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

EE 460N Spring 2017
Y. N. Patt, Instructor
Chirag Sakhuja, Sarbartha Banerjee, Jonathan Dahm, Arjun Teh, TAs
Exam 1
March 1, 2017

Name:____

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 sign the following. I have not given nor received any unauthorized help on this exam.

Signature:____

GOOD LUCK!

Name:___________________________________

Problem 1 (25 points): Answer any five of the six. Draw a line through the one you do not want graded.

Part a (5 points): A load-store ISA does not allow what?

Part b (5 points): Unaligned accesses. Part of the ISA or part of the microarchitecture? Explain.

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

Part d (5 points): J.E.Smith's branch predictor introduced the use of saturating 2-bit counters. What does the word "saturating" mean in this context, and why is it necessary?

Part e (5 points): McFarling modified my GAs predictor, creating g-share. What was his purpose for doing so?

Part f (5 points): Do processes having higher priority get increased privilege? If yes, explain why. If no, explain why not.

Name:___________________________________

Problem 2 (10 points): An Aggie hates the LC-3b, so he cuts the 16 wires labeled A and B on the data path, and grounds the wire labeled C in the microsequencer so the LC-3b can not function properly. (The data path, showing wires A and B is shown on the next page.)

Part a (2 points): If the 16 wires A are cut, which instruction(s) are impossible to function? Explain.

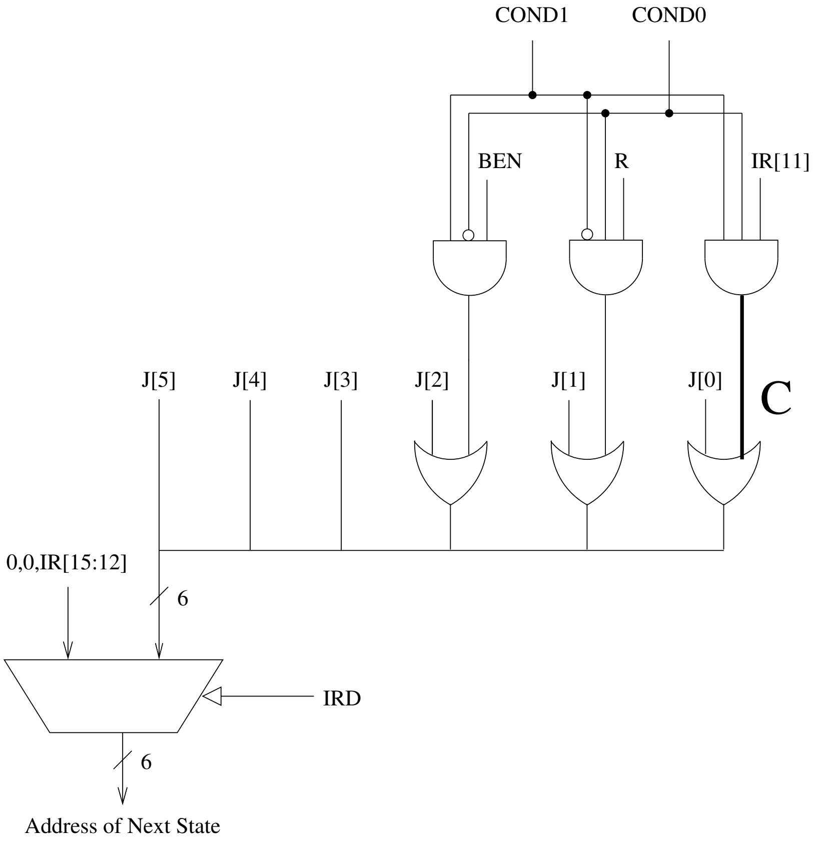
Instruction(s)	Explanation

Part b (4 points): If the 16 wires B are cut, which instruction(s) are impossible to function? Explain.

