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

ECE 306 Fall 2025
Instructor: Yale N. Patt
TAs: Madeleine Dreher, Evan Lai, Luke Mason
Final Exam
Dec. 12th, 2025

Name and EID: _____SOLUTION_____

Part A:	**Part B:**
Problem 1 (10 points): _____	Problem 6 (25 points): _____
Problem 2 (10 points): _____	Problem 7 (25 points): _____
Problem 3 (10 points): _____	
Problem 4 (10 points): _____	
Problem 5 (10 points): _____	
Part A Total: _____	Part B Total: _____

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. Only legible answers will be graded.

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.

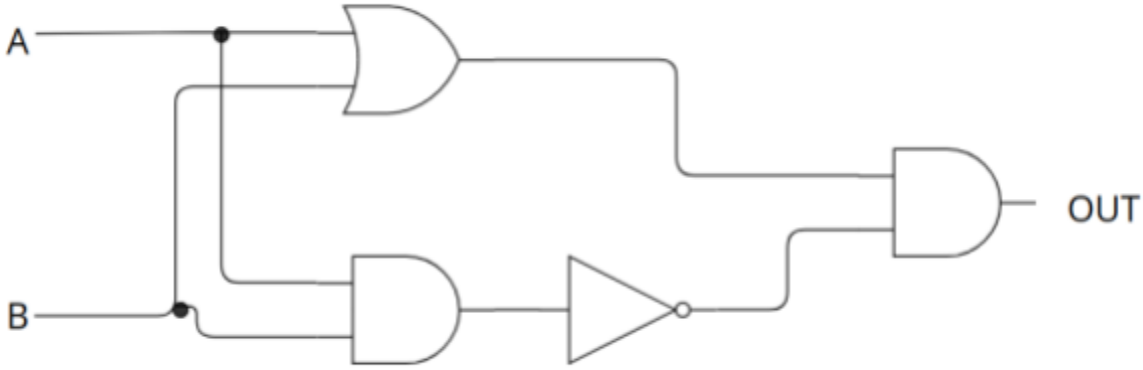
Signature: _____

GOOD LUCK!
(Have a good winter break)

Name: _____

Problem 1 (10 points): Answer the two following questions about logic and transistor circuits.

Part A (5 pts):
Look at the following logic circuit. It has two inputs A and B, and one output OUT. This circuit will NOT be used in Part B of this question.



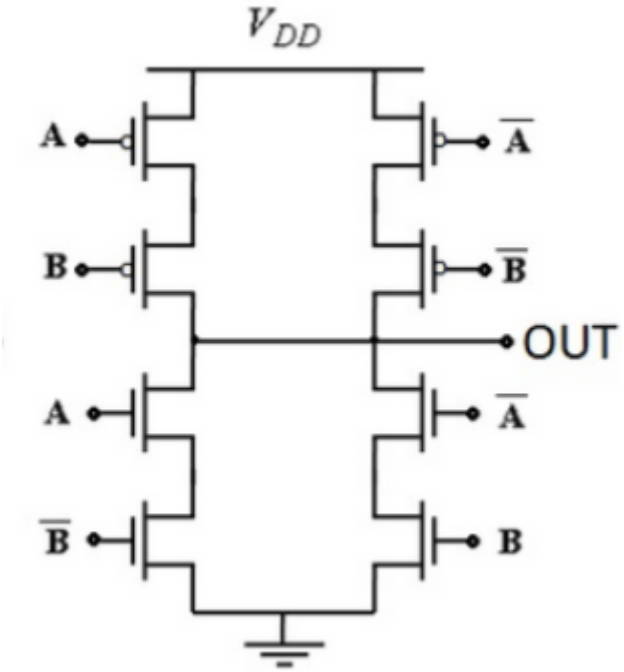
Complete the truth table for the logic circuit. **You DO NOT have to use all rows of the truth table.**

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

Name: _____

Part B (5 pts):

The following transistor circuit has two inputs A and B, and one output OUT.



