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 2
April 17, 2019

Name: Aniket Deshmukh

Problem 1 (20 points):______

Problem 2 (20 points):______

Problem 3 (30 points):______

Problem 4 (30 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: [illegible]

GOOD LUCK!

Name:______________________________

Problem 1 (20 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): In an asynchronous system, a device controller in the Idle state receives the Bus Grant (BG) signal. If its device does not want the bus, it passes the BG signal to the next device at the same priority level. The controller can stop asserting the BG signal and return to Idle (a) when it receives the SACK signal or (b) when it stops receiving the BG signal. Using one of these is problematic. Which one and why?

(a) When it receives SACK
Assume device A, B, C on the same priority line. C requests bus. A, B forward bus grant. C get BG, wait on BBSY. B gets SACK from C, ~~waits~~ goes to Idle. Since A is far away, it still forwards BG and B may now get the bus!

Part b (5 points): The VAX ALU performs addition on 32 bit, 2's complement integers. It can perform BCD arithmetic on 8 BCD digits with the assistance of a register that is hard-wired to x66666666. What is the purpose of the value x66666666. Please be specific.

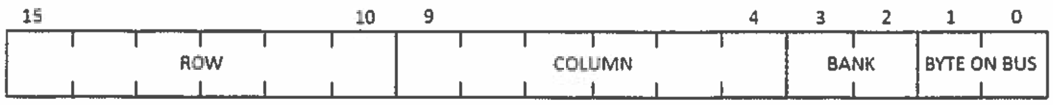
Helps propogate carry, since binary addition on BCD digits does not produce a carry if the sum is < 16.

Part c (5 points): A microarchitecture executes instructions in program order with a ten-stage pipeline and obeys the requirement for precise exceptions. Assume we have ten instructions in various stages of execution. Instruction 1 is in the last stage of the pipeline, about to complete execution. Instruction 10 has just entered the pipeline. Instruction 4, in the 7th pipeline stage, takes a page fault. What should the microarchitecture do before turning control over to the exception service routine to process the page fault.

Allow instructions # 1-3 to complete. Flush instructions 5-10. Now, take PC, PSR as seen by instruction 4 and push it onto the stack. Take the exception!

PROBLEM CONTINUES ON NEXT PAGE

Part d (5 points): You are given a 16-bit physical memory with the address bits broken down as follows:



The elements of an array containing 32-bit integers are accessed. Program A accesses consecutive elements in the array. Program B accesses elements at even indices. Memory takes more than 3 cycles to access. Given both programs access the same number of elements, which one takes longer to complete the array accesses? Explain.

Program A : Sequential acces hit in bank 0,1, 2,3,0,1,2,3...
Program B : Accesses at add indices hit in bank 0,2,0,2,0,2...
B has more bank conflicts ∴ takes longer.

eg:

	A		B
0	x 3000	0	x 4000
1	x 3004	2	x 4008
2	x 3008	4	x 4010
3	x 300B	6	x 4018

Name:________________________________

Problem 2 (20 points):
Part a (10 points): A 64-byte physical memory has a direct mapped cache consisting of 4 cache lines, where the line size is L bytes. The cache is initially empty.

Six byte-accesses were made to the cache, as shown in the table below on the left. After the six accesses have completed, the state of the Tag Store is as shown below on the right.

Your job: Determine L, the line size of the cache. Also, fill in the missing address bits of the six accesses.

Cache line size: $\boxed{2B}$

6bits = 3bits + 2bits + L
(TAG) (IDX)

The six accesses

Access	Hit/Miss	Address					
1	M	0	0	0	0	1	0
2	M	0	1	0	0	0	0
3	H	0	0	0	0	1	1
4	M	1	1	1	0	0	1
5	M	1	0	1	0	0	0
6	M	1	1	0	1	1	1

in cache

State of Tag Store after the six accesses

Set	Valid	Tag bits		
0	1	1	0	1
1	1	0	0	0
2	~~0~~	~~0~~	~~1~~	0
3	1	1	1	0

IGNORE!

