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

EE 306, Fall 2019
Yale Patt, Instructor
TAs: Sabee Grewal, Arjun Ramesh, Joseph Ryan, Chirag Sakhuja, Meiling Tang, Grace Zhuang
Final Exam, December 13, 2019

Name: Solution

Part A:

Problem 1 (10 points):_____

Problem 2 (10 points):_____

Problem 3 (10 points):_____

Problem 4 (10 points):_____

Problem 5 (10 points):_____ **Part A (50 points):**

Part B:

Problem 6 (25 points):_____

Problem 7 (20 points):_____

Problem 8 (20 points):_____

Problem 9 (20 points):_____ **Parb B (85 points):**

Total (135 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.

I will not cheat on this exam.

_____
Signature

GOOD LUCK!
(HAVE A GREAT SEMESTER BREAK)

Name:_____

Problem 1. (10 points):

Part a. (2 points): How many of the 15 LC-3 instructions assert the LD.MDR control signal during its instruction cycle? Explain in 10 words or fewer.

15, because LD.MDR is asserted during fetch

Part b. (2 points): An Aggie is trying to assemble a program but the program won't assemble. The assemble-time error reads "`LD R0, A. error: cannot encode as 9-bit 2's complement integer`". What's the issue? Explain in 20 words or fewer.

The label A is too far from the instruction

Part c. (2 points): At the end of the clock cycle in which state 18 is executed, why does the MAR contain the value of the PC at the start of the clock cycle, and not the value PC+1 since 1 is added to the PC during the clock cycle? Explain in 25 words or fewer. Be specific.

Flip flops are edge triggered, so PC and MDR change simultaneously

Part d. (4 points): A student decides to design the RLC-1 (Really Little Computer 1). RLC-1 instructions will be 8 bits wide and use 3 bits for the opcode. Additionally, the RLC-1 will have 4 general purpose registers.

How many instructions ~~are~~ can be in the RLC-1 ISA?

$2^3 = 8$

The student wants the RLC-1 ISA to include an instruction that does the following: `DR <-- SR1 + SR2`. Can the RLC-1 have such an instruction? Explain in 20 words or fewer.

No, because each register takes 2 bits to encode, so anything of this format would require 9 bits

Name:_____

Problem 2. (10 points):

Consider the following program written in LC-3 assembly language:

```
                .ORIG x3000
                AND R5, R5, #0
                LEA R0, ARRAY
                LD  R1, N
                LDR R2, R0, #0
                NOT R2, R2
                ADD R2, R2, #1

LOOP            LDR R3, R0, #0
                ADD R3, R3, R2
                BRnp DONE
                ADD R0, R0, #1
                ADD R1, R1, #-1
                BRp LOOP

                ADD R5, R5, #1
DONE            ST  R5, OUTPUT
                HALT

ARRAY           .BLKW #20
N               .FILL #20
OUTPUT          .BLKW #1
                .END
```

What must be the case for 1 to be stored in OUTPUT? Answer in 15 words or fewer.

When all the elements in ARRAY are the same

Name:_____

Problem 3. (10 points):

In each of the five small programs below, assume M[x4000] has been initialized with a value before the programs are run. The value may represent a 2's complement integer, an address, an ASCII code, a bit vector, a floating point number, or an instruction. For each of the five programs below, put an $\times$ in the box corresponding to what the value in M[x4000] represents.

Part a. (2 points):

```
    .ORIG x3000
    LDI R0, A
    OUT
    HALT
A   .FILL x4000
    .END
```

- ☐ 2's Complement Integer
- ☐ Address
- ☒ ASCII Code
- ☐ Bit Vector
- ☐ Floating Point
- ☐ Instruction

Part b. (2 points):

```
    .ORIG x3000
    LDI R0, B
    LDR R1, R0, #0
    HALT
B   .FILL x4000
    .END
```

- ☐ 2's Complement Integer
- ☒ Address
- ☐ ASCII Code
- ☐ Bit Vector
- ☐ Floating Point
- ☐ Instruction

Part c. (2 points):

```
    .ORIG x3000
    LDI R0, C
    NOT R0, R0
    ADD R0, R0, #1
    HALT
C   .FILL x4000
    .END
```

- ☒ 2's Complement Integer
- ☐ Address
- ☐ ASCII Code
- ☐ Bit Vector
- ☐ Floating Point
- ☐ Instruction

Part d. (2 points):

```
    .ORIG x3000
    LDI R0, D
    PUTS
    HALT
D   .FILL x4000
    .END
```

- ☐ 2's Complement Integer
- ☒ Address
- ☐ ASCII Code
- ☐ Bit Vector
- ☐ Floating Point
- ☐ Instruction

Part e. (2 points):

```
    .ORIG x3000
    LD R0, E
    JSRR R0
    HALT
E   .FILL x4000
    .END
```

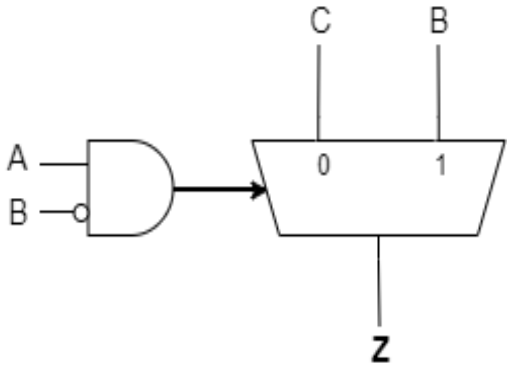
- ☐ 2's Complement Integer
- ☐ Address
- ☐ ASCII Code
- ☐ Bit Vector
- ☐ Floating Point
- ☒ Instruction

Name:_____

Problem 4. (10 points):

Part a. (5 points): A logic circuit and incomplete truth table are shown below. The logic circuit has three inputs: A, B, and C.