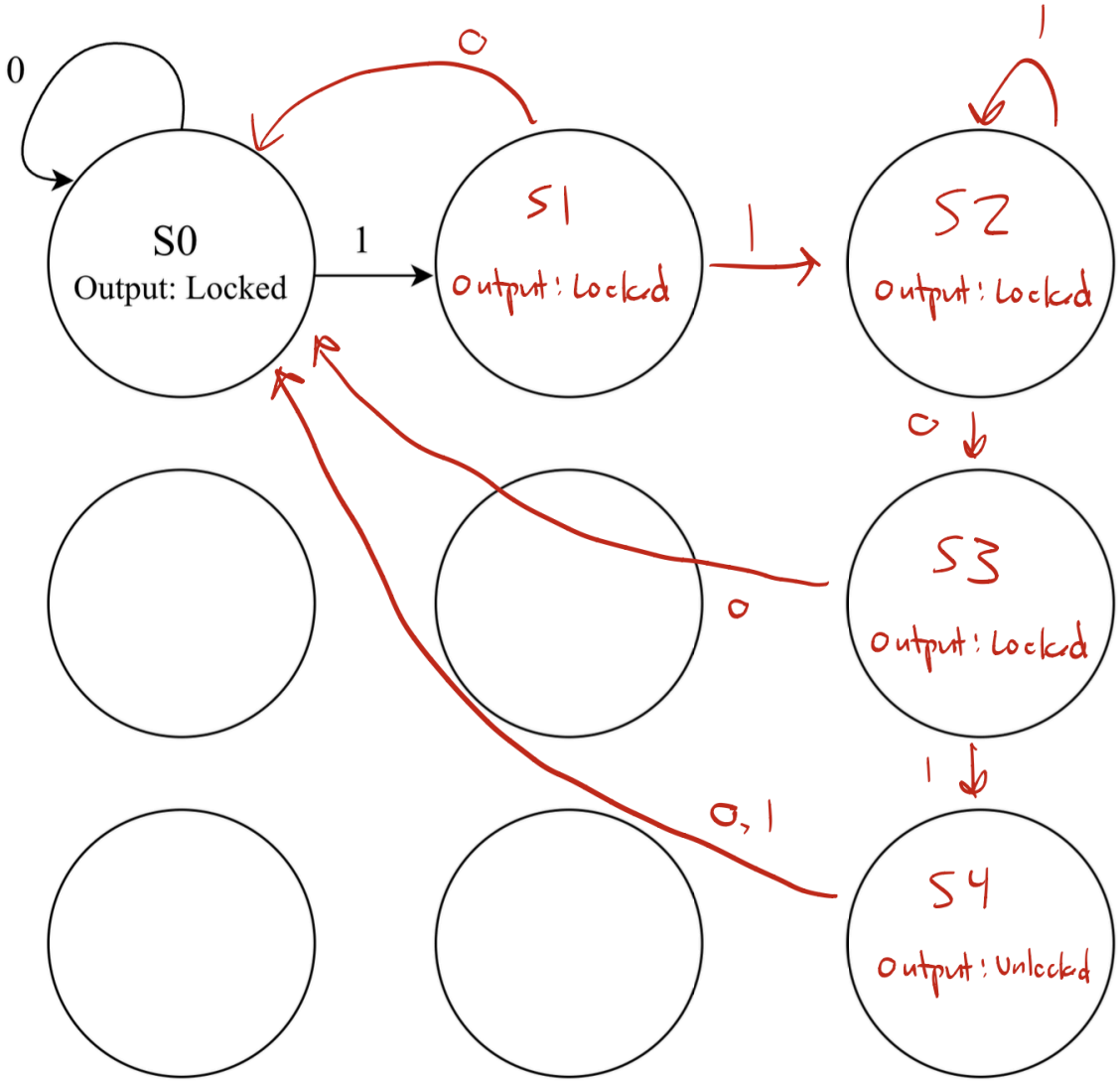
Complete the truth table for the transistor circuit. **You DO NOT have to use all rows of the truth table.**

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	1

Name: _____

Problem 2 (10 points): Design a finite state machine (FSM) for a lock that opens when the user enters the input 1101. The FSM receives one bit at a time as its input. Its output indicates whether the lock is currently locked or unlocked. **The initial state of the lock is the top left state.**

- Before the correct sequence is entered, the lock is locked.
- Once the sequence 1101 has been entered, the lock becomes unlocked, regardless of what the previous inputs were.
    a. Examples of a valid sequence include **both** '11101' and 00001101'
- After the lock is unlocked, **the lock returns to the initial state after any input.**
- You may use as many states as you need.



