EECS 470 *Midterm Exam* *Answer Key*

Fall 2024

Name: ______________________________________ Uniqname: ____________________

Sign the honor code:

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

____

#	Points
1	**/ 12**
2	**/ 8**
3	**/ 10**
4	**/ 15**
5	**/ 10**
6	**/ 10**
7	**/ 15**
8	**/ 20**
Total	**/100**

NOTES:

- Open book and open notes
- Calculators are allowed, but no smart watches, cell phones, earbuds, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for this exam.
- There are **14** pages including this one.
- **Be sure to show your work and explain what you've done when asked to do so.**

1) Fill in the blank or circle the <u>best</u> answer [12 points, +2 per correct answer]

A. A processor with a frequency of 250 MHz will have a clock period of __ 4ns ____

B. As you increase the size of the ROB, you expect the IPC of the processor on a given workload to go **up / down**, and you expect the clock period to go **up / down**.

C. In the P6 scheme, the result is written to the ARF when it is **issuing / completing / committing**.

D. In the R10K scheme, the result is written to the PRF when it is **issuing / completing / committing**.

E. Increasing the number of cores by a factor of 8 for a program that is 20% serial and 80% parallelizable results in a _3.33_x overall speedup

$1/(0.2 + 0.8/8) = (1/0.3) = 10/3 = 3.333$x speedup.

F. The head of the ROB in a P6 architecture points to the **oldest / newest** instruction that has **dispatched / completed / committed**.

Extra credit:

G. A *P6-style* processor capable of simultaneously running 2 threads on the same processor needs a register file size of **32 / 48 / 64 / 80 / 128** if there are 48 architected registers (24 per hardware thread) and each thread has a ROB size of 16.

2) Extending VeriSimpleV [8 points]

A team of intelligent baboons at Microchimp Solutions is developing a new state-of-the-art processor based on the 5-stage VeriSimpleV pipeline you developed in Project 3. They have implemented caches and eliminated structural hazards from memory. Forwarding still occurs to the EX stage, and branches are still resolved in the MEM stage. In addition, the company has decided to add a small set of floating-point operations to the processor. To maintain a short clock period, floating point operations are pipelined, and are resolved in the memory stage.

Assume a given program consists of 25% loads, 20% stores, 20% branches, 25% integer ALU operations, and 10% floating-point operations. If 30% of the branches are taken and 15% of all instructions for each instruction type are dependent on the instruction in front of them, **what is the expected CPI of the processor on this program?** Show your work. Assume static not-taken branch prediction.

CPI = 1 (for all instructions)
+ (Load * followed by dependant instruction * 1 cycle)
+ (Branches * which are taken * branch penalty in cycles)
+ (flops * followed by dependant instruction * 1 cycle)

CPI = 1 + 0.25 * 0.15 + 0.20 * 0.30 * 3 + 0.10 * 0.15
= 1.2325

3) Wonka's Chocolate Factory [10 points]

A. A software team at Wonka's chocolate factory is developing a new Oompa Loompa scheduling application. The team has prototyped the application on a single-processor system and found that it does not meet their performance targets. However, they are hoping to get a significant performance boost by parallelizing their code over four cores.

   Some fraction f of the application can execute in parallel over the **four cores** with perfectly linear speedup (i.e. it is four times as fast as on the single processor system). The remaining fraction executes on a single core at the same speed as the development system. **What must the fraction f be to double overall performance?**

   Applying Amdahl's law: $2 = 1 / (1-f) + f/4$. Solving for f, we find that $f=2/3$

B. A hardware team at Wonka's chocolate factory is trying to design a new high-performance processor for their server farms. Their new processor executes at 1000 MIPS at an operating frequency of 2 GHz and a voltage of 1.3V. At this operating point, the chip requires 80W of power. Assume that operating frequency and voltage are linearly related, and performance is proportional to frequency.

   The server farm can budget at most 60W for each processor. **What is the performance (in MIPS) of the processor if voltage and frequency are scaled to meet the 60W power budget?** Round to the nearest whole number.

