EECS 470 *Midterm Exam*

Winter 2014

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	**/12**
Page 3	**/10**
Page 4	**/10**
Page 5	**/13**
Page 6	**/11**
Page 7	**/11**
Page 8	**/6**
Page 9	**/9**
Pages 10 & 11	**/18**
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 ***11*** 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. Fill-in-the-blank or circle the *best* answer **[12 points, -2 per wrong/blank, minimum 0]**

   a. Given an 8-KB, two-way associative cache with 16-byte lines, you will need ____8__ bits to index the cache. If virtual and physical addresses were both 32-bits in size, the cache would need *about* 1KB **/ 2KB / 4KB / 8KB / 16KB** to store all the tags.

   b. When running a given program on a processor which has implemented the P6 scheme, it is discovered that the program is fetching twice as many instructions as it is committing. The branch prediction rate for this program is about 95% and only 10% of all instructions are branches. Given that which of the following can you most reasonably be sure is true?
      - ***The mispredictions are mostly from the same branch***
      - THIS PROCESSOR HAS A ROB WITH MORE THAN 128 ENTRIES.
      - ***The mispredictions are tightly grouped (they mostly occur one after the other or at least soon after another misprediction)***
      - ***The processor is using a purely global predictor***

   c. You would expect a wire 2mm long with a 200nm^2 cross-sectional area to have a **higher /** LOWER **/ identical** resistance than a wire 1mm long and 50nm^2 cross-sectional area.

   d. The period of a 2GHz clock is ____0.5_____ns.

   e. Say you have an ISA where all instructions are 32-bits, there are 32 general purpose registers, and all immediate values are 16-bits. If your instruction set consisted of nothing other than instructions that used two GPRs and one immediate, you could have up to 64 **/ 256 / 512 / 1024 / 4096** instructions in your ISA.

   f. A processor using the R10K algorithm has 64 RoB entries and 16 RS entries. Assuming the ISA has 32 architected registers, you can be reasonably certain that the RAT uses ***128 /*** 224 ***/ 256 / 408 / 512*** bits to store tags (not including valid bits and the like).

   g. Arguably, the largest problem with using the original version of Tomasulo's algorithm for a modern processor is that it ***requires too large of a RoB / requires too large of a branch predictor /*** CAN'T PROVIDE PRECISE EXCEPTIONS ***/ won't work with floating point operations***.

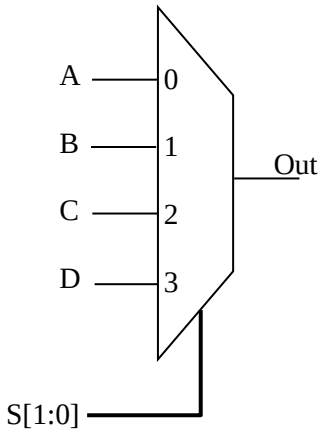
2. Given the following design changes to a simple out-of-order pipeline (and assuming no other changes to the pipeline or workload), what would be the effect on the i) number of instructions committed (N_{inst}), ii) the cycles-per-instruction (CPI), iii) clock period (t_{clk}), and iv) time to execute a given program (t_{CPU}). For each possible effect, indicate one of the following: no change (Ø), equal or greater (↑), equal or less (↓), or not enough information to determine (?). Provide the best/most likely answer. Leave the boxes with X's blank.
[10 points, -1 per wrong or blank answer, minimum 0]

Design Change	**N_{inst}**	**CPI**	**t_{clk}**	**t_{cpu}**
Change from the original Tomasulo's algorithm to the R10K scheme	Ø	↓*	X	↓*
An in-order processor goes from 6 stages to 8 stages.	Ø	↑	↓	?↓
Remove all caches	Ø	↑	↓	↑
Do what's needed to replace short, hard-to-predict branches with CMOV (assume the ISA has CMOV in it already)	↑	↓	Ø	X
Increase the size of the RoB while keeping the RS constant in size.	Ø	↓	↑	↓ ?

IN SOME CASES MORE THAN ONE ANSWER WAS REASONABLE, IN WHICH CASE ALL ARE LISTED. FOR THE ONES WITH A "*" WE TOOK "?" THIS, THOUGH WE PROBABLY SHOULDN'T HAVE.