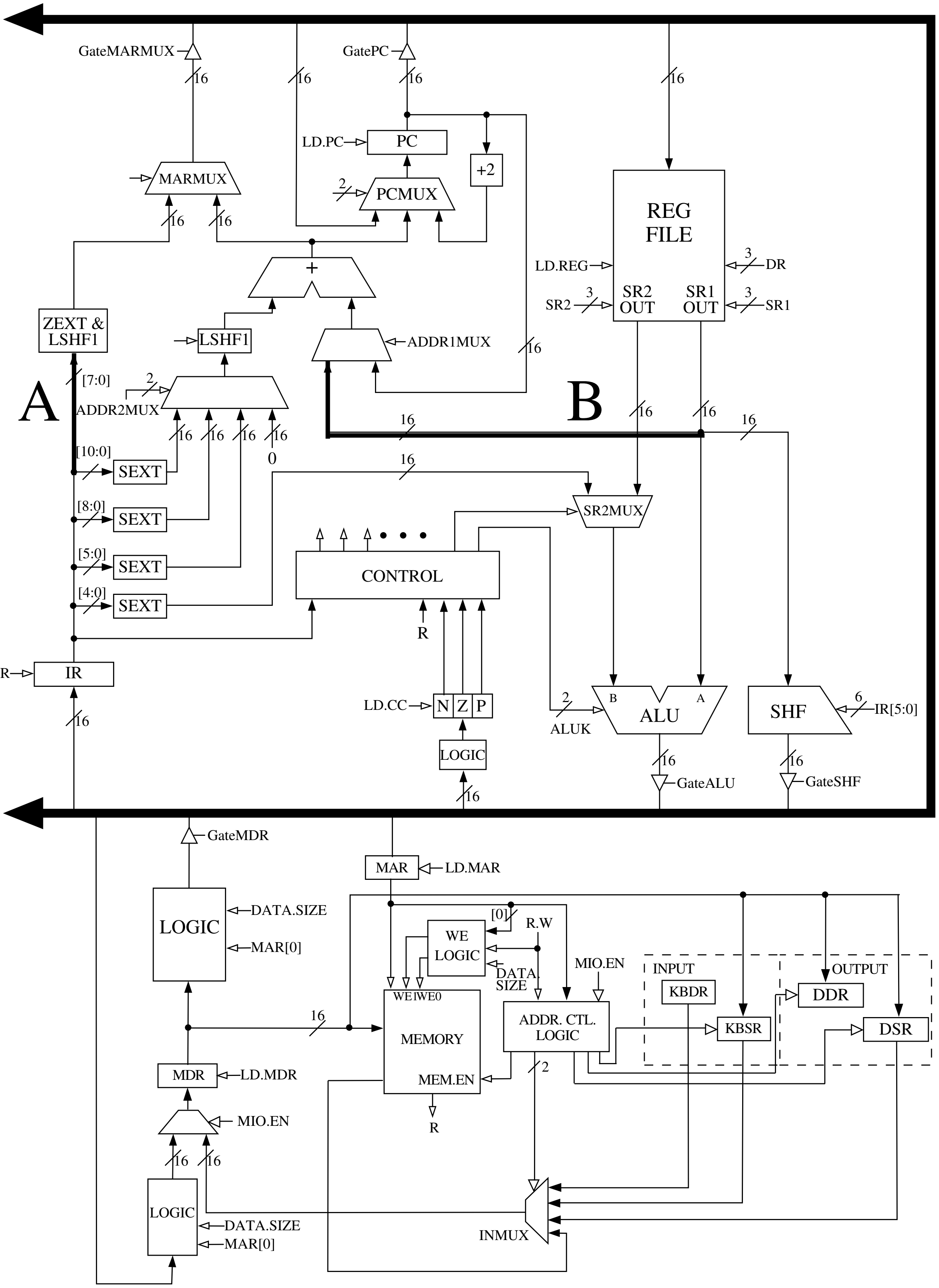
Instruction(s)	Explanation

Part c (4 points): If wire C is grounded, which instruction(s) are impossible to function? Explain.

Instruction(s)	Explanation



Name:________________________________


GateMARMUX
16
16
GatePC
16
16
LD.PC
PC
+2
MARMUX
2
PCMUX
16
16
REG
FILE
LD.REG
3
DR
3
SR2
SR2
OUT
SR1
OUT
3
SR1
+
ZEXT &
LSHF1
LSHF1
ADDR1MUX
16
[7:0]
A
2
ADDR2MUX
B
16
16
16
16
16
16
16
16
16
0
[10:0]
SEXT
16
[8:0]
SEXT
SR2MUX
[5:0]
SEXT
CONTROL
[4:0]
SEXT
R
LD.IR
IR
16
LD.CC
N
Z
P
2
ALUK
B
A
ALU
SHF
6
IR[5:0]
LOGIC
16
16
GateALU
16
GateSHF
GateMDR
MAR
LD.MAR
LOGIC
DATA.SIZE
MAR[0]
[0]
R.W
WE
LOGIC
DATA.
SIZE
MIO.EN
INPUT
OUTPUT
KBDR
DDR
WE1WE0
ADDR. CTL.
LOGIC
16
MEMORY
KBSR
DSR
MDR
LD.MDR
MEM.EN
2
R
MIO.EN
16
16
LOGIC
DATA.SIZE
MAR[0]
INMUX