PROBLEM CONTINUES ON NEXT PAGE

Name:________________________

Part b (10 points): Replace the direct-mapped cache with a 2-way set associative cache with LRU replacement. Physical address space has not changed. We still have four cache lines, but the line size may have changed. The cache is initially empty.

Again, six byte-accesses were made to the cache, as shown in the table below. After the six accesses have completed, the state of the Tag Store is as shown in the next table.

Your job: Determine L, the line size of the cache. Also, fill in the missing address bits of the six accesses.

Cache line size: $\boxed{1B}$

6bits = 5bits + 1bit + L
(TAG) (IDX)

The six accesses

Access	Hit/Miss	Address					
1	M	1	1	1	0	1	0
2	H	1	1	1	0	1	0
3	M	0	1	0	1	0	0
4	M	0	0	0	1	1	0
5	M	1	0	1	0	0	1
6	H	1	0	1	0	0	1

→ evicted

in cache

State of Tag Store after the six accesses

Set	Valid	Tag bits					Valid	Tag bits				
0	1	0	0	0	1	1	1	0	1	0	1	0
1	1	1	0	1	0	0	~~0~~	~~1~~	~~1~~	~~1~~	~~0~~	~~1~~

IGNORE

Name:________________________________

Problem 3 (30 points): There are times when more than one process wants to be able to read a single page of data, but occasionally wants to write to that page also. For example, suppose process X has page A in physical frame M. Suppose process Y has page B that happens to contain the same data as page A. The O/S could allocate a second frame of memory for process Y's page B. However, since the contents of process X's page A and process Y's page B contain the same information, the O/S can save a frame of memory if it simply has the corresponding PTEs point to the same frame, and designates in their respective PTEs that these pages are sharing the same frame (marking them both as Read-Only), even though the two processes have nothing to do with each other. If later, one of the processes wants to write to that page, an exception will occur, and the O/S can at that time allocate a second frame, copying the contents of the frame that both processes were reading. In the vernacular, this is referred to as a "copy-on-write." That is, until the data on the two pages are different (due to writing), there is no reason to copy the page to a second frame. This approach is often called lazy copying since the O/S does not copy until absolutely necessary.

The O/S action of designating PTEs to point to the same frame is **copy-on-write setup**. The actual copying of the contents of the shared frame to a second frame is **performing the copy-on-write**. In this problem, we will only deal with **copy-on-write setup.**

Suppose we augment the LC-3b with VAX-like virtual memory. "VAX-like" so we don't have to deal with such large numbers. For example, we will use 16 bits for virtual addresses, 14 bits for physical addresses, and page size of 256 bytes. Process virtual space: x0000 to x7FFF, System space: x8000 to FFFF.

A PTE is 2 bytes with the following format:

15	14 – 10	9 – 8	7 – 6	5 – 0
V	00000	PROT	00	PFN

The PROT bits specify:

No Access:	00
Read-Only:	01
Read & Write:	10

Assume that SBR = 0x0100, Process X's Process Space Page Table Base Register (PBR) = 0x9500,
Process Y's PBR = 0x9700.
System Length Register (LR) = 0x18, Process X's Length Register (LR) = 0x3A, Process Y's LR = 0x18.

Part a (4 points): How many bits should SBR be? How many bits should Process X's PBR be?

SBR : 14 bit (in physical memory) Bonus: 6 bits [Page tables in VAX are page aligned]
PBR : 16 bits (in virtual memory) 8 bits

PROBLEM CONTINUES ON NEXT PAGE

Name:__________________________________

Parts b), c) and d) on the next page assume the contents of memory are as shown below.

