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

EE 460N Spring 2019
Y. N. Patt, Instructor
Aniket Deshmukh, Chester Cai, Mohammad Behnia, TAs
Exam 1
February 27, 2019

Name: ANIKET DESHMUKH

Problem 1 (25 points):_______

Problem 2 (10 points):_______

Problem 3 (15 points):_______

Problem 4 (25 points):_______

Problem 5 (25 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 (25 points): Note: For each of the five answers below, if you leave the box empty, you will receive one point of the five.

Part a (5 points): Performance is a function of several different attributes, one of which is cycle time. Make the cycle time smaller (frequency higher) and performance should improve. But a second attribute runs counter to cycle time. Make cycle time better and this gets worse. Make this better, and cycle time gets worse. What is this second attribute, and why does it run counter to cycle time with respect to performance?

CPI. Smaller cycle time means less work per cycle. Therefore, each instruction takes more cycles (to complete the same amount of total work). In pipelined processors, this translated to number of stages.

Part b (5 points): A data type is a representation of information in a form such that the ISA has instructions that operate on that representation. Can a doubly-linked list be a data type? If yes, what is required to make it so. If no, why not?

Yes.
Need instructions in the ISA that operate on doubly-linked lists.

Part c (5 points): Is the Reorder Buffer part of the ISA or part of the microarchitecture? Explain.

Microarchitecture.
The ROB is responsible for implementing in-order retirement - the exact implementation of how this is done is irrelevant to the ISA.

Part d (5 points): Speculation used to be a no-brainer. If you don't know how to move forward, speculate. If you guess right, performance improves. If you guess wrong, you are no worse off than if you just did nothing and waited until you knew how to move forward. This argument no longer has the ring of truth that it used to have. Why?

Speculation costs energy. Energy is a first order concern in processors today.

Part e (5 points): In moving from the LC-3 to the LC-3b, I discarded LDI and STI. You recall that LDI R2,A used the contents of the memory location labeled A as the address of the data to be loaded into R2. That is, A is a pointer variable, a very useful construct. With LDI and STI gone, can A still be a pointer variable, or has the LC-3b lost that very important construct? If A can no longer be a pointer variable, explain why pointer variables are no longer needed. If A can still be a pointer variable, explain how I can load and store from/to the address contained in A.

A can still be a pointer variable:

```
        LEA     R2, A
        LDW     R2, R2, #0  ; get pointer
STW/LDW         R2, R2, #0  ; get data
```

Name:______________________________

Problem 2 (10 points):

Assume we are executing instructions out-of-order on IBM's 360/91, using the method of Tomasulo, but NOT including the Reorder Buffer (ROB) which I added 20 years later.

ADD takes one cycle each to Fetch, Decode and Store Result, 4 cycles to execute. MUL takes one cycle each to Fetch, Decode and Store Result, 6 cycles to execute. Data forwarding is supported.

The following code segment attempts to execute, but the first MUL instruction fails during the 5th clock cycle of its 6 cycles of execution.

```
ADD R1,R2,R3
MUL R4,R1,R2
ADD R2,R5,R6
MUL R4,R5,R6
```

Does this present a problem? Hint: Yes, which is why IBM had to give special dispensation to the release of their 360/91 product.

What is the problem. Explain precisely, but concisely. Giving the "name" of the problem without further explanation is not sufficient.

Cannot implement precise exceptions.
(ADD R2, R5, R6) writes to the register file before the first MUL instruction fails. Upon failure, the ISA specifies the state of the machine needs to be reset to that before the first MUL executes. This is no longer possible as the value in R2 which has been overwritten by the second ADD cannot be recovered.

Name:______________________________

Problem 3 (15 points): Consider a processor with a four stage pipeline: Fetch, Decode, Execute, and Store Result. Fetch, decode and store take one clock cycle each. The execute stage for ADD takes two clock cycles.

The processor has a branch predictor. The target of the branch instruction is calculated during the Decode stage so the predicted address of the next instruction can be loaded into the PC at the end of the Decode clock cycle. The next instruction can then be fetched during the next clock cycle.