Your job: Complete the truth table so that it reflects the behavior of the logic circuit.



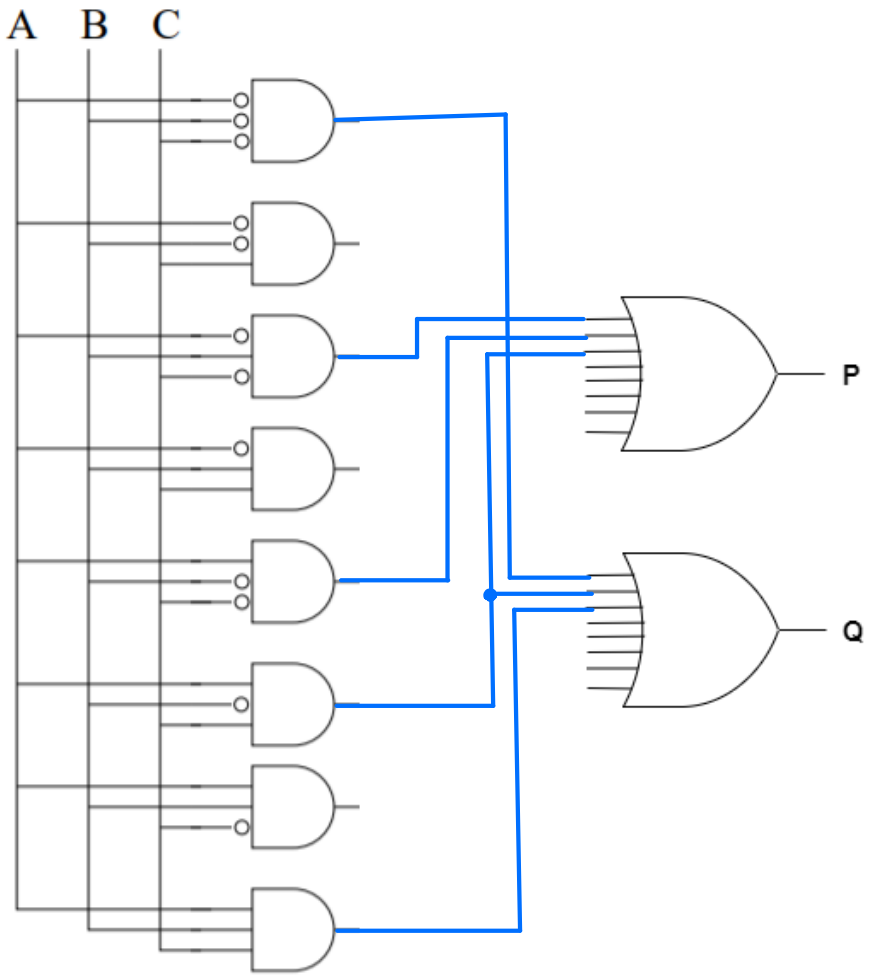
A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Name:_____

Part b. (5 points): A multi-output truth table and a logic circuit are shown below. The logic circuit is missing all the connections from the outputs of the AND gates to the inputs of the OR gates.

Your job: Draw the connections so that the logic cirucit implements the truth table.

A	B	C	P	Q
0	0	0	0	1
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1



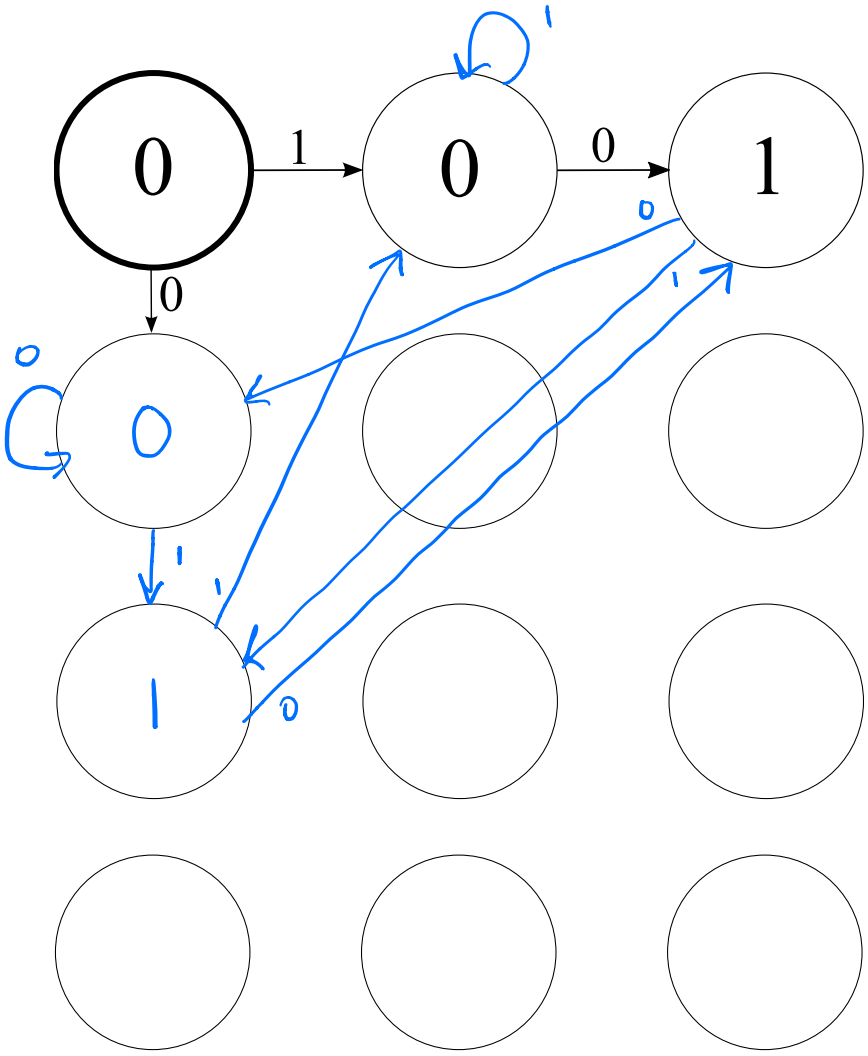
Name:_____

Problem 5. (10 points):

A synchronous finite state machine has a single input and a single output. One input value is provided each cycle. The output of the finite state machine is 1 each time the input provided is different from the previous value, i.e., from 0 to 1, or from 1 to 0. The synchronous finite state machine outputs 0 at all other times.