   P ~ f^3
   Frequency scale = (60/80)^(⅓) = 0.909
   New performance = 0.909 * 1000 MIPS = 909 MIPS

4) Short Answer [15 points, +2.5 points per correct answer]

Answer the following questions in at most two sentences (they can be shorter).

A. Why is it that in Scoreboarding, the entry can only be freed in the writeback stage?

There is no ROB to track the instruction after it leaves the FUST, so the scoreboard must track the instruction and keep the association between the functional unit and corresponding instruction.

B. What would happen if you tried to improve the Scoreboarding scheme by freeing a Scoreboard entry earlier?

In short, it would not work. Some valid reasons are:

- Couldn't associate functional units to corresponding instructions
- Sourcing of values would be incorrect
- Would get WAR and WAW hazards

Recall that in the P6 design, the Reservation Station entry can be freed when an instruction enters the execute stage. Our examples using the P6 design assume that every functional unit is fully pipelined. Let's imagine that one of the functional units was designed to support a special accelerator instruction that took multiple cycles to execute, during which no subsequent instruction could start execution in that functional unit.

C. How would you change the algorithm to free Reservation Station entries and maintain correctness? Your answer can be three sentences long.

You would need to implement proper backpressure to ensure that an entry doesn't issue to that functional unit until the current in-flight instruction has completed.

If they assume each RS entry is associated with a specific functional unit, you would have to wait to free that entry until the functional unit finished executing.

[problem continues on next page]

Recall the R10K microarchitecture we covered in class, logical (architected) registers are mapped to physical registers (in multiple different map tables). The lifetime of a logical-physical mapping is the length of time during which the mapping stays unchanged (e.g. `r5` maps to `p2`).

D. How many times can a physical register be read during the lifetime of a given logical-physical pair?

Infinitely many times.

E. How many times can a physical register be assigned a new value during the lifetime of a given logical-physical pair?

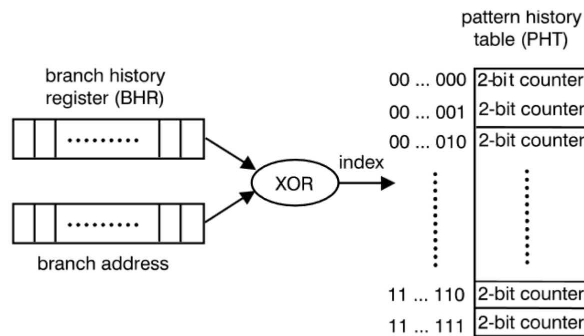
Once, when it is written the first time. The lifespan is over when the logical register is renamed to another physical register with a new value.

F. Why are the answers from problems D and E important for the operation of the R10K microarchitecture in terms of resource allocation?

This defines how register renaming must work. We must allocate a new physical register for each new write to an architected register, and then we can recycle the register once its lifetime is over. This allows us to maintain proper functionality with a finite number of physical registers.

5) Dynamic Branch Prediction [10 points]

You are designing a microprocessor that implements dynamic branch prediction via GShare branch predictor with a **kibibit (2^10 bit) pattern history table**:



[problem continues on the following page]

Your processor can checkpoint for up to four inflight branches. You would like to extend your GShare branch predictor to enable recovery of the BHR (branch history register) upon branch mispeculation. To do this:

- Upon reset, the bmask-reg gets assigned `bmask_reg = 4'b0000`.
- As branches are decoded in (speculated) program order, the `bmask_reg` is updated by setting the first unset bit (e.g., `4'b1010 -> 4'b1110`). The branch keeps a copy of this new `bmask_reg` with it as it travels through the pipeline, as well as `bmask_mask = 4'b0100` which specifies which bit was set.
- When a branch completes, and speculation is confirmed:
  - If speculation is correct: set `bmask_reg = bmask_reg XOR bmask_mask` for both the `bmask_reg` in decode, and the copy stored by each branch instruction in flight.

Suppose a branch is determined to have been mispredicted during its execution phase:

A. How many additional flip-flops are required in the branch predictor to support recovering the BHR to its pre-speculated state, and how are they used?

