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

EE 382N.1 Fall 2018
Dam Sunwoo, Instructor
Pritesh Chhajed, TA
Exam 1
October 10, 2018

Name: SOLUTION EID:_____

Problem 1 (15 points):_____

Problem 2 (15 points):_____

Problem 3 (10 points):_____

Problem 4 (20 points):_____

Problem 5 (15 points):_____

Problem 6 (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: I DONT KNOW is a valid answer that automatically gives you 15% of the maximum possible grade. Be sure to either erase, or cross-out everything else and write I DONT KNOW in, large, capital letters. A blank answer does not get you the 15%. No partial I DONT KNOW credit (only on entire questions/parts that have a marked point value on the exam)

Note: The exam is open book, open notes, bring anything type of exam, but there are constraints: Do not ask anyone or anything for help. You may only search for text in PDFs or other material that you have downloaded. NO web searches. Do not share any material related to the exam with anyone (or upload it anywhere). The exam will be made public soon.

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 (15 points): Answer the following questions.

Part a (4 points): Under out-of-order execution, how do we enable precise state recovery for store instructions?

Writes to memory (caches) are buffered until the store instruction becomes the oldest executed inst in ROB.

Part b (4 points): If the desired functional unit is busy, scoreboarding stalls the frontend of the pipeline. The Tomasulo algorithm does not stall the pipeline in this case (at least not immediately). How is this stall avoided with the Tomasulo algorithm?

Instructions wait in Reservation Stations until FUs become available.

Part c (7 points): In two-level adaptive branch prediction, the Branch History Register (BHR) is typically updated *speculatively* using *predicted* branch outcomes (taken or not taken), whereas the Pattern History Table (PHT) is updated with the *actual* branch outcomes when they become available. Why is this?

BHR is updated speculatively as actual outcome may not be available yet. (due to pipelining). Using "some" info is still better than nothing. Typical high accuracy helps.

Name:_____

Problem 2 (15 points): Trade-offs

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 a (5 points):

What is the Cycles Per Instruction (CPI) of this CPU running this target workload?

$0.1 \times 3 + 0.1 \times 5 + 0.8 \times 1 = 1.6$

1.6

Part b (5 points): An improvement in the divider makes the `DIV` instruction take 1 cycle to execute, but causes the overall cycle time to increase by 30%. Does this new design make the workload run faster than the original CPU? Explain.

New ~~perf~~ exec time :

$(0.1 \times 3 + 0.1 \times 1 + 0.8 \times 1) \times 1.3 = 1.56 < 1.6$ Faster

Part c (5 points): An improvement in both the multiplier and divider makes the `MUL` instruction take 1 cycle and the `DIV` 2 cycles, respectively, but causes the overall cycle time to increase by 50%. Does this new design make the workload run faster than the original CPU? Explain.

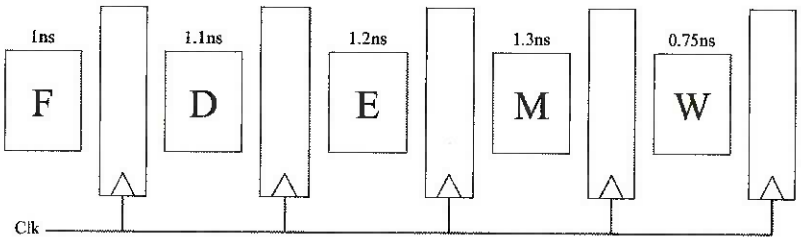
$(0.1 \times 1 + 0.1 \times 2 + 0.8 \times 1) \times 1.5 = 1.65 > 1.6$

Slower

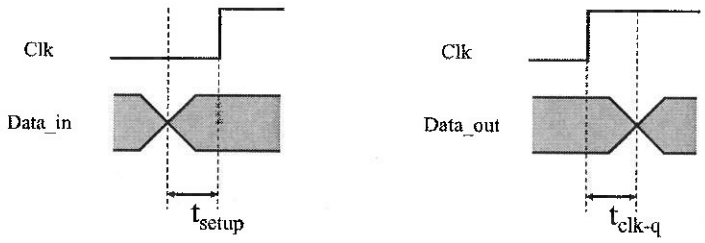
Name:_____

Problem 3 (10 points): Pipelining

Consider an in-order 5-stage pipeline (F-D-E-M-W) processor. The critical path delay for the combinational logic in each stage is shown below.