Your job: Complete the synchronous finite state machine.

We have provided twelve states. Use as many as you need. We have also provided the initial state (shown in bold) and a few of the state transitions and corresponding outputs.



Name:_____

Problem 6. (25 points):

Part a. (5 points): Shown below are a main program starting at M[x3000] and a keyboard interrupt routine starting at M[x1000]. Before the main program was run, the operating system enabled interrupts and loaded M[x0180] with x1000.

```
        .ORIG x3000
LOOP    BR LOOP
        HALT
        .END
```

```
        .ORIG x1000
        ST R0, SaveR0
        LDR R0, R6, #0
        ADD R0, R0, #1
        STR R0, R6, #0
        LD R0, SaveR0
        RTI
SaveR0  .BLKW #1
        .END
```

A key is pressed. Does the main program halt? Why or why not? Explain in 20 words or fewer.

Yes, because the service routine increments the return address on the stack (popped during RTI)

Part b. (5 points): An Aggie tried to write a **recursive** subroutine which, when given an integer n, returns the sum of the first n positive integers. For example, for $n = 4$, the subroutine returns 10 (i.e., $1 + 2 + 3 + 4$). The subroutine takes the argument n in R0 and returns the sum in R0.

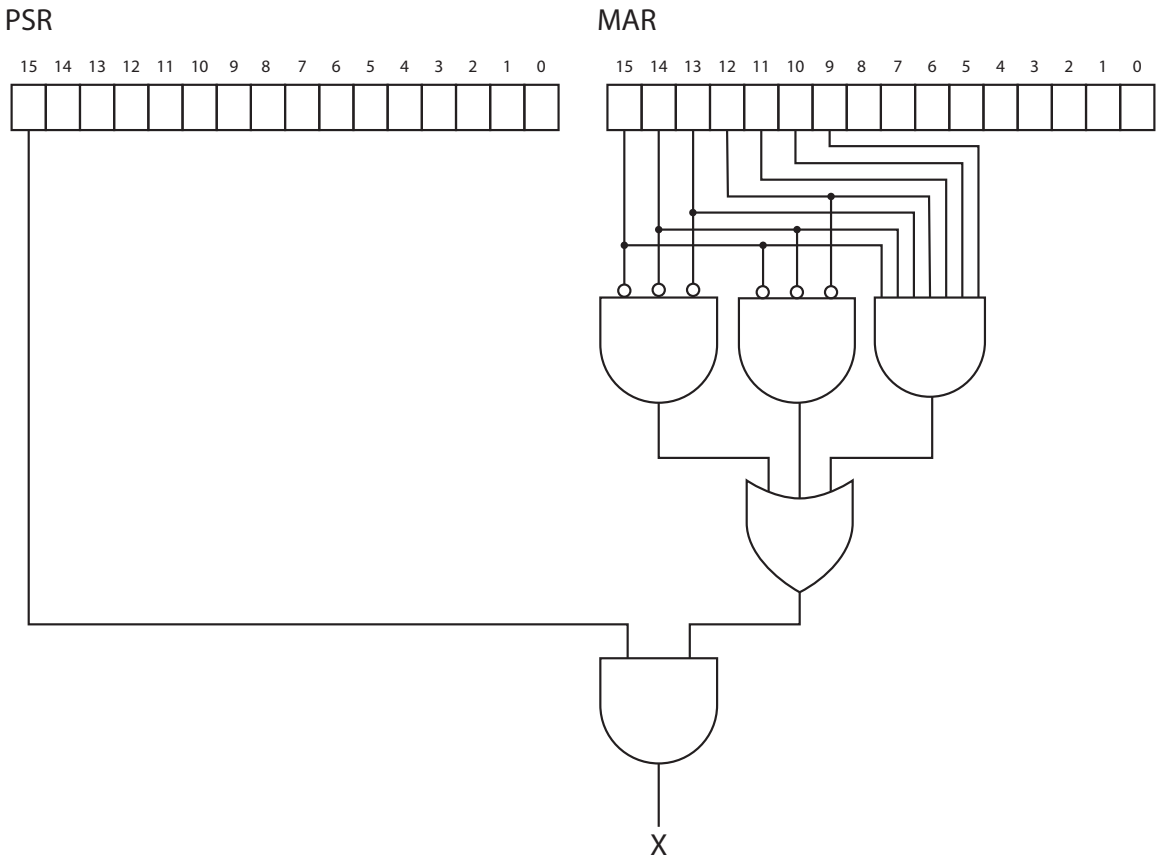
```
SUM     ADD R6, R6, #-1
        STR R7, R6, #0
        ADD R6, R6, #-1
        STR R1, R6, #0

        ADD R1, R0, #0
        ADD R0, R0, #-1
        JSR SUM
        ADD R0, R0, R1

        LDR R1, R6, #0
        ADD R6, R6, #1
        LDR R7, R6, #0
        ADD R6, R6, #1
        RET
```

Unfortunately, the recursive subroutine does not work. What is the problem? Explain in 15 words or fewer.

There is no base case

Name:_____

Part c. (5 points): The LC-3 contains the following logic.

