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

EE 460N Spring 2013
Y. N. Patt, Instructor
Faruk Guvenilir, Sumedha Bhangale, Stephen Pruett, TAs
Exam 1
March 6, 2013

Name: Solution

Problem 1 (20 points): 20

Problem 2 (20 points): 20

Problem 3 (20 points): 20

Problem 4 (20 points): 20

Problem 5 (20 points): 20

Total (100 points): 100

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: Solution

GOOD LUCK!

Name: Solution

Problem 1 (20 points)

Part a (5 points): A microarchitecture is predicting whether a branch is taken or not taken using a single saturating 2-bit counter. The last five branches were: taken, taken, taken, taken, not taken. What does the branch predictor predict? **Circle one:** [✓ Taken]/Not Taken. Explain.

Counter Saturates to strongly taken after the 4 taken BR, remains on weakly taken after the not taken BR

Part b (5 points): The LC-3b data path has several tri-states connected to the bus: Gate_MDR, Gate_PC, Gate_ALU, to name a few. What is the maximum number of these signals that can be asserted in a single clock cycle if you are sourcing the bus in that cycle?

Answer: 1

Explain.

Only 1 BUS, Garbage will be on bus if more than 1.

Part c (5 points): If you were asked to design the HEP processor, which branch predictor would you use? Explain.

None! HEP switches threads every cycle; when it resumes a thread, the BR for that thread already resolved.

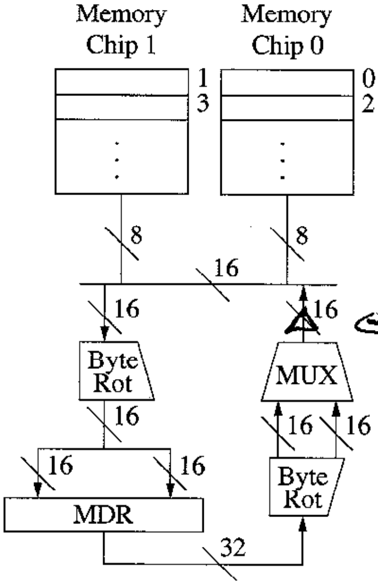
Part d (5 points): To perform a DRAM access, do you always need to assert the Row Address Strobe (RAS) so that the high bits of address are applied to the DRAM chip? **Circle one:** Yes/[✓ No]. Why or why not? Explain.

On a row buffer hit, only CAS asserted because the correct row already open in row buffer.

Name: Solution

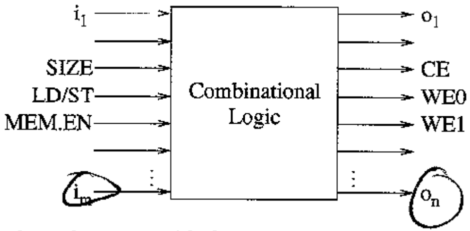
Problem 2 (20 points)

A 2 MB, byte addressable physical memory consists of two 1MB memory chips (20 bit address, 8 bits of data), connected to a 16 bit data bus. The processor is a 32-bit machine, i.e., the ALU processes 32-bit data, registers are 32 bits wide, etc. The instruction set allows byte, half-word, and 32-bit word loads via LD8, LD16, and LD32 instructions. Note that Memory Chip 0 is connected to the low 8 bits of the bus, and Memory Chip 1 is connected to the high 8 bits of the bus. You should assume anything that is not explicitly stated should be treated as discussed in class and as treated for the unaligned question on the problem set.



+1 Extra Credit for realizing GateMUX missing (tristate Buffer)

To accomplish all memory accesses, the system requires a control logic unit containing **m inputs** (i_1 through i_m) and **n outputs** (o_1 through o_n), as shown below:



Note that a few of the control signals have been provided.

Part a (5 points): Identify all n control signals. Unaligned access is allowed.

LD.MDR (4 bits, 1 for each byte)
Rotate
MUX

Name: Solution

Problem 2 continued

Part b (6 points): Identify all m inputs to the control in the table below. Note that LD/ST, SIZE, and MEM.EN have been provided. Note that some rows in the table may not be needed.

Control Signal	Purpose	Values
LD/ST	Load or Store	Load, Store
SIZE	Data Size	Byte, Halfword, Word
MEM.EN	Enable Memory	No, Yes
MAR[0]	Unaligned or not	0, 1
Access	Which access to MEM	1st / 2nd / 3rd
—		
—		

Part c (9 points): We wish to process the instruction LD32 R1, A. Recall LD32 means load 32 bits of data from A into the 32-bit register R1.

Construct those rows of the m-input, n-output truth table that are needed to guarantee that the correct 32 bits of data are loaded into the MDR as a result of LD32 R1,A. Assume a little endian ISA. Note that some rows in the table may not be needed.

