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

EE 460N Fall 2018
Y. N. Patt, Instructor
Chirag Sakhuja, John MacKay, Aniket Deshmukh, Mohammad Behnia, TAs
Exam 1
October 10, 2018

Name:____

Problem 1 (20 points):_______

Problem 2 (15 points):_______

Problem 3 (20 points):_______

Problem 4 (20 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 (20 points):

Part a (5 points): When discussing out-of-order execution, some have misconstrued the notion of hazard and talked about ”hazards,” and in particular, “WAR Hazards.” I prefer Professor Kuck’s term for this: “antidependencies.” Why are they not a problem and create no latency when implementing out of order execution with the Tomasulo algorithm?

Part b (5 points): In a modern microarchitecture, where 4 instructions are fetched each cycle, performance is optimized if we can maximize instruction supply. That is, no instruction supply misses, 100% branch prediction, and no packet breaks. What causes a packet break? In ten words or fewer, please.

Part c (10 points): Little Computer Inc. has a CPU that takes 3 cycles to execute a `MUL` instruction, 5 cycles to execute a `DIV` instruction, and 1 cycle to execute all other instructions. There is no pipelining. The target workload has an instruction mix containing 10% `MUL` instructions, 10% `DIV` instructions, and 80% others.

Part c.i (5 points): What is the Cycles Per Instruction (CPI) of this CPU running this target workload?

Part c.ii (5 points): An improvement in the multiplier makes the `MUL` instruction take 1 cycle to execute but causes the overall cycle time to increase by 20%. Does this new design make the workload run faster than the original CPU? Explain.

Name:____________________________________

Problem 2 (15 points):

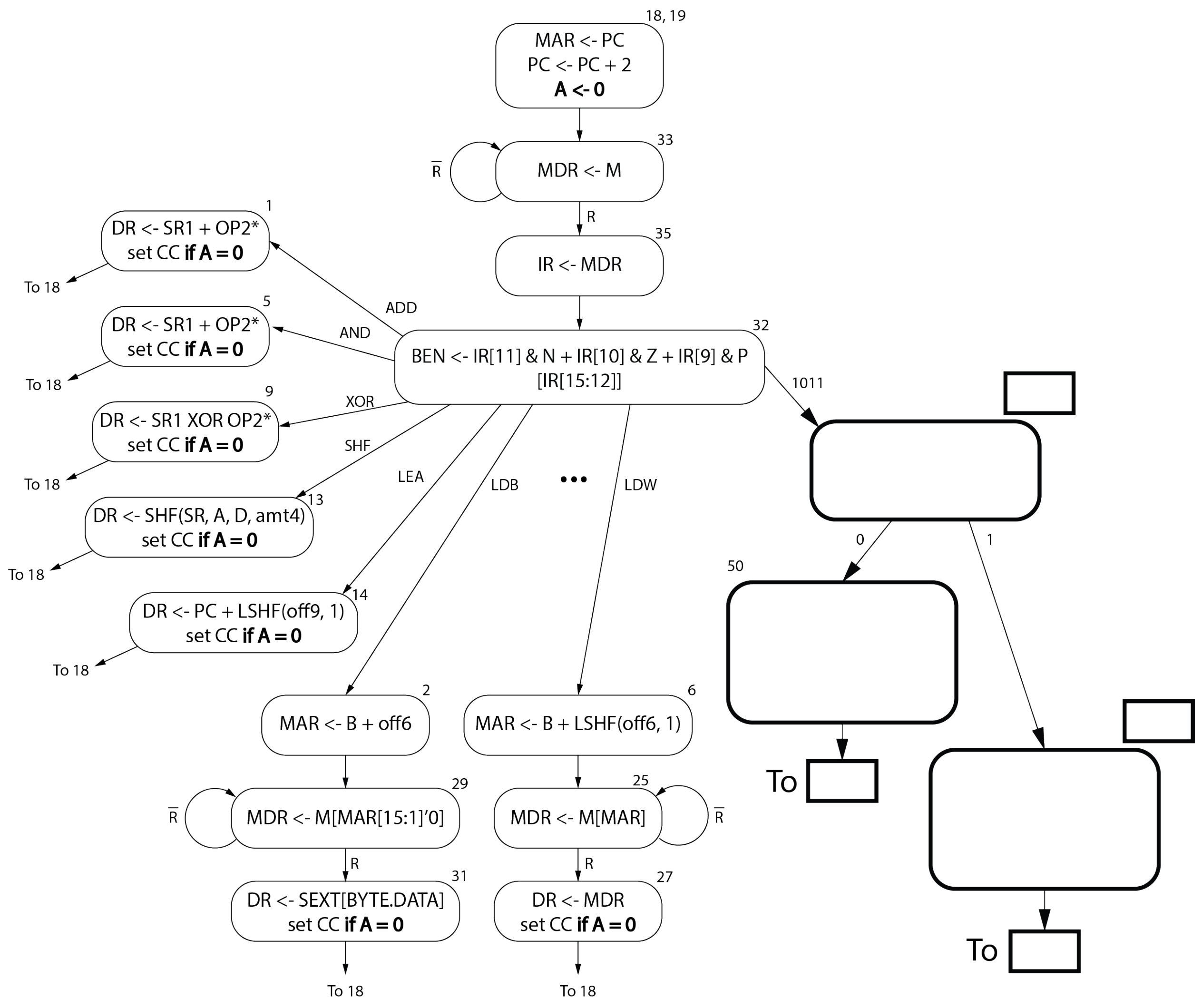
We can add predicated execution to the LC-3b using the unused opcode 1011, much like the prefixes in the x86 ISA. That is, if we wish to predicate instruction X, we put the following prefix just before instruction X in our program:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	N	Z	P	0	0	0	0	0	0	0	0	0