The timing diagram below shows the instruction flow for a sequence ADD.1, BR, ADD.2 if the branch is predicted correctly.

	n	n+1	n+2	n+3	n+4
ADD.1	F	D	E	E	S
BR		F	D		
ADD.2				F	

The timing diagram below shows the instruction flow for the same sequence if the branch is predicted incorrectly. Data forwarding is implemented so the condition codes of the ADD.1 instruction are available in cycle n+4 so the correct next PC can be loaded at the end of cycle n+4. ADD.2 can be fetched in cycle n+5.

	n	n+1	n+2	n+3	n+4	n+5
ADD.1	F	D	E	E	S	
BR		F	D	-	-	
ADD.2						F

Note: In class on Monday, I showed the case where the PC of ADD.2 could be loaded at the end of cycle n+3, which would allow ADD.2 to be fetched in cycle n+4. With the appropriate logic, this can certainly be done. However, your job here is to assume ADD.2 is fetched in cycle n+5.

Part a (9 points): The following program segment is executed on the processor, with R0 initialized to 3.

```
LOOP    ADD R0, R0, #-1
        BRp LOOP
        ADD R1, R1, #3
```

How many clock cycles does the program segment take if the processor does not have a branch predictor?

$5 \cdot 4 = \boxed{20}$

How many clock cycles does the program segment take if the processor has a Last Time taken branch predictor? Assume the last branch before this program segment is executed was taken.

$20 - 2 \times 2$ Two mispredictions avoided

16

↕ Same as the predictions will be the same.

How many clock cycles does the program segment take if the processor has a saturating 2-bit counter branch predictor? Assume the 2-bit counter is in state 10 just before this program segment is executed.

16

Note: We have included several timing diagrams on the next two pages which you can use if you wish for scratch work. They will not be graded.

Name:______________________________

Part b (6 points): The following program segment is executed on the processor, with R0 and R1 initialized to 3.

```
LOOP1   ADD R0, R0, #-1
        BRp LOOP1
LOOP2   ADD R1, R1, #-1
        BRp LOOP2
        ADD R2, R2, #3
```

How many clock cycles does the program segment take if the processor has a Last Time taken branch predictor? Assume the last branch before this program segment is executed was taken.

$\boxed{29}$

How many clock cycles does the program segment take if the processor has a saturating 2-bit counter branch predictor? Assume the 2-bit counter is in state 10 just before this program segment is executed.

29 - 2×1 (One less misprediction)

$\boxed{27}$

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	F	D	E	E	S															
		F	D																	
						F	D	E	E	S										
							F	D												
											F	D	E	E	S					
												F	D							
																F	D	E	E	S

] I0

] I1

] I2

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Last time taken

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	F	D	E	E	S																									
		F	D																											
				F	D	E	E	S																						
					F	D																								
							F	D	E	E	S																			
								F	D																					
												F	D	E	E	S														
													F	D																
																	F	D	E	E	S									
																		F	D											
																				F	D	E	E	S						
																					F	D								
																									F	D	E	E	S	

] Incorrect

] predictions

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Name:__________________________________

Problem 4 (25 points):

An out-of-order processor executes its instructions based on Tomasulo's original algorithm (i.e. no in-order retirement). The ISA specifies 8 registers, R0 to R7. The execute stage of the pipeline contains one pipelined adder and one pipelined multiplier. This allows one ADD/MUL operation to start executing every clock cycle.

- Fetch and Decode take one cycle each.
- ADD takes 4 cycles to execute.
- MUL takes 5 cycles to execute.
- Both functional units have three-entry reservation stations and are initially empty.
- Reservation station entries are allocated at the end of decode, in a top to bottom manner. Entries are freed and available in the cycle following their result broadcast.
- Data forwarding is implemented. Functional units broadcast results to the bus at the end of the last cycle of execution. This allows dependent instructions to start executing immediately after the value of the dependent source is known.
- When allocating a reservation station entry for ADD DR, SR1, SR2 the order of the operands SR1 and SR2 is maintained.
- The write-back bus supports only one result being stored at a time.