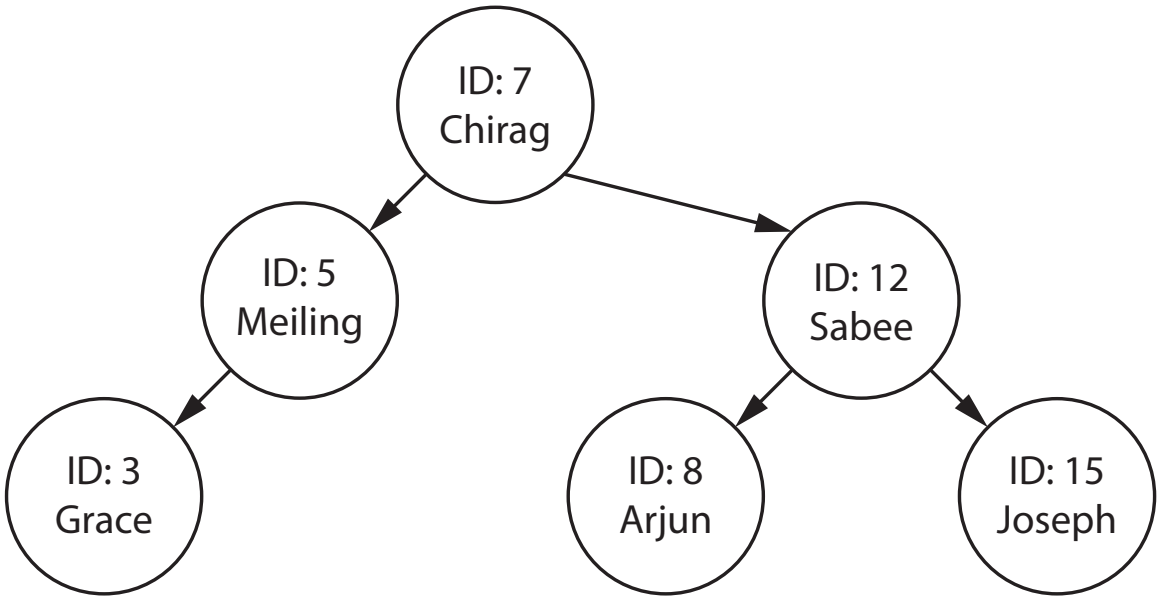


When does X = 1? Answer in 15 words or fewer.

When you are attempting to access privileged memory or a device register from user mode

Name:_____

Part d. (10 points): A *binary tree* is made up of nodes that contain values and pointers to at most two other nodes. An example is shown below.