SIZE	LD/ST	MEM.EN	MAR[0]	Access	CE	WE1 WE0	MUX	Rotate	LD.MDR
W	L	1	0	1st	1	0 0	X	0	0011
W	L	1	0	2nd	1	0 0	X	0	1100
W	L	1	1	1st	1	0 0	X	1	0001
W	L	1	1	2nd	1	0 0	X	1	0110
W	L	1	1	3rd	1	0 0	X	1	1000

Name: Solution

Problem 3 (20 points)

Assume a 256-byte, byte-addressable virtual memory system of the same style as the VAX discussed in class. Physical memory address space is 128 bytes, consisting of 16 page frames. We will assume virtual address space is divided equally between a single process (user) region starting at VA 0x00, and a single system region starting at VA 0x80 (i.e., two regions, not four).

Each page requires a one-byte PTE, as shown below:

4 bits PFN

7	6 ... 0
V	?

Note that the actual location of the PFN within the PTE is not specified. But, the bits of PFN are continuous within the PTE.

Like the VAX, the user space Page Table is in System Virtual Memory, and the System Page Table is in physical memory.

The table below lists the sequence of accesses to physical memory required by **n** consecutive LD_Byte instructions (excluding instruction fetch) to addresses in user virtual space. Exactly one of those instructions resulted in a TLB miss (the TLB only contains PTEs for user space). No page faults occurred during these accesses.

Note the inconsistency here. Access 3 would be system access if part of page walk, which conflict with access 8 being a user access.

	Physical Address	Data	TLB Hit?
Access 1	1111001	10101010	✓
2	1101000	10001110	✓
3	0111101	11010001	✓
4	1000001	10000001	✓
5	1011001	10001010	
6	0011101	11100101	
7	1001000	10111011	
8	0111001	10101110	✓

Part a (3 points): What is **n**? Explain.

$$\boxed{6, \left(5 \text{ hits} \cdot 1 \frac{\text{access}}{\text{hit}}\right) + \left(1 \text{ Miss} \cdot 3 \frac{\text{access}}{\text{miss}}\right) = 8 \text{ accesses}}$$

Part b (3 points): Identify which of the accesses in the table resulted in a TLB hit by putting a check mark in the corresponding rows of the column labeled TLB hit.

Part c (4 points): What is the data returned by the LD_Byte instruction that resulted in the TLB miss?

3 accesses in a page walk!

Answer: $\boxed{1011\ 1011}$

Name: Solution

Problem 3 continued

Part d (4 points): Two of the virtual addresses listed below correspond to accesses incurred in getting from the LD_Byte instruction that resulted in the TLB miss to thc actual data required. Please circle them.

3 bits offset

Region 0 – user

Region 1 – system

Virtual Address	
11000001	
00010101	
[✓ 00101000]	← user region, offset matches
01011101	
01101001	
[✓ 10111101]	← system region, offset matches

Part e (3 points): What is the UBR (User Space Page Table Base Register)?

Answer: $\boxed{xB8}$

Part f (3 points): What is the SBR (System Space Page Table Base Register)?

Answer: $\boxed{x52}$

User VPN × PTE SIZE + UBR = System VA (Part of user page table)

0101 × 1 + UBR = 10111101

UBR = 10111101 − 0101 = 10111000 = x B8

System VPN × PTE SIZE + SBR = PA (part of system page table)

0111 × 1 + SBR = 1011001

SBR = 1011001 − 0111 = 1010010 = x 52

Name: Solution

Problem 4 (20 points)

Consider a microarchitecture for an out-of-order processor, in the Tomasulo style. The data path has one adder and one multiplier. Neither is pipelined. The adder requires 4 clock cycles to execute, the multiplier requires 6 clock cycles to execute. All instructions require one cycle each for Fetch, Decode, and WriteBack. The WB bus can accommodate one result per WB cycle. Reservation station entries are allocated in program order. Instructions remain in the reservation stations until their results are written in the WB stage, and are then available for replacement by other instructions needing a reservation station slot.

We wish to execute a program fragment consisting of five instructions I1 to I5. All instructions are of the form **OPCODE DR,SR1,SR2.** Figure 1 lists the five instructions in program order. In processing these five instructions, a sequence of writes to the Reservation Stations and to the Register Alias Table occurred. Figure 2 lists that sequence in the order they occurred. Note that writes to the Reservation Stations occur at the end of Decode/Rename and writes to the Register Alias Table occur during Write Back. (Other writes occur to the Register Alias Table, but we will not concern ourselves with them in this problem).