Physical Address	Contents	Physical Address	Contents	Physical Address	Contents
....	....	....	....	....	....
0x0110	0x8235	0x2160	0x8218	0x2920	0x8224
0x0112	0x801B	0x2162	0x0876	0x2922	0x0828
0x0114	0x0A2C	0x2164	0x094F	0x2924	0x8215
0x0116	0x8007	0x2166	0x0835	0x2926	0x7856
0x0118	0x8001	0x2168	0x8238	0x2928	0x8245
0x011A	0x8000	0x216A	0x8224	0x292A	0x8038
0x011C	0x021C	0x216C	0x8012	0x292C	0x810F
0x011E	0x8212	0x216E	0x0323	0x292E	0x002F
0x0120	0x8003	0x2170	0x810F	0x2930	0x020F
0x0122	0x0211	0x2172	0x0023	0x2932	0x4493
0x0124	0x8120	0x2174	0x010F	0x2934	0x23AB
0x0126	0x8109	0x2176	0xFEC7	0x2936	0xEF7C
0x0128	0x8127	0x2178	0x800E	0x2938	0xEE23
0x012A	0x8121	0x217A	0x5678	0x293A	0x800F
0x012C	0x0228	0x217C	0x17BD	0x293C	0x000F
0x012E	0x8129	0x217E	0x821E	0x293E	0x881E
....	....	....	....	....	....

PROBLEM CONTINUES ON NEXT PAGE

Name:______________________________

Part b (6 points): The O/S wishes to setup page 0x12 in process Y's process address space as a copy of page 0x34 in process X's process address space. **What is the PFN for page 0x34 in process X's address space?**

VA_1 = x9500 + x34x2 = x9568 (VPN = x15)

PA_1 = x0100 + x15x2 = x012A: x8121

PA_2 = x21 & x68 = x2168 : x8238

$\boxed{x38}$

Part c (12 points): Now that the O/S has found where in physical memory the source page (page 0x34) is located, it wishes to **setup the copy-on-write**. Your job: Set up the PTE for page 0x12, the destination page, in process Y's address space.

Where is this PTE located in Process Y's virtual address space?

VA_1 = x9700 + x12x2

$\boxed{x9724}$

What is the physical address of this PTE?

PA_1 = x100 + 17x2 = x012E: x8129

$\boxed{x2924}$

Specify the bits in the PTE:

PA_2 = x29 & x24

PTE:

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

valid   Read-only   x38 (PFN)

Part d (8 points): At the time the snapshot of memory on the previous page is taken, frame 0x0F in physical memory is another frame whose permissions are setup to be read-only for process X and Y via copy-on-write. Your job: Locate the pages in process X and process Y that are mapped to frame 0x0F.

What is the VPN of the page mapped to frame 0x0F in process X's address space?

x2168 ← for x34
x216A ← for x35
⋮
x2170 ←

$\boxed{x38}$

What is the VPN of the page mapped to frame 0x0F in process Y's address space?

x2924 ← for x12
x2926 ← for x13
⋮
x292C ←

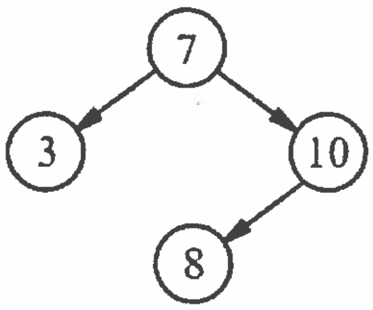
$\boxed{x16}$

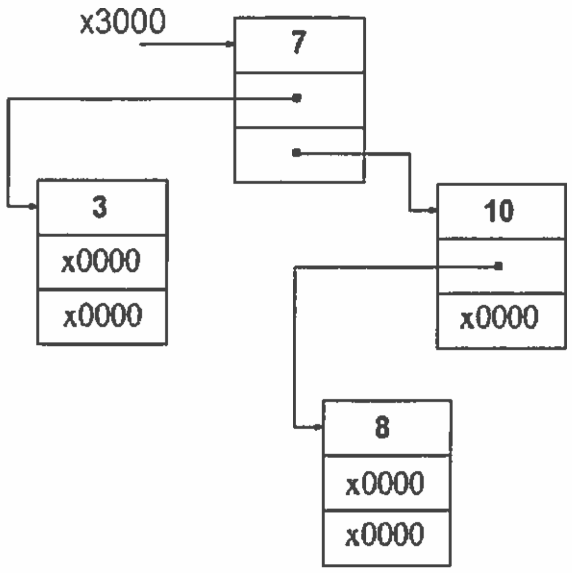
Name:____________________________________