Flip-flops are used to latch the data. You need to consider the setup time and clock-to-Q delay of the register in your cycle time calculation. Setup time (t_{setup}) is defined as the minimum amount of time before the clock's active edge that the input data must be stable for it to be latched correctly. For an edge-triggered flip-flop, the clock-to-Q delay (t_{clk-q}) is the time it takes for the register output to be in a stable state after a clock edge occurs. See the figure below:



For this design, assume $t_{setup} = 0.15$ns and $t_{clk-q} = 0.05$ns, respectively.

Part a (4 points):

What is the highest frequency at which this pipeline can operate correctly?

Cycle time = 1.3 + 0.15 + 0.05 = 1.5 ns

0.67 GHz

Part b (6 points):

We are considering adding data forwarding (bypassing) to this design. Performance evaluation showed that implementing data forwarding from the Execute stage to the Decode stage can improve Instructions Per Cycle (IPC) by 15% for our workloads. However, forwarding requires adding a multiplexer at the end of the Decode stage, which would add 0.4ns to the critical path. Note that the forwarded data of the Execute stage is on the critical path of the Execute stage.

If we add this forwarding logic, what would be the highest operating frequency? Would you recommend implementing data forwarding in this design for performance? Why or why not?

Data forwarded from Execute stage is on critical path.

Cycle time = 1.2ns + 0.15ns + 0.05ns + 0.4ns = 1.8ns → 0.56 GHz

$1.15 \times \frac{1.5\text{ns}}{1.8\text{ns}} = 0.958\ldots < 1$ Data forwarding does NOT improve performance.

Name:_____

Problem 4 (20 points): Branch Prediction

An LC-3b system has a GAg (global BHR and global PHT) 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:

INDEX[3] = BHR[3] xor PC[10]

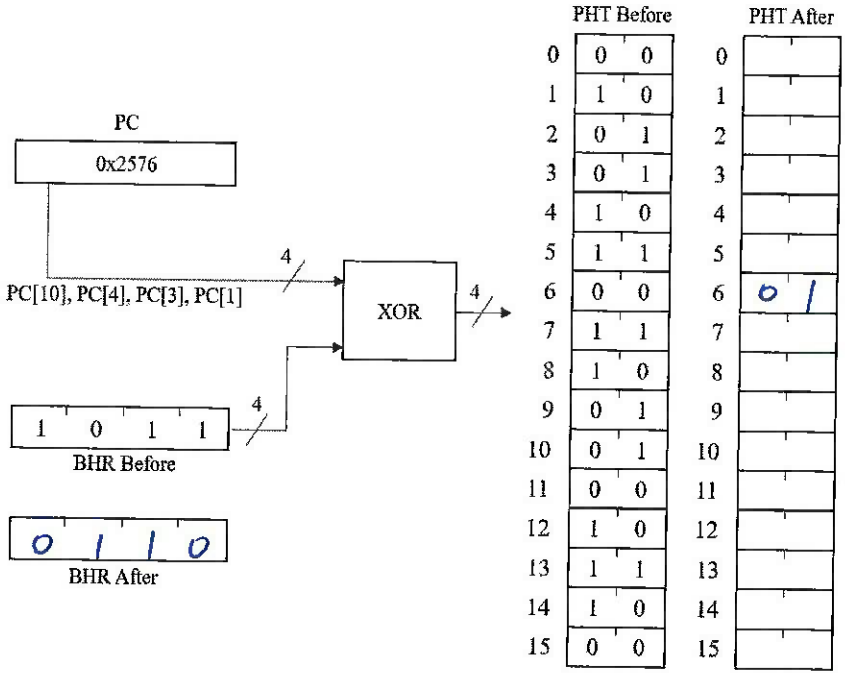
INDEX[2] = BHR[2] xor PC[4]

INDEX[1] = BHR[1] xor PC[3]

INDEX[0] = BHR[0] xor 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. The PHT entries are standard saturating 2-bit counters that increment or decrement by 1 unless saturated.

(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 the reasoning behind using the speculative value or 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.)



Part a (6 points):

Given the snapshot of the branch predictor shown above, what is the index into the PHT? 6

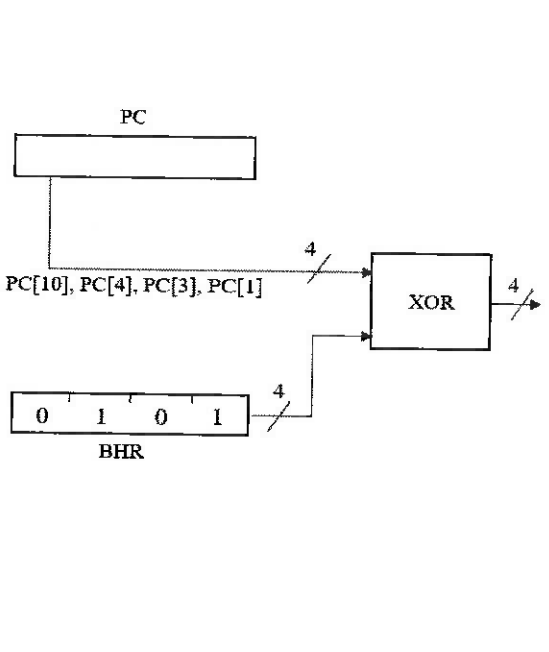
Is the branch predicted taken or not taken? Not taken

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.