	Opcode	DR	SR1	SR2
I1	ADD	R5	R6	R7
I2	MUL	R2	R2	R5
I3	MUL	R2	R2	R5
I4	MUL	R2	R4	R6
I5	MUL	R5	R5	R7

Figure 1: Program Fragment

Access	Data Traffic	
R/S I1	1 — 7	1 — 13
R/S I2	1 — 2	0 α —
R/S I3	0 π —	0 α —
R/S I4	1 — 3	1 — 7
RAT I1	α 20	
RAT I4	σ 21	
R/S I5	1 — 20	1 — 13
RAT I2	π 40	
RAT I3	ρ 800	
RAT I5	σ 260	

←2×20=40

Note that this happens late, MUL R/S fills up.

Figure 2: Program Trace

Your job is to complete both tables. Note that one source and one destination in Figure 1 are already filled in, and many of the entries of Figure 2 are partially filled in.

Entries in Figure 2 are of the form "R/S source1 source2," or "RAT tag value." For example,

R/S 1 - 7 1 - 13

means a Reservation Station entry was written, and valid data values 7 and 13 obtained from the two registers identified in the corresponding instruction.

RAT α 20

means the value 20 was written to the Register Alias Table entry having the tag α.

PROBLEM IS CONTINUED ON THE NEXT PAGE!

See dataflow graph on next page.

Name: Solution

Problem 4 continued

To help you complete the tables, we have included a mapping of the reservation stations, and the contents of registers R0 through R7 before and after the program fragment executes. Note that there are three reservation station entries for the multiplier and three for the adder. You are not required to fill in the reservation stations. We include this figure only to provide you with additional information you might find useful in solving the problem.

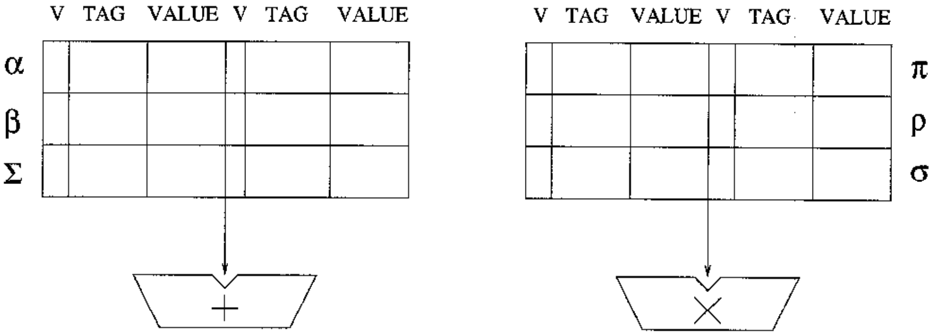
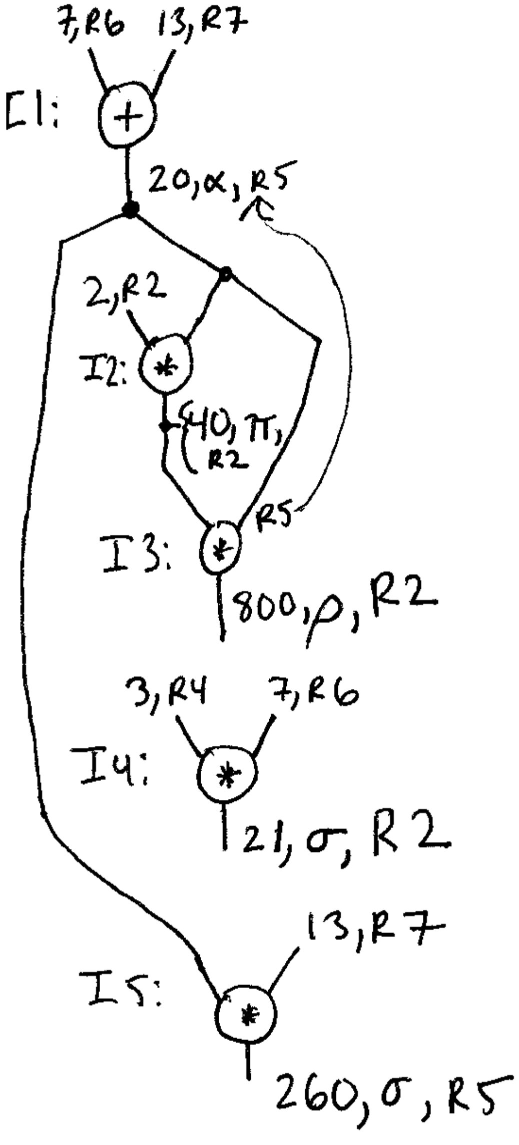


Figure 3: Reservation Stations



	BEFORE	AFTER
R0	4	4
R1	1	1
R2	2	21
R3	6	6
R4	3	3
R5	5	260
R6	7	7
R7	13	13

Figure 4: Contents of Registers

*I4 ready immediately, must be 3·7=21

*Only R2 and R5 change.