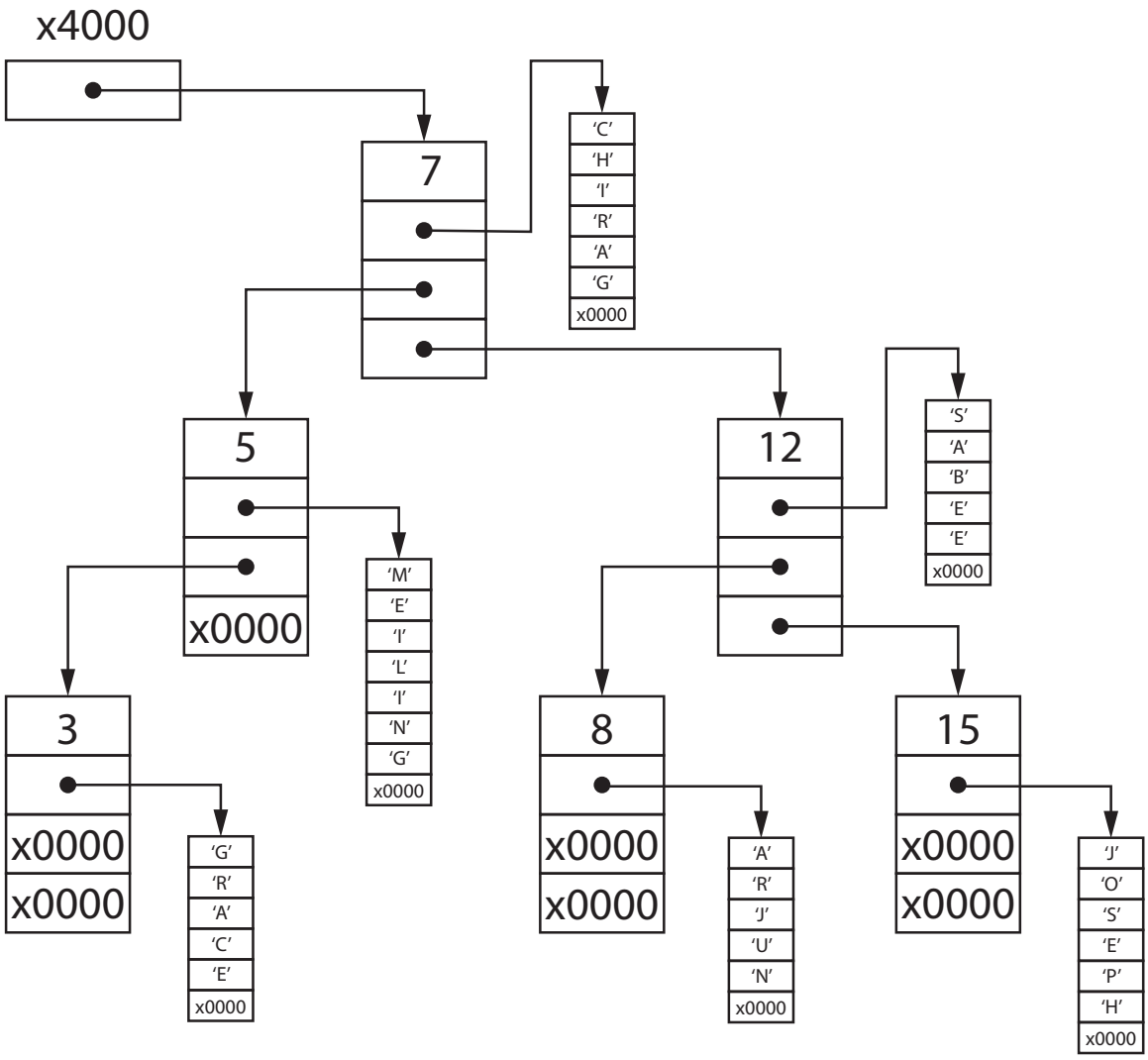


Each node can point to a node to the left and/or a node to the right. These nodes are referred to as the *left child* and *right child* respectively. In our example above, the node labeled "Sabee" has a left child "Arjun", a right child "Joseph", and an ID 12. If a binary tree has the property that for every node, its value is greater than the value of its left child and less than the value of its right child, it is called a *binary search tree*.

Note that the figure shown above is a binary search tree. Specifically, for every node, the ID is greater than the ID of its left child and less than the ID of its right child.

The binary search tree shown above can be implemented as shown on the following page.

Name:_____



Each node consists of four contiguous words of memory. The first word contains the ID, the second word is a pointer to a character string, identifying the person represented by the node, the third word contains a pointer to the left child, and the fourth word contains a pointer to the right child. If there is no left (or right) child, that word contains x0000. A list head (in this case location x4000) contains a pointer to (i.e., the address of) the top node (called the *root*) of the binary search tree. In this case, the root is the node labeled "Chirag."

Name:_____

Given an ID in R0, the following subroutine finds the name of the person associated with that ID, and loads the address of the character string of that person into R5. Note that some instructions in the subroutine are missing.

Your job: Fill in the missing instructions. You can assume that the binary search tree contains the node whose ID is in R0.

```
          .ORIG x3100
SEARCH    ST R1, SaveR1
          ST R2, SaveR2
          ST R3, SaveR3
          NOT R0, R0
          ADD R0, R0, #1
          LDI R1, ROOT
LOOP      LDR R2, R1, #0
          ADD R3, R0, R2
          BRz A
          BRp B
          LDR R1,R1, #3
          BR LOOP
B         LDR R1,R1,#2
          BR LOOP
A         LDR R5, R1, #1
          LD R1, SaveR1
          LD R2, SaveR2
          LD R3, SaveR3
          RET
ROOT      .FILL x4000
SaveR1    .BLKW #1
SaveR2    .BLKW #1
SaveR3    .BLKW #1
          .END
```

Name:_____

Problem 7. (20 points):

In the following subroutine, arguments are passed in R0 and R1, and the result is returned in R0 and R1.

```
          .ORIG x4000
          ST R2, SAVER2
          AND R2, R2, #0
          NOT R1, R1
          ADD R1, R1, #1
LOOP      ADD R0, R0, R1
          BRn DONE
          ADD R2, R2, #1
          BRnzp LOOP
DONE      NOT R1, R1
          ADD R1, R1, #1
          ADD R1, R0, R1
          ADD R0, R2, #0
          LD R2, SAVER2
          RET
SAVER2    .BLKW #1
```

Part a. (10 points): What does the subroutine do? Answer in 15 words or fewer.

Divide R0 by R1 and store quotient in R0 and remainder in R1

Name:_____

Part b. (10 points): Before the program is executed, R0 is initialized to 5, and R1 is initialized to 3.

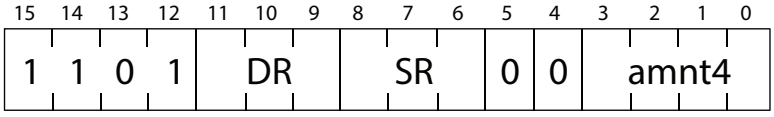
Your job: Fill in the missing entries. Register values should show the values loaded at the END of the cycle. Control signals should be represented by "x" if they are "don't cares." Memory accesses take five cycles.

Clock Cycle	State	Information
1	18	LD.MAR 1, LD.PC 1, GatePC 1, GateALU 0, GateMARMUX 0, PCMUX 10
10	3	Bus x400E, LD.MAR 1, LD.MDR 0, GatePC 0, GateMARMUX 1, ADDR1MUX 0, ADDR2MUX 01
11	23	LD.MDR 1, LD.REG , GatePC 0, GateALU 1, SR2MUX X, ALUK 11
23	28	MAR x4001, MDR x5440, MIO.EN 1, R.W 0, R 1
26	5	Bus x0000, LD.MDR 0, LD.REG 1, GatePC 0, GateALU 1, SR2MUX 1
34	30	IR x927F, LD.IR 1, LD.PC 0, GateMDR 1, GatePC 0
66	0	IR x8002, PC x4006
67	18	

Name:_____

Problem 8. (20 points):

We use the unused opcode to introduce an LSHF instruction to the LC-3 with the instruction encoding shown below.



The LSHF instruction left shifts the value in SR by the number of bits specified by amnt4, and stores the result into DR. As you recall from Programming Lab 1, when you shift n bits to the left, you insert n 0's from the right.

The modifications to the data path and state machine to implement the LSHF instruction are shown on the following two pages.

Part a. (15 points): Fill in the empty boxes in the incomplete state machine to implement the LSHF instruction.

