Name _____

We strongly recommend that you do not read the solutions before attempting the exam on your own!

CS4290 / CS6290 / ECE4100 / ECE6100

Midterm I SOLUTIONS

Fall 2022

September 29, 2022

Part A: Short Questions (25 points)

Question A.1 (10 points)

For each of the following questions, answer **True** or **False**. **Briefly justify your answer for credit.**

1. After many years of research, branch prediction was finally proven to universally outperform every known instance of multipath execution.

False, predicated execution is sometimes more effective. If the average cost of recovering from misprediction ends up higher than that of executing both paths, the latter approach is preferable.

2. Both Tomasulo's algorithm and scoreboarding enable correct out-of-order execution, but Tomasulo's algorithm is more effective in extracting ILP.

True, Tomasulo **eliminates** WAW and WAR hazards, unlike scoreboarding, thus improving the ability of the processor to convert the available ILP to higher IPC.

3. Compiler code transformations can only be effective in improving performance if the transformed code consists of fewer instructions compared to the original code.

False, Iron Law indicates that the number of instructions comprising the program is only one of three factors affecting performance. Fewer slower instructions could overall hamper performance.

4. IPC is a metric that can always be used to accurately evaluate the performance of parallel programs executing on a multiprocessor system.

False, spinning on a lock results in high IPC, but does not correspond to higher performance.

5. "Flynn's bottleneck" refers to the bottleneck imposed by the number of architectural registers on the maximum ILP a processor can extract.

False, Flynn's bottleneck states that the number of instructions executed per cycle is capped by the number of instructions fed into the pipeline per cycle.

Question A.2 (9 points)

The Iron Law of processor performance states that program's runtime is defined as the following product: (Instructions/Program) x (Cycles/Instruction) x (Time/Cycle)

Assume we have a system with a scalar pipelined processor and we are considering a number of modifications. For each of these modifications, mark how each of the three Iron Law terms is affected. Your three options are: **(i) Likely increases, (ii) likely decreases, (iii) no effect. Briefly justify your answers for credit.**

	Instructions/Program	Cycles/Instruction	Time/Cycle
Break each pipeline stage into two	No effect	Incr – more cycles to traverse pipeline	Decr – potentially shorter critical path
Add a huge BTB to cover all branches	No effect	Decr – fewer stalls as branch target is more often known	Incr – access latency may increase critical path
Use a compiler that performs extensive loop unrolling	Decr (slightly) Note: remember the # of instructions per program in the Iron Law refers to **dynamic** instruction count, not static.	Decr – less branch prediction OR Incr – instruction bloat hurts iCache locality	No effect

Name _____

Question A.3 (6 points)

Assume an N-stage M-way superscalar in-order pipeline with N>10 and without any bypass paths.

(i) The decode stage, which checks for RAW dependencies, is at stage 4. How many pairwise checks must be performed at each cycle to determine whether any of the instructions at the decode stage must stall? Assume that the last stage's writeback happens in the first half of the cycle, as in lab 2.

