1EECS 470 *Midterm Exam*

Winter 2020

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:

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" for the last question. Clearly mark which one you want graded or we will grade the first one.**

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

a) If, in the algorithm we are calling R10K, the RRAT had only one port for writing values, you could only have one instruction that writes a register ***issue / dispatch / start execution / complete execution / [✓ retire]*** per cycle.

b) Adding more reorder buffer entries will reduce the amount of time that the processor stalls due to ***[✓ structural hazards] / data hazards / control hazards*** but may decrease the ***number of branch delay slots / clock period / [✓ clock frequency]***.

c) When using the P6 algorithm, if you have B reorder buffer entries, S reservation stations, and A architected registers you can expect to have ***$B*\log_2(S)$ / [✓ $A*\log_2(B)$] / $(S+A)*\log_2(B)$ / $(S+B)*\log_2(A)$*** bits needed in the register alias table. Don't include valid bits or other status bits, just the information used to store the rename location.

d) In the R10K scheme an instruction might get its data from one of two places. Those

two places are the __CDB___________, or the ______PRF________.

e) Given an 32-KB two-way associative cache with 32-byte cache lines and a 64-bit

address space there will be ___9___ bits used for the index. If that same cache were

four-way associative you'd need __8____ bits to be used for the index.

f) Say when using the R10K scheme instruction "A" is a load that misses in the cache and the next 1000 instructions are "in the shadow" of that load. A structural hazard will likely occur while A is being executed. That structural hazard will most likely be due to a lack of enough ***[✓ Reservation stations] / Reorder buffer entries / Physical registers / CDBs***.

g) The period of a 4 GHz clock is _____0.25___________ ns.

h) Consider a CMOV instruction which takes 3 register arguments. If all instructions were 32 bits and the architected register file (ARF) had 256 entries, *approximately* what percent of all encodings would the CMOV instruction use assuming any register could be used for any of the three register arguments? ***0.02% / [✓ 0.4%] / 2% / 5% / 50%***

2) Now you are the compiler [11 points]

Consider the following assembly code

```
1. R1 = R2 / R3
2. R0 = MEM[R1]
3. R1 = R2 + R3
4. R0 = R0 + R1
5. R1 = MEM[R1]
6. R1 = R0 * R1
7. R4 = R0 + R1
8. MEM[R3] = R2 // a store instruction
```

a. Assuming parallelism is limited only by true dependencies, what is the ILP available in this code? Justify your answer by drawing a dependency graph (using the instruction number to represent the instructions). **[4]**

```
1   3   8        ILP=5/8.
2   5
4
6
7
```

b. Rewrite the assembly to eliminate all false dependencies in the code segment while maintaining the same behavior. **[7]**

You must:

- Not reorder, remove, or add instructions (only changing register numbers)
- Be sure that once your code segment ends, the values in registers R0-R4 remain exactly the same as they were after the above segment executed. So if R0=5 after the above code segment, R0 must still be five after your code segment finishes. Any memory writes must also be identical.
- Only use registers R0-R8.

1. R5=R2/R3
2. R6=MEM[R5]
3. R7=R2+R3
4. R0=R6+R7
5. R8=MEM[R7]
6. R1=R0*R8
7. R4=R0+R1
8. MEM[R3]=R2

3) Short answers [11 points]

In the paper "Combining Branch Predictors," one of the options considered is something called "gshare-best". Say you have a gshare predictor that uses a 2-bit saturating counter and is 2KB in size.