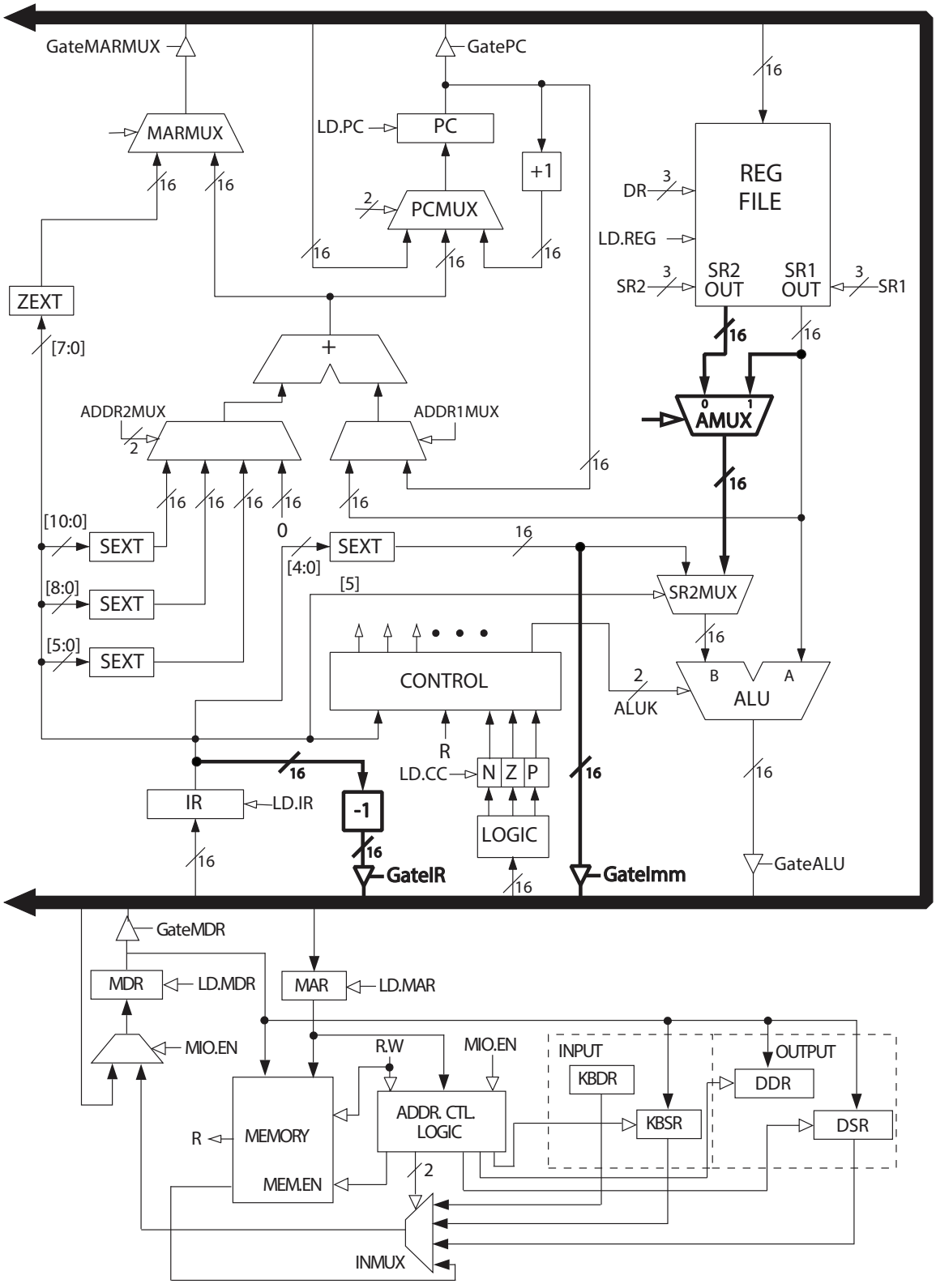
Part b. (5 points): Fill in the control signals for the states labeled B, D, and E on the state machine. If the value of a control signal does not matter, fill it with an X.

	LD.MAR	LD.MDR	LD.IR	LD.REG	LD.CC	LD.PC	GatePC	GateALU	GateMARMUX	GateIR	GateImm	ADDR1MUX	ADDR2MUX		ALUK		AMUX
State B	0	0	0	0	1	0	0	0	0	0	1	X	X	X	X	X	X
State D	0	0	0	1	0	0	0	1	0	0	0	X	X	X	0	0	1
State E	0	0	1	0	0	0	0	0	0	1	0	X	X	X	X	X	X