$$2 * M^2 * (N - 5) + 2 * \sum_{i=1}^{M} M - i$$

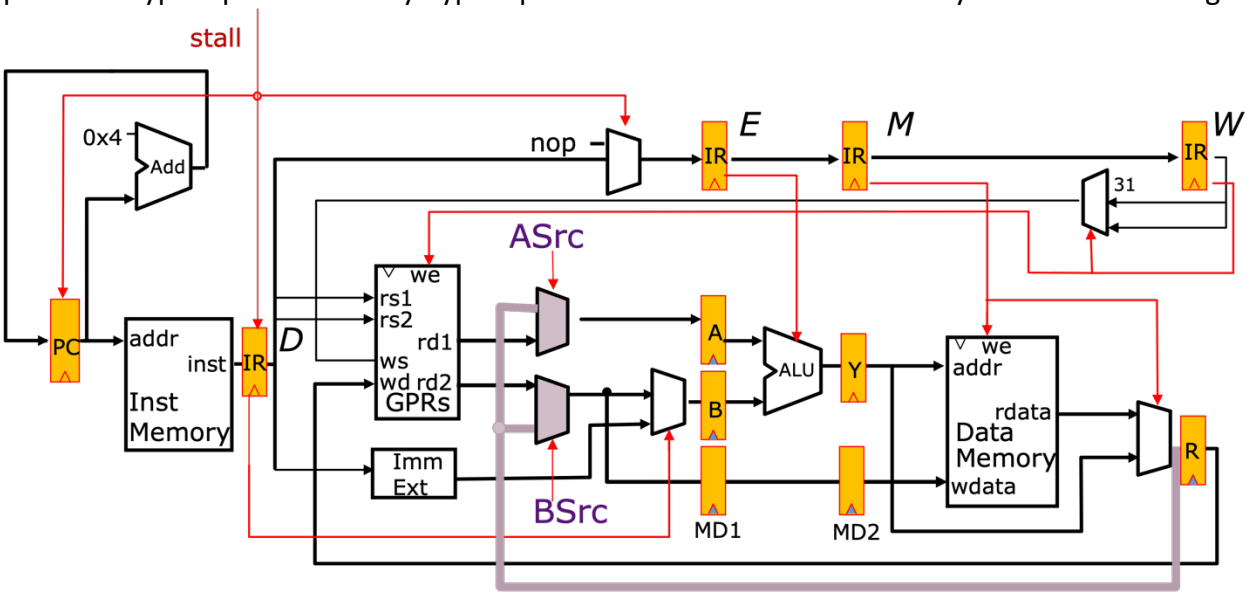
Checks with all pipes of stages ahead of the decode stage, plus checks within decode stage (two operands of younger insn with one dest of older insn)

(ii) The pipeline features branch prediction support, and branch resolution occurs at stage 6. In the event of branch misprediction, what is the **maximum** number of instructions in the pipeline that may have to be flushed?

5*M + M-1 = 6M – 1
M instructions from each of the preceding stages, plus up to M-1 instructions within the stage where the resolution occurs.

Part B: 5-Stage Pipeline (20 points)

The following figure shows a scalar in-order 5-stage pipeline with **partial bypassing** – note the provided bypass paths: the only bypass path available is from the Memory to the Decode stage.



Assume that, unlike Lab 2, the same register in the register file *cannot* be both written and read at the same cycle. **Ignore all instructions that change control flow (Jumps and Branches) for all questions in Part B.**

Question B.1 (4 points)

Give an example sequence of two or more instructions that will lead to a stall in this pipeline. Clearly specify in your example

(i) what dependence causes the stall, and

(ii) for how many cycles.

A few different examples possible here – simplest one is just two back-to-back ALU instructions with RAW dependence, which would cause 1 cycle stall.

Name _____

Question B.2 (6 points)

Write down the *complete* logical expression that describes when the pipeline must stall as a function of the following terms (***Note: you may not need to use all the terms***):
IR_D.op_type, IR_E.op_type, IR_M.op_type, IR_W.op_type,
IR_D.src1_needed, IR_D.src1_reg, IR_D.src2_needed, IR_D.src2_reg,
IR_E.dest_needed, IR_E.dest_reg, IR_M.dest_needed, IR_M.dest_reg, IR_W.dest_needed, IR_W.dest_reg.
IR_D, IR_E, IR_M, and IR_W represent the instructions in the latches preceding the Decode, Execute, Memory and Writeback stage, respectively.

stall =

IR_E.dest_needed **and** [(IR_D.src1_needed **and** IR_D.src1_reg == IR_E.dest_reg) **or** (IR_D.src2_needed **and** IR_D.src2_reg == IR_E.dest_reg)]
or
IR_W.dest_needed **and** [IR_D.src1_needed **and** (IR_D.src1_reg == IR_W.dest_reg **and not** X) **or** (IR_D.src2_needed **and** IR_D.src2_reg == IR_W.dest_reg **and not** Y)]

Where

X = IR_D.src1_needed **and** IR_{DE}.src1_reg == IR_M.dest_reg
Y = IR_D.src2_needed **and** IR_{DE}.src2_reg == IR_M.dest_reg

Note: Conditions X and Y are there to avoid unnecessary stalls if there's a RAW both between ID and Mem, and between ID and WB. The forward path from Mem can provide the latest value.