Name:_____

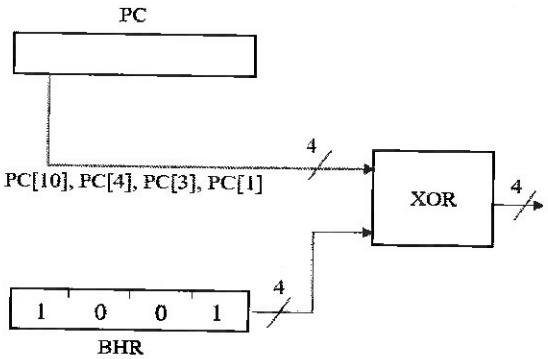
Part b (14 points):

The branch predictor initially in the state shown below executes five branch instructions at addresses 0x3004, 0x300a, 0x3012, 0x301c, and 0x3028, in sequence.



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

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.



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

PROBLEM CONTINUES ON NEXT PAGE

Name:_____

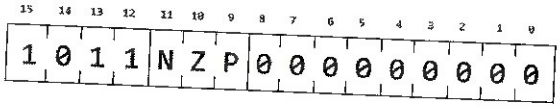
Your job: Using the before and after state of the branch predictor shown on the previous page, complete the table below with the predicted and actual branch direction values for the five branches.

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

Name:_____

Problem 5 (15 points):

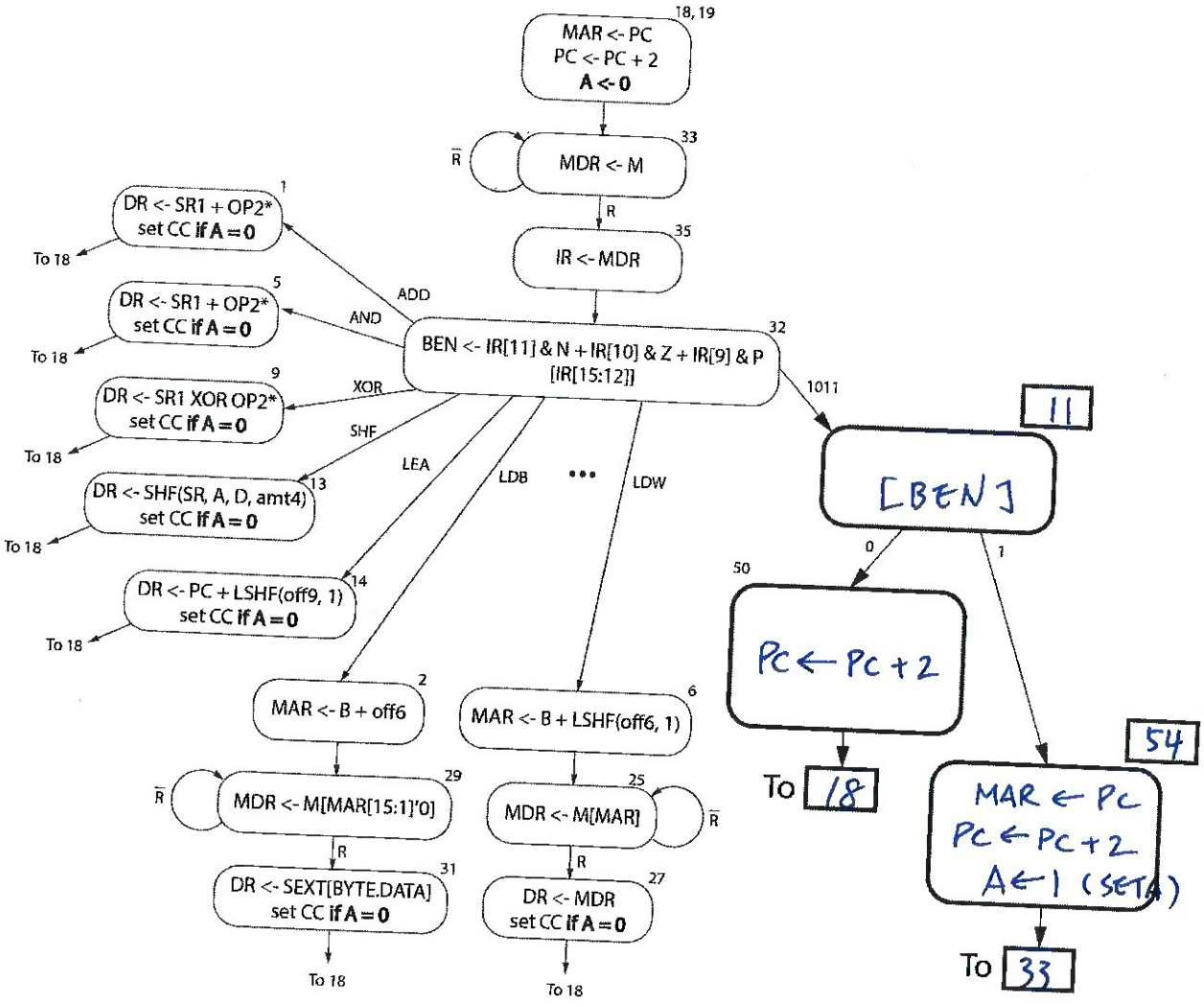
Predicated execution is an alternative to conditional branches. Each instruction that is associated with a predicate will only be executed if the predicate is true. We can add predicated execution to the LC-3b using the unused opcode 1011, much like the prefixes in the x86 ISA. That is, if we wish to predicate instruction X, we put the following prefix just before instruction X in our program:



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.

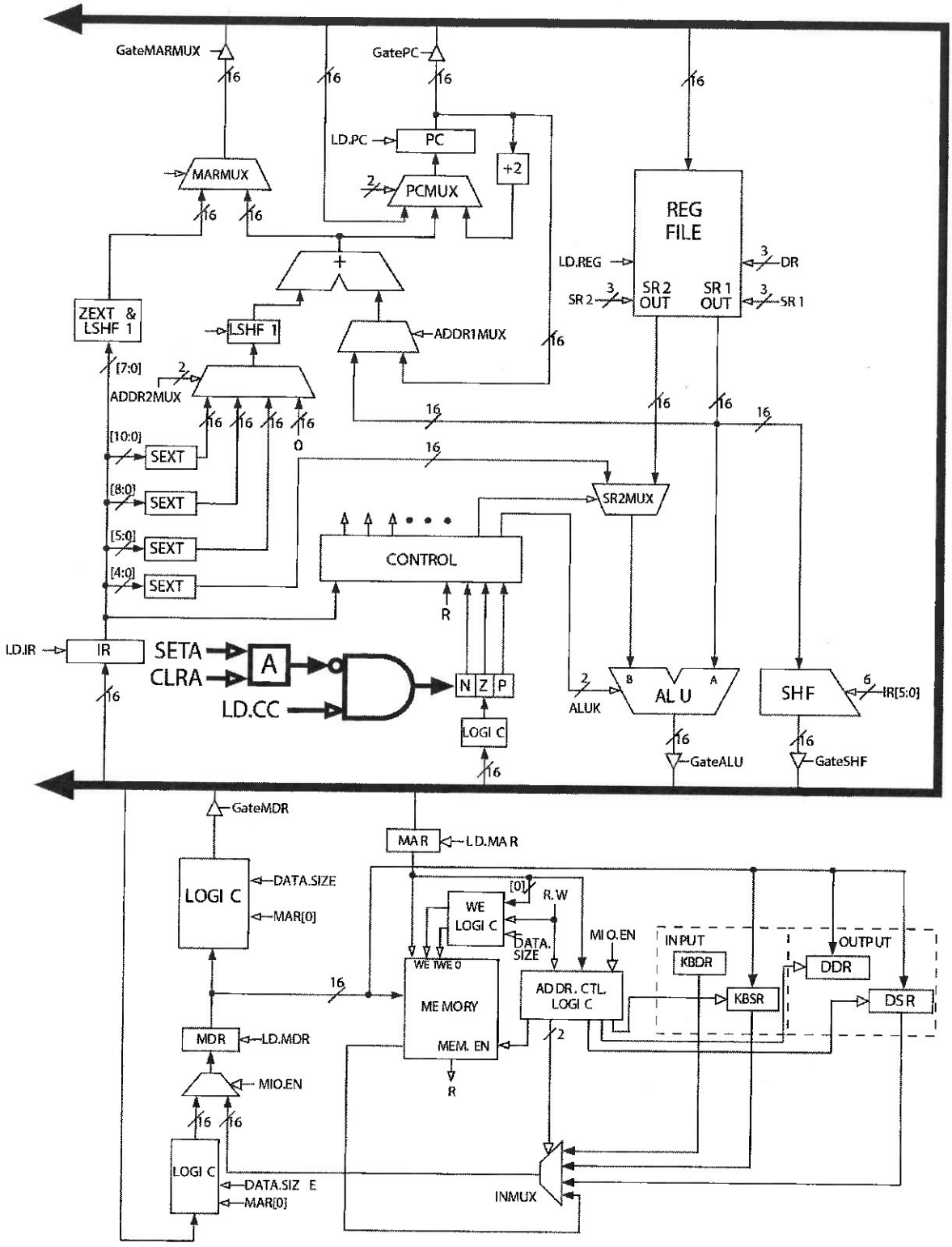
Whether or not instruction X is executed, we wish to preserve the condition codes so we can predicate additional instructions based on the same condition. Therefore, if instruction X is executed, it must not set condition codes. To make this work, we need to augment the data path (next page) with a flip-flop 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 6 (25 points): Tomasulo/Out-of-Order