PROBLEM CONTINUES ON NEXT PAGE

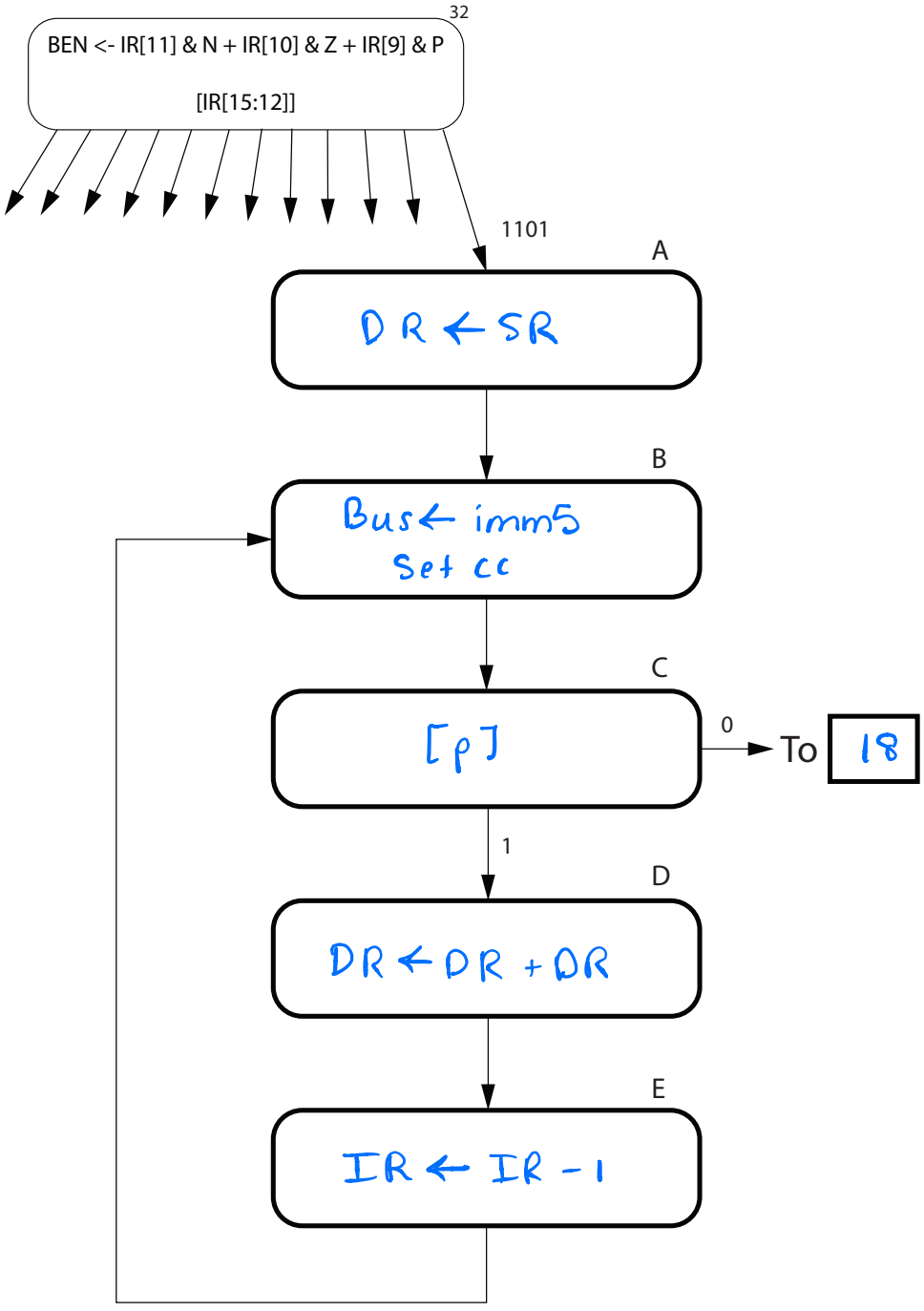
Name:_____

The modifications to the data path are shown below in bold.



PROBLEM CONTINUES ON NEXT PAGE

Name:_____


32
BEN <- IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
1101
A
DR ← SR
B
Bus ← imm5
Set cc
C
[p]
0
To
18
1
D
DR ← DR + DR
E
IR ← IR - 1

Name:_____

Problem 9. (20 points):

The LC-3 computer executes a program, starting with the fetch of the first instruction in clock cycle 1. The table below shows the use of the bus in several clock cycles. Memory accesses take five clock cycles.

Your job: Fill in the missing entries with the values on the bus during those clock cycles.

Clock Cycle	Bus
1	x5020
8	xE1FF
10	x5020
11	x5021
18	x3000
20	x5022
21	x5020
27	x5022
34	x5020
36	x0000
37	x5023
40	x2FFF
47	x2FFE
53	x0180
60	x1000
67	x8000
85	x5023