If checkpointing: (9 bits) x (4 checkpoints) = 36 flip-flops

If a branch is mispeculated, restore the BHR to the snapshot taken when the branch was dispatches

If doing the "extra credit" solution: 3 - one flop for each checkpoint but the last. 4 is also an appropriate answer. These are used to keep previous history so you can recover it

I. Extra Credit: How do you do this *without* copying the BHR?

(Second approach given to previous answer) The 9-bit BHR is extended by the extra 3 to 4-bits. Only the first 9 bits of the 13-bit shift register are used to access the global history. On a mispeculation, shift over the appropriate number of bits to recover the previous global history.

B. On detection of mispeculation during execution, what step(s) must be taken to recover the BHR to its pre-speculated state?

For checkpointing approach:

- Copy the checkpointed BHR into the main BHR
- Clear all checkpoints for branches that follow the mispeculated branch

For "extra credit" approach:

- Set dist to the distance between the tail pointer and the mispeculated pointer.
- Shift BHR by -dist
- Set tail pointer to the BCID of the mispeculated branch.

6) SystemVerilog Recursive Generate [10 points]

SystemVerilog supports **generate constructs**. These blocks allow hardware designers to instantiate hardware blocks inside a module conditioned on elaboration time constants, such as parameters. This allows for *recursively defined modules* (of course, the recursion must terminate, or else the hardware module will be infinitely large). For example, we can define a module which sums a list of 32-bit integers:

```
module list_sum #(parameter LEN=32)
(
    input  [LEN-1:0] [31:0] i_list,
    output           [31:0] o_sum
);

  generate
    if (LEN == 1) begin // Base Case
      assign o_sum = i_list[0];
    end else begin // Recursive Case
      logic [31:0] rec_sum;
      list_sum #(LEN-1) u_rec_list_sum (
        .i_list(i_list[LEN-2:0]),
        .o_sum(rec_sum)
      );
      assign o_sum = rec_sum + i_list[LEN-1];
    end
  endgenerate

endmodule
```

Rewrite the conditional generate statement in `list_sum` so that the complexity of the resulting hardware is proportional to `log(LEN)` in circuit depth. You may assume `LEN` is always equal to 2^a for some non-negative integer, *a*. You do not need to re-write the module header.

```
if (LEN == 1) begin // Base Case
    assign o_sum = i_list[0];
end else begin // Recursive Case
    logic [31:0] rec_sum_a, rec_sum_b;
    list_sum#(LEN / 2) u_rec_list_sum_a(
      .i_list(i_list[LEN-1:LEN/2]),
      .o_sum(rec_sum_a)
    );
    list_sum#(LEN / 2) u_rec_list_sum_b(
      .i_list(i_list[LEN/2-1:0]),
      .o_sum(rec_sum_b)
    );
    assign o_sum = rec_sum_a + rec_sum_b;
end
```

7) SystemVerilog UART Design [15 points]

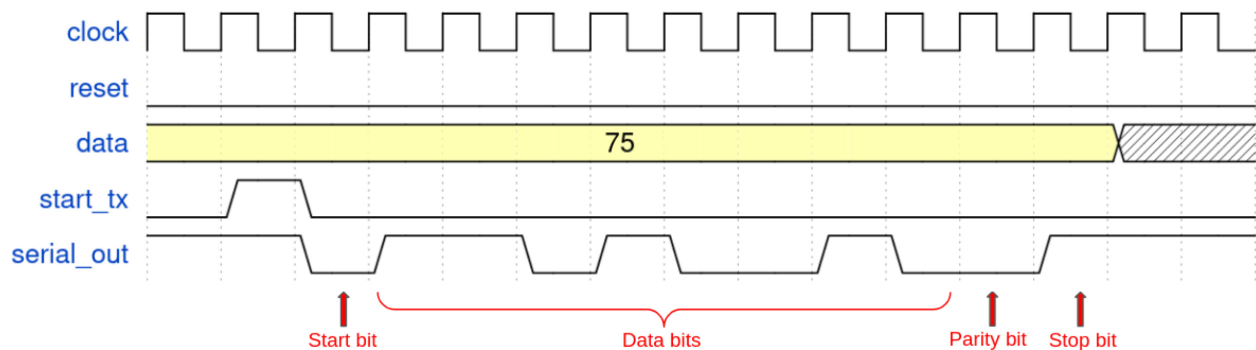
A UART (universal asynchronous receiver/transmitter) is a hardware device used for allowing serial communication between several devices. UARTs are commonly used today for embedded systems applications, and microcontrollers will often have one or more UARTs available.

