EECS 470 *Midterm Exam*

Winter 2024

Name: ________**Key**__________________________ unique name: ________________

Sign the honor code:

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

____

NOTES:

- Open book and Open notes
- Calculators are allowed, but no smart watches, cell phones, ear buds, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam.
- There are ***10*** pages including this one.
- **Be sure to show your 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) **Select all** [10 points, -1 for each incorrectly marked (or unmarked) answer, minimum 0]

Clearly fill in the rectangle of ***all*** correct answers for each question.

a) In the P6 scheme, an individual RAT entry might be written when an instruction…
   - ☐ Enters the out-of-order system
   - ☐ Is sent from an RS to an execution unit
   - ☐ Completes execution and is broadcast on the CDB (if including ready bit, take all answers)
   - ☐ Retires

b) In the R10K scheme, an individual PRF entry might be written when an instruction…
   - ☐ Enters the out-of-order system (if including removing from free list, take all answers)
   - ☐ Is sent from an RS to an execution unit
   - ☐ Completes execution and is broadcast on the CDB
   - ☐ Retires (if including adding to free list, take all answers)

c) In the R10K scheme using "back pointers", the RS value field might be written when an instruction…
   - ☐ Enters the out-of-order system
   - ☐ Is sent from an RS to an execution unit
   - ☐ Completes execution and is broadcast on the CDB
   - ☐ Retires

d) A *disadvantage* of increasing the number of architected registers is:
   - ☐ You need more bits to encode the register number in an instruction
   - ☐ You will have more spills and fills
   - ☐ The time to access the register value will likely go up
   - ☐ You'd expect to have more false dependencies

e) In the P6 scheme, an instruction might get its value from where?
   - ☐ The PRF
   - ☐ The ARF
   - ☐ The CDB
   - ☐ The ROB

2) **Fill-in-the-blank or circle the one best answer** [15 points, -2 per wrong/blank, minimum 0]

a) Adding more reservation station entries will reduce the amount of time that the processor stalls due to ***[✓ structural hazards] / data hazards / control hazards*** but may increase the ***PRF size / [✓ clock period] / clock frequency.***

b) The period of a 5 GHz clock is _______0.2________ ns.

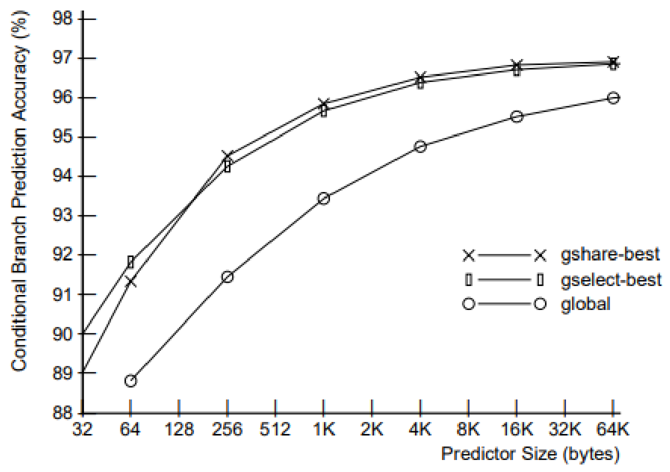
c) Given a 16-KB four-way associative cache with 64-byte cache lines and a byte-addressed 40-bit address space, there will be ___6____ bits used for the index. If that same cache were direct-mapped you'd need ___8___ bits to be used for the index.

d) If a given application were perfectly parallelizable for 60% of its execution time on one processor and perfectly serial for the rest of the time, you would expect the application to have a speedup of ____2.174 or 1/0.46_____ on 10 processors and ____2.5 or 1/0.4_______ on an infinite number of processors.

e) Consider a local history predictor where the BHT is indexed by 10 bits of the PC and each entry has 4 bits of history. The PHT is a standard 4-state predictor. We would need a total of __4096__ bits of storage for the BHT and a total of __32__ bits of storage for the PHT.

f) If the multiplier is in the critical path for the processor's clock period, decreasing the number of pipeline stages used to do a multiply can be expected to ***[✓ increase] / decrease / not affect*** the clock period while ***increasing / [✓ decreasing] / not affecting*** the CPI.