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 each take a single cycle. The ADD instruction takes 4 cycles to execute, and 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. In this design, the tags point to reservation station entries instead of separate physical registers.

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

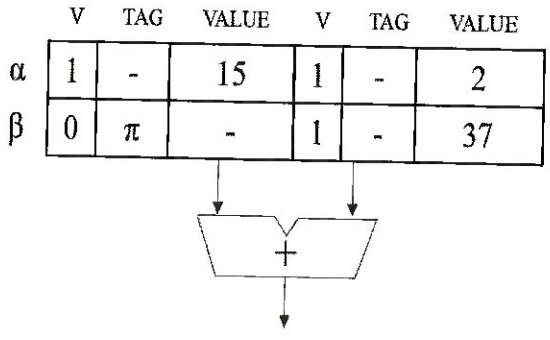
	V	Tag	Value
R0	1	-	15
R1	1	-	2
R2	0	α	-
R3	0	π	-
R4	0	β	-
R5	1	-	37
R6	1	-	10
R7	1	-	3

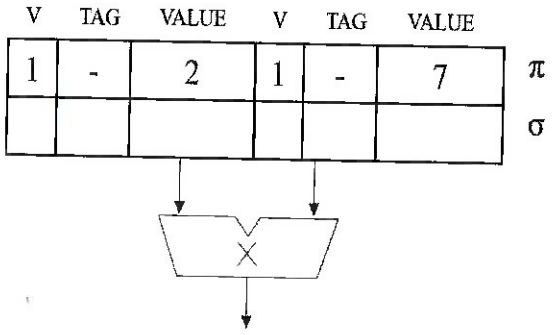
After Cycle 7

	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:





Name:_____

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	MUL	R3	R1	R2
I2	ADD	R2	R0	R1
I3	ADD	R4	R3	R5
I4	ADD	R1	R1	R2
I5	MUL	R2	R5	R7

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	E	E	E	E	E	R3								
I2		F	D	E	E	E	E	E*	R2							
I3			F	D	–	–	–	–	E	E	E	E	R4			
I4				F	D	D*	D*	D*	D*	E	E	E	E	R1		
I5					F	F*	F*	F*	F*	D	E	E	E	E	E	R2

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.

I5 can start executing earlier.

I4 will still take the same time due to RAW hazard.