UART messages are transmitted as "frames," which are serial sequences of bits with a defined length and specific delimiters. Many frame configurations are possible. In this problem, you will consider a UART frame with the following characteristics:

- One start bit – logic low, indicates the start of a new frame
- Eight data bits – the data to be transmitted, in **little-endian bit order** (least significant bit first)
- One parity bit – an error-checking mechanism, logic high if the number of **data bits** which are high is **odd** (referred to as *even parity*, since the additional high parity bit creates an even number of 1s in the frame)
- One stop bit – logic high, indicates the end of the frame

When no frame is being transmitted, the UART holds the line **high**.

Here is an example of a UART frame transmitting the number 75 (ascii 'K', binary 01001011). One bit is sent per clock cycle:



Implement a SystemVerilog module, `uart_tx`, defined on the following page. We have given you an enum of states for your implementation.

- Assume the input data signal is registered outside of the module (i.e. the data signal is valid throughout the entirety of the UART frame).
- When reset is asserted, the module should transition the idle state. The reset must be synchronous.

```
module uart_tx (
     input  logic        clock,       // one bit sent per clock
     input  logic        reset,       // return UART to idle state
     input  logic [7:0] data,         // data to be transmitted
     input  logic        start_tx,    // begin sending (data valid)
     output logic        serial_out   // data tx output
);
     typedef enum {IDLE, START, DATA, PARITY, STOP} state_t;
     state_t state, state_n;
     (ONE OF MANY SOLUTIONS)

     logic [2:0] data_count;

     always_comb begin
          state_n    = state;
          serial_out = 1;
          case (state)
               IDLE: begin
                    state_n    = start_tx ? START : IDLE;
                    serial_out = 1;
               end
               START: begin
                    state_n    = DATA;
                    serial_out = 0;
               end
               DATA: begin
                    state_n    = data_count == 7 ? PARITY : DATA;
                    serial_out = data[data_count];
               end
               PARITY: begin
                    state_n    = STOP;
                    serial_out = ^data;
               end
               STOP: begin
                    state_n    = IDLE;
                    serial_out = 1;
               end
          endcase
     end

     always_ff @(posedge clock) begin
          if (reset) begin
               state      <= IDLE;
               data_count <= 0;
          end else begin
               state      <= state_n;
               data_count <= state == DATA ? data_count + 1 : 0;
          end
     end

endmodule
```

8) R10K Cycles [20 points]

On the next pages, you will find a set of charts showing a snapshot of an MIPS R10K-like microarchitecture after one cycle executing a sequence of instructions. You must advance this machine 5 additional 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:

- Assume the machine is a **2-wide superscalar** (e.g., it can dispatch, issue, complete, and commit at most 2 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 architectural registers, a 5-entry ROB, 4 reservation stations, and 8 physical registers.
- There are **2 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 all reservation stations begin empty.
- Assume the initial map table is `rX → pX`.
- Assume the ROB is initially empty (i.e., that `h = t`).
- Assume there is no bypassing in the X stage, but C will bypass to S through the physical register file.
- Assume reservation stations are freed as early as possible and can be reused as soon as they are freed.
- The **tail of the ROB points to the youngest instruction**, not the next empty spot.

Here is the instruction sequence:

```
(1) R3 = R1*R4
(2) R2 = R2*3
(3) R3 = R2+R4
(4) R1 = R3+R1
(5) R4 = R2+R1
```

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 committed. Cycle 1 has been completed for you.