3) **Short answer** [15 points]



a) In the "Combining Branch Predictors" paper by McFarling, he compared a scheme called "gselect" to a scheme called "gshare". Explain what the "gselect-best" and "gshare-best" schemes referred to in the figure to the right are. **[5]**

Gselect is a branch predictor that concatenates the global history with selected bits of the PC and uses the result to index into a pattern history table which makes the prediction.
Gshare uses the global history xor'ed with the PC to index in the pattern history table. The number of PC bits determines the number of PHT entries, and the number of bits of history can be less than or equal to the number of PC bits.
The "-best" versions of these predictors are the versions that McFarling found to work best after adjusting the number of bits of global history and PC used.

b) Briefly explain why we don't use the most significant bits of an address to index a branch predictor. **[5]**

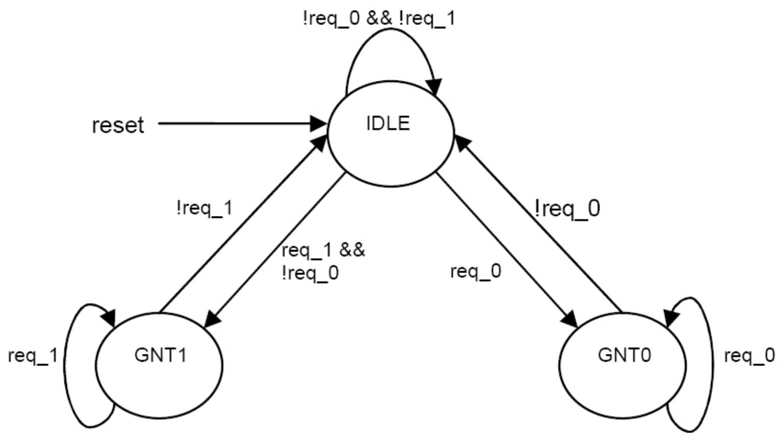
If we use the most significant bits of the address as the index, branches near each other in memory will be much more likely to alias to the same predictor. Because of spatial locality, branches near each other in memory are likely to be used near each other in time. In fact, most of the time it would be like having only one branch predictor. Thus, we'll get a lot of aliasing and thus a much lower prediction rate. *Note: this was graded in part for clarity, a lot of you were close to the right idea but couldn't really explain it in a clear way.*

c) Branch alignment is a compiler optimization that converts strongly biased taken branches to strongly biased not-taken branches by rearranging instruction layout in memory. How can this optimization be used to improve BTB performance? (Keep your answer to a few sentences.) **[5]**

Changing the bias from strongly taken to strongly not-taken lets us avoid storing the targets of those branches at all. This lets us store other things in the BTB, thus increasing the BTB's hit rate.

4) **SystemVerilog** [10 points]

Write a SystemVerilog module named ARB which implements the following state transition diagram. The outputs are `gnt_0` (which should be a 1 if and only if the current state is GNT0) and `gnt_1` (which should be a 1 if and only if the current state is GNT1). In addition to correctness, it will be graded in part for good coding style.



```
module ARB (
        input clock, reset,
        input req_0, req_1,
        output gnt_0, gnt_1
);
        logic [1:0] state, next_state;
        assign gnt_0 = state[0];
        assign gnt_1 = state[1];

        always_comb begin
           next_state = state; // don't forget to avoid a latch (could also use default case)
           case (state)
              2'b00: next_state = req_0 ? 2'b01 : req_1 : 2'b10 : 2'b00;
              2'b01: next_state = req_0 ? 2'b01 : 2'b00;
              2'b10: next_state = req_1 ? 2'b01 : 2'b00;
           endcase
        end

        always_ff @(posedge clock) begin
           if (reset) begin
              state <= 0;
           end else begin
              state <= next_state;
           end
        end

endmodule
```