3. Write a Verilog ***module*** which implements each of the following devices. You are to keep the signal names the same as they are in the provided figures. Your code should be reasonably efficient. *Minor* syntax errors will be ignored. **[10 points]**

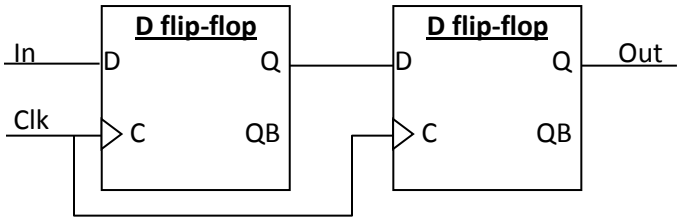
a. The 4 to 1 MUX shown below. **[5]**




```
module MUX41(S,A,B,C,D,Out);
        input A,B,C,D;
        input [1:0] S;
        output Out;

        assign Out= S==2'd0?A:
                    S==2'd1?B:
                    S==2'd2?C:
                    D;
endmodule
```

b. The registers shown below. **[5]**




```
module reg2(Clk, In, Out);
        input Clk, In;
        output reg Out;
        reg tmp;

        always@(posedge Clk)
        begin
                tmp <= #1 In;
                Out <= #1 tmp;
        end
endmodule
```

4. Consider the following pseudo-assembly code. **[8 points]**

```
R1=R2+R3
R2=R1+R1
R3=R1+8
R4=R3+R4
R4=R1+R3
R2=R2+R3
R3=R3+R8
```

In class we discussed that *programs* can be evaluated for ILP independent of a computer implementation. ILP of a program is the average number of instructions that could be executed in parallel. So if there were 2 instructions and they could be executed at the same time, the ILP would be "2", but if they couldn't be (due to some dependency) the ILP would be 1.

a. What is the ILP available in this code assuming there is ***no*** renaming (and thus all name dependencies need to be respected)? Show your work. **[4]**

7/4 (DEPENDENCY GRAPH IS OF HEIGHT 4, SO AVERAGE IS 7/4)

b. What is the ILP available in this code assuming renaming ***is*** happening? Show your work. **[4]**

7/3 (UPON REMOVING FALSE DEP. BETWEEN 4TH AND 5TH INSTRUCTION, HEIGHT IS NOW 3)

5. Consider the pipeline you were to implement for your third programming assignment, but assume that the structural hazard has been removed and *branches are resolved in the execute stage*. A given program consists of 25% loads, 10% stores, 10% branches and ~~60%~~ 55% ALU operations. If 40% of the branches are not-taken and 30% of all instructions are dependent on the instruction in front of them, what is the expected CPI of the processor on this program? Show your work **[5 points]**

BASE+DATA HAZARDS+BRANCH SQUASHES. BRANCH TAKEN ONLY SQUASHES 2 INSTR. NOW.

1+.25*.3+.1*.6*2 =1.195

6. Consider a non-superscalar processor implementing the P6 algorithm. This processor has 32 architected registers, 64 RoB entries and 8 reservation stations. List all input and output that will be used to implement the RAT by filing in the table below. We've done one signal for you as an example. **[11 points]**

Description of signal	Input/ Output	Why it's needed	How many bits?
Architected destination register	Input	It needs to be renamed	5
CLK	INPUT	HAS SYNC LOGIC, NEEDS A CLOCK	1
RESET/FLUSH	INPUT	USED ON RESET AND MISPREDICT. ALL ENTRIES POINT TO ARF.	1
ARCH SOURCE REG #1	INPUT	SOURCE VALUE TO BE RENAMED	5
ARCH SOURCE REG #2	INPUT	OTHER SOURCE VALUE TO BE RENAMED	5
ARCH. DEST. REG VALID	INPUT	SHOULD ASSIGN A ROB ENTRY THIS CYCLE*	1
ROB TAIL	INPUT	NEEDED TO ASSIGN INSTRUCTION A ROB NUMBER IN RAT	6
ROB SOURCE #1	OUTPUT	ROB NUMBER/TAG FOR SOURCE #1 REG	6
ROB SOURCE #2	OUTPUT	ROB NUMBER/TAG FOR SOURCE #2 REG	6
ARCH SOURCE #1 IN ARF	OUTPUT	SHOULD USE VALUE IN ARF FOR S1, NOT ROB OUTPUT	1
ARCH SOURCE #1 IN ARF	OUTPUT	SHOULD USE VALUE IN ARF FOR S2, NOT ROB OUTPUT	1
CLEAR ENTRY X	INPUT	ARF # TO CLEAR, USED BY COMMIT WHEN UPDATING RAT	5
ENABLE CLEAR OF X	INPUT	ENABLE THE ABOVE SIGNAL	1
HEAD OF ROB	INPUT	SEE IF WE SHOULD CLEAR RAT ENTRY	6