- How many bits of the global history *might* be used as part of indexing this predictor? Provide a range if there is more than one possible answer. **[2]**

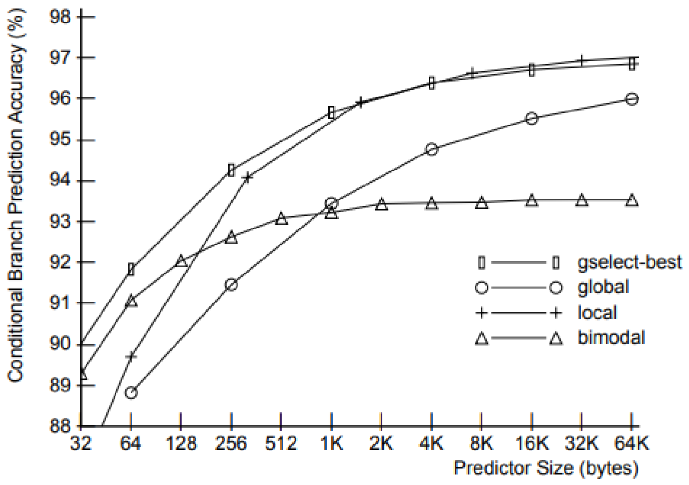
  0 to 13

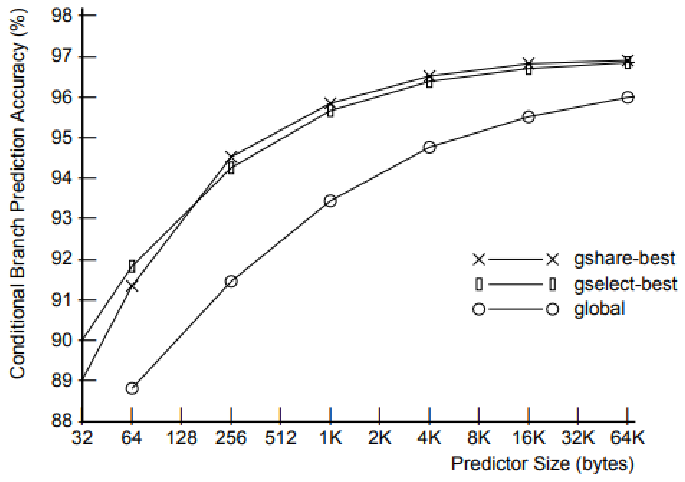
- How many bits of the PC *might* be used as part of indexing this predictor? Provide a range if there is more than one possible answer. **[2]**

  13

- Given the graph to the right, what indexing option can you be certain gshare-best did not actual utilize? Briefly explain. **[2]**

  More than 0 bits of global history are used, otherwise the result for gshare-best at 2KB would be the same as bimodal.





4) More short answer [12 points]

a) Provide a taken/not-taken pattern consisting of exactly 3 branches where a 2-bit saturating up-down predictor will do better than a 1-bit predictor. Assume the 2-bit starts as weakly not-taken and the 1-bit starts as not-taken. Your answer should be of the form (T, NT, T). **[4]**

NT, T, NT

b) In the P6 scheme, when a mispredicted branch at the head of the ROB commits, what changes are made to the following structures, if any? **[3]**

- ARF
  None

- RAT
  Clear it (so everything points to the appropriate ARF value)

c) In the P6 scheme, when an instruction commits, what changes are made to the following structures, if any? If updates are conditional, make that clear. Assume the committing instruction is not a mispredicted branch. **[5]**

- RS
  None.

- RAT
  If the committing instruction is in RoB entry X and modifies architected register Y, then if RAT[Y]==X, clear RAT[Y]. Otherwise do nothing.

- ARF
  Copy the RoB's result value into the appropriate ARF.

5) **Tournament of champions** [12 points]

Consider a variant of the tournament branch predictor used in class.

- The BHT is indexed with 10 bits of the PC
- The local PHT is indexed by 8 bits and uses a standard 4-state state machine.
- The Global history register is 8 bits.
- The tournament selector is indexed with 8 bits of the address and is a 2-state state machine.

a) What is the size of the BHT, in bits? **[2]**

$2^{10}*8=2^{13}$

b) What is the size of the local PHT, in bits? **[2]**

$2^{8}*2=2^{9}$

c) Say a given conditional branch was predicted by this predictor and was actually taken.

- Under what circumstances, if any, would it be possible that none of the structures (BHT, local PHT, Global history register, global PHT, tournament selector) would need to be updated? Be specific. **[4]**

  Assume for the BHT and GHR that "taken" is indicated by a 1.