Question B.3 (6 points)

An instruction has an *i*-dependency when, at the cycle it first attempts to advance through the decode stage, it has a true dependence with an instruction currently residing *i* pipeline stages ahead of it. Thus, an instruction may have a 1-dependency, 2-dependency, or 3-dependency if it depends on the output of an instruction currently in the EX, MEM, or WB stage, respectively.

What is the CPI of Question B.1's 5-stage pipeline under the following assumptions?

- 10% of instructions *only* have a 1-dependency.
- 25% of instructions *only* have a 2-dependency.
- 20% of instructions *only* have a 3-dependency.
- All aforementioned dependencies are between ALU operations.

Justify your answer to receive credit.

1-dependency and 3-dependency always stalls for one cycle
2-dependency never stalls

CPI = 1 + 10% * 1 + 20% * 1 = 1.3

Question B.4 (4 points)

Using the definition for i-dependency introduced in Question B.3, what is the CPI of Question B.1's 5-stage pipeline under the following assumptions?

- 15% of instructions have *both* a 1-dependency and a 2-dependency.
- 5% of instructions have *both* a 1-dependency and a 3-dependency.
- All aforementioned dependencies are between ALU operations.

Justify your answer to receive credit.

CPI = 1 + 15% * 3 + 5% * 1 = 1.5

The tricky bit of this question is the 3-cycle cost for 15% of the instructions. You can't forward a value and store it until a following cycle. So when you can forward the 1-dependency from Mem, you can't forward the 2-dependency anymore, as the instruction is in WB, so the forwarding path is not useful for instructions with both 1- and 2-dependency.

Name _____

Part D: Out-of-Order Execution (30 points)

Question D.1 (6 points)

Suppose you have an ISA that introduces a new **MAX** instruction with the following syntax and semantics:

MAX R1, R2, R3 ; R1 = max(R1, R2, R3)

For the following instruction stream, identify **all** RAW (Read After Write), WAR (Write After Read) and WAW (Write After Write) dependencies in the following table.

If there are multiple dependencies, you must list them all.

Do not worry about memory dependencies for this question. The dependencies between I4 and older instructions have been filled for you.

I1: LW R0, 0(R1)
I2: LW R4, 42(R0)
I3: MAX R0, R2, R5
I4: ADD R0, R2, R0
I5: MAX R4, R3, R1

Earlier (Older) Instruction

Current Instruction	I1	I2	I3	I4
I1				
I2	RAW			
I3	RAW, WAW	WAR		
I4	WAW	WAR	RAW, WAW	
I5	--	RAW, WAW	--	--

Instruction Dependencies

The tricky bit here was to realize the format of the new instruction: it has three source operands

Next, we investigate the operation of an **Out-of-Order Processor with an ROB**. The processor holds all data values in a physical register file (**PRF**), and uses a register alias table (**RAT**) to map from architectural to physical register names. A free list is used to track which physical registers are available for use. A reorder buffer (**ROB**) contains the bookkeeping information for managing the out-of-order execution (but it does not contain any register data values).

The processor has the following pipeline stages

- **Fetch:** The **next PC** is fetched. **The processor features a BHT in this stage**, which is used to determine the next instruction that will be fetched whenever a branch instruction is encountered.
- **Decode:** The instruction from the Fetch Buffer is decoded.
- **Issue:** The instruction from the Decode Buffer is added into the ROB.
- **Execute:** The instructions in the ROB which are ready to execute start executing. Branch Instructions that were mis-predicted update the next PC to be fetched this cycle.
- **Writeback:** Execute units writeback results on the CDB which updates the ROB and Physical Register File.
- **Commit:** Instructions are committed in-order from the ROB. The BHT is updated in this stage.