THERE ARE A NUMBER OF OTHER SIGNALS ONE MIGHT CONSIDER. FOR EXAMPLE, ROB FOR DEST REGISTER. BUT THAT'S JUST AN INPUT (ROB TAIL) AND SO ISN'T NEEDED (BUT IT'S NOT WRONG EITHER). WE GAVE ONE POINT PER USEFUL LISTED (MAX OF 11) AND TOOK OFF POINTS FOR WRONG ANSWERS ONLY IF *REALLY* WRONG.

7. An instruction can be said to be "in the shadow" of an earlier instruction if it is directly or indirectly data dependent on that earlier instruction. Say you have a non-superscalar processor which implements the R10K algorithm. It supports an ISA with 32 architected registers and has 64 RoB entries, 8 reservations stations and 96 PRF entries. Say a high-latency load is at the head of the RoB and the processor has stopped dispatching instructions because of this load. How many instructions would you expect to have in the RoB if: **[4 points]**

    a. All instructions after this load are in the load's shadow? **[1]**
    ASSUMING INSTRUCTIONS LEAVE THE RS WHEN ISSUING, 9.

    b. No instructions after this load are in the load's shadow? **[1]**
    64

    c. Half the instructions after this load are in the load's shadow? **[2]**
    UNCLEAR AS IT ISN'T OBVIOUS IF EVERY OTHER ARE IN THE SHADOW. 17 IS PROBABLY THE BEST ANSWER, BUT ANY WELL JUSTIFIED ANSWER WAS TAKEN.

8. Early branch resolution **[7 points]**
    a. If we are implementing early branch resolution using Branch Register Alias Tables (BRATs), at what point will we need to allocate a BRAT? **[3]**

    DISPATCH

    b. In order to recover the PRF's free list, we typically have a free list associated with each BRAT. If we wish to be able to simply copy that free list over to the main free list on a misprediction we need to do a number of things. *Circle **all** the things that need to happen*. You may assume this processor is not superscalar. **[4, no partial credit]**

    - UPON DISPATCH OF THE BRANCH, COPY THE RAT'S FREE LIST TO THE BRAT'S FREE LIST AFTER UPDATING THE FREE LIST TO REFLECT ANY DESTINATION REGISTER(S) THE BRANCH MIGHT HAVE.
    - Upon dispatch of a branch, copy the RAT's free list to the BRAT's free list ***before*** updating the free list to reflect any destination register(s) the branch might have.
    - Update the BRAT's free list as other instructions ***dispatch***.
    - UPDATE THE BRAT'S FREE LIST AS INSTRUCTIONS COMMIT.

9. Say we have the following code segment in pseudo-assembly:

```
        If(R1!=R2) goto NEXT
        R5=R5+R1
NEXT:
```

And say we've got the following instruction available to us: **CMOV(Rx,Ry,Rz)** where the instruction sets Ry=Rz if and only if Rx!=0.

Rewrite the above code to remove the branch. You may only use R6 and R7 as a "scratch" registers: all other registers are holding live values. You may not read or write memory and you may assume you have all the standard ALU operations (add, subtract, multiply, NOT, AND, etc.) available to you. **[6 points]**

LOTS OF REASONABLE ANSWERS HERE. ONE WOULD BE:

```
R6=R1-R2
R7=R5+R1
R6=NOT(R6)
CMOV(R6,R5,R7)
```

10. Consider the following pseudo-assembly code:

```
        r3=1000
        r5=0
        r6=0
Top:    r1=MEM[r3+0]              // Load #1
        if(r1==0) goto Nxt        // Branch 1
        r6=r6+1
Nxt:    r2=MEM[r1]                // Load #2
        if(r2>0) goto Lst         // Branch 2
        r6=r6+1
Lst:    r3=r3+8
        r5=r5+r2
        if(r3<500000) goto Top    // Branch 3
```

The predictors all use the least significant bits of the PC other than the word-offset. Predictors and patterns are all initialized to all zeros (not taken).

- Branch #1 follows the pattern TTTTNN forever (starting with taken).
- Branch #2 follows the pattern TNTNTN forever (starting with taken).