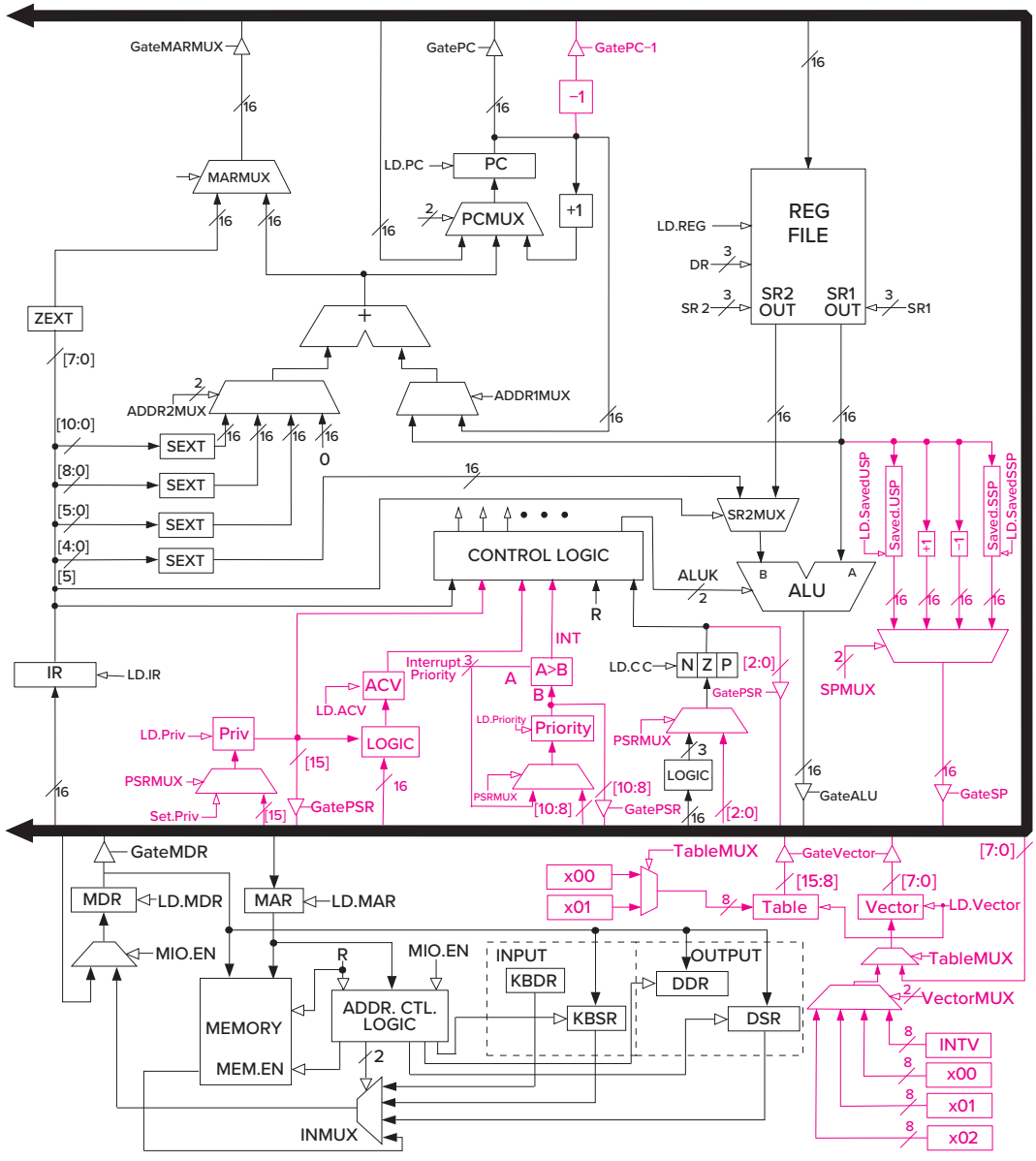


Figure C.8 LC-3 data path, including additional structures for interrupt control.

Section C.7.1 describes the flow of processing required to initiate an interrupt. Section C.7.3 describes the flow of processing required to initiate an exception.

C.7.1 Initiating an Interrupt

While a program is executing, an interrupt can be requested by some external event so that the normal processing of instructions can be preempted and the control can turn its attention to processing the interrupt. The external event requests

the event that causes the program that is executing to stop. Interrupts are events that usually have nothing to do with the program that is executing. Exceptions are events that are the direct result of something going awry in the program that is executing. The LC-3 specifies three exceptions: a privilege mode violation, an illegal opcode, and an ACV exception. Figure C.7 shows the state machine that carries these out. Figure C.8 shows the data path, after adding the additional structures to Figure C.3 that are needed to make interrupt and exception processing work.

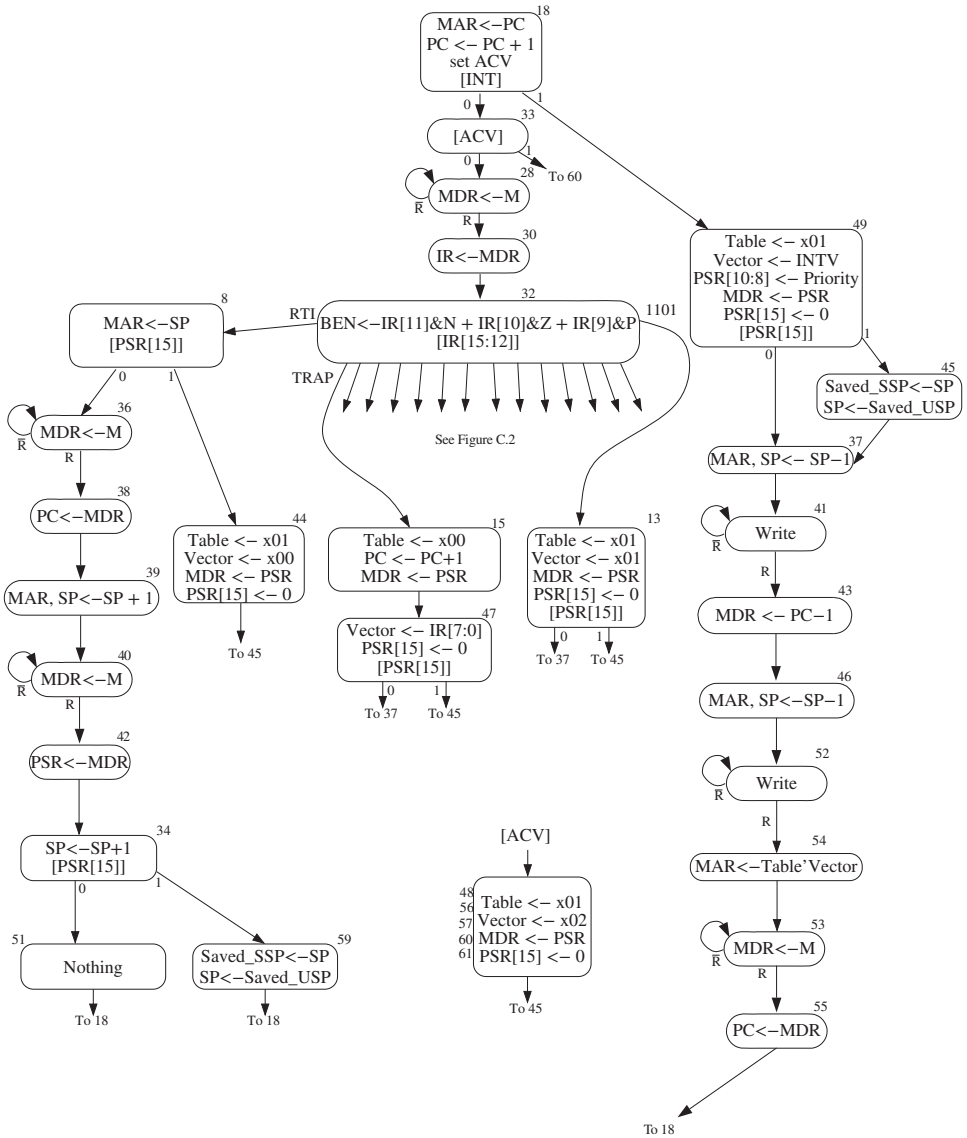


Figure C.7 LC-3 state machine showing interrupt control.