Problem 4 (30 points): A Binary Search Tree is a useful data structure for searching quickly for a value in a sorted list. The Binary Search Tree has the property that for every node in the tree, if we consider that node as the root of two subtrees, all nodes in its left subtree have values smaller than the value in the root, and all nodes in its right subtree have values greater than the value in root, as shown in the example below. The root has value 7, there is one node in the left subtree, its value is 3. There are two nodes in the right subtree, their values are 8 and 10. If we consider the node with value 10 as a root, it has one node in its left subtree whose value is 8. There are no nodes in the right subtree of 10.

Searching for a value is a simple matter of querying a root, and then moving on to the root of its left subtree or the root of its right subtree, depending on whether the value you are searching for is smaller or greater than the root.

We represent each node in the Binary tree with three words of memory: its value, the pointer to the root of the left subtree, and the pointer to the root of the right subtree. If there is no subtree, the pointer is x0000 (the null pointer). The actual implementation in memory for the tree whose root has the value 7 is also shown below.



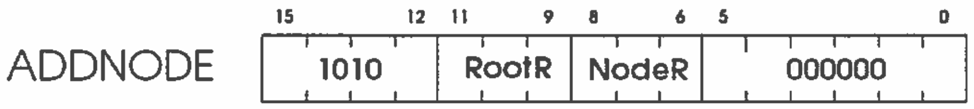


PROBLEM CONTINUES ON NEXT PAGE

Name:___________________________________

It is useful to be able to add a value to the sorted list, which means adding a node to the Binary Search Tree, while retaining the property that we can search for a value in the manner discussed above. The mechanism is to search for the value until we find a null pointer and insert the node by replacing the null pointer with the address of the node being inserted. In the example on the previous page if we wanted to insert the value 9, we would traverse the tree until we got to the node whose value is 8. At this point, we would replace the null pointer of the right subtree of 8 with the address of the new node whose value is 9.

Your job: Implement a new instruction ADDNODE to the LC-3b, which will insert a value as described above into a binary search tree. We will use unused opcode 1010. The format of the instruction is shown below:



RootR contains the address of the root node of the Binary Search Tree. NodeR contains the address of the first word of the node to be inserted. We can assume that the node to be added consists of three words: a value, and two null pointers. We also assume that if RootR is null, we do not want to insert this node anywhere so we are done. The procedure for adding a new node into our Binary Search Tree is as follows:

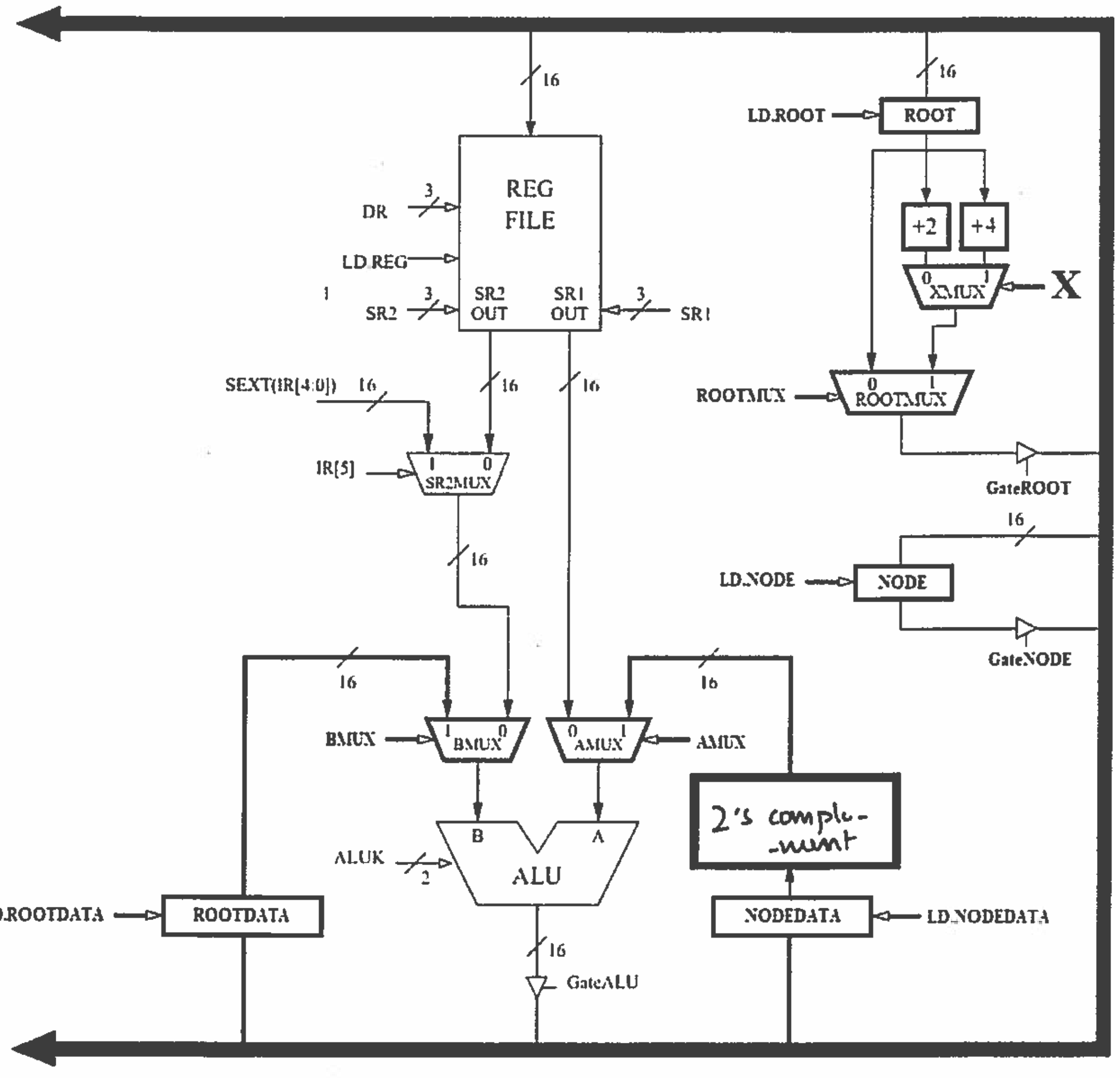
```
while(Root != 0) {     // exit if Root is NULL
    if(Node.Data == Root.Data) return;     // node exists, do not add duplicate
    if(Node.Data < Root.Data) {
        if(Root.Left == 0) { Root.Left = Node; return; }     // insert node here
        else Root = Root.Left;     // so you can traverse left subtree
    }
    if(Node.Data > Root.Data) {
        if(Root.Right == 0) { Root.Right = Node; return; }     // insert node here
        else Root = Root.Right;     // so you can traverse right subtree
    }
}
```

PROBLEM CONTINUES ON NEXT PAGE

Name:______________________

Part a (7 points): To implement ADDNODE, we need to make changes to the data path, the state machine, and the microsequencer. We show below the changes necessary to the data path. They are: four registers (NODE, NODE-DATA, ROOT, ROOTDATA) with their respective Load Enable and Gate signals, four muxes and their control signals (AMUX, BMUX, ROOTMUX, X), and an unspecified logic block. NODE and ROOT contain pointers to nodes.

Your job: Fill in the missing logic block.