Name:___________________________________

Problem 3 (15 points): In this problem we add an SAg 2-level branch predictor to the LC-3b. Recall that the S means the branches are partitioned into sets, where all branches in a set share the same BHR. In our case, we have 8 sets, and therefore 8 BHRs. Bits 13, 10, and 7 of a branch's address determine which set a branch belongs to. For example, a branch at address x9E84 uses BHR 3 because bit 13 is 0, and bits 10 and 7 are 1.

The current state of the BHRs are shown below. The direction (taken = 1, not taken = 0) of the most recent branch is the right-most bit of its respective BHR.

0	0	1	1	1
1	1	1	0	0
2	0	1	0	1
3	1	1	1	1
4	0	1	0	1
5	0	0	1	1
6	0	1	0	0
7	1	0	0	1

BHRs
(before)

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

PHT
(before)

0				
1				
2				
3				
4				
5				
6				
7				

BHRs
(after)

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

PHT
(after)

Part a (5 points): The PC contains x360E, and the instruction fetched is a branch. Does the branch predictor predict taken or not taken? On what information is your answer based? Please be specific.

Part b (5 points): The branch at x360E is taken. Compute the new BHRs and PHT in the figure above after this branch completes execution. It is only necessary to show the "after" entries that have changed.

PROBLEM CONTINUTED ON NEXT PAGE

Name:___________________________________

Part c (5 points): Execution of the program results in four more branches (for a total of five) being executed. They are at locations xCC48, xD028, x4842, and x6974. Each of the five branches retires before the next branch is fetched. The table below shows the prediction and direction for each of these five branches. Your job: complete the table below. Do not make any changes to the figures on the previous page.

Branch at address	**Prediction**	**Actual**
x360E	(Answer in part a)	Taken
xCC48		Taken
xD028		Taken
x4842		Not Taken
x6974		Not Taken

Name:___________________________________

Problem 4 (25 points): An out-of-order processor executes its instructions according to the Tomasulo algorithm. The ISA specifies 8 registers, R0 to R7. The microarchitecture contains one pipelined adder and one pipelined multiplier. Pipelining allows an add (or multiply) instruction to initiate execution each clock cycle.

- Fetch and Decode take one cycle each.
- ADD execution takes 3 cycles
- MUL execution takes 5 cycles.
- For ADD and MUL, if two instructions are ready to dispatch to the same functional unit in the same cycle, the older instruction is dispatched and the younger instruction waits.
- For ADD and MUL, one cycle is needed to write the result to a destination register. Only one result can be written in a single clock cycle. If two instructions want to write results in the same clock cycle, the older instruction writes, and the younger is stored in a buffer and is available for writing in the following cycle.
- Data forwarding is not implemented.

The adder and multiplier each have 3-entry reservation stations. Note: if an instruction is of the form ADD Rx, Ry, Rz or MUL Rx, Ry, Rz, and Ry contains valid data 74 and Rz contains valid data 27, the reservation station entry has the form:

V	TAG	VALUE	V	TAG	VALUE
1	—	74	1	—	27

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 table below contains a program of seven instructions that are executed.

I1	ADD			
I2				R4
I3				
I4				R3
I5				
I6			R2	
I7	ADD	R5		

PROBLEM CONTINUTED ON NEXT PAGE

Name:___________________________________

Three snapshots of the machine are shown: (a) before execution, (b) after clock cycle 5, and (c) after clock cycle 9. Note that some information in the program (shown on the previous page), in the register file, and in the reservation stations are missing.