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

R10K Data Structures Cycle 1

ROB							
ht	#	Instruction	T	Told	S	X	C
h	1	R3 = R1 * R4	p5	p3			
t	2	R2 = R2 * 3	p6	p2			
	3	R3 = R2 +R4					
	4	R1 = R3 + R1					
	5	R4 = R2 + R1					

Map Table	
Reg	T+
r1	p1+
r2	p2+
r3	p5
r4	p6

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p3
r4	p4

Free List
~~p5, p6~~, p7, p8

Reservation Stations					
#	busy	op	T	T1	T2
1	y	mul	p5	p1+	p2+
2	y	mul	p6	p2+	
3	n				
4	n				

CDB
T

R10K Data Structures Cycle 2

ROB							
ht	#	Instruction	T	Told	S	X	C
h	1	R3 = R1 * R4	p5	p3	c2		
	2	R2 = R2 * 3	p6	p2			
	3	R3 = R2 +R4	p7	p5			
t	4	R1 = R3 + R1	p8	p1			
	5	R4 = R2 + R1					

Map Table	
Reg	T+
r1	p8
r2	p6
r3	p7
r4	p4+

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p3
r4	p4

Free List

Reservation Stations					
#	busy	op	T	T1	T2
1	y	mul	p5	p1+	p2+
2	y	mul	p6	p2+	
3	y	alu	p7	p6	p4+
4	y	alu	p8	p7	p1+

CDB
T

R10K Data Structures Cycle 3

ROB							
ht	#	Instruction	T	Told	S	X	C
h	1	R3 = R1 * R4	p5	p3	c2	c3+	
	2	R2 = R2 * 3	p6	p2	c3		
	3	R3 = R2 +R4	p7	p5			
t	4	R1 = R3 + R1	p8	p1			
	5	R4 = R2 + R1					

Map Table	
Reg	T+
r1	p8
r2	p6
r3	p7
r4	p4+

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p3
r4	p4

Free List

Reservation Stations					
#	busy	op	T	T1	T2
1	n				
2	y	mul	p6	p2+	
3	y	alu	p7	p6	p4+
4	y	alu	p8	p7	p1+

CDB
T

R10K Data Structures Cycle 4

ROB							
ht	#	Instruction	T	Told	S	X	C
h	1	R3 = R1 * R4	p5	p3	c2	c3+	
	2	R2 = R2 * 3	p6	p2	c3	c4+	
	3	R3 = R2 +R4	p7	p5			
t	4	R1 = R3 + R1	p8	p1			
	5	R4 = R2 + R1					

Map Table	
Reg	T+
r1	p8
r2	p6
r3	p7
r4	p4+

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p3
r4	p4

Free List

Reservation Stations					
#	busy	op	T	T1	T2
1	n				
2	n				
3	y	alu	p7	p6	p4+
4	y	alu	p8	p7	p1+

CDB
T

R10K Data Structures Cycle 5

ROB							
ht	#	Instruction	T	Told	S	X	C
h	1	R3 = R1 * R4	p5	p3	c2	c3+	c5
	2	R2 = R2 * 3	p6	p2	c3	c4+	
	3	R3 = R2 +R4	p7	p5			
t	4	R1 = R3 + R1	p8	p1			
	5	R4 = R2 + R1					

Map Table	
Reg	T+
r1	p8
r2	p6
r3	p7
r4	p4+

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p3
r4	p4

Free List

Reservation Stations					
#	busy	op	T	T1	T2
1	n				
2	n				
3	y	alu	p7	p6	p4+
4	y	alu	p8	p7	p1+

CDB
T
p5

R10K Data Structures Cycle 6

ROB							
ht	#	Instruction	T	Told	S	X	C
	1	R3 = R1 * R4					
h	2	R2 = R2 * 3	p6	p2	c3	c4+	c6
	3	R3 = R2 +R4	p7	p5	c6		
	4	R1 = R3 + R1	p8	p1			
t	5	R4 = R2 + R1	p3	p4			

Map Table	
Reg	T+
r1	p8
r2	p6+
r3	p7
r4	p3

Arch. Map	
Reg	T+
r1	p1
r2	p2
r3	p5
r4	p4

Free List

Reservation Stations					
#	busy	op	T	T1	T2
1	y	alu	p3	p6+	p8
2	n				
3	y	alu	p7	p6+	p4+
4	y	alu	p8	p7	p1+

CDB
T
p6