The processor uses a 4-entry BHT with a simple 1-bit predictor, indexed by PC[3:2].

For Questions D-2, D-3 and D-5, you need to update the processor state (in all tables) after the event described in that question has occurred.

The starting state in each question is the same, and the event specified in each question is the ONLY event that takes place for that question.

```
        0x1000  LW   R1, 0(R2)
        0x1004  ADD  R3, R2, R4
        0x1008  ADD  R1, R3, R3
        0x100C  BNEZ R4, skip
        0x1010  LW   R3, 0(R2)
skip:   0x1014  SUB  R1, R3, R2
        0x1018  ADD  R4, R3, R1
        0x101C  ADDI R4, R4, 4
```

Name _____

Question D.2 (5 points)

Fill out the new contents of
(i) ROB, (ii) PRF, (iii) Register Free List, (iv) RAT and (v) ROB head/tail pointers
when the first ADD operation completes (before the LW operation that is ahead of it).

Physical Registers		
P0	7475	p
P1	2013	p
P2	42	p
P3	7	p
P4		
P5	2020	p
P6		
P7		
P8		

Free List
P7
P8

RAT		
	Before	After
R1	P6	
R2	P1	
R3	P5	
R4	P3	

BHT	
00	0
01	1
10	0
11	1

```
        0x1000  LW   R1, 0(R2)
        0x1004  ADD  R3, R2, R4
        0x1008  ADD  R1, R3, R3
        0x100C  BNEZ R4, skip
        0x1010  LW R3, 0(R2)
skip:0x1014  SUB R1, R3, R2
        0x1018  ADD R4, R3, R1
        0x101C  ADDI R4, R4, 4
```

next to commit: RS id 0

next available: RS id 4

Reorder Buffer (ROB)										
RS id	**use**	**ex**	**op**	**p1**	**PR1**	**p2**	**PR2**	**Rd**	**LPRd**	**PRd**
0	x		LW	p	P1			R1	P0	P4
1	x	x	ADD	p	P1	p	P3	R3	P2	P5
2	x		ADD	p	P5	p	P5	R1	P4	P6
3	x		BNEZ	p	P3					
4										
5										
6										

Question D.3 (8 points)

Fill out the new contents of
(i) ROB, (ii) PRF, (iii) Register Free List, (iv) RAT and (v) head/tail pointers
when the next instruction issues.

Physical Registers		
P0	7475	p
P1	2013	p
P2	42	p
P3	7	p
P4		
P5		
P6		
P7		
P8		

Free List
~~P7~~
P8

RAT		
	Before	After
R1	P6	P7
R2	P1	
R3	P5	
R4	P3	

BHT	
00	0
01	1
10	0
11	1

```
        0x1000  LW   R1, 0(R2)
        0x1004  ADD  R3, R2, R4
        0x1008  ADD  R1, R3, R3
        0x100C  BNEZ R4, skip
        0x1010  LW R3, 0(R2)
skip:0x1014  SUB R1, R3, R2
        0x1018  ADD R4, R3, R1
        0x101C  ADDI R4, R4, 4
```

next to commit: RS id 0

next available: RS id 5

Reorder Buffer (ROB)										
RS id	**use**	**ex**	**op**	**p1**	**PR1**	**p2**	**PR2**	**Rd**	**LPRd**	**PRd**
0	x		LW	p	P1			R1	P0	P4
1	x		ADD	p	P1	p	P3	R3	P2	P5
2	x		ADD		P5		P5	R1	P4	P6
3	x		BNEZ	p	P3					
4	x		SUB		P5	p	P1	R1	P6	P7
5										
6										

The next instruction is at skip, because the branch is taken!

Question D.4 (2 points)

Assuming the state that you reached **at the end of the previous question (D.3)**, how many new instructions can be issued before any older instruction **commits**? **Why?**

Only one, because there's only one physical register left in the free list.