	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

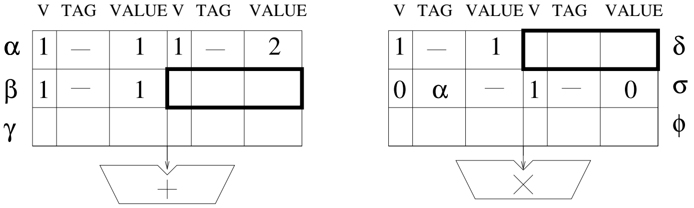
(a) Beginning

	V	TAG	VALUE
R0	1	—	0
R1	1	—	1
R2	0	δ	—
R3	0		—
R4	1	—	4
R5	0		—
R6	1	—	6
R7	0	β	—

(b) After cycle 5

	V	TAG	VALUE
R0	1	—	0
R1	0	γ	—
R2	1	—	4
R3	1	—	3
R4	1	—	4
R5	0		—
R6	0		—
R7	0	β	—

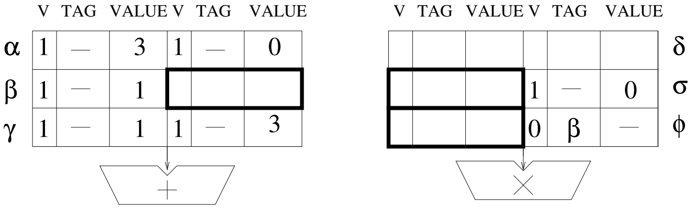
(c) After cycle 9



	V	TAG	VALUE	V	TAG	VALUE
α	1	—	1	1	—	2
β	1	—	1			
γ						

V	TAG	VALUE	V	TAG	VALUE	
1	—	1				δ
0	α	—	1	—	0	σ
						ϕ

(b) After cycle 5



	V	TAG	VALUE	V	TAG	VALUE
α	1	—	3	1	—	0
β	1	—	1			
γ	1	—	1	1	—	3

V	TAG	VALUE	V	TAG	VALUE	
						δ
			1	—	0	σ
			0	β	—	ϕ

(c) After cycle 9

Part a (20 points): Your job: Fill in the missing information in the program (shown on the previous page), and in the bolded boxes in the register file and reservation stations at each of the snapshots.

Part b (5 points): The figure below shows, for each clock cycle, which phase of the instruction cycle each of the seven instructions are in. Complete the figure. We have provided 20 clock cycles. Only use what you need.

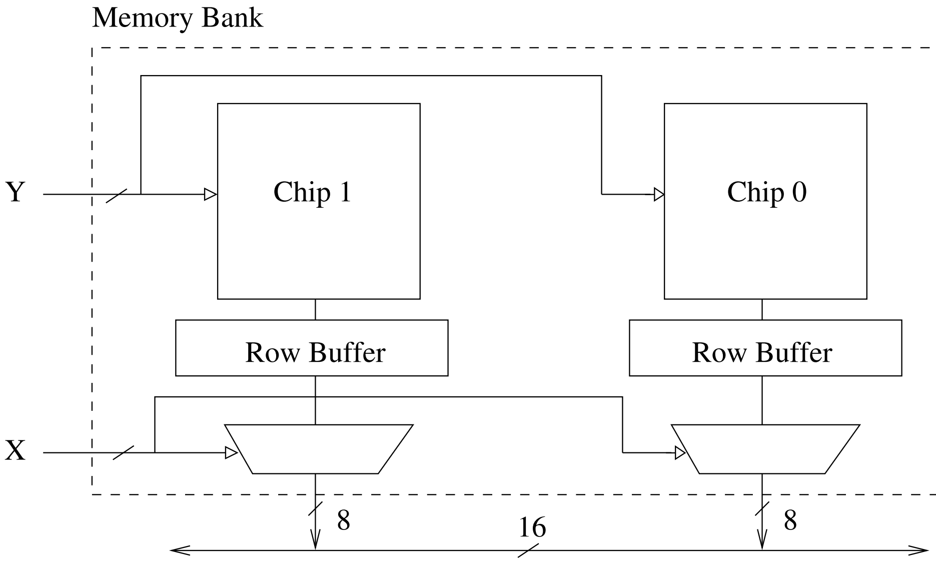
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1	F	D																		
I2		F	D																	
I3			F	D																
I4				F	D															
I5					F	D														
I6						F	D													
I7							F	D												

Name:____________________________________

Problem 5 (25 points): Suppose we have a 4-way interleaved DRAM memory, with bits[2:1] designating the bank. All address bits are as shown. Note, the question marks below; in part b, you are going to have to determine how many row bits and how many column bits.

15 11	10 ?	? 3	2 1	0
Rank	Row	Column	Bank	Byte on Bus

Recall that a single memory bank has the following form:



A row access takes 13 cycles, and a subsequent column access takes 3 cycles.

