Department of Electrical and Computer Engineering
The University of Texas at Austin

ECE 460N Spring 2025
Instructor: Yale N. Patt
TAs: Luke Mason, Jenna May, Roy Mor, Rathna Sivakumar, Margaret Lee
Exam 1
February 26th, 2025

Name: _________SOLUTION :)____

Problem 1 (20 points): _______

Problem 2 (10 points): _______

Problem 3 (15 points): _______

Problem 4 (25 points): _______

Problem 5 (30 points): _______

Total (100 points): _____________

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

Please read the following sentence, and if you agree, sign where requested:
I have not given nor received any unauthorized help on this exam.

Signature: ____

GOOD LUCK!

Name: ____

Problem 1 (20 pts): Answer the following questions.
For each answer, if you leave the box empty, you will receive one point of the five.

Part a (5 pts):
You have the ability to increase cycle time by 10%, and in return, allow a function to be performed in one cycle rather than two. Should you ever do this? If so, in which case would you?

Depends how often you use this function.
Also, it’s possible this function’s latency gets masked by a pipeline.

Part b (5 pts):
In order to retire an instruction that is in the reorder buffer, what two things are required?

Instruction finishes executing, and previous instructions have retired.

Part c (5 pts):
Most arithmetic instructions in the LC-3b (ADD, AND, XOR, etc.) require three addresses, two for the two source operands and one for the destination. Not all ISAs explicitly express all three in the instruction. In fact, a zero-address machine expresses none of them. For example, an add instruction would just have the opcode ADD with no addresses. In such an ISA how does the microarchitecture know where to get the source operands and where to store the result?

Stack, or perhaps elsewhere as specified by the ISA/uarch.

Part d (5 pts):
Gate_MARMUX = 1 is required at some point in the execution of which instructions?

LDW, LDB, STW, STB, LEA, TRAP

Name: ____

Use these blank diagrams for scratch work on problems 2 and 3. Nothing on this page will be graded.

2A proof

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	E	W													
I2		F	D	E	E	W												
I3			F	D	*	*	E	E	W									
I4				F	*	*	D	E	E	W								
I5							F	D	E	E	W							
I6								F	D	*	E	E	W					

2C proof

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	E	W													
I2		F	D	E	E	W												
I3			F	D	E	E	W											
I4				F	D	*	E	E	W									
I5					F	*	D	*	E	E	W							
I6							F	*	D	E	E	W						

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1																		
I2																		
I3																		
I4																		
I5																		
I6																		

Name: ____

Problem 2 (10 pts):
Consider an in-order pipelined machine.

- The pipeline has the following four stages: FETCH, DECODE, EXECUTE, and WRITEBACK.
- FETCH, DECODE, and WRITEBACK each take 1 cycle..
- EXECUTE is pipelined and takes 2 cycles.
- There is no data forwarding. The destination register is updated at the end of WRITEBACK, and dependent instructions can begin EXECUTE in the next cycle

Use this LC-3b program to answer the following questions. You may use the blank timing diagrams on the previous page for scratch work, and they will not be graded.

I1	AND R3, R0, R6
I2	AND R6, R1, R4
I3	AND R2, R6, R1
I4	XOR R3, R5, R6
I5	LSHF R7, R0, #2
I6	ADD R7, R5, R3

Part a (4 pts): How many cycles does the above LC3-b program segment take to execute?

13

Part b (2 pts): Does splitting the work of a stage into multiple stages decrease cycle time? Explain.

Depends if it was the critical path before we split it.

Part c (4 pts): Given the above code, a compiler decides to swap two instructions to achieve more optimal performance. Which two should be swapped, and what is the speedup? *Speedup = old/new.*

Instructions:
I3 and I5
I3 and I4

Speedup:
13/12

Name: ____________________________________

Problem 3 (15 pts):

An in-order pipelined LC-3b processor has 5 stages: Fetch, Decode, Execute, Memory, and Writeback. Each stage can process one instruction at a time. Branch instructions execute during the **Writeback** stage. In addition, it implements a branch predictor, which predicts **in Decode** if a branch will be taken.

Correctly predicted *Taken* BR:

Cycle:	1	2	3	4	5	6	7
I1 (BR)	F	D	E	M	W		
I2		▨	F	D	E	M	W

Incorrectly predicted BR:

Cycle:	1	2	3	4	5	6	7
I1 (BR)	F	D	E	M	W		
I2		▨	▨	▨	▨	F	D

Correctly predicted *Not Taken* BR:

Cycle:	1	2	3	4	5	6	7
I1 (BR)	F	D	E	M	W		
I2		F	D	E	M	W	

Note: If the branch correctly predicts not-taken, the next instruction has already been fetched so there is no delay.