  - BHT is all 1s (all branches taken)
  - Local PHT entry (indexed by 11111111) is strongly taken
  - GHR is all 1s (all branches taken)
  - Global PHT entry (indexed by 11111111) is strongly taken.
  - Local and global predictor made the same prediction (which they would have in this case)

- Under what circumstances, if any, would it be possible that tournament selector would get updated but none of the other structures (BHT, local PHT, Global history register, global PHT) would need to be updated? Be specific. **[4]**

  As noted above, the only way none of the other structures would be updated leads both predictors to predict "taken". So the tournament predictor wouldn't update as the local and global predictors agree. So this is impossible.

6) On ISAs and microarchitecture [8 points]

Branches with "branch delay slots" (also called "delayed branches") were added as a feature when in-order pipelined architectures were a common microarchitecture. **[8 points]**

a) Explain how having a "delayed branch" instruction helped with performance on an in-order pipeline. Your answer must be 40 words or less. **[3]**

Because the instruction(s) after the branch is always executed, on a mispredict there are fewer instructions that need to be squashed.

b) What changes would you have to make to the R10K microarchitecture to support a "delayed branch" instruction (in addition to a normal branch)? Be specific. **[5]**

You would need to put the instruction after the branch into the machine (fetch, dispatch, etc.) even if the branch is predicted taken. Further, when a branch is predicted not-taken but that is a mis-prediction, you would have to not squash when the branch hit the head of the ROB, but instead when the delayed instruction(s) were at that head of the ROB.

There are a few other ways of dealing with this (selective squashing of all instructions other than the delayed instructions for example) but a correct answer should address both what happens on a branch predicted to be taken (delayed instructions need to be executed and committed) and how to handle a branch predicted as not-taken that is actually taken (those delayed instructions need to be executed, ideally without re-fetching them).

c) Consider the pipelined processor you did as part of project 3 (including the structural hazard). Say 25% of all instructions were loads, 10% were branches, and 10% 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 60% of the time. What would you expect the CPI to be? Show your work and put your final answer in a box. **[5]**

CPI=1+structural hazards + control hazards + data hazards
CPI=1+0.35+(0.1)(0.6)(3)+(0.25)(0.2)(1)=1.58

7) Modulo by observation—SystemVerilog problem [10 points]

You are to design SystemVerilog module that has a 32-bit shift register. The module has the following one-bit inputs each sampled on the rising edge of clock: reset, enable, and serial_in. If reset is a 1, the 32-bit shift register is cleared (set to 0). If reset is 0 and enable is 1, then the bits will be shifted to the left, with the shift-in value placed in the least-significant bit of the shift register. If reset is 0 and enable is 0, the shift register will hold its value.

The module output is a one-bit value which is a "1" if the shift register has a value that is a multiple of 5.

An example is show below.

reset	**enable**	**serial_in**	**Sequence**	**Value in shift register**	**Is a multiple of five?**
1	0	x		32'd0	1
0	1	1	1	1	0
0	1	0	10	2	0
0	1	1	101	5	1
0	1	0	1010	10	1
0	1	1	10101	21	0
0	0	X	10101	21	0