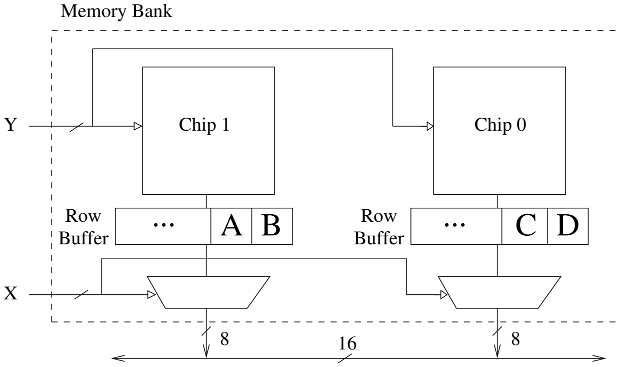
We store sequentially in this memory, starting at location x8000, an array of 64 English words, each containing 7 letters. Each word is stored in 8 consecutive locations, one location each for the corresponding ASCII code, and one location for a null terminator (x00). For example the word "Compute" would be stored as:

x43
x6F
x6D
x70
x75
x74
x65
x00

PROBLEM CONTINUTED ON NEXT PAGE

Name:__________________________________

Part a (6 points): Suppose we were to load the contents of location x8000. After the load completes, A, B, C, and D are bytes of data in the row buffer. What are the addresses of the locations containing those bytes of data?



Address containing A:

Address containing B:

Address containing C:

Address containing D: x8000

Part b (19 points): We wish to determine how many of the 64 English words end in the letter ‘e’ with a minimum number of memory accesses.

Part b1 (7 points): If we end up having to load 8 rows into the row buffer, how many bits must have been used to specify the row, and how many bits must have been used to specify the column?

Row bits: Column bits:

Part b2 (6 points): How many clock cycles are necessary to access this information from memory, assuming memory accesses are sent to DRAM in order of increasing addresses? (Note: Your answer will depend on how much you can use interleaving.)

Clock cycles:

Part b3 (6 points): Suppose we interchange the column bits and the bank bits. Now how many clock cycles are necessary to access this information from memory, assuming memory accesses are sent to DRAM in order of increasing addresses?

Clock cycles:

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

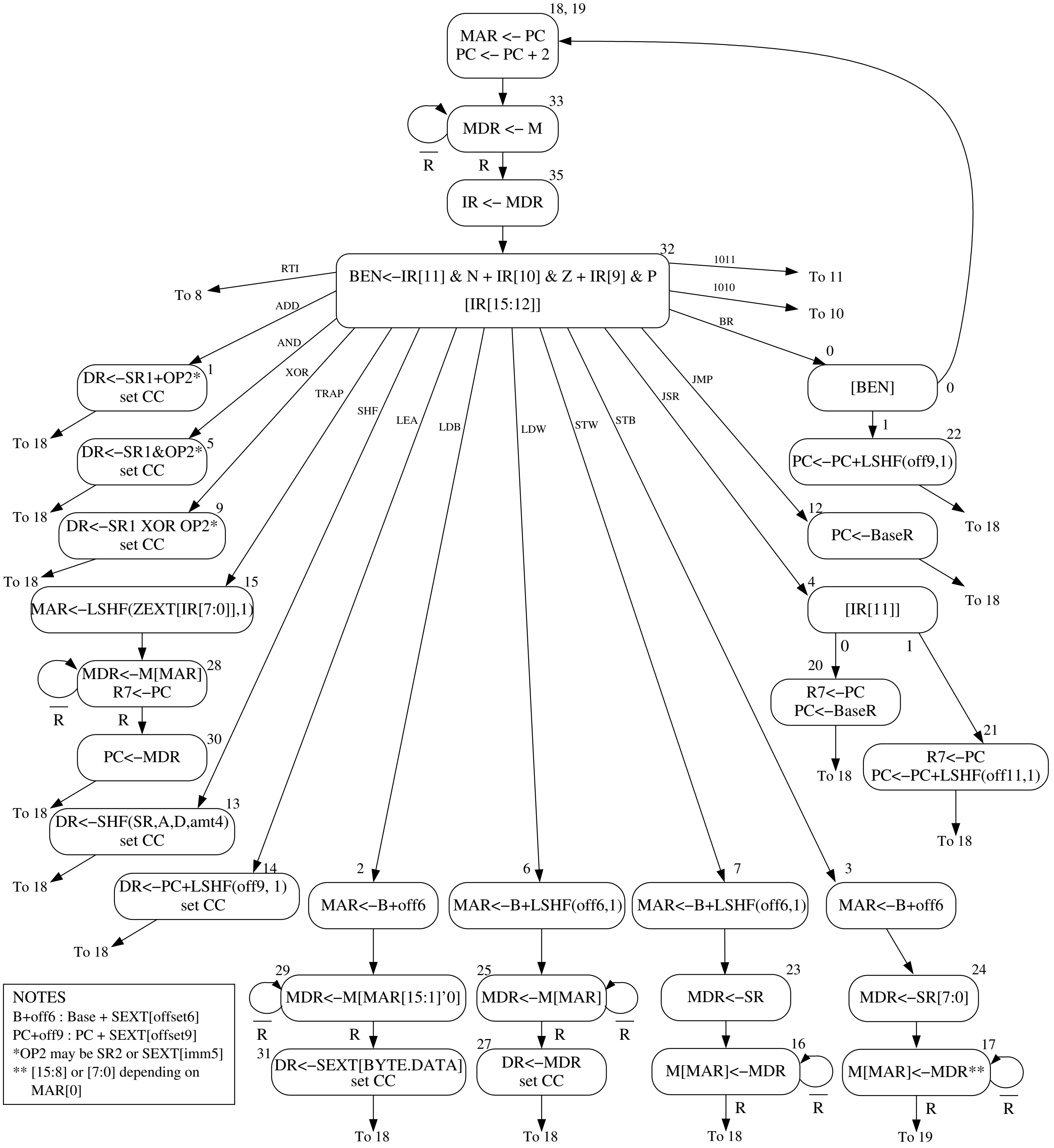
Figure 1: LC-3b Instruction Encodings

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