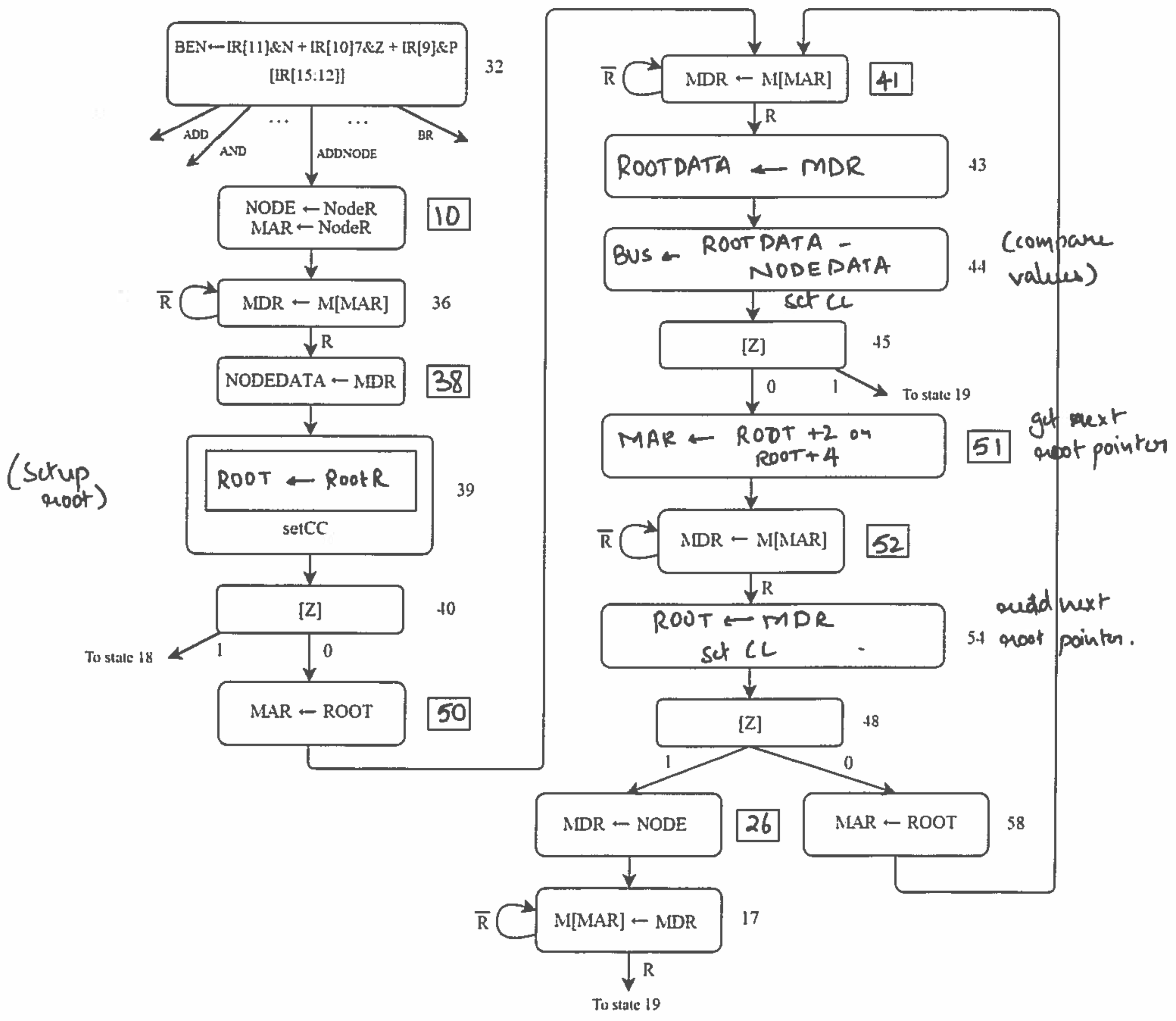
PROBLEM CONTINUES ON NEXT PAGE

Name:________________________________

Part b (16 points): With respect to the state machine, we need to add additional states after state 32.