You are now to consider 2 branch predictors:

- **Predictor 1:** A 1-bit bimodal predictor with 8 entries. Each entry is a 1 bit predictor.
- **Predictor 2:** A local pattern history predictor. The BHT has 16 entries, each with 2 bits of history. The predictors are each 1 bit.

What are the expected prediction *rates* for each of the following (percentage of time right)? Your answers must be correct within 1.0%. **[9 points, -2 per wrong or blank box, min 0]**

	Predictor 1	*Predictor 2*
Branch 1	**2/3**	**1/2**
Branch 2	**0**	**2/3**
Branch 3	**1**	**5/6**

11. Consider the following state of a machine implementing what we've called the R10K algorithm with a retirement RAT.

RAT

Arch Reg #	Phy. Reg #
0	~~12~~ 1
1	3
2	~~9~~ 0
3	~~10~~ 6
4	5

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
~~0~~	~~0~~	~~N~~	~~7~~	~~0~~	← TAIL
~~1~~	~~4~~	~~N~~	~~8~~	~~0~~	
~~2~~	~~8~~	~~Y~~	~~9~~	~~2~~	
~~3~~	~~12~~	~~Y~~	~~--~~	~~--~~	
~~4~~	~~16~~	~~N~~	~~11~~	~~0~~	
~~5~~	~~20~~	~~Y~~	~~10~~	~~3~~	
~~6~~	~~24~~	~~Y~~	~~12~~	~~0~~	
7	60	N	0	2	← HEAD
8	64	N	1	0	

RRAT

Arch Reg #	Phy. Reg #
0	~~2~~ 8
1	3
2	~~4~~ 9
3	6
4	5

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	+	N	7	Y	36	11	4
1	*	Y	-1	Y	6	7	0
2	+	N	7	N	7	8	1
3	+	Y	5	Y	7	0	7
4	+	Y	-12	N	0	1	8

PRF

Phy Reg #	Value	Free	Valid
0	5	~~Y~~ N	N
1	5	~~Y~~ N	N
2	4	~~N~~ Y	~~Y~~ N
3	5	N	Y
4	6	~~N~~ Y	~~Y~~ N
5	7	N	Y
6	-1	N	Y
7	~~5~~ -6	~~N~~ Y	N
8	~~5~~ -12	N	~~N~~ Y
9	36	N	Y
10	1	~~N~~ Y	~~Y~~ N
11	4	~~N~~ Y	N
12	2	~~N~~ Y	~~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 60. Say that the instruction in memory location 60 is R2=R1+R4 and in 64 is R0=R0+R2. Update the machine to the state where the branch has left the RoB, and the instructions at memory 60 and 64 have dispatched but not started execution. *When selecting a PRF use the lowest numbered physical register available*, otherwise when making an arbitrary decision, just be sure it is legal. ***Be sure to update the head and tail pointers!*** **[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.

This is a copy of the previous state. You may use this one or the previous one but be sure to cross out (with a BIG X) the one you don't want graded.

RAT

Arch Reg #	Phy. Reg #
0	12
1	3
2	9
3	10
4	5

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
0	0	N	7	0	← HEAD
1	4	N	8	0	
2	8	Y	9	2	
3	12	Y	--	--	
4	16	N	11	0	
5	20	Y	10	3	
6	24	Y	12	0	
7					← TAIL
8					

RRAT

Arch Reg #	Phy. Reg #
0	2
1	3
2	4
3	6
4	5

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	+	N	7	Y	36	11	4
1	*	Y	-1	Y	6	7	0
2	+	N	7	N	7	8	1
3							
4							

PRF

Phy Reg #	Value	Free	Valid
0	5	Y	N
1	5	Y	N
2	4	N	Y
3	5	N	Y
4	6	N	Y
5	7	N	Y
6	-1	N	Y
7	5	N	N
8	5	N	N
9	36	N	Y
10	1	N	Y
11	4	N	N
12	2	N	Y

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 if 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 60. Say that the instruction in memory location 60 is R2=R1+R4 and in 64 is R0=R0+R2. Update the machine to the state where the branch has left the RoB, and the instructions at memory 60 and 64 have dispatched but not started execution. *When selecting a PRF use the lowest numbered physical register available*, otherwise when making an arbitrary decision, just be sure it is legal. ***Be sure to update the head and tail pointers!*** **[18]**