5) **5-stage pipeline** [11 points]

Consider the pipelined processor you did as part of project 3 (*including the structural hazard*). Say 20% of all instructions were loads, 15% were branches, 10% were stores, and that the program was quite long (millions and millions of instructions). 40% of all instructions are data dependent on the instruction in front of them (and are otherwise not dependent on any other instruction), and branches are taken 40% of the time.

a) Given the above, what would you expect the CPI to be? Show your work and put your final answer in the box provided. **[4]**

Base CPI is 1 plus cycles stalling: 1 * (0.2 + 0.1) for structural hazards from loads plus stores, 3 * 0.15 * 0.4 for taken branches, and 1 * 0.2 * 0.4 for data dependent loads. This gives: 1 + (0.2 + 0.1) + (3 * 0.15 * 0.4) + (0.2 * 0.4) = 1 + 0.3 + 0.18 + 0.08 = 1.56

Note that there would also be decreased stalling in the case where a dependent load followed by a dependent load absorbs the data stall for the second load into the first load's structural hazard stall. However, this was not required for full points. This would reduce the CPI to: 1.56 - (0.2 * 0.2 * 0.4 * 0.4) = 1.5536

CPI = __ 1.56 _______________

b) Now say that branches are resolved in the decode stage and we are forwarding values needed by branches to the decode stage if the value has been computed. Everything else is the same as in part a of this problem. What would you now expect the CPI to be? Show your work, *briefly explain your answer*, and put your final answer in the box provided. **[7]**

If branches resolve in decode, we get the following changes:
1) Reduce the number of squashes for each taken branch by 2 (probability 0.15 * 0.4)
2) Stall an extra cycle for all data dependencies for branches (probability 0.15 * 0.4)

Our CPI decreases by (2 cycles * 0.15 * 0.4) and increases by (1 cycle * 0.15 * 0.4).
New CPI = 1.56 - 0.12 + 0.06 = 1.50

CPI = ____ 1.50
____

6) **CMOV** [9 points]

Consider the following code. Use the CMOV instruction (defined below) to remove one of the two branches (leaving only one). You may rewrite the code as needed but it needs to do the same thing it was doing initially and your code should not fault on a data load when this code wouldn't fault. Of the 32 registers, R0-R25 are all live after this code segment (so their values after your changes need to be the same as they are now).

Your CMOVs and compares should be of the following form (where Rx etc. is a register and C is an immediate).

- `Rx=(Ry>C) ; Rx=1 if Ry > C, otherwise Rx=0. C is an`
  `          ; immediate and the ">" can be replaced by "=="`
  `          ; or "<".`
- `CMOV(Rx,  Ry ←Rz)     ; If Rx != 0, Ry=Rz, else noop.`
- `CMOV(!Rx, Ry ←Rz)     ; If Rx == 0, Ry=Rz, else noop.`

You can cross out lines and replace them as you wish (rather than rewriting the whole thing).

```
  top:
    R3=MEM[R2+0]
    R4=MEM[R2+4]
    R2=R2+8
    R5=R5+R4
    ~~If(R5<0) GOTO skip~~       R26 = R5 < 0;
                                 R27 = R5 + R6;
                                 R28 = R2 + 4;
    ~~R5=R5+R6~~                 CMOV(!R26, R5 ← R27);
    ~~R2=R2+4~~                  CMOV(!R26, R2 ← R28);
  skip:
    If(R3==0) GOTO top
    R2=MEM[R3+0]
    done:
    R5=R6+1
              ...
```

7) **Prediction is difficult, especially if it's about the future**. [12 points, -1 for each blank or wrong answer.]

```
             r2 = 0
             r4 = 0
             r5 = 0
      bob:   r3 = (r2 % 2)
             if(r3 == 0)goto next         //Branch 1
             r4 = r4 + 1
      next:  r5 = r5 + r4
             r2 = mem[r5 + 0]            //Load
             if(r4 < 500000000) goto bob //Branch 2
```

The predictors are addressed by the least significant bits of the PC other than the word-offset. Predictors are all initialized to 0s (not taken or strongly not taken).