Name: _____

Problem 3 (10 points): With the current LC-3 datapath, suppose ADDR1MUX is broken and always selects the BaseR input, **NEVER passing the PC input through**.

Your Job: For each of the instructions below, decide whether they would still execute correctly given the broken ADDR1MUX. If they would, explain in fewer than 10 words. If they would not, list the registers and/or memory locations that would end up with incorrect values, as well as the incorrect value they would end up with.
Example: Incorrect, R7 would end up with Mem[BaseR]

STR R0 R1 #0

It would execute correctly. STR uses BaseR to calculate its address.

LDI R2 #5

It would execute incorrectly. LDI uses PC+offset9 to calculate its address.
R2 would end up with M[M[BaseR+5]]
Instead of M[M[PC+1+5]]

ADD R3 R3 #1

It would execute correctly, ADD doesn't use an address.

BRnzp #-2

It would execute incorrectly. BR uses PC+offset9 to calculate its address.
PC would end up with BaseR-2
Instead of PC+1-2

Name: _____

Problem 4 (10 points): The following program counts the number of elements in an array that **are equal to a value M**. The program is given three inputs: the array length is stored in memory location x4002, the target value M is stored in memory location x4000, and the base address of the array is stored in memory location x4003. The program must compute and store the number of elements equal to M in memory location x4001. Assume that the array contains at least one element.

Part A (7 points): Complete the program by filling in the missing instructions.

Part B (3 points): Modify the program to count the number of array elements **greater than M** by replacing a single instruction. Please cross out the original instruction and write your new instruction next to it.

```
        .ORIG x3000

        LDI R0, START
        LDI R1, LENGTH
        LDI R2, M
        AND R4, R4, #0

REP     LDR R3, R0, #0
        NOT R3, R3
        ADD R3, R3, #1
        ADD R3, R3, R2
        BRnp SKIP  BRzp SKIP

        ADD R4, R4, #1

SKIP    ADD R0, R0, #1
        ADD R1, R1, #-1
        BRp REP

        STI R4, STORE
        HALT

START   .FILL x4003
LENGTH  .FILL x4002
M       .FILL x4000
STORE   .FILL x4001
```

Name: _____

Problem 5 (10 points): You are given the assembly language program shown below:

```
            .ORIG     x3000
            LD        R1, A
            NOT       R1, R1
            ADD       R1, R1, #1
            LD        R2, B
            NOT       R2, R2
            ADD       R2, R2, #1
            TRAP           x20        ;GETC
            ADD       R3, R0, R2
            BRz       PRINT1
            ADD       R3, R0, R1
            BRz       PRINT2
            BRnzp     PRINT3
PRINT1      TRAP           x21        ;OUT
PRINT2      TRAP           x21        ;OUT
PRINT3      TRAP           x21        ;OUT
DONE        TRAP           x25        ;HALT
A           .FILL     x41         ;ASCII code for 'A'
B           .FILL     x42         ;ASCII code for 'B'
            .END
```

Your Job: What does the program output in the scenarios provided below? Assume the User types a character after GETC is called.

User types the character 'A' on the keyboard AA
User types the character 'B' on the keyboard BBB
User types the character 'C' on the keyboard C

Name: _____

Problem 6 (25 points): Suppose we have two programs: a user program located at x3000 and an interrupt service routine located at x1000. Both programs share access to the user stack, **whose stack pointer is stored in memory location x5000**. Recall user stack begins at xFDFF.

The user program runs in an infinite loop, continuously checking the stack for new data. If it determines that the stack is not empty, it pops the value from the top of the stack and outputs it to the display. The keyboard interrupt service routine pushes the input character onto the stack. Both the user program and the interrupt service routine are shown below.