Instruction[11:9] act in a way similar to those bits in a conditional branch instruction. In the case of a conditional branch, the branch is taken only if the condition is satisfied. In this case, instruction X is executed only if the condition is satisfied.

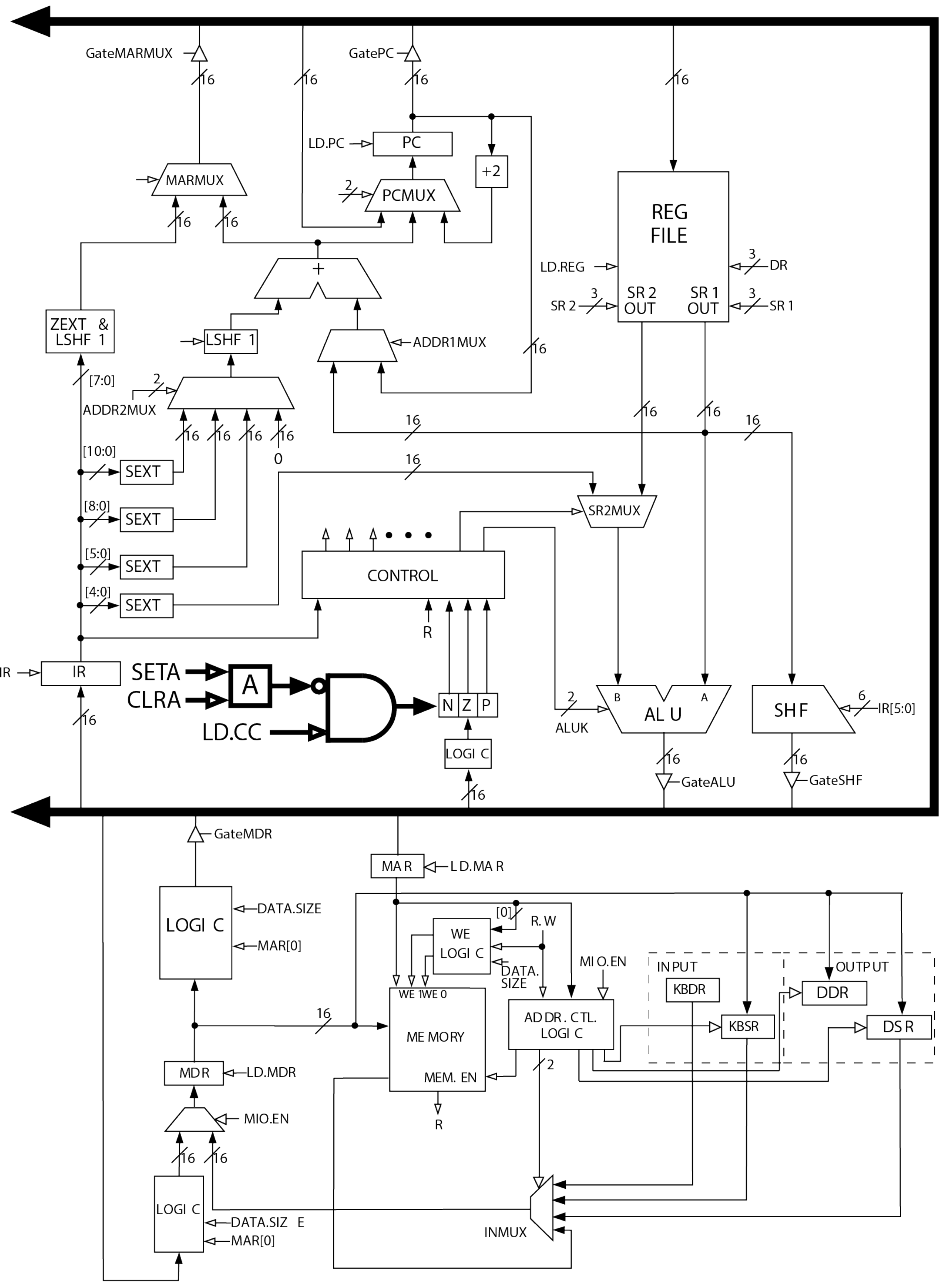
Whether or not instruction X is executed, we wish to preserve the condition codes so we can predicate additional instructions based on the same condition. Therefore, if instruction X is executed, it must not set condition codes. To make this work, we need to augment the data path (next page) with a flip-flop `A`, an AND gate, and two new control signals `SETA` and `CLRA`. `SETA` sets `A` to 1; `CLRA` sets `A` to 0.