Name _____

Question D.5 (6 points)

Fill out the new contents of
(i) ROB, (ii) PRF, (iii) Register Free List, (iv) RAT and (v) head/tail pointers
when the first instruction in the ROB throws an exception.

Physical Registers		
P0	7475	p
P1	2013	p
P2	42	p
P3	7	p
P4		
P5		
P6		
P7		
P8		

Free List
P7
P8
P4
P5
P6

RAT		
	Before	After
R1	P6	P0
R2	P1	
R3	P5	P2
R4	P3	

BHT	
00	0
01	1
10	0
11	1

```
        0x1000  LW   R1, 0(R2)
        0x1004  ADD  R3, R2, R4
        0x1008  ADD  R1, R3, R3
        0x100C  BNEZ R4, skip
        0x1010  LW R3, 0(R2)
skip:0x1014  SUB R1, R3, R2
        0x1018  ADD R4, R3, R1
        0x101C  ADDI R4, R4, 4
```

next to commit: RS id 0

next available: RS id 0

Reorder Buffer (ROB)

RS id	use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
0	~~x~~		~~LW~~	~~p~~	~~P1~~			~~R1~~	~~P0~~	~~P4~~
1	~~x~~		~~ADD~~	~~p~~	~~P1~~	~~p~~	~~P3~~	~~R3~~	~~P2~~	~~P5~~
2	~~x~~		~~ADD~~		~~P5~~		~~P5~~	~~R1~~	~~P4~~	~~P6~~
3	~~x~~		~~BNEZ~~	~~p~~	~~P3~~					
4										
5										
6										

Question D.6 (3 points)

Assume the processor also had hardware support for RAT snapshots, and a snapshot was taken at the time the LW instruction (at PC 0x1000). Considering Question D.5's scenario, what is the benefit of having a RAT snapshot corresponding to an instruction raising an exception?

Recovery is faster with a RAT snapshot. Instead of sequentially walking back the register renamings in the ROB to retrieve the correct RAT state, the RAT can be instantly restored from the snapshot.

Part C: Branch Prediction (25 points)

Assume a processor equipped with an 8-entry Branch History Table (BHT) with 2-bit counters, indexed using PC[4:2]**.** The BHT is initially in the following state:

000	**11**
001	**00**
010	**01**
011	**01**
100	**11**
101	**00**
110	**01**
111	**10**

Figure C-1: BHT

Consider the following code snippet:

label	**PC**	**Instruction**	**note**
start:	0x101C	ADDI R1, R1, -1	
	0x1020	AND R2, R1, 1	//Bitwise AND operation
	0x1024	BNEZ R2, target1	//Branch to PC *target1* if R2 $\neq$ 0
	0x1028	ADD R0, R0, R1	
	0x102C	J target1	//Jump to PC *target1*
	…		
target1:	0x2A74	AND R3, R1, 3	//Bitwise AND operation
	0x2A78	XOR R2, R3, 2	//Bitwise XOR operation
	0x2A7C	BEQZ R2, loop	//Branch to PC *loop* if R2 == 0
	0x2A80	SUB R3, R3, R1	
loop:	0x2A84	BNEZ R1, start	//Branch to PC *start* if R1 $\neq$ 0

Assume initial register values R0=0, R1=100.

Name _____

Question C.1 (9 points)

Fill out a row of the following table for each of the first nine branch instructions executed in the program.