Your job: Complete the state machine for ADDNODE shown below. Note: States 26, 34, and 36-63 in the control store are available.
Also, identify the control signals needed to perform the actions specified in state 58 in the box below.

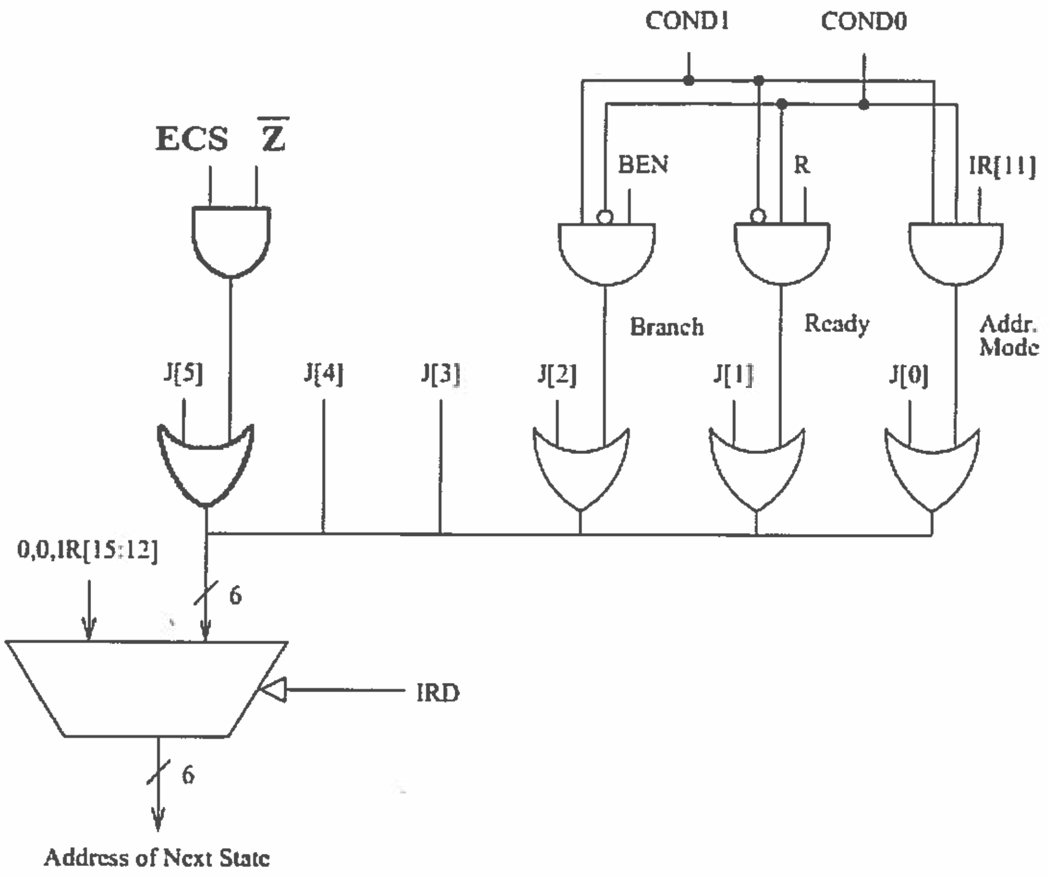
ROOTMUX = 0 , GATE ROOT = 1 , LD.MAR = 1 . Rest 0.



PROBLEM CONTINUES ON NEXT PAGE

Name:______________________________

Part c (7 points): To handle the new micro-branches, we need to augment the microsequencer with an additional control signal ECS (Extra Control Signal).



ECS is asserted in a number of states in the state machine. Use as many entries as you need to indicate these states.

40	45	52		

The mux signal **X** in the datapath can be replaced with an existing signal from the LC3-b datapath. Which signal?

[M]