14 cycles

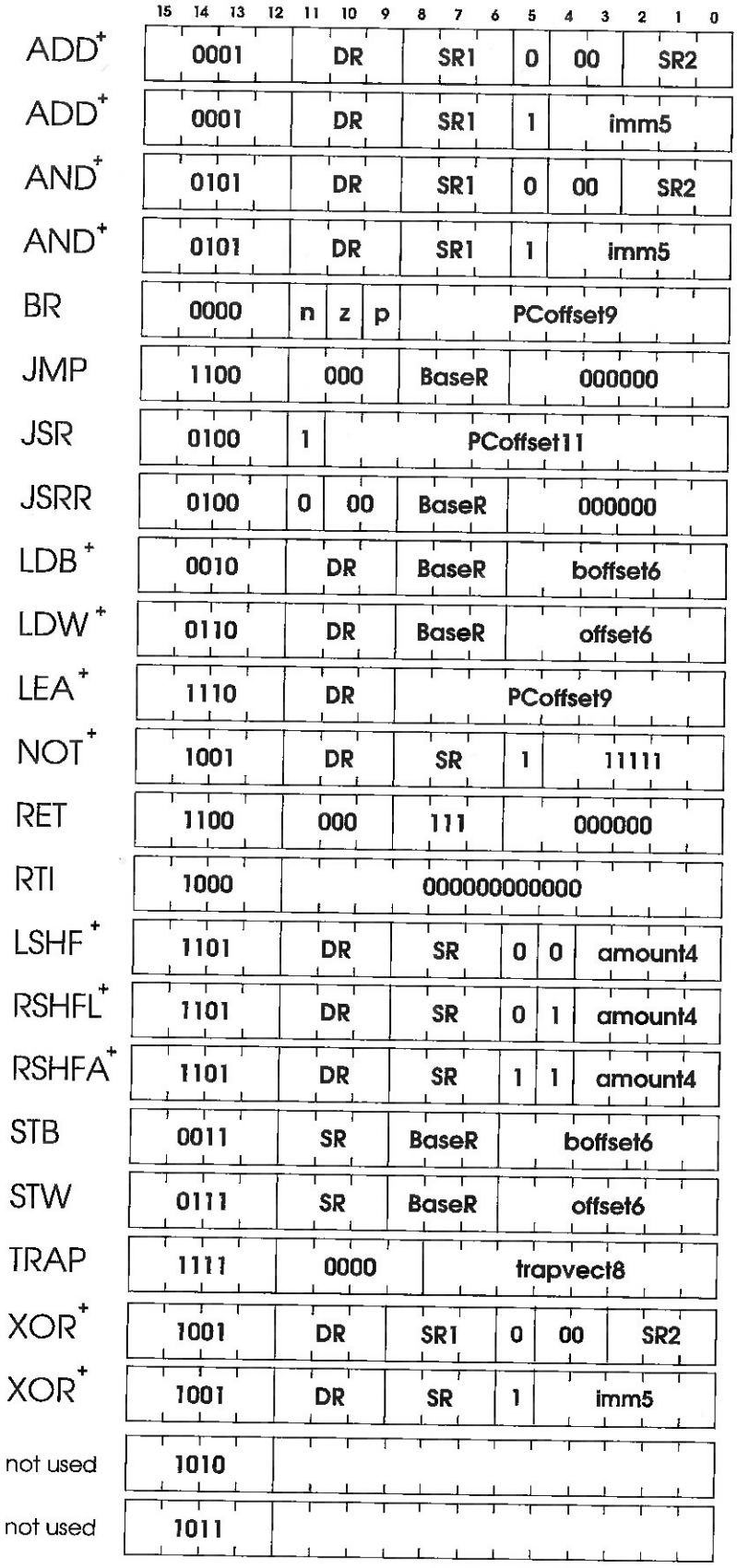


Figure 1: LC-3b Instruction Encodings

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	