Branch instruction PC	BHT index	Prediction (T/NT)	Actual branch direction	Updated BHT entry	Correct prediction? (Y/N)
0x1024	001	NT	T	01	N
0x2A7C	111	T	NT	01	N
0x2A84	001	NT	T	10	N
0x1024	001	T	NT	01	N
0x2A7C	111	NT	T	10	N
0x2A84	001	NT	T	10	N
0x1024	001	T	T	11	Y
0x2A7C	111	T	NT	01	N
0x2A84	001	T	T	11	Y

100 = 64 + 32 + 4 → bottom-most bits three bits we care about for branch 0x2A7C are 00
First four iterations:
11 XOR 10 = 01 → NT
10 XOR 10 = 00 → T
01 XOR 10 = 11 → NT
00 XOR 10 = 10 → NT

Branches at PC 0x1024 and 0x2A84 alias into the same BHT entry

Question C.2 (4 points)

What is the total branch misprediction ratio when this program runs to completion? Justify your answer for credit.

$$\frac{\text{Total branches mispredicted}}{\text{Total branch instructions executed}} = \frac{81}{100 * 3}$$

Justification:
0x1024 interleaves between T, NT. 1st time mispredict 2/2, next times 1/2 → 51 mispreds
0x2A84 is always T (99 times), NT the last time. Mispredicts first two times and very last time → 3 mispreds
0x2A74 is NT, T, NT, NT (repeats). 1st time mispredict 3/4, every next time 1/4 → 27 mispreds

Question C.3 (3 points)

The prediction accuracy of the branch at PC 0x1024 using the BHT alone turns out to be low. How would you enhance the branch predictor to improve that branch's prediction accuracy? Describe your enhanced mechanism and **briefly explain** why it would perform better than Figure C-1's baseline 2-bit BHT.

Correct answers would mention leveraging history/temporal correlation. GHR or LHR would work well to get close to 100% prediction accuracy.

Question C.4 (9 points)

The following figure shows a hypothetical 9-stage pipeline. The pipeline also features a BTB and a BHT at different stages. Your goal is to deduce at which stage the BHT and BTB reside, as well as deduce the stages where the branch direction and target are known.

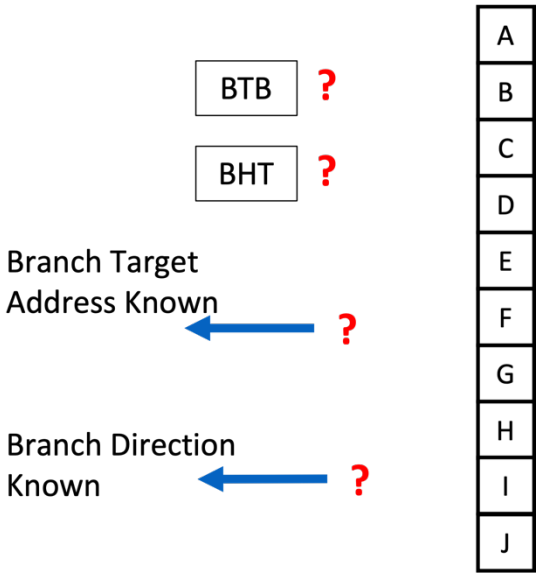


Figure C-2: 9-Stage Pipeline with Branch Prediction

Name _____

The following table reports the number of pipeline bubbles incurred depending on the accuracy of branch direction and target prediction.

BTB Hit?	(BHT) Predicted Taken?	Actually Taken?	Pipeline bubbles
Y	Y	Y	1
Y	Y	N	6
N	Y	Y	3
N	N	N	0

Based on the above information, answer the following questions:

1. Select the correct answer. The pipeline's fetch stage (A)
   a. Always speculatively fetches PC+4.
   b. Stalls until it is confirmed that the fetched instruction is not control-altering.

2. The branch target address is determined at stage _D_ because _when the BTB is missed, there is a 3-cycle penalty__.

3. The BHT is updated at stage _G_ because _that's when the branch direction is resolved, since the penalty of mispredicting the branch direction is 6 cycles_.

4. The BHT is used for branch direction prediction at stage _B_ because __the penalty for correct branch direction prediction of taken branch is 1 cycle_.

5. The BTB is used for branch target prediction at stage _B_ because ___(same reason as above)_____.

Note: answering A/B or B/A for questions 4/5 would also be correct (in addition to B/B), as it's possible to know only one of two pieces of information (direction or target) early on but not be able to convert this knowledge to a correct next PC fetch before stage B.