The table below contains a six instruction program segment.

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

PROBLEM CONTINUES ON NEXT PAGE

Name:________________________________

Three snapshots of the register alias table are provided: one before the first instruction is fetched, one at the end of cycle 7, and one after the 6th instruction stores its result.

	V	Tag	Value
R0	1	-	0
R1	1	-	1
R2	1	-	2
R3	1	-	3
R4	1	-	4
R5	1	-	5
R6	1	-	6
R7	1	-	7

Before Cycle 1

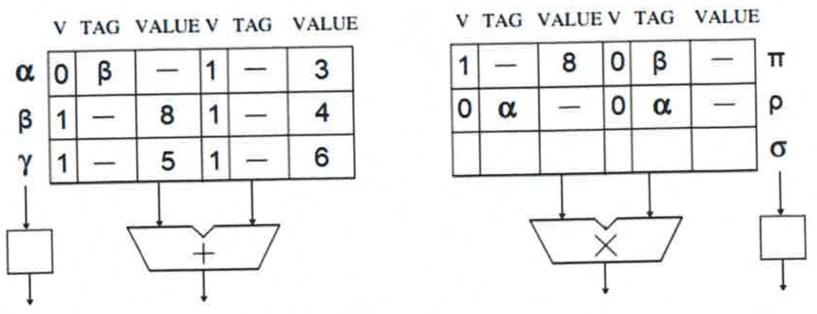
	V	Tag	Value
R0	0	ρ	-
R1	1	-	8
R2	0	β	-
R3	1	-	3
R4	1	-	4
R5	0	γ	-
R6	1	-	6
R7	1	-	7

End of Cycle 7

	V	Tag	Value
R0	1	-	225
R1	1	-	8
R2	1	-	12
R3	1	-	3
R4	1	-	4
R5	1	-	11
R6	1	-	6
R7	1	-	7

After Execution

Here is a snapshot of the reservation stations at the end of cycle 7:



	V	TAG	VALUE	V	TAG	VALUE
α	0	β	—	1	—	3
β	1	—	8	1	—	4
γ	1	—	5	1	—	6

V	TAG	VALUE	V	TAG	VALUE	
1	—	8	0	β	—	π
0	α	—	0	α	—	ρ
						σ

PROBLEM CONTINUES ON NEXT PAGE

Name:____________________________

Part a (15 points): Determine the six instructions that were executed.

Part b (2 points): Determine the missing entries in the register alias table, shown in bold face.

Part c (8 points): Complete the timing diagram for the execution of the six instructions. Write the stage each instruction occupies during each cycle. If an instruction is storing a result, write the name of the register written in that cycle. If an instruction is stalled in a stage, write an asterisk (*) in that cycle.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1	F	D	E	E	E	E	R1													
I2		F	D				E	E	E	E	R2									
I3			F	D							E	E	E	E	E	R0				
I4				F	D	E	E	E	E	R5										
I5					F	D					E	E	E	E	R0					
I6						F	D								E	E	E	E	E	R0

Note: ADD and MUL can start together (I3, I5) as they use different Functional Units.

Name:__________________________________

Problem 5 (25 points): The very simple FOR loop, present in all high-level languages, has two parameters important to controlling the number of iterations of the loop: the number of instructions in the loop body (BLOCK_SIZE) and the number of iterations of the loop body (ITERATION_COUNT).

We wish to add a REPEAT instruction to the LC-3b ISA that the compiler can use to implement the iteration control of the FOR loop. The FOR is translated into the REPEAT instruction followed by the loop body.

We assume there are no control instructions in the loop body. Also, to make this problem work, we will have to assume there are no store instructions in the loop body, something you should not waste any time thinking about.

The REPEAT instruction has the following format, using the unused opcode 1011:

15–12	11–9	8–6	5	4–0
1 0 1 1	**0 0 0**	**SR**	**1**	**imm5**

The ITERATION_COUNT is stored in SR as an unsigned integer. The BLOCK_SIZE of the loop body is expressed as an unsigned integer in bits[4:0]. You can assume for purposes of this exam question that neither is 0.

