EECS 470 *Midterm Exam*

Fall 2019

Name: _____ unique name: _____

Sign the honor code:

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

_____

Scores:

NOTES:

- One sheet of notes allowed
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don't spend too much time on any particular problem.
- You have about 90 minutes for the exam.
- There are ***12*** pages including this one.
- **Be sure to show work and explain what you've done when asked to do so.**
- **If you need to use the back of any page, please indicate it**

1. Fill in the blanks. Check the boxes for multiple-choices. [22 points, 2 points per entry]

a. When using P6 style register renaming, if you have 32 reorder buffer entries, 10 reservation stations, and 16 architectural registers you can be sure that you will not need more than _____16_____ entries in the physical register file.

b. When using R10k style register renaming with checkpoint and no early-branch recovery, you check whether branches are mispredicted at ( )

- ❑ execution stage
- ❑ complete stage
- ❑ retire stage

To recover the branch you squash ( ) in the pipeline. (select all that apply)

- ❑ Reorder Buffer
- ❑ Reservation Station
- ❑ Map Table
- ❑ Architectural Map Table
- ❑ Free List

Meanwhile, you recover ( ) from saved checkpoints. (select all that apply)

- ❑ Reorder Buffer
- ❑ Reservation Station
- ❑ Map Table
- ❑ Architectural Map Table
- ❑ Free List

c. One advantage scoreboarding has over an in-order pipeline is

_____our-of-order execution_____

d. 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 ( ).

- ❏ number of outstanding branches
- ❏ clock period
- ❏ clock frequency

e. If a ROB is constantly stalling because it's full, briefly list two ways that can increase performance other than adding more ROB entries:

___increase retire width___, ___reduce # pipeline stages__, ___increase # functional units__.

f. Does scoreboarding support precise state?

- ❏ Yep
- ❏ Nope

g. Latency and throughput are very different such that _____throughput_____ can exploit parallelism while the other one can't.

2. Power and Performance [14 points]

Blizzard Entertainment has been receiving a lot of complaints about poor online experience in their game World of Warcraft, so they are trying to optimize their code by making use of multi-core processors. Assume that 80% of the execution time is spent on TCP packet processing, which can be perfectly parallelized across all of the cores on a processor. The rest of the program is serial, and the performance is directly proportional to available memory bandwidth.

a. Assume they had only a single core on the processor initially. To add another same core to the processor while maintaining the same overall power consumption for cores, they plan to adjust the voltage of the cores. What percentage should they drop the voltage to afford a second core? (You can assume the frequency drop proportionally with the voltage) [4 points]

$1 - \sqrt[3]{0.5}$

b. A new core design consumes 30% less power but is also 20% slower. By applying two cores with the new design, what is the speedup they can achieve compared to the original single-core design? In terms of energy efficiency (throughput / Watt) of cores, does it win or lose? [5 points]

$$S = \frac{1}{0.2+\frac{0.8}{2}} \times 0.8 = \frac{4}{3}$$

Energy efficiency gain $= \frac{4/3}{(1-30\%)*2} = 0.95$, it is losing

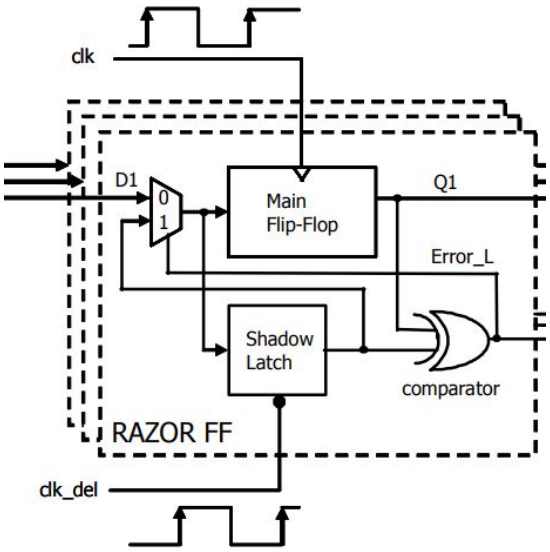
c. Recent advances in memory technology can double memory bandwidth. With the new memory technology, what is the speedup they can achieve by applying the two cores in B. this time compared to the original single-core design with the advanced memory? In terms of energy efficiency (throughput / Watt) of cores, does it win or lose? [5 points]