Answer the questions on the next page.

PROBLEM CONTINUES ON NEXT PAGE

Name: ____

Consider these four possible predictors:

- Always Taken
- BTFN (Backward Taken Forward Not)
- 2-bit saturating counter
- Gshare indexing an 8-entry table of 2-bit saturating counters, using this hash function: XOR bit 5, 3, and 1 of the PC with bits 2, 1, and 0 of the BHR (Branch history register).

Where applicable, 2-bit saturating counters are initialized to binary 01, and the BHR is initialized to all 0s. Counters and the BHR update when a branch is executed.

Your job: In each program segment, determine which branch predictor the processor used. Initial registers and condition codes are unknown. There are no data dependencies. If you leave a box blank, you will get one point.

Part a (5 pts):

- The program segment contains 5 instructions.
- The program completes in 7 cycles.

Predictor:
Always Taken

Part b (5 pts):

- The program segment contains an unknown number of instructions.
- Three of the instructions are branches, in some order.
- Each branch is executed twice: once taken and once not taken.
- The branch predictor is **correct on 5 of the 6** predictions.
- Each branch completes execution before the next one is predicted.

Predictor:
Gshare

Part c (5 pts):

- The program segment contains 5 instructions.
- Only one of the instructions is a branch. This is a backward branch executed twice: taken the first time and not taken the second time.
- The program completes in 21 cycles.

Predictor:
2-bit Counter

Name: ______________________________________

Problem 4 (25 pts):

You've been tasked to add an instruction that generates a bitmap for a 16-element array. Each bit represents whether an element in the array is greater than a threshold. The least significant bit represents the 0th element, and the most significant bit represents the 15th element. Each element in the array is an LC-3b word, or 16 bits. Both the elements in the array and the threshold are unsigned.

The instruction has two source operands. One of them is the threshold for the bitmap and the other is the base address of the array. You will figure out which source register corresponds to each operand in part b.

Instruction Encoding:

15 — 12	11 — 9	8 — 6	5 — 3	2 — 0
1010	DR	SR1	000	SR2

The datapath has been augmented with the changes in bold on the following page. A shift register has been added to the datapath, which shifts the P bit into the least significant position when LD.SR is set. Two temp registers have also been added. Additionally, the ALU now supports a subtract operation (A-B).