User Program:

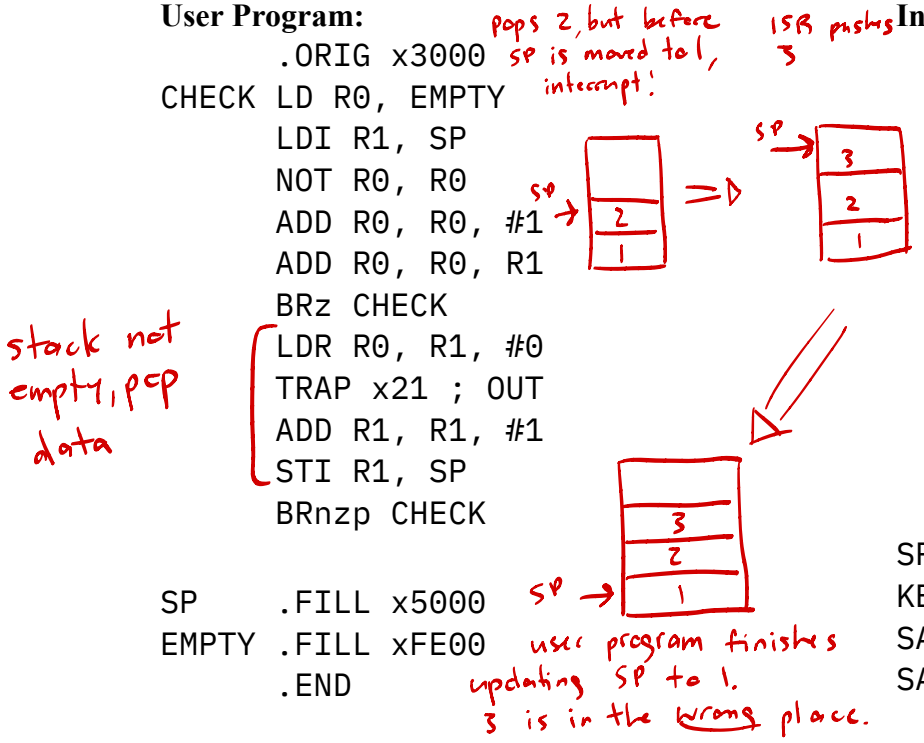
```
        .ORIG x3000
CHECK   LD R0, EMPTY
        LDI R1, SP
        NOT R0, R0
        ADD R0, R0, #1
        ADD R0, R0, R1
        BRz CHECK
        LDR R0, R1, #0
        TRAP x21 ; OUT
        ADD R1, R1, #1
        STI R1, SP
        BRnzp CHECK

SP      .FILL x5000
EMPTY   .FILL xFE00
        .END
```

Interrupt Service Routine:

```
        .ORIG x1000
        ST  R0, SAVE_0
        ST  R1, SAVE_1
        LDI R0, KBDR
        LDI R1, SP
        ADD R1, R1, #-1
        STR R0, R1, #0
        STI R1, SP
        LD R0, SAVE_0
        LD R1, SAVE_1
        RTI

SP      .FILL x5000
KBDR    .FILL xFE02
SAVE_0  .BLKW #1
SAVE_1  .BLKW #1
        .END
```



Part A (10 points): Each program functions correctly alone. However, a problem sometimes arises depending on when the interrupt happens with respect to the user program. Describe the sequence of events that leads to this problem. Assume that the interrupt service routine cannot be interrupted.

If the user program determines that the stack isn't empty, it pops data from the stack and outputs it. It modifies the SP, but before it can store it back to x5000, the ISR runs and loads the OLD value of SP. It pushes data onto the stack and updates SP to x5000. When the user program runs again, it will finish updating the SP. Now, the new data that the ISR pushed is in the incorrect spot with respect to the SP.

THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: _____

Part B (12 points): Complete the two routines that would solve this problem. The first blank instruction in the user program represents a call to ROUTINE_1, while the second blank instruction in the user program represents a call to ROUTINE_2.

Routine 1 must enable interrupts: Lets pending interrupts get serviced before we load from x5000

Routine 2 must disable interrupts: Prevents keyboard routine from interrupting the user program before it has updated the SP.