$$S = \frac{1}{1/9+\frac{8/9}{2}} \times 0.8 = 1.44$$

Energy efficiency gain $= \frac{1.44}{(1-30\%)*2} = 1.02$, it is winning

3. Verilog Implementation [15 points]

Razor is a very famous publication that came out of Michigan. While you can discover the purpose of Razor at a later time, the key idea is that a Razor flip-flop has a regular **flip-flop** and a shadow **latch** that is clocked at a slightly delayed clock. The main flip-flop registers the signal at the posedge of the clock, while the shadow latch latches the signal when the delayed clock is low (the latch is transparent when delayed clock is high, conversely). Then the output of the main flip-flop and the shadow latch are compared. If the two results differ, an error signal is generated, and the output of the shadow latch will be stored in the main flip-flop and the shadow latch in the next cycle instead of its regular input. The block diagram of a Razor flip-flop is shown below.



a. Implement a Razor FF in SystemVerilog with the IOs given below. For full credit, your code should be fully synthesizable, reasonably efficient and not contain any latches other than the shadow latch (Hint: usually we do not allow latches, but this time we are asking you to write a latch). You can omit the reset signal in the implementation. [5 points]

```
module Razor_FF (    input D1,
                     input clk,
                     input clk_del,
                     output logic Q1,
                     output logic Error_L );
  logic real_d1;
  logic shadow_latch;
  assign real_d1 = (error_detected) ? shadow_latch : D1;
  assign error_detected = Q1 ^ shadow_latch;
  always_ff @(posedge clk)
        Q1 <=  real_d1;
  always_comb
        if (clk_del)
              shadow_latch = real_d1;
```

```
////////OR THE BETTER way IS//////////
//always_latch
//        if (clock_delayed)
//                shadow_latch <= real_d1;
//////END BETTER WAY////////////////
endmodule
```

b. Use the 1-bit Razor FF module above to create a 32-bit Razor FF. The interface is given below. [3 points]

```
module Razor_FF_32   (       input [31:0] D,
                             input clk,
                             input clk_del,
                             output logic [31:0] Q,
                             output logic Error_L );
logic [31:0] full_error;
Razor_FF S [31:0]  (.D1(D), .clk(clk), .clk_del(clk_del),.Q1(Q), .Error_L(full_error));
assign Error_L = | full_error;

endmodule
```

c. Assume the input of the Razor FF comes from a rotating priority selector that selects 1 requesters from 4 requesters. Each request has 32-bit data and when a request is valid, the corresponding bit in the request signal will be high. The priority is determined by an internal 5-bit counter. The requester whose index equals to the counter value has the highest priority, then the priority goes from counter + 1, counter + 2… counter + n. For example, when the counter value is 2, request[2] has the highest priority, request [3] has the second highest priority, and request [1] has the lowest priority. The output valid signal is high only if the priority selector is granting to a request. Your code should be reasonably efficient and synthesizable (no latches). [7 points]

```
module rotate_priority_selector (
        input clock, reset,
        input [3:0] request, //request valid bits
        input [3:0] [31:0] data, //request data
        output valid_gnt, // granted valid bit
        output [31:0] D, // granted data
);
        logic [1:0] counter;

        always_ff @(posedge clock) begin
                if (reset) begin
                        counter <= #1 0;
                end else begin
                        counter <= #1 counter + 1;
                end
        end
/************ Your code here ***********/
        logic [1:0] index; //to prevent overflow
        always_comb begin
                valid_gnt = |request;
                D = 32'b0;
                for (int i = 0; i < 4; i = i + 1) begin
                        index = counter + i;
                        if (request[index]) begin
                                D = data[index];
                                break;
                        end
                end
        end

endmodule
```

4. Advanced ROB [29 points]

A ROB needs to store a lot of metadata about the instructions in-flight and can often become a bottleneck in a pipeline due to its limited size. A design team at CacheMoney Inc. is attempting to increase the number of in-flight instructions drastically without having an enormous ROB. Modern processors have ROBs that only keep track of certain types of instructions. Now, remember that the purpose of a ROB is to keep precise state in case of an exception (i.e. memory access errors or branch mispredictions) that would cause a pipeline flush and rollback. They presume, therefore, a ROB only has to keep track of instructions that may cause an exception. Other instructions are tagged with an identifier for an older excepting instruction and are being tracked separately. They may not retire until its associated excepting instruction has retired. For this question, assume a MIPS R10K style microarchitecture with 6-way superscalar issue width. To facilitate their design, they track a reference count on each Told value, that is each time an instruction reads Told from the map table it is incremented, and each time an instruction issues it reduces the reference count on its source operands.