The state machine for this addition is shown below. Only relevant states are shown, and some information (state numbers, and activity in a state) is missing. **Your job:** Fill in the missing information.



Name:____________________________________

We augment the datapath with a new flip-flop, `A`, and AND gate, and the control signals `CLRA` and `SETA`. The changes to the datapath are in bold.



Name:____________________________________

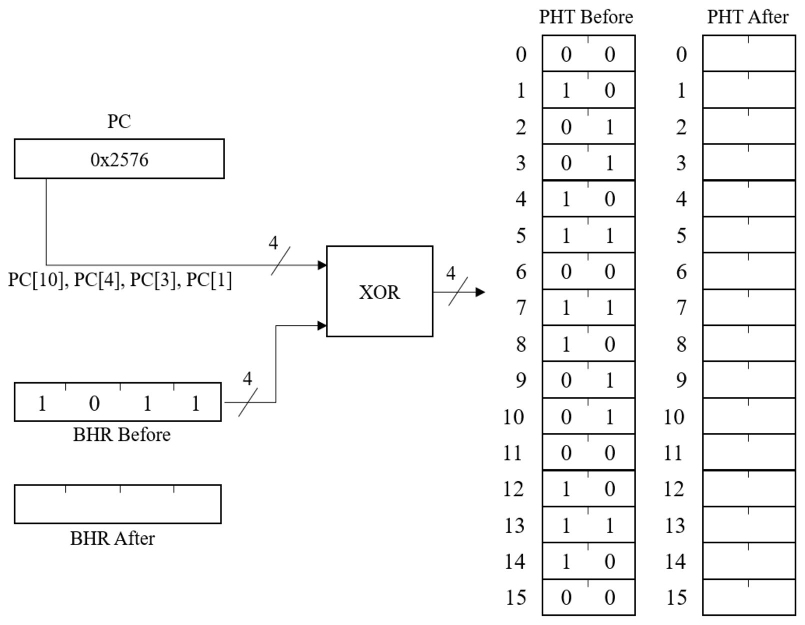
Problem 3 (20 points):

An LC-3b system has a GAg two-level adaptive branch predictor, with one modification. We index into the PHT with a vector created (like g-share) by XOR-ing the four bits of the BHR with four bits taken from the address of the current branch instruction, as follows:

$$\text{INDEX}[3] = \text{BHR}[3] \text{ xor } \text{PC}[10]$$
$$\text{INDEX}[2] = \text{BHR}[2] \text{ xor } \text{PC}[4]$$
$$\text{INDEX}[1] = \text{BHR}[1] \text{ xor } \text{PC}[3]$$
$$\text{INDEX}[0] = \text{BHR}[0] \text{ xor } \text{PC}[1]$$

The rightmost bit of the BHR comes from the most recent branch. When a branch instruction is encountered, the BHR is updated with the result of the branch predictor, i.e., the **speculative** (predicted) result. That result is shifted into the BHR, and the BHR is shifted one bit to the left. Information about the oldest branch is shifted out. The PHT is updated with the actual result of the branch.

(We hasten to add that when the predicted branch is actually determined, that information should replace the predicted value in the BHR. Unfortunately, this branch predictor was designed by an Aggie, and he never understood that the branch prediction is better than no information, but better still is the actual direction of the branch. So, in this problem we are stuck with the Aggie's design, i.e., values in the BHR are always predicted values.)



	PHT Before	PHT After
0	0 0	
1	1 0	
2	0 1	
3	0 1	
4	1 0	
5	1 1	
6	0 0	
7	1 1	
8	1 0	
9	0 1	
10	0 1	
11	0 0	
12	1 0	
13	1 1	
14	1 0	
15	0 0	

Part a (6 points):

