EECS 470 *Midterm Exam*

Name: ___________________________________

Scores:

#	Points
1	/35
Section 2	
1	/15
2	/20
3	/10
4	/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 80 minutes for the exam.
- **Be sure to show work and explain what you've done when asked to do so.**
- **The last page is a duplicate of question #2. Clearly mark which one you want graded!**

Section I – Short Answer – 35 points

1. You are given a typical 5-stage pipeline where branches are resolved in memory and we only forward to the EX stage. What would be the CPI of a program where 10% of instructions are loads, 20% are branches, 50% of all loads are followed by dependent instructions, and 60% of branches are taken? Assume branches are predicted not-taken. **[6]**

2. List one advantage and one disadvantage of a stack machine compared to a register (load-store) machine. **[5]**

3. Consider a local 1-bit predictor that is indexed off of the PC only. If that predictor starts in an unknown state, how many mis-predicts might it get (maximum) on the following pattern: T,NT,NT, NT, T, T, NT? Assume no other branch indexes into the same predictor (i.e. there is no aliasing). **[6]**

4. Briefly describe how the number of reservation stations impacts the CPI of a processor and the clock frequency of that processor. Your answer should be no more than two or three sentences in length. **[6]**

5. Say we have a BTB used in parallel with a tournament branch predictor. The BTB is fully tagged while the predictor is not. During the fetch stage say we encounter a branch that is in the BTB and is predicted taken by both the 2-bit local and 2-bit global predictors of the tournament predictor. If the branch is actually taken which of the following might be changed? Clearly circle your answer(s) **[6]**

    a. BTB target address
    b. BTB tag
    c. Local branch predictor
    d. Local branch history table
    e. Global branch predictor
    f. Tournament predictor
    g. Global history register

6. Consider the following statement:

    > *"Because of register renaming there is much less of a need for having a large set of general purpose registers – the physical register file can be much larger than the architected register file and not have any impact on the ISA."*

    Explain when having a larger set of architected registers can help but having a larger set of physical registers would not. **[6]**

Section II – Longer Answer – 65 points

1. Consider the following programs

Program A	**Program B**
R1=MEM[R2+0]	R1=MEM[R2+0]
R2=R1+4	R2=R3+4
R3=R5+R7	R3=R4+R1
R4=R2+6	R4=R6+6
R5=R4+R3	R5=R7+R8
R6=R8+R5	R6=R9+R10
R7=R6+19	R7=R8+19
R8=R7+R6	R8=R9+R3

Say we have an out-of-order machine (Using Tomasulo's II) with 3 RSs and 6 ROB entries. Further, say the first load stalls for a **long** time. Both programs will come to a halt waiting for the load to finish. For each program indicate 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. Be sure to **clearly** show/explain your work. **[15]**

2. In this problem we are using what we are calling Tomasulo's third algorithm. Say the ROB, RS, RAT and physical register file are as follows:

RAT

Arch Reg. #	Phys. Reg. #
0	2
1	0
2	8
3	1
4	10
5	9

ROB

Buffer Number	PC	Done with EX?	Arch Reg #	Previous RAT value
0	12	Yes	1	4
1	16	No	2	7
2	24	No	N/A	N/A
3	60	Yes	2	5
4	64	Yes	5	6
5	68	No	4	3
6	--	--	--	--
7	--	--	--	--

← **Head** (buffer 0)

← **Tail** (buffer 5)

RS

RS#	Op type	Op1 ready?	Op1 PRN/value	Op2 ready?	Op2 PRN/value	Dest. PRN	ROB
0	DIV	Yes	2	Yes	12	5	1
1	Branch	No	5	Yes	9	N/A	2
2	ADD	No	5	Yes	12	10	5
3	--	--	--	--	--		

Phy. Register File

Phy. Reg #	Value
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	10
9	11
10	12
11	13

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.
- **ROB** is the associated ROB entry for this instruction.
- **Previous RAT value** is the value that was in the RAT for the destination architected register *before* this instruction wrote to the RAT.

Say that sometime later *only* the following have changed:

- The branch is discovered to have mispredicted
- All instructions which can retire do so
- An instruction, R1=R2+R3 is renamed and put into the RS and ROB. This instruction comes after the branch-mispredict and has a PC of 28.

Update the above table(s) to reflect the changes this would cause. Be sure to clearly label the ROB head and tail. When arbitrary decisions need to be made just be sure you make a legal decision. You are to assume an instruction leaves the RS when it finishes executing. *An extra copy of this page can be found at the end of the exam*. Please mark which version you would like us to grade. **[20]**

3. Consider a gshare predictor where:
    - The global history to date is 011
    - The instructions are 32-bits long each.
    - Memory is byte-addressed.
    - The current PC is 0x1F8, it is a branch, and it will be taken.
    - The local bits used are the three least significant bits of the PC other than the word offset and they are used with the LSB on the right.
    - The underlying predictor is the two-bit predictor shown on page 198.
    - The current branch is an indirect (and always taken) branch
    - The current predictor table is:

Index	Predictor
0	01
1	10
2	01
3	10
4	11
5	01
6	11
7	00

Will the branch be predicted as taken or not taken? How will the above table be changed (it at all) once the branch executes? **[10]**

4. Consider the following code:

```
loop:    r1 ← r6+r4          // inst A
         r2 ← r3+r1          // inst B
         r3 ← r3+r1          // inst C
         r4 ← r4+r2          // inst D
         r3 ← r3+r2          // inst E
         if(r4!=0)goto loop       // inst F
exit:    halt
```

a) Draw a *data dependency* (not control or name dependency) diagram for the above code assuming that the loop was taken the first time and not taken the second. Use the instruction name (A, B, etc.) and the loop iteration number (1 or 2) to describe each executed assembly instruction (so A1, B1, B2, etc.). Draw arrows from an instruction to the instruction(s) it is *directly* dependent on. **[7]**

A1	A2
B1	B2
C1	C2
D1	D2
E1	E2
F1	F2

b) Say we are predicting not-taken and there is a 3 cycle penalty for misprediction. And say that other than mispredicted branches our processor always finishes 1 instruction per cycle. If the branch is taken many many times, what would be the CPI of the processor on this code assuming the loop were? **[3]**

c) Say we considered replacing the branch with a branch that had three branch delay slots. Show how you would rewrite the code to get the best performance while maintaining code behavior. Assuming there is now no penalty for misprediction, what is the CPI of your program? Do *not* count any noops you may have inserted as instructions. **[6**]

d) In part c we assumed that the misprediction penalty would be removed with the addition of the branch with 3 delay slots. Is this a reasonable assumption? Explain your answer in a sentence or two. **[4]**