IMPORTANT: EXAMINE ALL PARTS OF THE PROBLEM BEFORE DESIGNING YOUR SOLUTION

PROBLEM CONTINUES ON NEXT PAGE

Name: ____

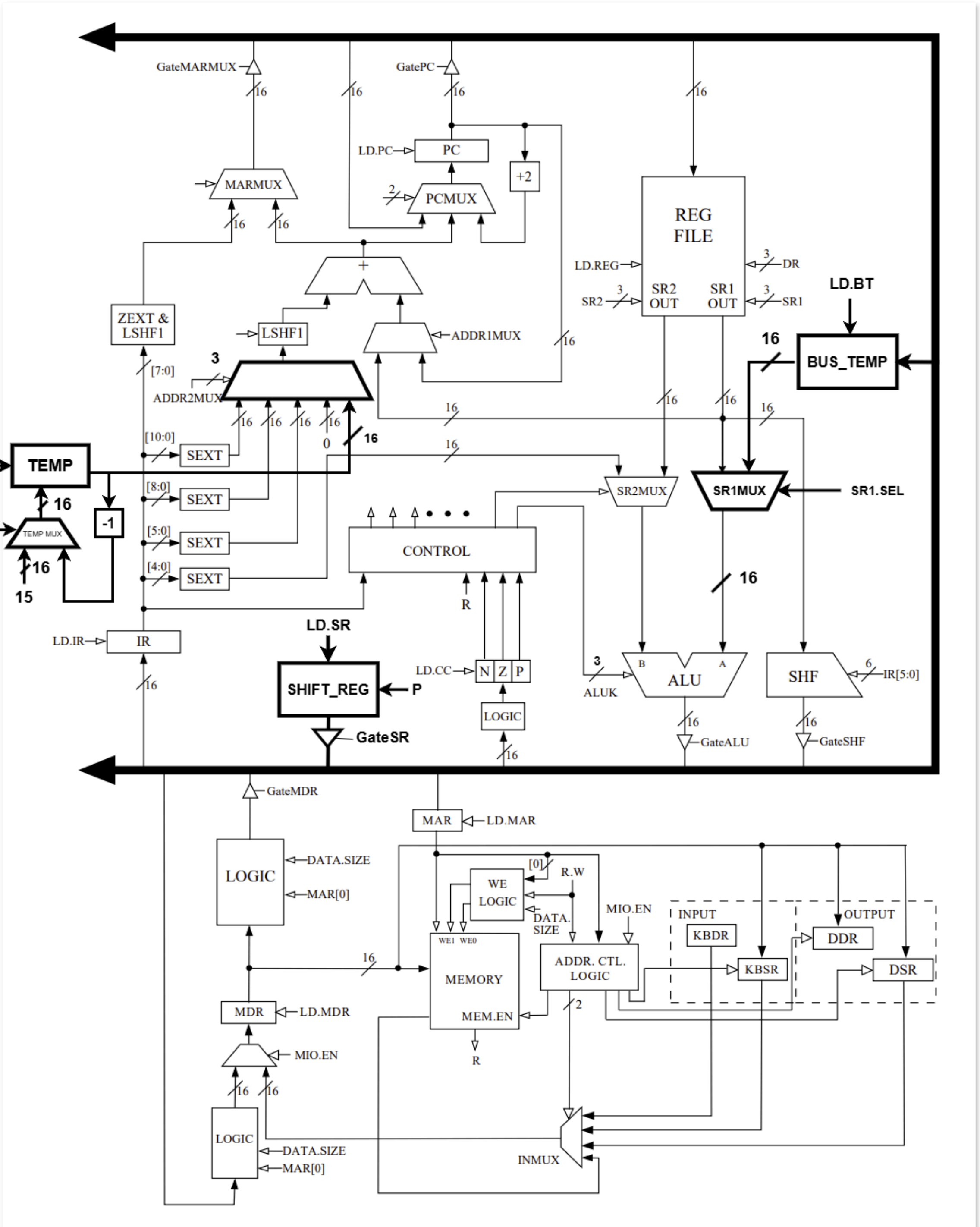
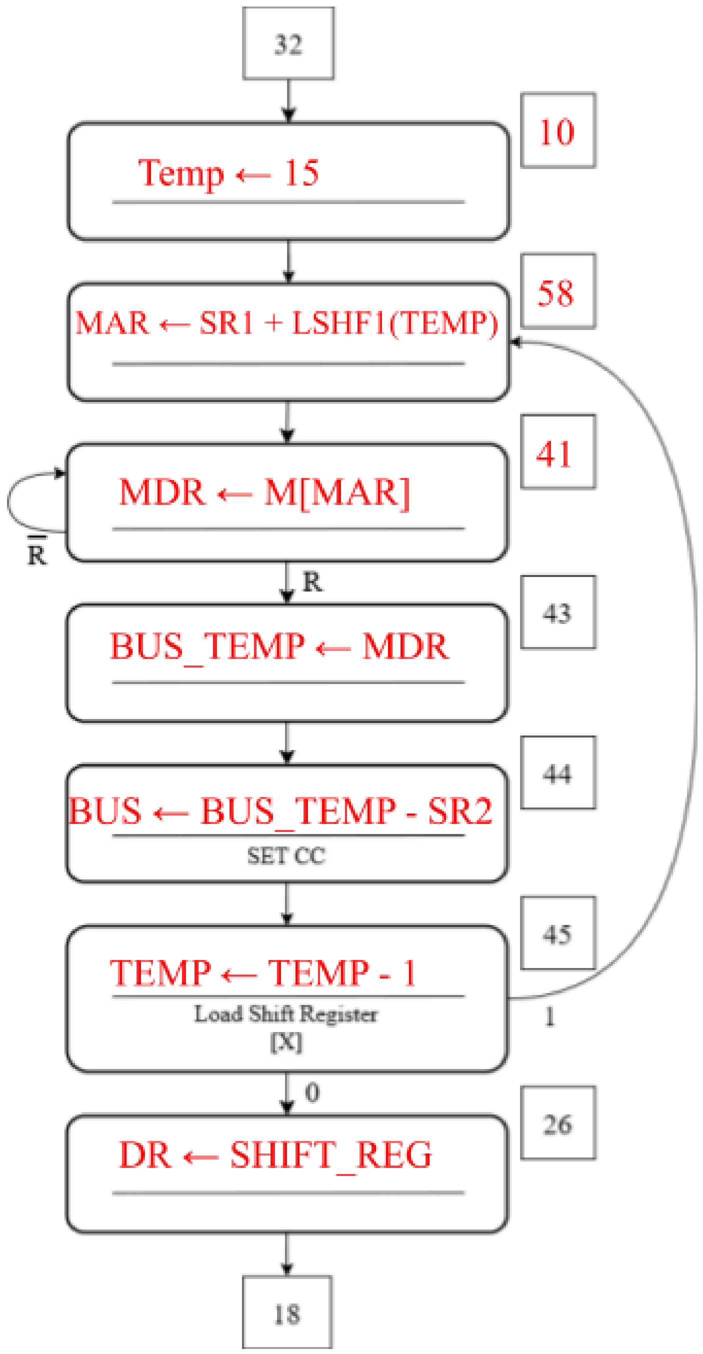