*Only way to get 21: 7x3

*Only way to get 260: 20x13

* Need 20 from R5 in I4 or I5, so I3 must write to R2 (800 unused)

Name: Solution.

Problem 5 (20 points):

We wish to add a new instruction BLOCKADD to the LC-3b ISA. BLOCKADD sums the integers contained in a block of contiguous memory locations, writes the results to a destination register and sets the condition codes based on the sum. Its format is as follows:

	15 12	11 9	8 7 6	5	0
BLOCKADD	1 0 1 0	DR	BaseR	0 0 0	SR2

where 1010 is the opcode, BaseR contains the starting address of the block of memory, and SR2 contains the number of 16-bit integers in the block.
Modifications to the data path required by BLOCKADD are shown below in boldface.

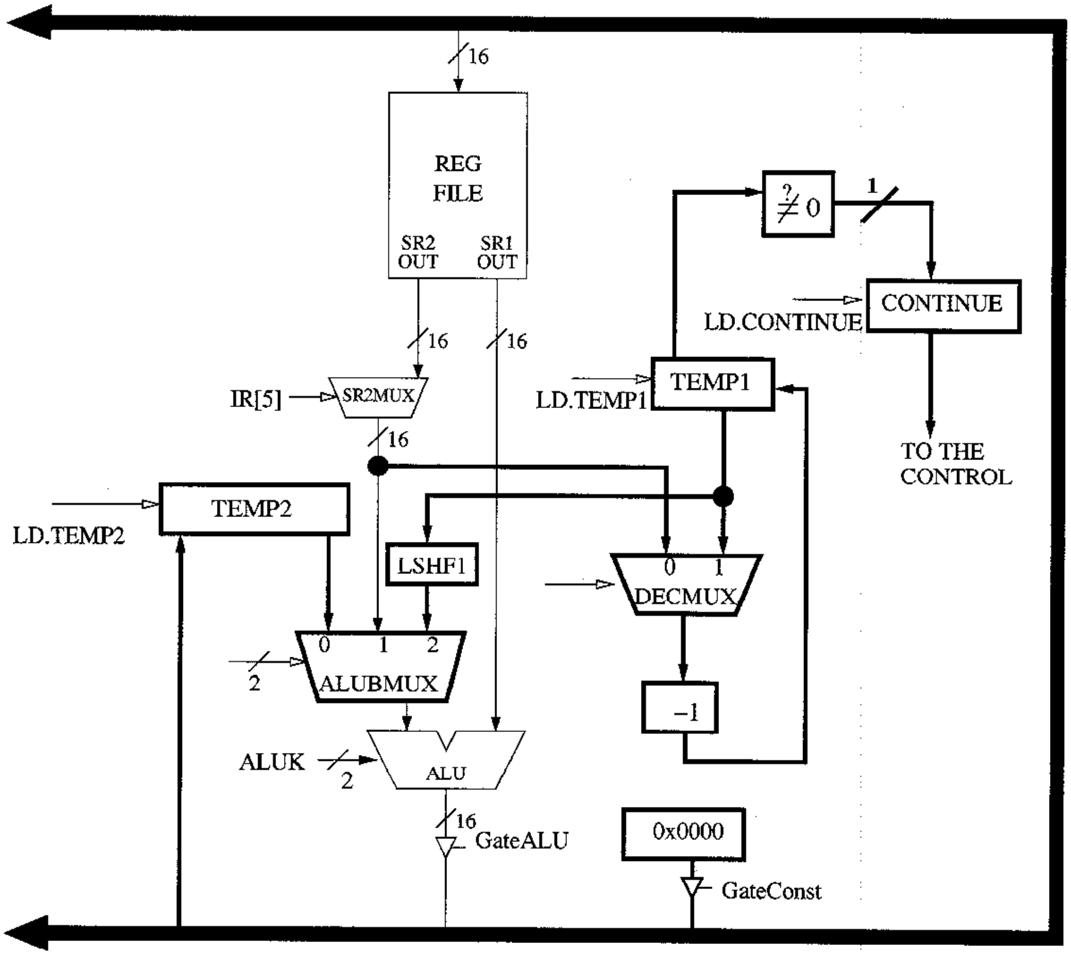


Figure 5: Modified datapath to support BLOCKADD instruction

Note: Assume SR2 will always be ≥ 1.

Name: Solution

Problem 5 continued:

Three things must be done to complete the task of implementing BLOCKADD.

Part a (9 points): Five new states are required in the state machine. Fill in the missing information in the state machine below. Don't forget to give each state a state number (aka control store address).