User Program:

```
        .ORIG x3000

CHECK   [                    ]
        [                    ]

        LD R0, EMPTY
        LDI R1, SP
        NOT R0, R0
        ADD R0, R0, #1
        ADD R0, R0, R1
        BRz CHECK
        LDR R0, R1, #0
        TRAP x21 ; OUT
        ADD R1, R1, #1
        STI R1, SP
        BRnzp CHECK

SP      .FILL x5000
EMPTY   .FILL xFE00
        .END
```

ROUTINE_1

```
ST R0, SAVE_R0
ST R1, SAVE_R1

LDI R0, KBSR
LD R1, MASK
AND R0, R0, R1
NOT R1, R1
ADD R0, R0, R1
STI R0, KBSR

LD R0, SAVE_R0
LD R1, SAVE_R1
RTI

KBSR    .FILL xFE00
MASK    .FILL xBFFF
SAVE_0  .BLKW 1
SAVE_1  .BLKW 1
```

set KBSR[14]

ROUTINE_2

```
ST R0, SAVE_R0
ST R1, SAVE_R1

LDI R0, KBSR
LD R1, MASK
AND R0, R0, R1
STI R0, KBSR

LD R0, SAVE_R0
LD R1, SAVE_R1
RTI

KBSR    .FILL xFE00
MASK    .FILL xBFFF
SAVE_0  .BLKW 1
SAVE_1  .BLKW 1
```

clear KBSR[14]

Main idea: Load SP → Modify SP → Store SP
Must be executed as a single, indivisible step.
This means that it CANNOT BE INTERRUPTED.

THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: _____

Part C (3 points): Should ROUTINE_1 and ROUTINE_2 be implemented as regular subroutines that are invoked with JSR or JSRR, or should they be implemented as new TRAP routines that are invoked with a TRAP instruction? Explain your answer in 20 words or fewer.

TRAP. Both routines must access ICBSR, which resides in system space. So ROUTINE_1 and ROUTINE_2 must execute in priviteged mode.

Name: _____

Problem 7 (25 points): A new instruction, **MYSTERY,** is added to the LC3**.** We don't know what this instruction does, but we know these things are true:

1. MYSTERY **replaces the STI instruction** in the ISA and uses the same opcode.
2. An exception occurs if MYSTERY is executed in unprivileged mode (user mode).
3. Memory is accessed three times during the *full* instruction cycle of MYSTERY.
4. The addition of MYSTERY adds no more than 7 new states.

MYSTERY instruction is executed at PC=x1000, and you record some signals on certain cycles. 8 measurements are taken during the full instructions cycle of MYSTERY. You don't know on what cycles these measurements are taken, but you know they are in chronological order.

Measurement #1: GatePC/1, LD.MAR/1
Measurement #2: IRD/1
Measurement #3: COND2/1 (COND1 and COND0 unknown)
Measurement #4: LD.MAR/1, BUS = x2345, ADDR2MUX/ZERO
Measurement #5: LD.MDR/1, All gate signals are 0
Measurement #6: GateMARMUX/1, MARMUX/ZEXT(IR[7:0]), LD.MAR/1
Measurement #7: R.W/Write
Measurement #8: J/18, all other control signals are 0, Current State = 63

Part A (5 points): In under 15 words, describe the purpose of this instruction:

Stores address at location specified by BaseR into the trap vector table at ZEXT(IR[7:0])
(In other words: create new trap instruction with vector IR[7:0] with trap routine at M[BaseR])

How to arrive at this answer:
M2 must be state 32.
M1 then must be state 18. So, these are not useful to us.
M3 COND2/1 means we're checking ACV, PSR[15], or INT