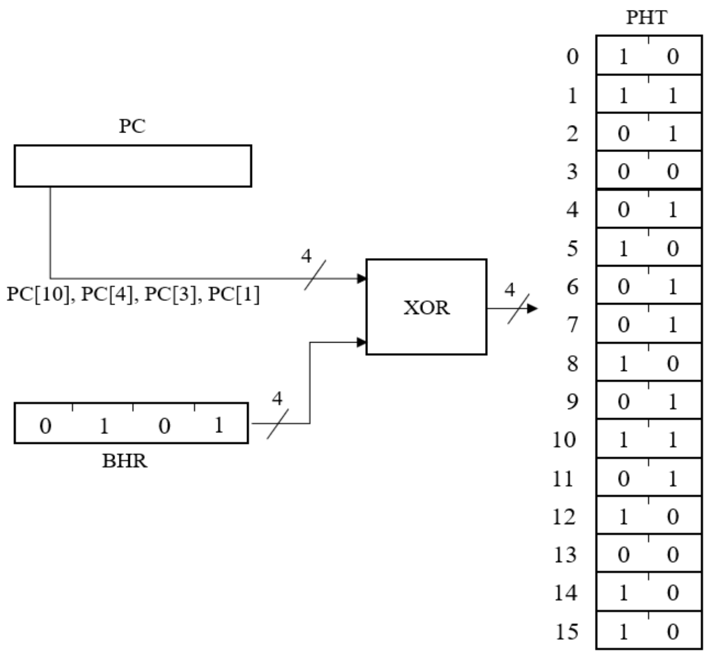
Given the snapshot of the branch predictor shown above, what is the index into the PHT? [ ]

Is the branch predicted taken or not taken? [ ]

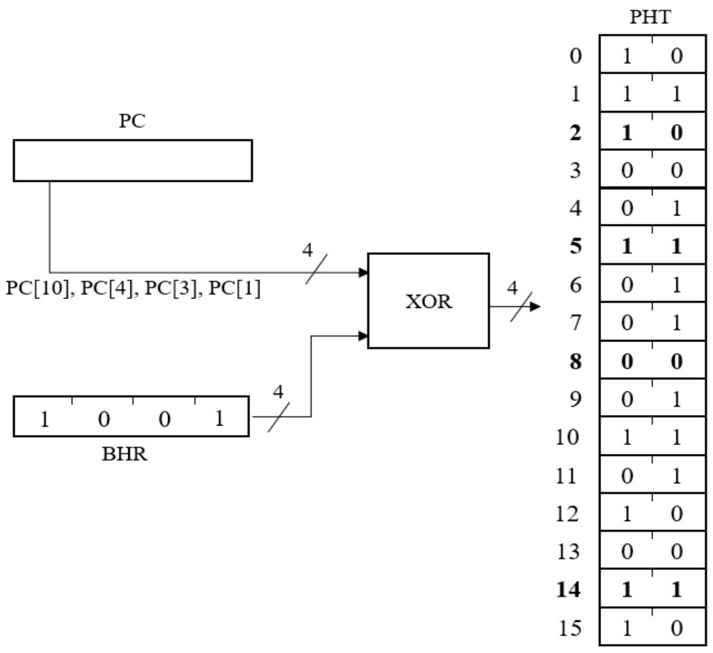
Assume the branch is actually taken. Show the BHR after this branch is processed. Show the updated PHT entry in the column labeled PHT after as a result of this branch. It is not necessary to copy the other 15 entries in the PHT after that are unchanged as a result of this branch.

PROBLEM CONTINUES ON NEXT PAGE

Part b (14 points): The branch predictor is initially in the state shown below executes five branch instructions at addresses 0x3004, 0x300a, 0x3012, 0x301c, and 0x3028.



After executing these five branch instructions, the state of the branch predictor is as shown below. Entries 2, 5, 8, and 14 are the only ones that changed.



PROBLEM CONTINUES ON NEXT PAGE

Name:____________________________________

Your job: Using the before and after state of the branch predictor shown on the previous page, complete the table below with the predicted and actual branch direction values for the five branches.

Branch PC	**Prediction (Taken/Not Taken)**	**Actual (Taken/Not Taken)**
0x3004		
0x300a		
0x3012		
0x301c		
0x3028		

Name:____________________________________

Problem 4 (20 points):

Assume memory locations x1000 to x1009 contain values as shown:

x1000	A
x1001	B
x1002	C
x1003	D
x1004	E
x1005	F
x1006	G
x1007	H
x1008	I
x1009	J

Assume:

- Memory is byte-addressable, little endian
- Memory address and data buses are independent and 16-bits each
- It takes 2 cycles after an address is sent to memory to latch a row address
- It takes 4 cycles to open a new row
- It takes 1 cycle to read the row buffer
- Data is on the data bus in the next cycle after the row buffer is read
- A new address can be supplied on the address bus in the same cycle that data is returned on the data bus

If we access memory in sequence starting with location x1000, we observe the following behavior on the memory address and data buses.

Time	Addr	Data
0	0x1000	
1	0x1002	
8	0x1004	BA
9	0x1006	DC
10	0x1008	FE
11		HG
18		JI

Part a (5 points): Given the access latencies above, and restricting the number of banks and columns to their minimum size, how many bank bits (B), column bits (C), row bits (R) and byte-on-bus bits (E) make up the 16 bit address to memory.