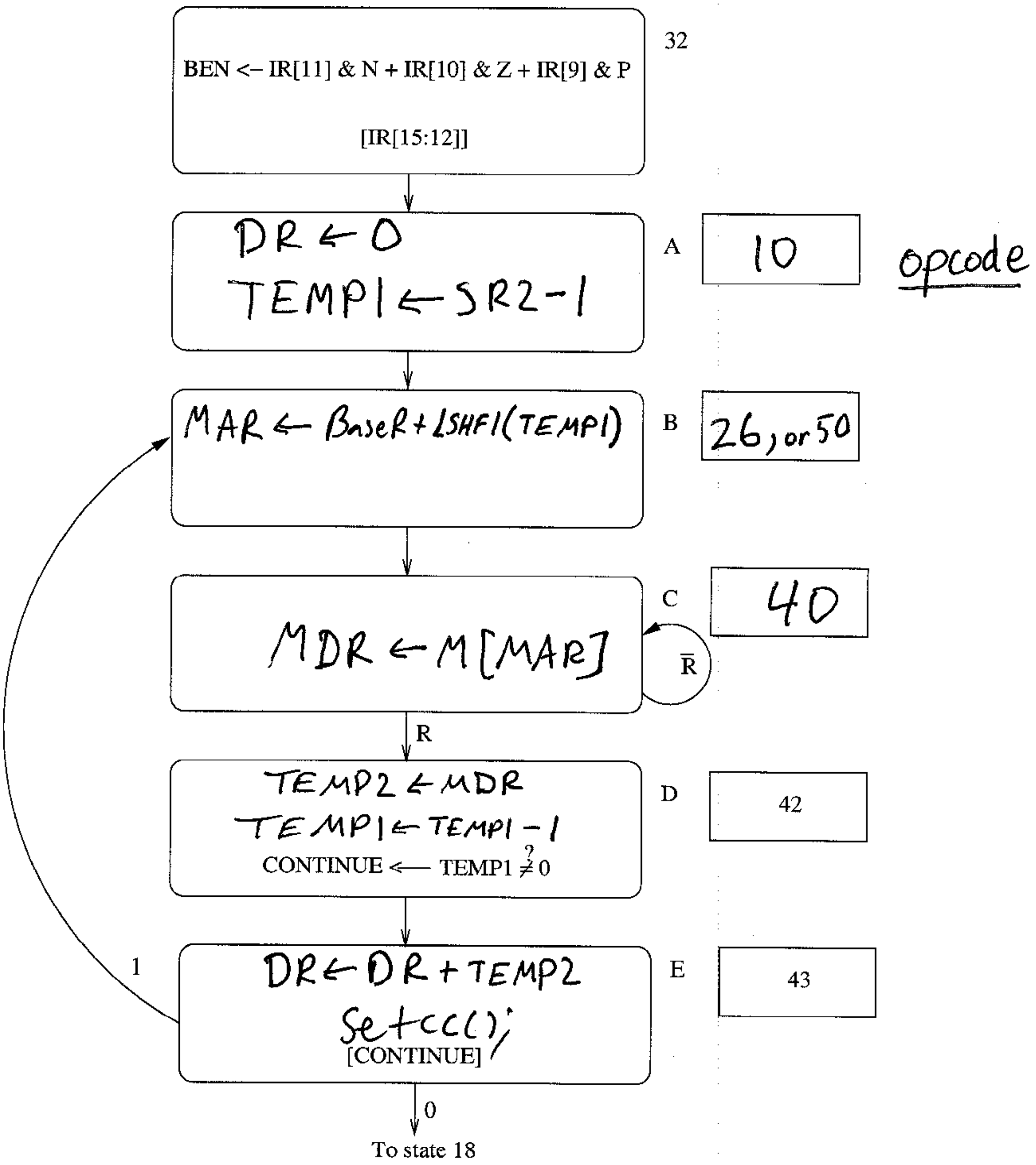


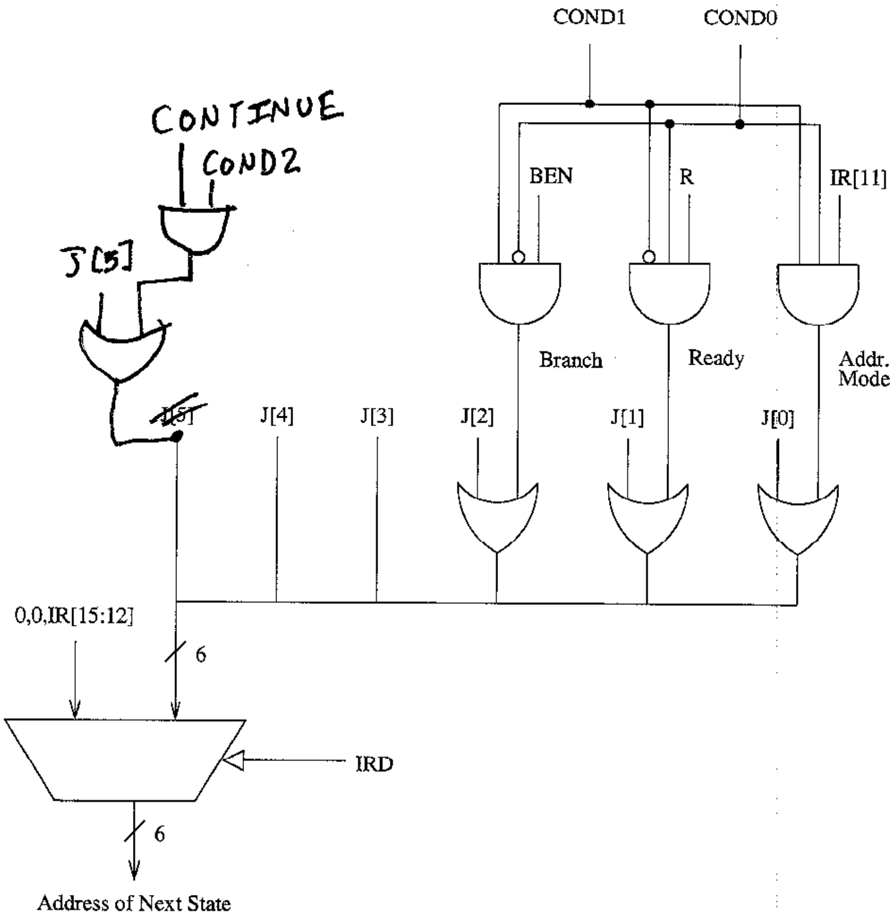
Figure 6: State diagram for BLOCKADD instruction

[TEMP1 ← TEMP1 − 1] Can be in state D or E.

Name: Solution

Problem 5 continued:

Part b (6 points): Augment the Microsequencer as necessary, without using any muxes. Hint: cond (see part c below) is a 3 bit field. Assume 50 chosen for State B.



Part c (5 points): Fill out the control signals that constitute the microinstructions for the five new states in the state machine. List any signals that are don't cares for a particular state as 0.

	COND	J	LD.MAR	LD.MDR	LD.REG	LD.CC	GateMDR	GateALU	DRMUX	SR1MUX	ALUK	MIO.EN	R.W	DATA.SIZE	LD.TEMP1	LD.TEMP2	LD.CONTINUE	GateConst	ALUBMUX	DECMUX
state A	000	110010	0	0	1	0	0	0	0	0	00	0	0	0	1	0	0	1	00	0
state B	000	101000	1	0	0	0	0	1	0	1	00	0	0	0	0	0	0	0	10	0
state C	001	101000	0	1	0	0	0	0	0	0	00	1	0	1	0	0	0	0	00	0
state D	000	101011	0	0	0	0	1	0	0	0	00	0	0	1	1	1	1	0	00	1
state E	100	010010	0	0	1	1	0	1	0	0	00	0	0	0	0	0	0	0	00	0

	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 7: 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	