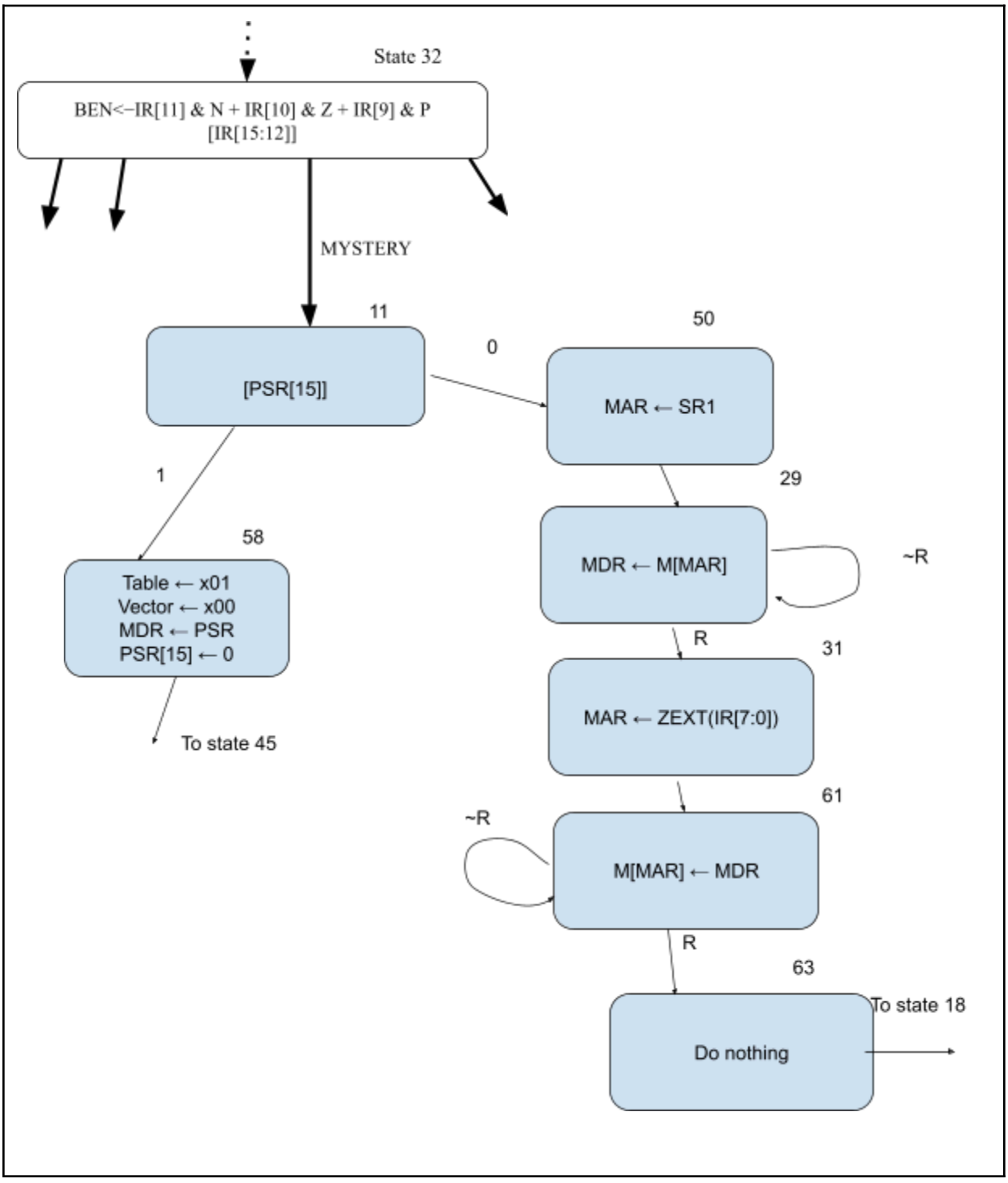
- There aren't enough added states (we're given it's <= 7 new states) to set ACV and use that for exception checking. PSR[15] is the easier option and fits the criteria. If you used ACV you may receive partial credit if used correctly.

M4 tells us we must output BaseR to bus, rather than PC because we know PC is x1000
M5 is loading into MDR, but it must not be from the bus since we aren't putting anything on the bus. This must be one of the memory cycle states.
M6 Now we load the trap vector into MAR
M7 must be a memory cycle state where we write - we store x2345 into the trap table with this
M8 is state 63 and does nothing, simply returning back to state 18.

THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: _____

Part B (11 points): Draw the sequence of states that is added for this instruction, beginning with the arrow from state 32. You are allowed to use any of the following unused state numbers: 11, 19, 29, 31, 50, 58, 61, 62, and 63. If you go to a state that already exists, draw an arrow labeled "To state #X", where X is the state number.



THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: _____

Part B explanation: Opcode is 11, so we arrive at state 11 from state 32 due to IRD. We must check for privilege, else cause an exception. If PSR[15] is set, we are in user mode, so go to state 58 to set up privilege exception (copy of state 44 from RTI). Otherwise, go to state 50 to continue the instruction. We must use state 58 and 50 here because checking PSR[15] overrides J[3] in the uSequencer to be 1, effectively adding 8 to the J bits (50 → 58). From part A, we know the other 5 states. The two memory access state pairs must be 29→31 and 61→63 respectively, because we are given that the last state must be 63, and we need to use those pairs of states for memory accesses (similar to how uSequencer handles PSR[15] checking, except checking for R bit overrides J[2]). Lastly, go to state 18.

Other innovative ideas will receive partial credit. For example:

- Using ACV to check privilege, after loading MAR with the trap vector (which would be in privileged space), or even after loading MAR with the BaseR (which, if using good security practice, would also be in privileged space). I don't think this works out with the state numbers given, but I will give partial credit to those who attempted it, and maybe I'll be proven wrong!
- Rerouting 31 to state 16 rather than including 61 and 63 at all. Technically the problem description forces there to be a state 63 that does nothing before returning to state 18. But, for reasons discussed in part D, it would not be a bad implementation to make 31 go to 16, although ruled out by the problem constraints.

Part C (5 points): Assuming each memory access takes 5 cycles, how many cycles does MYSTERY take to execute in this case, from state 18 until state 18 is reached again?

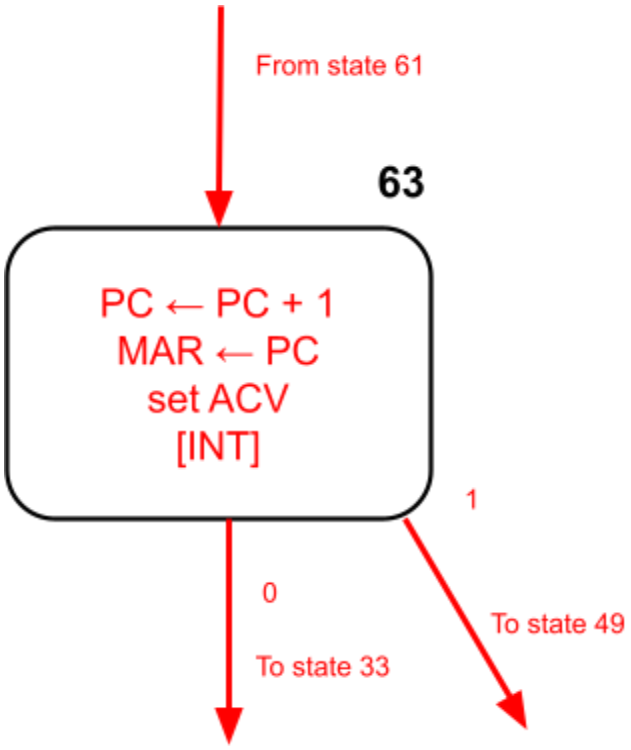
23 (32 if there is a privilege error)
If there is a privilege error, you would count the states all the way through the context switch until back to state 18.

Name: _____

Part D (4 points): To improve the performance of MYSTERY, state 63 can be changed to do something different. Fill in the diagram below for the new state 63 and draw/label any state transition arrows to/from other states.

The reason we have state 63 doing nothing in part B is because there is no way to return to state 18 immediately after a memory write. The microsequencer could only accomplish that if the memory cycle was on state 16, so when R is ready we would go to 18. However, state 16 already exists and does exactly that. Instead, make 63 have the same functionality as state 18 itself, and skip 18 afterward!

From the same line of thought, you can do even better. Instead of having state 61 and state 63 at all, simply go from state 31 to state 16. This would be a clever solution on this exam, although it was not expected since the beginning of the problem is constrained to do it the exact method above with state 63



This page is left blank intentionally. Feel free to use it for scratch work.
You may tear the page off if you wish.
Nothing on this page will be considered for grading.