a. What metadata do you need to track for non-excepting instructions and why? [5 points]

T and Told, just like before to support rollback, and the identifier for reasons mentioned before.

b. Aside from waiting for the older excepting instruction to retire, how can non-excepting instructions retire differently than normal R10K retirement? Why? [6 points]

Non-excepting instructions can retire OoO if it's not associated with an older excepting instruction or the older instruction retired AND when the ref count is 0. Because know you don't have to roll back and the ref count tells you there's no false dependency

c. Do excepting instructions in the ROB have to wait until all older instructions have retired to retire? Why? [5 points]

No, they only need to wait for all older excepting instructions in the ROB and the ref count to be 0 to retire for the reason non-excepting instructions can retire OoO

d. Do excepting instructions in the ROB have to wait until all older instructions have retired to roll back? Why? [5 points]

Not necessarily, in theory you can use methods like checkpointing to accommodate that since the older instructions don't need to be flushed.

e. Name one scenario in which this technique won't boost performance [3 points]

If you workload consists mostly of excepting instructions…

f. Would this technique be equally as effective on a P6 style microarchitecture? Why? [5 points]

No. Since there's no true register renaming in P6, tracking the non-excepting instructions and the ref count will be much harder

5. P6 Microarchitecture [20 pts]

On the next pages, you will find a set of charts showing a snapshot of a P6-like microarchitecture. You must advance this machine 6 clock cycles (to the end of cycle #6). Use the cycle-by-cycle state tables to record the contents of each hardware structure at the end of each clock cycle.

Assume the following:

- The machine is non-uniform in superscalar width across each stage. It can dispatch 3, issue 2, complete 2 and retire 3 instructions per cycle. If there are conflicts among instructions, the machine always selects the oldest instructions first.
- Ignore the fetch stage. Assume all instructions have been fetched and are ready for dispatch whenever the out-of-order core allows.
- This machine has 4 integer registers, a 6-entry ROB, and 5 reservation stations.
- There is 1 add functional units with a 1-cycle latency, and 1 fully-pipelined multiply functional unit with a 2-cycle latency (fully-pipelined means the multiply unit has 2 pipeline stages; it can issue a new multiply each cycle, however, multiplies take 2 cycles to execute).
- Assume there is no bypassing in the X stage, but C will bypass to S. Assume reservation stations are freed when they were in lecture and can be reused as soon as they are freed.
- If multiple values are on the CDB, list them all.
- Note that instructions 6-10 are the same as 1-5.

Here is the instruction sequence:

(1) R3 = R1 * R4
(2) R1 = R3 + R2
(3) R4 = R4 + 10
(4) R2 = R2 * R2
(5) R3 = R4 + R2
(6) R2 = R3 *R2
(7) R3 = R1 * R4
(8) R1 = R3 + R2
(9) R4 = R4 + 10
(10) R2 = R2 * R2
(11) R3 = R4 + R2
(12) R2 = R3 *R2

To save you time in writing, the instructions have been pre-filled into the ROB in the solution sheet – be sure to update the head and tail pointers correctly to show when instructions are dispatched and retired. Pay attention to the cycle number on each chart—be sure you fill them out in correct order!

If you make a mistake and need additional blank copies of the fill-in sheet, ask the exam proctor. Make sure the old sheets are torn up and the new ones are stapled to your exam!!!