Bank: $\boxed{}$ Column: $\boxed{}$ Row: $\boxed{}$ Byte on Bus: $\boxed{}$

Specify with the letters B, C, R, E which bits of the 16-bit address belong to each.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

PROBLEM CONTINUES ON NEXT PAGE

Name:___________________________________

Part b (5 points): If instead of accessing successive memory locations, we access every 16th location, our address stream will look like the following: 0x1000, 0x1010, 0x1020, 0x1030, 0x1040.

How many cycles does it take to complete this access stream?

Number of cycles

Part c (5 points): How would you change the assignments of address bits to bank, row, and column to minimize latency? Note: we are not changing the memory architecture, i.e., we keep the same number of bank, row, and column bits.

Again, specify with the letters B, C, R, E which bits of the 16-bit address belong to each.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Part d (5 points): How many cycles does it take to complete the access stream from part b after making the change in part c?

Number of cycles

Name:___________________________________

Problem 5 (25 points):

Consider an out-of-order processor which executes its instructions based on the Tomasulo algorithm. The ISA specifies 8 registers, R0 to R7. The execute stage of the pipeline contains one pipelined adder and one pipelined multiplier. Fetch and Decode take a single cycle each. The ADD instruction takes 4 cycles to execute; the MUL instruction takes 5 cycles to execute. Both ADD and MUL take an additional cycle to store the result into a register and/or any reservation stations waiting for it. There is no data forwarding/bypassing implemented in this system.

Only a single instruction can store its result at a time. If two attempt to store their results at the same time, the older instruction has priority. The newer instruction will stall in execute until it can store its result.

The adder and multiplier each have 2-entry reservation stations. The reservation stations are initially empty and are filled from top to bottom. Each instruction remains in the reservation station until the end of the cycle in which it writes its result to a register.

The state of the register file is shown before execution, after cycle 7, and after all five instructions have completed.

	V	Tag	Value
R0	1	-	15
R1	1	-	2
R2	1	-	7
R3	1	-	4
R4	1	-	5
R5	1	-	37
R6	1	-	10
R7	1	-	3

Before Cycle 1

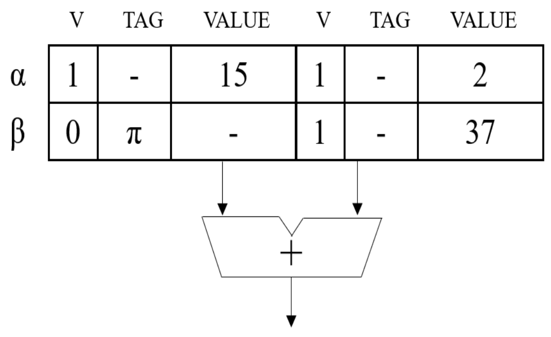
	V	Tag	Value
R0	1	-	15
R1	1	-	2
R2	0	α	-
R3	0	π	-
R4	0	β	-
R5	1	-	37
R6	1	-	10
R7	1	-	3

After Cycle 7

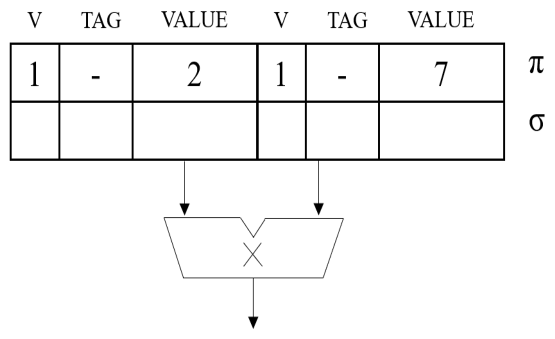
	V	Tag	Value
R0	1	-	15
R1	1	-	19
R2	1	-	111
R3	1	-	14
R4	1	-	51
R5	1	-	37
R6	1	-	10
R7	1	-	3

After Execution

The reservation stations are shown after cycle 7:



	V	TAG	VALUE	V	TAG	VALUE
α	1	-	15	1	-	2
β	0	π	-	1	-	37



V	TAG	VALUE	V	TAG	VALUE	
1	-	2	1	-	7	π
						σ

PROBLEM CONTINUES ON NEXT PAGE

The table below shows the cycles in which each functional unit is executing an instruction. An asterisk (*) indicates a stall.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adder				E	E	E	E	E*	E	E	E	E	E			
Multiplier			E	E	E	E	E				E	E	E	E	E	

Part a (12 points): Determine the five instructions that were executed.