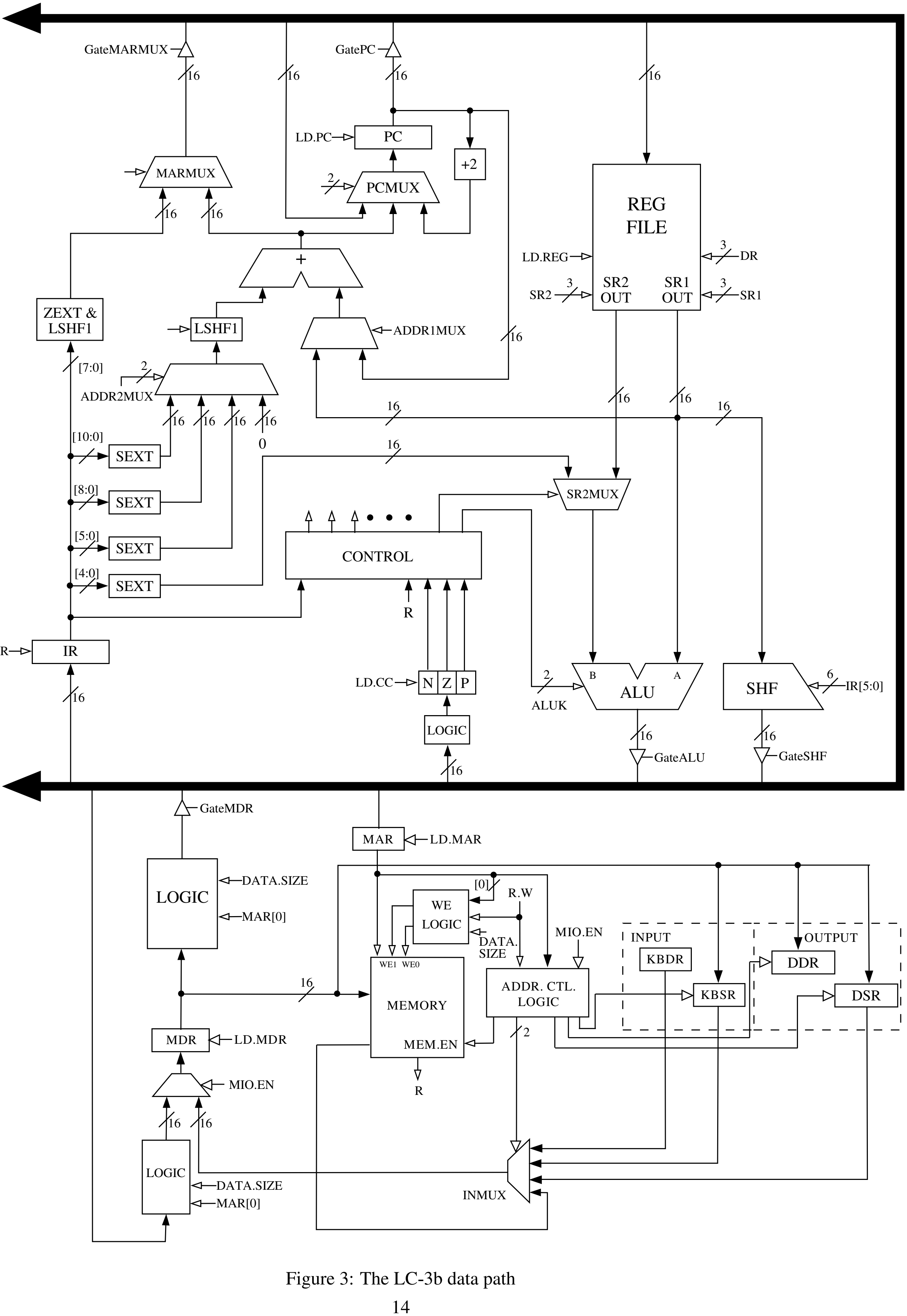


Figure 3: The LC-3b data path

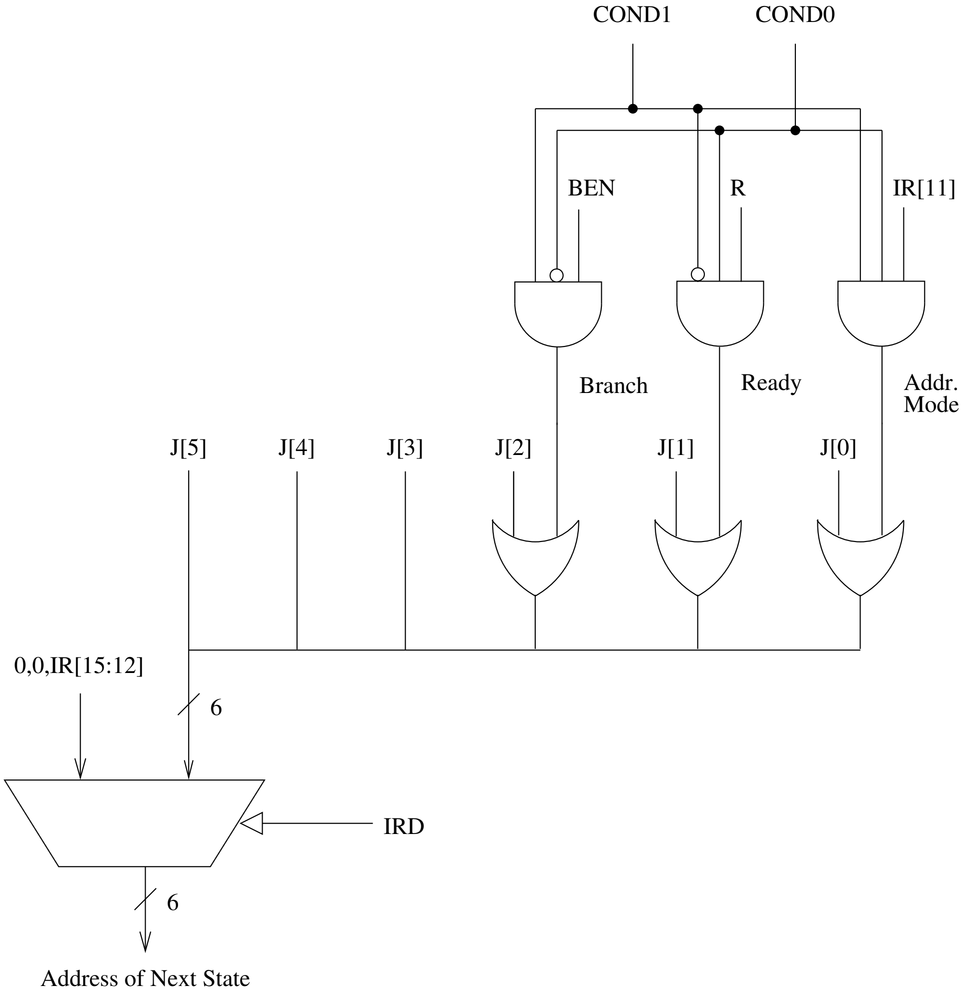


Figure 4: The microsequencer of the LC-3b base machine