On the next page, complete the SystemVerilog that would generate the correct output as described above. You should have the correct syntax but not be overly concerned by it. Style will be a factor. You may not use / or % in your solution.

```
typedef enum logic [2:0] {ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN}
remainder_state;

module serial_mult_of_five_display(
        input clock,
        input reset,
        input enable,
        input serial_in,
        output div_by_five
);

logic [31:0] shift_val, n_shift_val;
remainder_state state, n_state;

assign div_by_five = state == ZERO;
assign n_shift_val  = enable ? {shift_val[30:0], serial_in} : shift_val;
always_comb begin
        case(state)
                ZERO:   n_state = serial_in && shift_val[31]      ?      ZERO   :
                                                  serial_in        ?      ONE    :
                                                  shift_val[31]    ?      FOUR   : ZERO;
                ONE:    n_state = serial_in && shift_val[31]      ?      TWO    :
                                                  serial_in        ?      THREE  :
                                                  shift_val[31]    ?      ONE    : TWO;
                TWO:    n_state = serial_in && shift_val[31]      ?      FOUR   :
                                                  serial_in        ?      ZERO   :
                                                  shift_val[31]    ?      THREE  : FOUR;
                THREE:  n_state = serial_in && shift_val[31]      ?      ONE    :
                                                  serial_in        ?      TWO    :
                                                  shift_val[31]    ?      ZERO   : ONE;
                FOUR:   n_state = serial_in && shift_val[31]      ?      TWO    :
                                                  serial_in        ?      FOUR   :
                                                  shift_val[31]    ?       TWO   : THREE;
                default: n_state = state;
        endcase
end
always_ff @(posedge clock) begin
        if (reset) begin
                state       <= ZERO;
                shift_val   <= 32'h0;
        end
        else begin
                state       <= n_state;
                shift_val   <= n_shift_val;
        end
end

endmodule
```

8) **That last question** [18 points]

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	2
1	~~9~~ 8
2	~~4~~ 1
3	12
4	~~10~~ 6

ROB

Buffer Number	PC	Executed?	Dest. PRN	Dest ARN	
~~0~~	~~12~~	~~N~~	~~7~~	~~3~~	
~~1~~	~~16~~	~~N~~	~~8~~	~~1~~	
~~2~~	~~20~~	~~Y~~	~~12~~	~~3~~	
~~3~~	~~24~~	~~Y~~	~~--~~	~~--~~	
~~4~~	~~28~~	~~Y~~	~~9~~	~~1~~	
~~5~~	~~32~~	~~N~~	~~10~~	~~4~~	
6	60	N	0	2	← HEAD
7	64	N	1	2	
8					← TAIL

RRAT

Arch Reg #	Phy. Reg #
0	2
1	~~3~~ 8
2	4
3	~~5~~ 12
4	6

RS

RS#	Op Type	Op1 Ready?	Op1 PRN/value	Op2 Ready?	Op2 PRN/value	Dest PRN	ROB
0	~~+~~	~~N~~	~~7~~	~~Y~~	~~1~~	~~8~~	~~1~~
1	~~*~~	~~Y~~	~~2~~	~~Y~~	~~-1~~	~~7~~	~~0~~
2	~~+~~	~~Y~~	~~1~~	~~Y~~	~~1~~	~~10~~	~~5~~
3	~~+~~	~~Y~~	~~-1~~	~~Y~~	~~-2~~	~~0~~	~~6~~
4	~~+~~	~~N~~	~~0~~	~~Y~~	~~-1~~	~~1~~	~~7~~

PRF

Phy Reg #	Value	Free	Valid
0	3	~~Y~~ N	N
1	3	~~Y~~ N	N
2	1	N	Y
3	2	~~N~~ Y	~~Y~~ N
4	0	N	Y
5	-1	~~N~~ Y	~~Y~~ N
6	-2	N	Y
7	6	~~N~~ Y	N
8	6	N	~~N~~ Y
9	1	~~N~~ Y	~~Y~~ N
10	9	~~N~~ Y	N
11	9	Y	N
12	0	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 in it is valid. _A free entry must 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 R2=R2+R1. 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!***

<u>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.</u>

<u>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.</u>

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

ROB					
Buffer Number	**PC**	**Executed?**	**Dest. PRN**	**Dest ARN**	
0	12	N	7	3	← HEAD
1	16	N	8	1	
2	20	Y	12	3	
3	24	Y	--	--	
4	28	Y	9	1	
5	32	N	10	4	
6					← TAIL
7					
8					

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

RS							
RS#	**Op Type**	**Op1 Ready?**	**Op1 PRN/value**	**Op2 Ready?**	**Op2 PRN/value**	**Dest PRN**	**ROB**
0	+	N	7	Y	1	8	1
1	*	Y	2	Y	-1	7	0
2	+	Y	1	Y	1	10	5
3							
4							

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. *<u>A free entry must be marked as invalid</u>*.

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

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 R2=R2+R1. 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!***