	Opcode	**DR**	**SR1**	**SR2**
I1				
I2				
I3				
I4				
I5				

Part b (5 points): Complete the timing diagram for the execution of the five instructions. Indicate the stage each instruction is occupying 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, add an asterisk (*) to that cycle.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I1	F	D														
I2		F														
I3																
I4																
I5																

Part c (8 points): If an additional entry is added to the reservation station for both functional units, how many cycles would it take to execute the same five instructions? Explain.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD^{+}	0001				DR			SR1			0	00		SR2		
ADD^{+}	0001				DR			SR1			1	imm5				
AND^{+}	0101				DR			SR1			0	00		SR2		
AND^{+}	0101				DR			SR1			1	imm5				
BR	0000				n	z	p	PCoffset9								
JMP	1100				000			BaseR			000000					
JSR	0100				1	PCoffset11										
JSRR	0100				0	00		BaseR			000000					
LDB^{+}	0010				DR			BaseR			boffset6					
LDW^{+}	0110				DR			BaseR			offset6					
LEA^{+}	1110				DR			PCoffset9								
NOT^{+}	1001				DR			SR			1	11111				
RET	1100				000			111			000000					
RTI	1000				000000000000											
LSHF^{+}	1101				DR			SR			0	0	amount4			
RSHFL^{+}	1101				DR			SR			0	1	amount4			
RSHFA^{+}	1101				DR			SR			1	1	amount4			
STB	0011				SR			BaseR			boffset6					
STW	0111				SR			BaseR			offset6					
TRAP	1111				0000				trapvect8							
XOR^{+}	1001				DR			SR1			0	00		SR2		
XOR^{+}	1001				DR			SR			1	imm5				
not used	1010															
not used	1011															

