Tournament
     - 8-entry 2-bit up/down counter indexed by PC

Tournament selector 00=local, 11=global

ADR[27:29]	*Pred. state*
0	00
1	01
2	00
3	10
4	11→10
5	00
6	11
7	10

Local predictor 1st level table (BHT) 0=NT, 1=T

ADR[27:29]	*History*
0	001
1	101→110
2	100
3	110→001
4	110→101
5	001
6	111
7	101

Local predictor 2nd level table (PHT) 00=NT, 11=T

history	*Pred. state*
0	00
1	11
2	10
3	00
4	01→10
5	01→10
6	11
7	11

Global predictor table 00=NT, 11=T

history	*Pred. state*
0	11
1	10→01
2	00→10
3	00
4	00
5	11→10
6	11
7	00

Now consider the following program:

```
Bob:    R1=R2+R3
        if(R1==0) goto Next
        R3=R4+R5
Next:   if(R6==R3) goto Done
        goto Bob
Done:   ……..
```

Assume:

- The global history starts at 000.
- All histories have the right-most bit be the most recent branch.
- The first branch is taken every other time, and is taken the first time.
- The second branch is not taken the first time and taken after that.
- The always taken branch **is** predicted and updates the predictor.
- "Bob" is located at 0x100 and every instruction takes up 4 bytes.

Show the table's status once the "Done" label is reached. **[10]**

Branch Index	T/NT	Global PT	GPT	BHT	PHT	TS
1	1	000→001	11→11	101→011	11→11	01→01
3	0	001→010	10→01	110→100	11→10	10→10
4	1	010→101	00→01	110→101	10→11	11→10
1	0	101→010	11→10	011→110	00→00	01→01
3	1	010→101	01→10	100→001	01→10	10→10

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

RAT

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

ROB

Buffer Number	PC	Executed?	Dest ARN	Dest. PRN	Replaced Phy. Reg# (Back pointer)
0	~~12~~	~~Y~~	~~0~~	~~5~~	~~0~~
1	~~16~~	~~N~~	~~4~~	~~6~~	~~4~~
2	~~20~~	~~N~~	~~4~~	~~7~~	~~6~~
3	~~24~~	~~Y~~	~~—~~	~~—~~	~~—~~
4	~~80~~	~~Y~~	~~0~~	~~8~~	~~5~~
5	~~84~~	~~N~~	~~0~~	~~9~~	~~8~~
6	28	N	2	4	2
7	32	N	0	0	5
8					

Head=0 →6
Tail=5 →7

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	~~Add~~	~~Y~~	~~9~~	~~Y~~	~~17~~	~~6~~	~~1~~
1	~~Add~~	~~N~~	~~7~~	~~N~~	~~7~~	~~9~~	~~5~~
2	~~Add~~	~~N~~	~~6~~	~~Y~~	~~9~~	~~7~~	~~2~~
3	Add	Y	17	Y	35	4	6
4	Add	N	0	Y	10	0	7

PRF

Phy Reg #	Value	Free	Valid
0	7	N	Y→N
1	8	N	Y
2	9	N	Y
3	10	N	Y
4	1	N	Y→N
5	17	N	Y
6	66	N→Y	N
7	35	N	N→Y
8	11	N→Y	Y→N
9	70	N→Y	N
10	44	Y	N
11	55	Y	N
12	99	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

Say that the instruction in ROB #3 is a branch and it was mis-predicted: the next PC should have been 28. Say that the instruction in memory location 28 is R2=R0+R4 and in 32 is R0=R2+R3. Update the machine to the state where the branch has left the RoB, and the instructions at memory locations 28 and 32 have issued but not executed. When faced with an arbitrary decision, just be sure to make a legal choice. **[18]**

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. *Also, be sure to answer the following question.*

1. What was the instruction that is found at memory location 84? **[4]**

   R0=R4+R4

RAT

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

ROB

Buffer Number	PC	Executed?	Dest ARN	Dest. PRN	Replaced Phy. Reg# (Back pointer)
0	12	Y	0	5	0
1	16	N	4	6	4
2	20	N	4	7	6
3	24	Y	--	--	--
4	80	Y	0	8	5
5	84	N	0	9	8
6					
7					
8					

Head=0
Tail=5

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	Add	Y	9	Y	17	6	1
1	Add	N	7	N	7	9	5
2	Add	N	6	Y	9	7	2
3							
4							

PRF

Phy Reg #	Value	Free	Valid
0	7	N	Y
1	8	N	Y
2	9	N	Y
3	10	N	Y
4	1	N	Y
5	17	N	Y
6	66	N	N
7	88	N	N
8	11	N	Y
9	16	N	N
10	44	Y	N
11	55	Y	N
12	99	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