Cycle #1

ROB							
ht	#	Instruction	T	V	S	X	C
h	1	R3 = R1 * R4	R3				
	2	R1 = R3 + R2	R1				
t	3	R4 = R4 + 10	R4				
	4	R2 = R2 * R2					
	5	R3 = R4 + R2					
	6	R2 = R3 *R2					

Map Table	
Reg	T+
R1	ROB #2
R2	
R3	ROB #1
R4	ROB #3

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R3 = R1 * R4	R3			R1	R4
2	R1 = R3 + R2	R1	ROB #1			R2
3	R4 = R4 + 10	R4			R4	
4						
5						

CDB	
T1	
T2	

Cycle #2

ROB							
ht	#	Instruction	T	V	S	X	C
h	1	R3 = R1 * R4	R3		C2		
	2	R1 = R3 + R2	R1				
	3	R4 = R4 + 10	R4		C2		
	4	R2 = R2 * R2	R2				
t	5	R3 = R4 + R2	R3				
	6	R2 = R3 *R2					

Map Table	
Reg	T+
R1	ROB #2
R2	ROB #4
R3	ROB #5
R4	ROB #3

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R3 = R1 * R4	R3			R1	R4
2	R1 = R3 + R2	R1	ROB #1			R2
3	R4 = R4 + 10	R4			R4	
4	R2 = R2 * R2	R2			R2	R2
5	R3 = R4 + R2	R3	ROB #3	ROB #4		

CDB	
T1	
T2	

Cycle #3

ROB							
ht	#	Instruction	T	V	S	X	C
h	1	R3 = R1 * R4	R3		C2	C3	
	2	R1 = R3 + R2	R1				
	3	R4 = R4 + 10	R4		C2	C3	
	4	R2 = R2 * R2	R2		C3		
	5	R3 = R4 + R2	R3				

Map Table	
Reg	T+
R1	ROB #2
R2	ROB #6
R3	ROB #5
R4	ROB #3

t	6	R2 = R3 *R2	R2				

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R2 = R3 *R2	R2	ROB #5	ROB #4		
2	R1 = R3 + R2	R1	ROB #1			R2
3						
4	R2 = R2 * R2	R2			R2	R2
5	R3 = R4 + R2	R3	ROB #3	ROB #4		

CDB	
T1	
T2	

Cycle #4

ROB							
ht	#	Instruction	T	V	S	X	C
h	1	R3 = R1 * R4	R3		C2	C3+	
	2	R1 = R3 + R2	R1				
	3	R4 = R4 + 10	R4	[R4]	C2	C3	C4
	4	R2 = R2 * R2	R2		C3	C4	
	5	R3 = R4 + R2	R3				
t	6	R2 = R3 *R2	R2				

Map Table	
Reg	T+
R1	ROB #2
R2	ROB #6
R3	ROB #5
R4	ROB #3+

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R2 = R3 *R2	R2	ROB #5	ROB #4		
2	R1 = R3 + R2	R1	ROB #1			R2
3						
4						
5	R3 = R4 + R2	R3		ROB #4	R4	

CDB	
T1	ROB #3
T2	

Cycle #5

ROB							
ht	#	Instruction	T	V	S	X	C
h	1	R3 = R1 * R4	R3	[R3]	C2	C3+	C5
	2	R1 = R3 + R2	R1		C5		
	3	R4 = R4 + 10	R4	[R4]	C2	C3	C4
	4	R2 = R2 * R2	R2		C3	C4+	
	5	R3 = R4 + R2	R3				
t	6	R2 = R3 *R2	R2				

Map Table	
Reg	T+
R1	ROB #2
R2	ROB #6
R3	ROB #5
R4	ROB #3+

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R2 = R3 *R2	R2	ROB #5	ROB #4		
2	R1 = R3 + R2	R1			R3	R2
3						

CDB	
T1	ROB #1
T2	

4						
5	R3 = R4 + R2	R3		ROB #4	R4	

Cycle #6

ROB							
ht	#	Instruction	T	V	S	X	C
t	1	R3 = R1 * R4	R3				
h	2	R1 = R3 + R2	R1		C5	C6	
	3	R4 = R4 + 10	R4	[R4]	C2	C3	C4
	4	R2 = R2 * R2	R2	[R2]	C3	C4+	C6
	5	R3 = R4 + R2	R3		C6		
	6	R2 = R3 *R2	R2				

Map Table	
Reg	T+
R1	ROB #2
R2	ROB #6
R3	ROB #1
R4	ROB #3+

Reservation Stations						
#	Instruction	T	T1	T2	V1	V2
1	R2 = R3 *R2	R2	ROB #5			R2
2	R3 = R1 * R4	R3	ROB #2			R4
3						
4						
5	R3 = R4 + R2	R3			R4	R2

CDB	
T1	ROB #4
T2	