Part a (3 points):
What is the maximum size of the loop body?

31

Is this determined by the ISA or the microarchitecture? (Explain)

ISA. imm5 bits specified in the ISA.

What is the maximum number of iterations of the loop body?

$2^{16}-1$

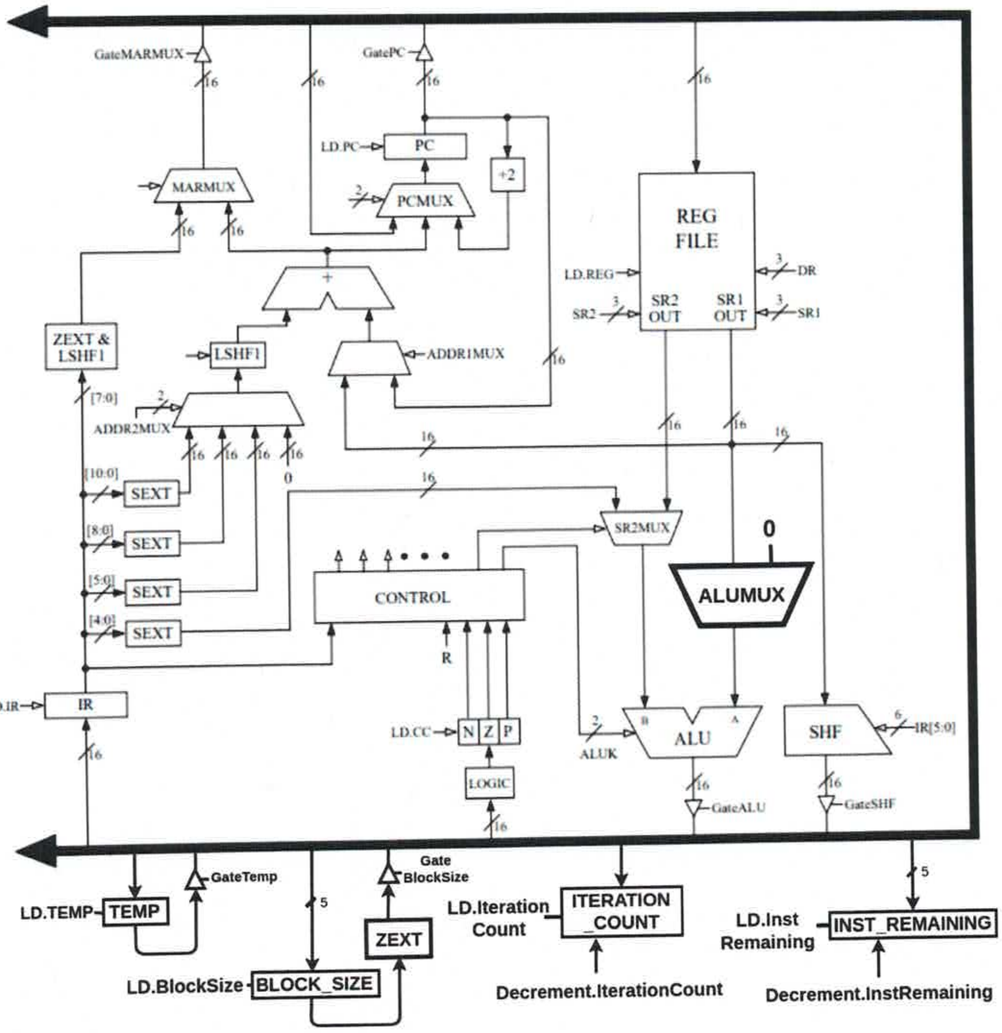
Is this determined by the ISA or the microarchitecture? (Explain)

ISA. Length of each register is determined by the ISA. (in the architectural register file)

PROBLEM CONTINUES ON NEXT PAGE

Name:__________________________________

To implement REPEAT, we need to make changes to the data path, the state machine, and the microsequencer. We show below all the changes necessary to the data path. They are: four registers (TEMP, BLOCK_SIZE, ITERATION_COUNT, INST_REMAINING), and ALUMUX. The registers are initialized to 0.