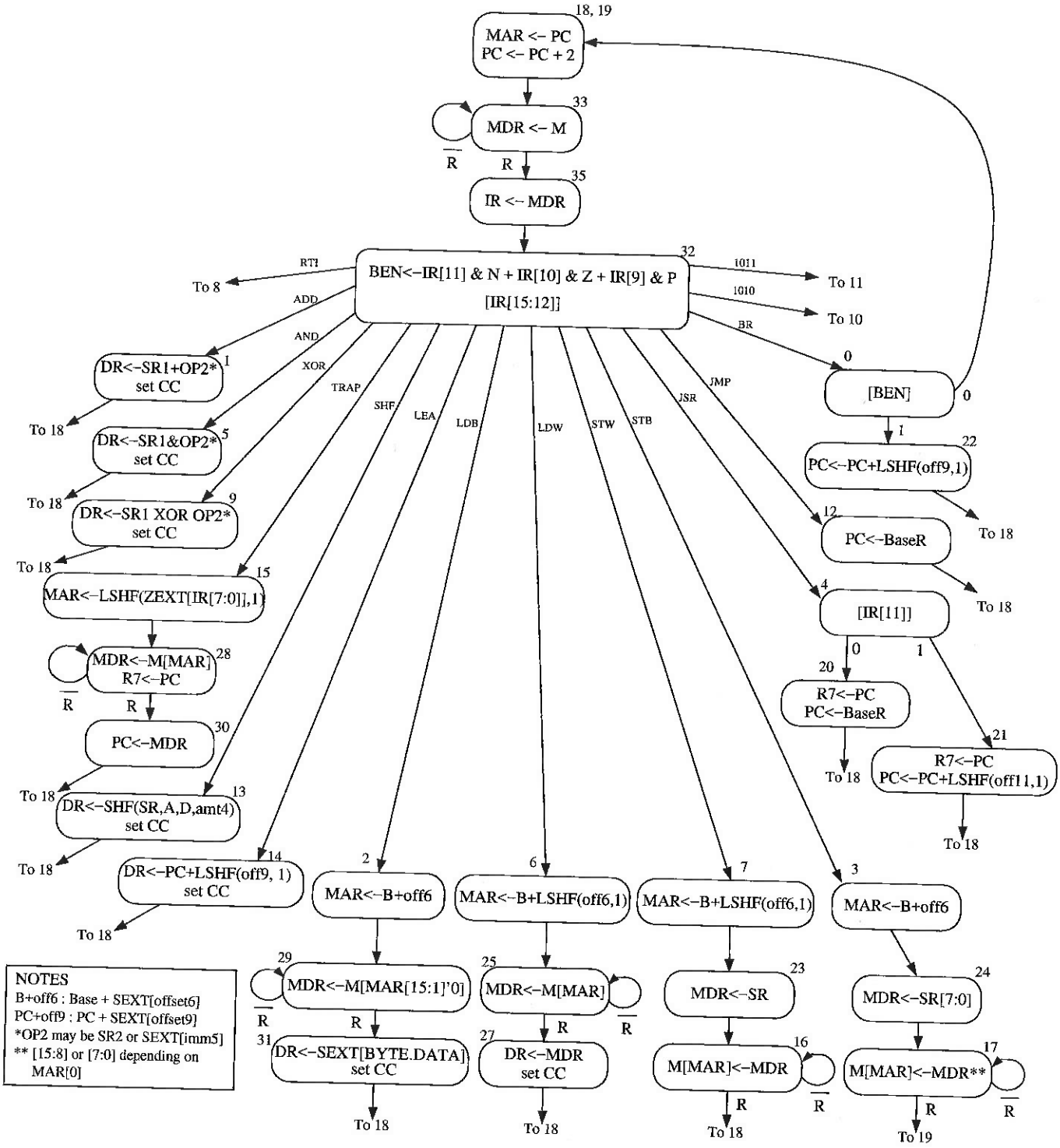


Figure 2: A state machine for the LC-3b

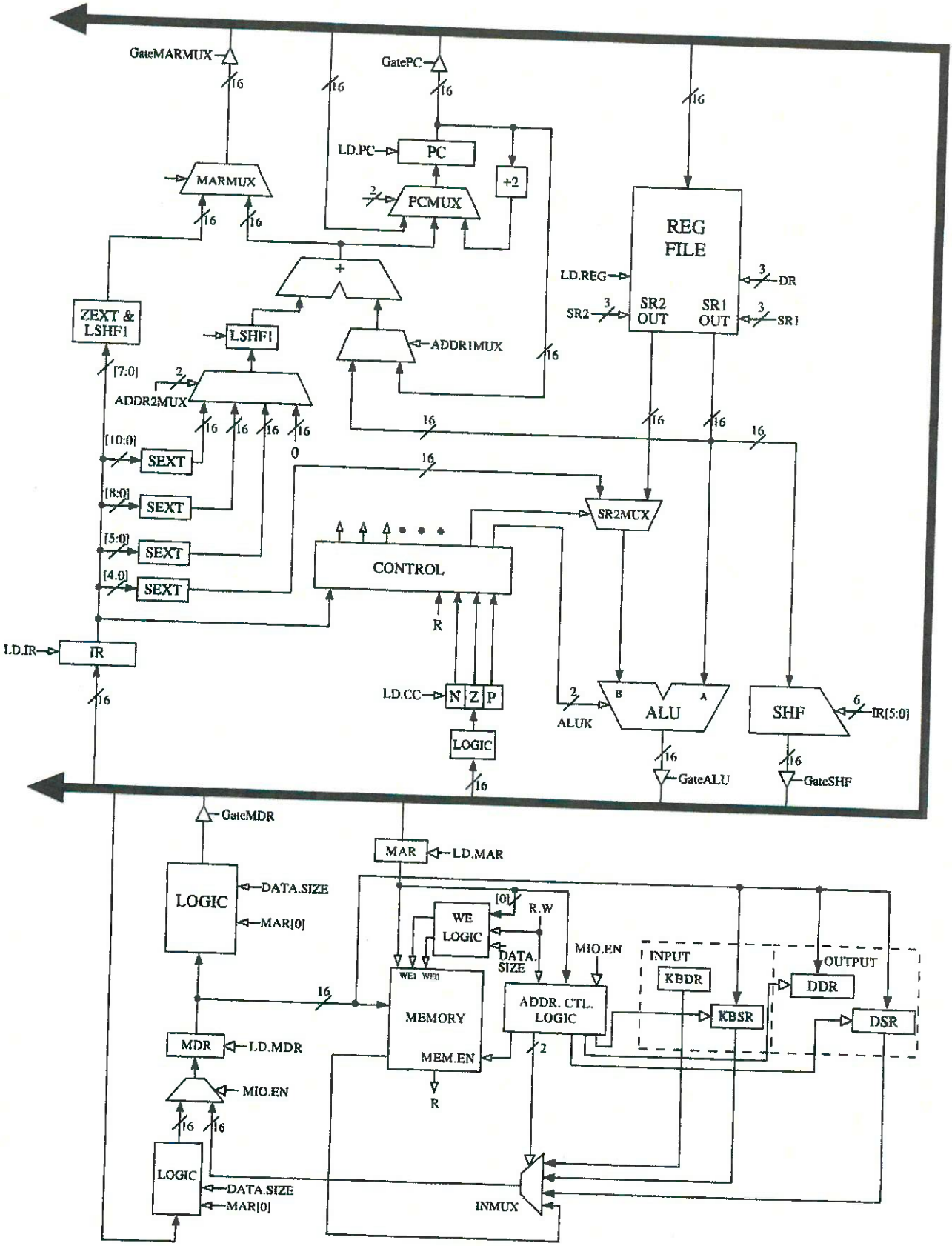