Figure C.3: The LC-3b data path

PROBLEM CONTINUES ON NEXT PAGE

Name: ____

Part a (15 pts): Fill in the missing entries in the state diagram, including the missing state numbers and any relevant micro-ops.



PROBLEM CONTINUES ON NEXT PAGE

Name: ____

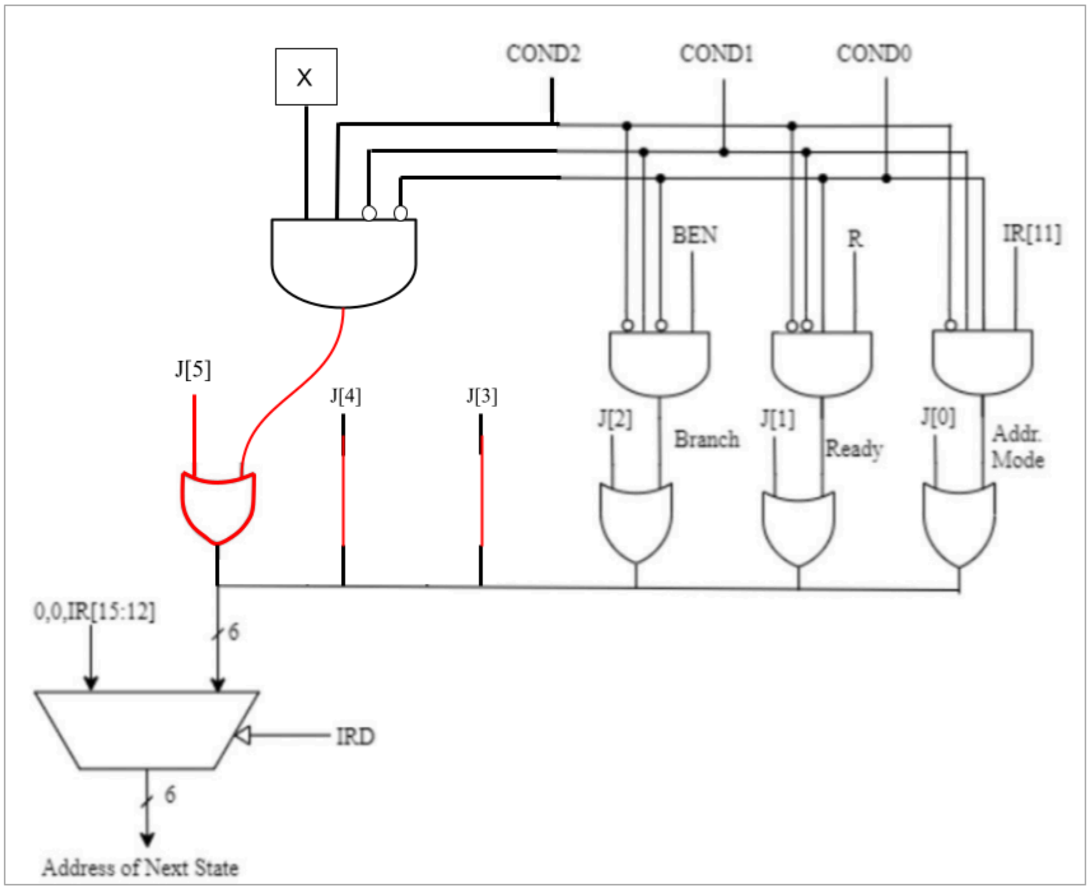
Part b (4 pts): Given what you know of the encoding and the datapath, which source operand is the threshold, and which is the base address of the array?

SR1 Base address
SR2 Threshold

Part c (6 pts): Explain the logic required to generate signal X. Then, complete the microsequencer.

Signal X: **Checking if TEMP != 0**

Microsequencer modification:



Name: ____________________________________

Problem 5 (30 pts):

A microarchitecture using Tomasulo's algorithm with ROB is executing a program.