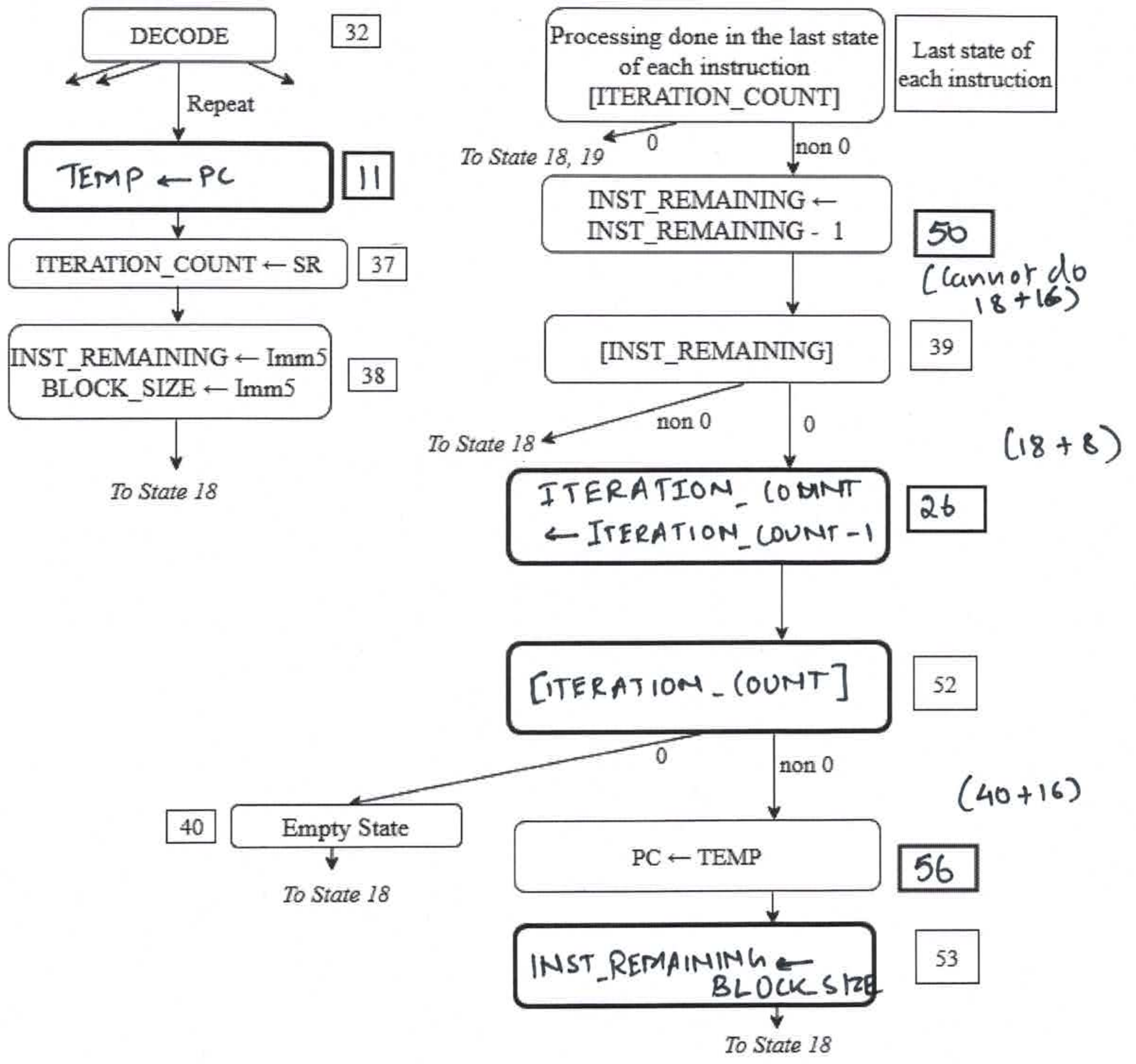


PROBLEM CONTINUES ON NEXT PAGE

Name:______________________________

Part b (16 points):
With respect to the state machine, we need to add additional states after state 32 (Decode), and after the last state of each instruction (the one that takes you to state 18 or 19). The empty state (state 40) does nothing. It is simply a transition state to state 18. **Your job: Fill in the missing entries in the state machine.**