Figure 3: The LC-3b data path

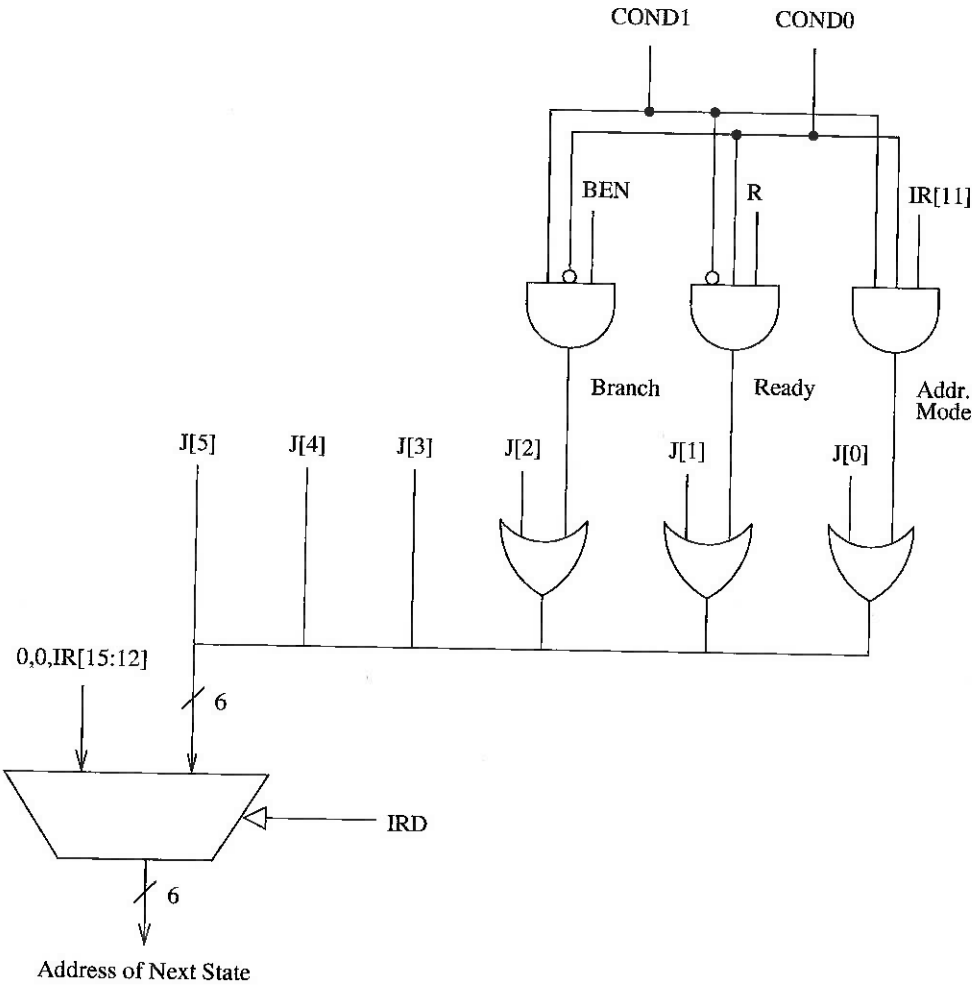


Figure 4: The microsequencer of the LC-3b base machine