Specifications are as follows:

- The pipeline has 6 stages:
  FETCH, DECODE, REGISTER RENAME, EXECUTE, WRITEBACK, and RETIRE.
- There is one pipelined adder and one pipelined multiplier.
- ADD takes 2 cycles to execute, MUL takes 3 cycles to execute.
- All other stages take one cycle each and can support one instruction at a time.
- During REGISTER RENAME, each instruction is renamed with a tag. Also, instructions are allocated to reservation stations and the ROB in a top-to-bottom manner.
- The reservation stations for ADD and MUL have 2 entries each.
- The tags (reservation stations) are α and β for ADD, and π and ρ for MUL.
- The result is broadcasted after WRITEBACK. Dependent instructions can begin execution the cycle after all the source values are written back.
- Reservation station entries are deallocated at the end of WRITEBACK. In the next cycle, the instruction waiting for that reservation station can be renamed. It will enter the reservation station in the cycle after it is renamed.
- The 'Executed' bit in the ROB is set at the end of WRITEBACK
- The 'Retired' bit in the ROB is set at the end of RETIRE.
- General purpose registers are 8 bits wide, which can store signed integer values from -128 to 127. An **overflow** exception is generated when an instruction produces a result outside of these bounds.

Complete parts a, b, and c on the following pages.

PROBLEM CONTINUES ON NEXT PAGE

Name: ______________________________________

Part a (12 points): Fill in the missing entries in the program.

Important: You are given incomplete snapshots of the RAT and ROB at different points during execution. You will need the information to complete the problem, but they will not be part of the grade. There is also scratch paper containing reservation stations and timing diagrams on page 14.

	INST	**DR**	**SR1**	**SR2**
I1	MUL	R3	R0	R1
I2	ADD	R2	R3	R1
I3	ADD	R4	R7	R3
I4	MUL	R1	R5	R5
I5	MUL	R0	R7	R1
I6	ADD	R0	R6	R3

RAT Before Cycle 1

	V	**Tag**	**Value**
R0	1	α	8
R1	1	β	1
R2	1	π	2
R3	1	ρ	3
R4	1	α	4
R5	1	β	5
R6	1	π	6
R7	1	ρ	7

RAT After Cycle 8

	V	**Tag**	**Value**
R0	0		8
R1	0		1
R2			2
R3	1	π	
R4	0		4
R5	1	β	5
R6	1	π	6
R7	1	ρ	7

RAT After Cycle 14

	V	**Tag**	**Value**
R0			
R1	1	ρ	
R2			
R3	1	π	
R4			
R5	1	β	5
R6	1	π	6
R7	1	ρ	7

ROB After Cycle 8

Instruction	Retired	Executed	Value
I1			
I2		0	
I3			
I4			
I5			
I6			

ROB After Cycle 14

Instruction	Retired	Executed	Value
I1			
I2			
I3			
I4			
I5		0	
I6		1	

PROBLEM CONTINUES ON NEXT PAGE

Name: ____

Part b (12 points): Complete the pipeline timing diagram for the execution of the program. **If an instruction causes an exception, circle the box where the exception is detected.**

FETCH	**F**	WRITEBACK	**WB**
DECODE	**D**	RETIRE to R#	**R#**
REGISTER RENAME	**RR**	Stall	**X**
MUL	**M**	ADD	**A**

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I1	F	D	RR	M	M	M	W	R3													
I2		F	D	RR	X	X	X	A	A	W	R2										
I3			F	D	RR	X	X	X	A	A	W	R4									
I4				F	D	RR	M	M	M	X	X	W	R1								
I5					F	D	X	RR	X	X	X	X	M	M	**M**						
I6						F	X	D	X	X	RR	A	A	W	X						

Exception generated in 15, but 16 or 17 fine as well. Bonus point probably given to anyone who correctly leaves off the retirement of I5 and I6.

Part c (6 points): What problem arises if there is no ROB in this example? How does the existence of the ROB solve it?

State of the computer is incorrect when we return from exception handler to re-execute I5.

ROB retires instructions in order, so state of reg file is up to date and no further.

Name: ______________________________________

Below are blank reservation stations timing diagrams for scratch work. Nothing on this page will be graded.

	V	Tag	Value	V	Tag	Value
α						
β						

	V	Tag	Value	V	Tag	Value
π						
ρ						

	V	Tag	Value	V	Tag	Value
α						
β						

	V	Tag	Value	V	Tag	Value
π						
ρ						

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I1	F	D	RR																		
I2		F																			
I3			F																		
I4				F																	
I5					F																
I6						F															

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I1	F	D	RR																		
I2		F																			
I3			F																		
I4				F																	
I5					F																
I6						F															