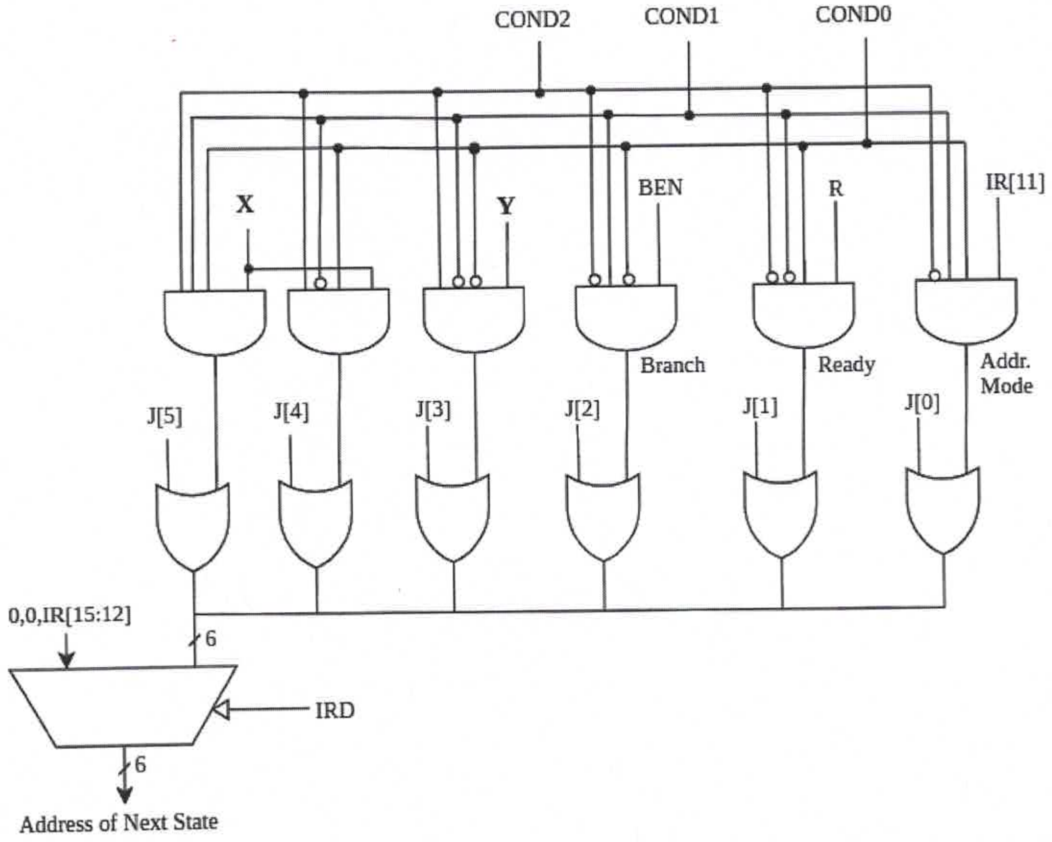
Note: States 26, 34, and 36-63 in the control store are available.



PROBLEM CONTINUES ON NEXT PAGE

Name:______________________________

Part c (6 points):
To handle the new branches in the state machine, we need to augment the microsequencer with new conditional control signals X, Y . **Your job: Explain what X and Y are.**



Explain what signal X is:

X = 1 if ITERATION_COUNT is non-0

Explain what signal Y is:

Y = 1 if INST_REMAINING is 0