You are to consider how different branch predictors will behave on this code under different circumstances:

- **Case 1**: The data from the load is even 5 times in a row and then odd once, then even 5 times in a row and then odd once, etc.
- **Case 2**: The data from the load is even 3 times in a row, then odd 3 times in a row, then even 3 times in a row, then odd 3 times in a row, etc.
- **Case 3**: The data from the load is a uniformly random integer.

You are now to consider 2 branch predictors:

- **Predictor 1**: A bimodal predictor with 8 entries, each 2 bits in size.
- **Predictor 2**: A bimodal predictor with 4 entries, each 1 bit in size.

What are the expected prediction rates for each of the following? Your answer must be correct within 0.5%.

	Case 1		**Case 2**		**Case 3**	
	Branch 1	**Branch 2**	**Branch 1**	**Branch 2**	**Branch 1**	**Branch 2**
Predictor 1	5/6 = 83%	100%	2/6 = 33%	100%	50%	100%
Predictor 2	5/6 = 83%	5/6 = 83%	3/6 = 50%	3/6 = 50%	50%	50%

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	
ARN	**PRN**
0	~~7~~ 10
1	9
2	~~4~~ ~~0~~ 1
3	~~11~~ 8
4	5

ROB					
Buffer Number	**PC**	**Executed?**	**Dest. PRN**	**Dest ARN**	
~~0~~	~~40~~	~~N~~ ~~Y~~	~~10~~	~~0~~	
~~1~~	~~44~~	~~N~~ ~~Y~~	~~9~~	~~1~~	
~~2~~	~~48~~	~~Y~~	~~8~~	~~3~~	
~~3~~	~~52~~	~~Y~~			
~~4~~	~~56~~	~~N~~	~~7~~	~~0~~	
~~5~~	~~60~~	~~Y~~	~~11~~	~~3~~	
6	80	N	0	2	← HEAD
7	84	N	1	2	
8					← TAIL

RRAT	
ARN	**PRN**
0	~~2~~ 10
1	~~3~~ 9
2	4
3	~~6~~ 8
4	5

RS							
RS#	**Op Type**	**Op1 Ready?**	**Op1 PRN/value**	**Op2 Ready?**	**Op2 PRN/value**	**Dest PRN**	**ROB**
~~0~~	~~*~~	~~Y~~	~~2~~	~~Y~~	~~-1~~	~~10~~	~~0~~
~~1~~	~~+~~	~~N Y~~	~~10 -2~~	~~Y~~	~~2~~	~~9~~	~~1~~
2	~~+~~	~~N~~	~~9~~	~~Y~~	~~6~~	~~7~~	~~4~~
3	+	Y	0	Y	6	0	6
4	+	N	0	Y	0	1	7

PRF			
PRN	**Value**	**Free**	**Valid**
0	3	~~Y~~ N	N
1	3	~~Y~~ N	N
2	1	~~N~~ Y	~~Y~~ N
3	2	~~N~~ Y	~~Y~~ N
4	-1	N	Y
5	6	N	Y
6	-2	~~N~~ Y	~~Y~~ N
7	6	~~N~~ Y	N
8	6	N	Y
9	~~1~~ 0	N	~~N~~ Y
10	~~9~~ -2	N	~~N~~ Y
11	9	~~N~~ Y	~~Y~~ N
12	0	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 value in it is valid. *A free entry must be marked as invalid*.

cted: the next PC should have been 80. Say that the instruction in memory location 80 is R2=R1+R4 and in 84 is R2=R2+R1. Update the machine to the state where the branch has left the ROB, and the instructions at memory 80 and 84 have been dispatched to the OoO core but not started execution. *When selecting a physical register, 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!*** ***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.***