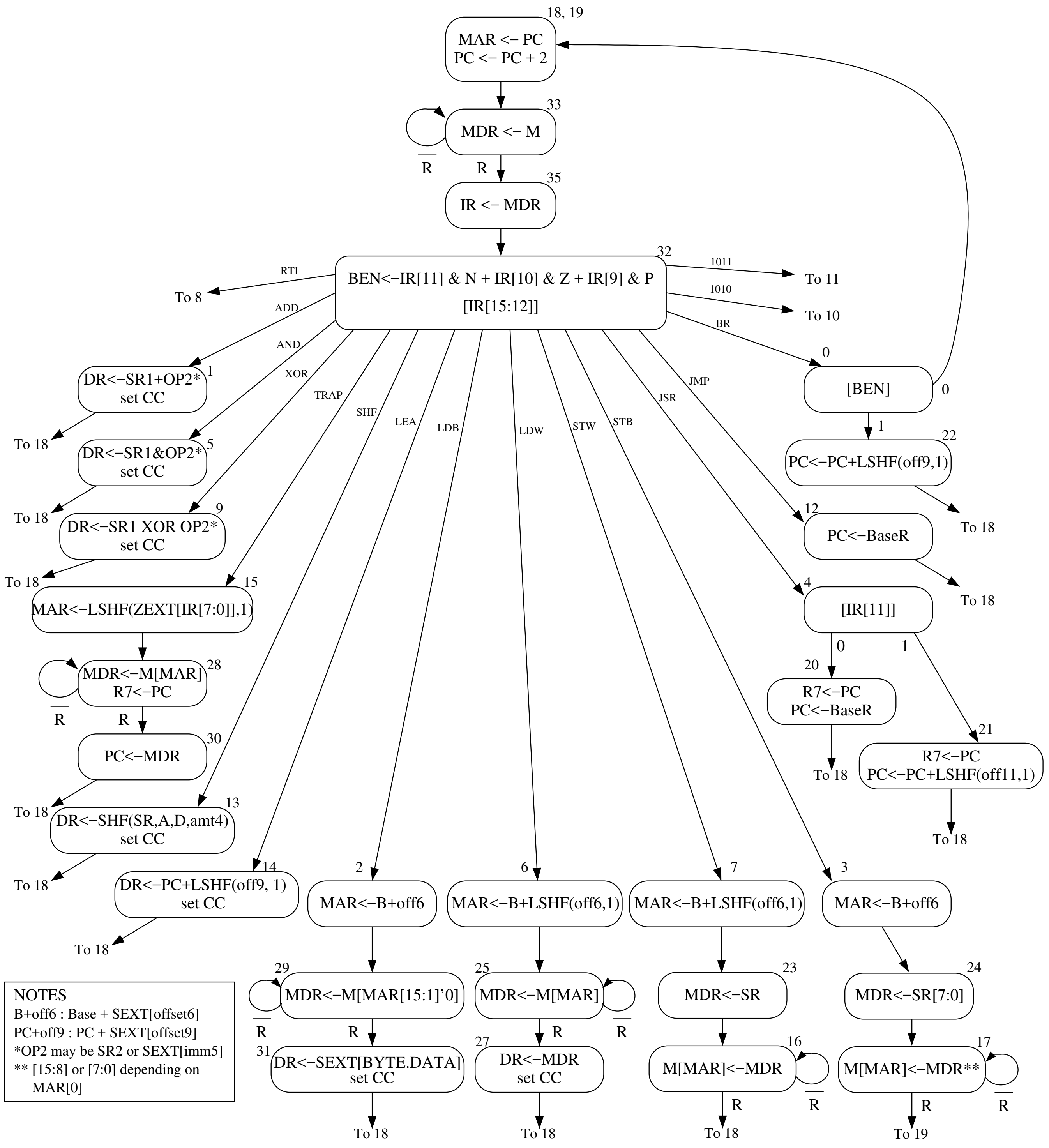
Figure 1: LC-3b Instruction Encodings

Table 1: Data path control signals

Signal Name	Signal Values	
LD.MAR/1:	NO(0), LOAD(1)	
LD.MDR/1:	NO(0), LOAD(1)	
LD.IR/1:	NO(0), LOAD(1)	
LD.BEN/1:	NO(0), LOAD(1)	
LD.REG/1:	NO(0), LOAD(1)	
LD.CC/1:	NO(0), LOAD(1)	
LD.PC/1:	NO(0), LOAD(1)	
GatePC/1:	NO(0), YES(1)	
GateMDR/1:	NO(0), YES(1)	
GateALU/1:	NO(0), YES(1)	
GateMARMUX/1:	NO(0), YES(1)	
GateSHF/1:	NO(0), YES(1)	
PCMUX/2:	PC+2(0)	;select pc+2
	BUS(1)	;select value from bus
	ADDER(2)	;select output of address adder
DRMUX/1:	11.9(0)	;destination IR[11:9]
	R7(1)	;destination R7
SR1MUX/1:	11.9(0)	;source IR[11:9]
	8.6(1)	;source IR[8:6]
ADDR1MUX/1:	PC(0), BaseR(1)	
ADDR2MUX/2:	ZERO(0)	;select the value zero
	offset6(1)	;select SEXT[IR[5:0]]
	PCoffset9(2)	;select SEXT[IR[8:0]]
	PCoffset11(3)	;select SEXT[IR[10:0]]
MARMUX/1:	7.0(0)	;select LSHF(ZEXT[IR[7:0]],1)
	ADDER(1)	;select output of address adder
ALUK/2:	ADD(0), AND(1), XOR(2), PASSA(3)	
MIO.EN/1:	NO(0), YES(1)	
R.W/1:	RD(0), WR(1)	
DATA.SIZE/1:	BYTE(0), WORD(1)	
LSHF1/1:	NO(0), YES(1)	

Table 2: Microsequencer control signals

Signal Name	Signal Values	
J/6:		
COND/2:	$COND_0$	;Unconditional
	$COND_1$	;Memory Ready
	$COND_2$	;Branch
	$COND_3$	;Addressing Mode
IRD/1:	NO, YES	



NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on
MAR[0]