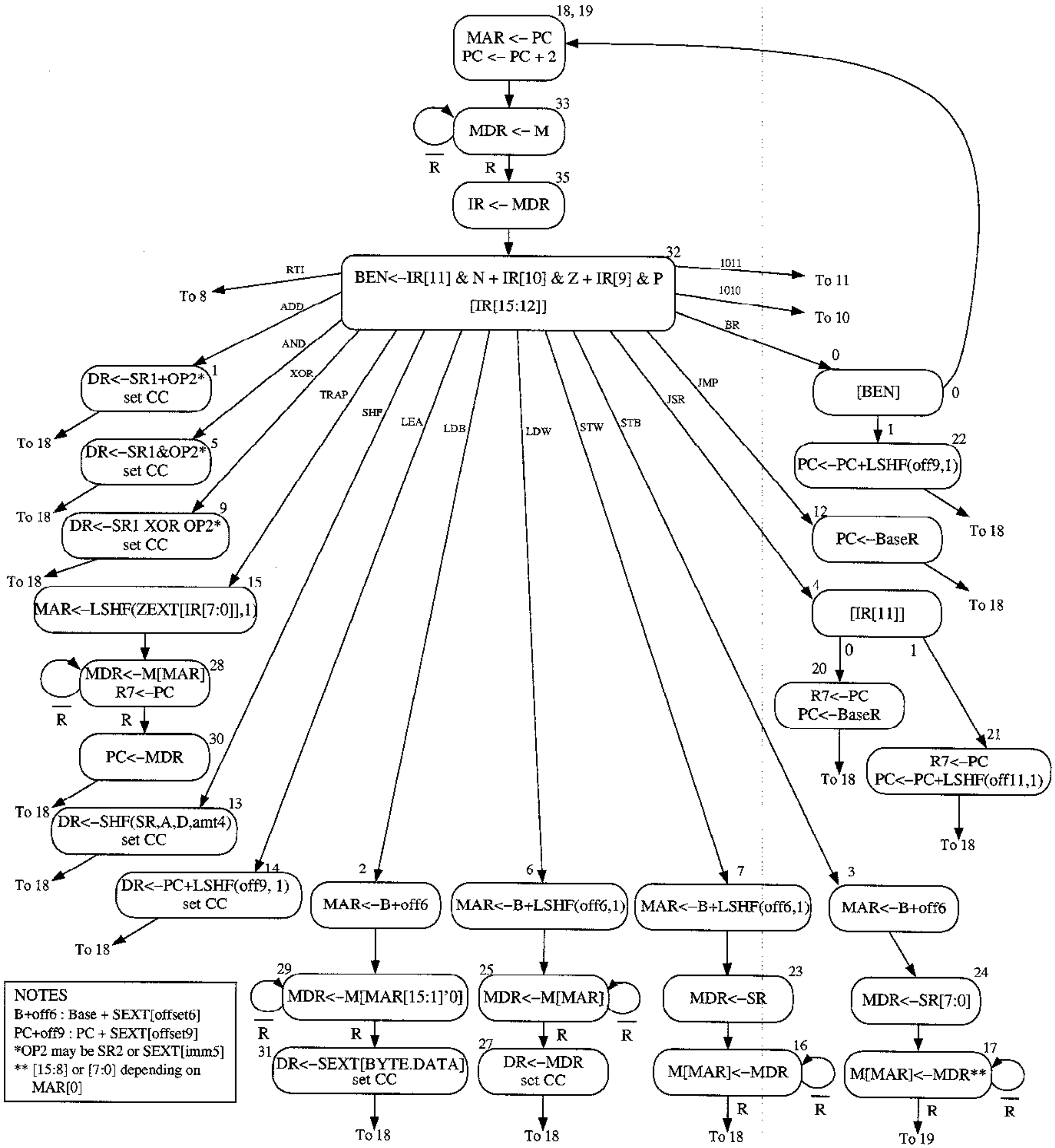


Figure 8: A state machine for the LC-3b

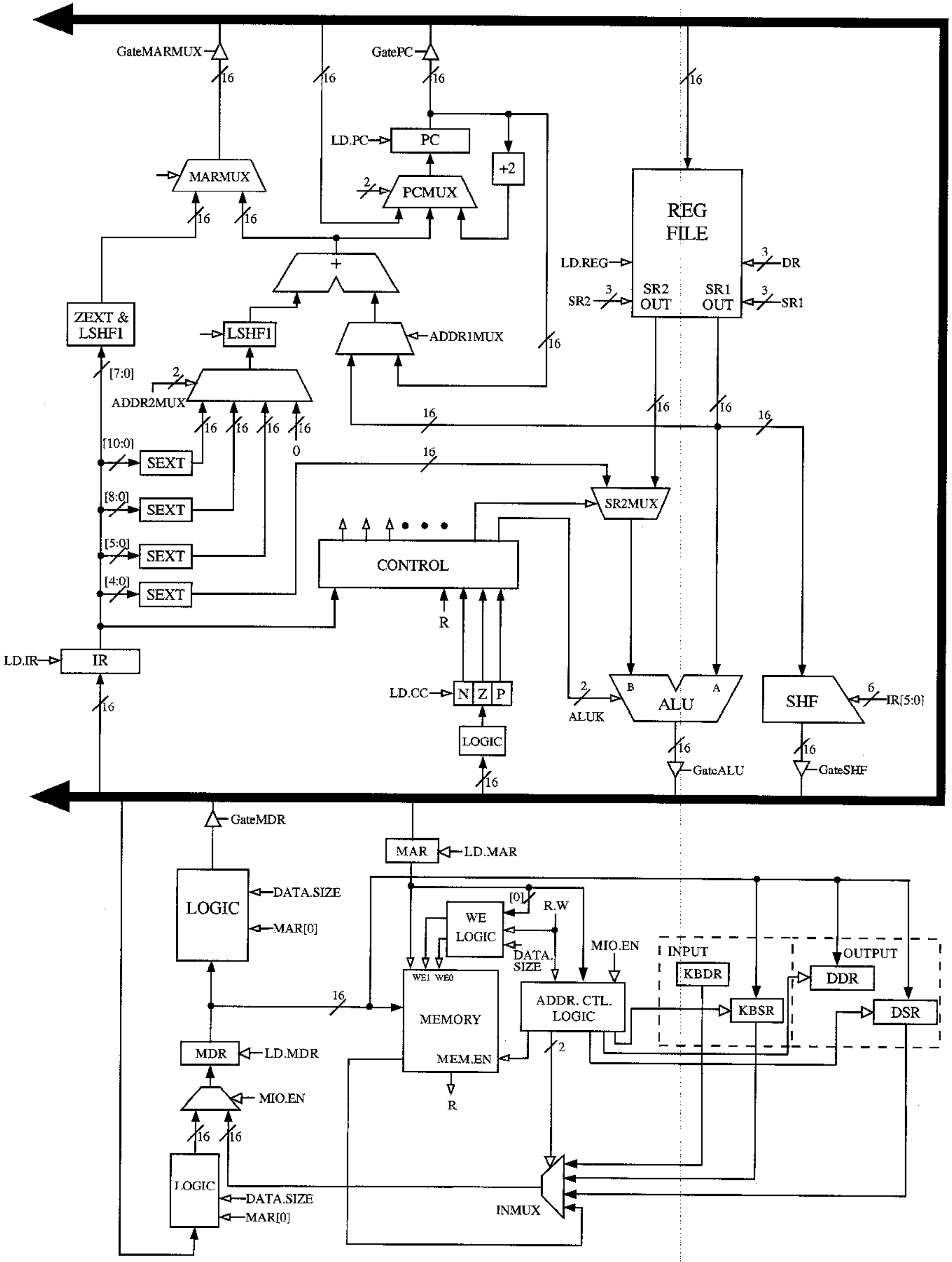


Figure 9: The LC-3b data path

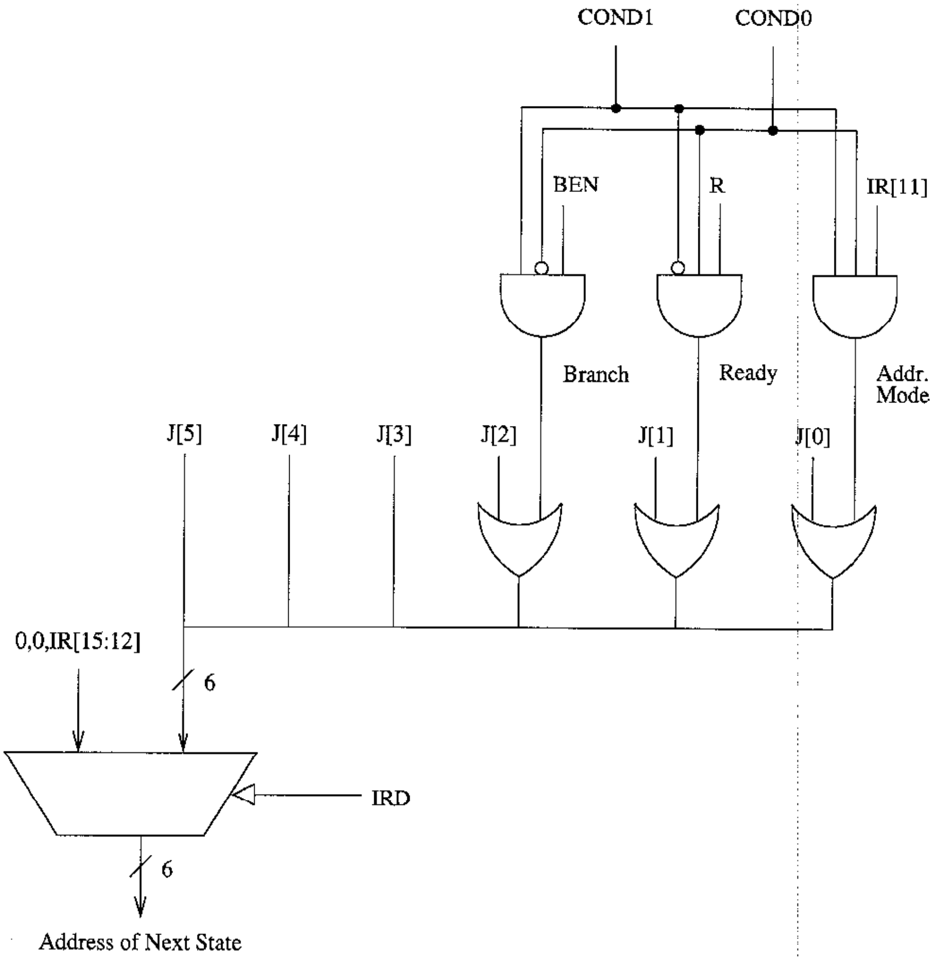


Figure 10: The microsequencer of the LC-3b base machine