Figure 2: A state machine for the LC-3b

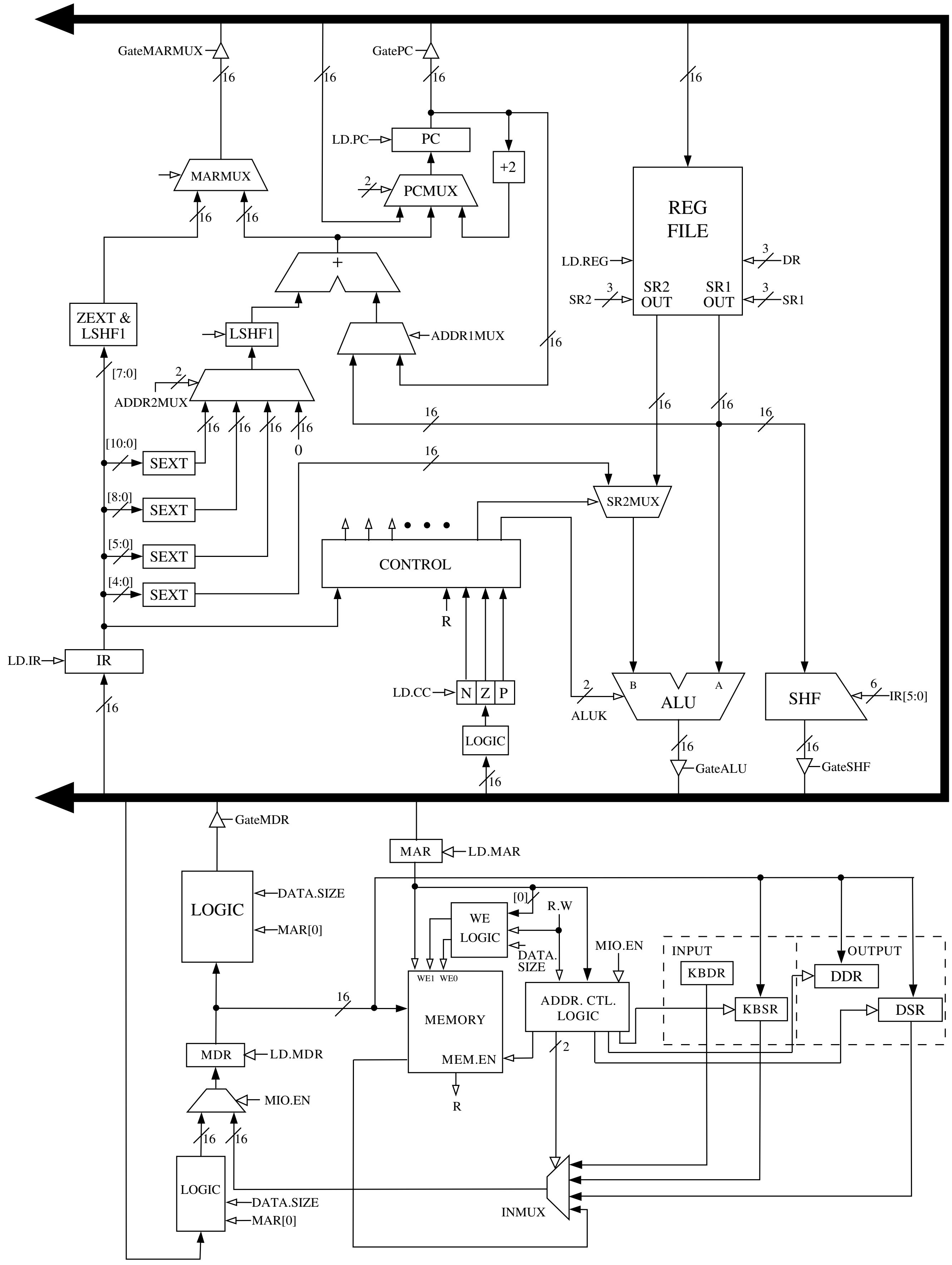


Figure 3: The LC-3b data path

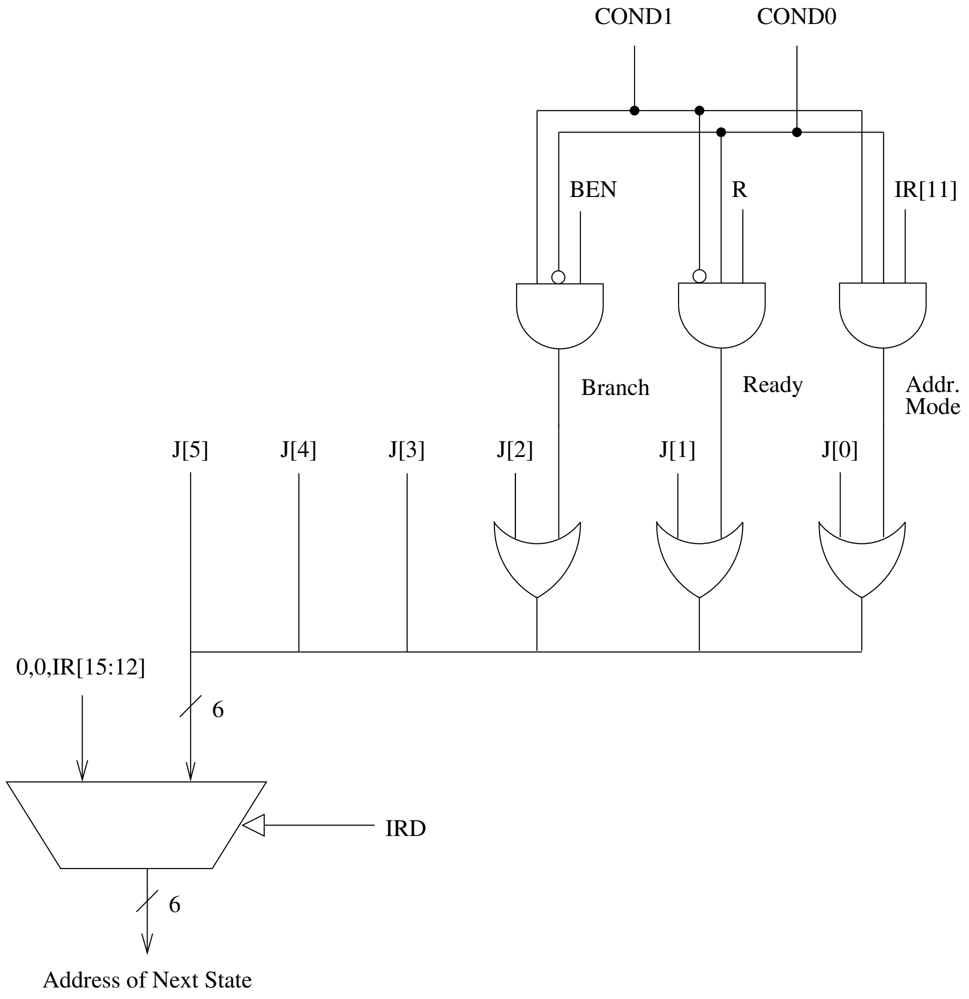


Figure 4: The microsequencer of the LC-3b base machine