

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

EE 306, Fall 2021

Yale Patt, Instructor

TAs: Sabee Grewal, Ali Fakhrzadehgan, Ying-Wei Wu, Michael Chen, Jason Math, Adeel Rehman

Final Exam, December 10, 2021

Name: \_\_\_\_\_

I would like to enroll in Professor McDermott's freshman design course in Spring 2022. **Circle one: Yes No**

I would like to enroll in 319K/312H in Spring, 2022. **Circle one: Yes No**

**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 (20 points): \_\_\_\_\_

Problem 7 (20 points): \_\_\_\_\_

Problem 8 (20 points): \_\_\_\_\_

Problem 9 (20 points): \_\_\_\_\_

**Part B (80 points):**

Total (130 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): The instructions in an ISA all have 8 bit opcodes. How many instructions can be specified in this ISA?

**Part b.** (2 points): An Aggie decided to debug their program by replacing each occurrence of the instruction `LD R0, A` with the two instruction sequence

```
AND R0, R0, #0
LD R0, A
```

How likely will this fix the problem so the program runs successfully? Explain in 10 words or fewer.

**Part c.** (6 points): An assemble time error occurs when the assembler fails to generate machine code (0s and 1s). A runtime error occurs during the actual execution of the program (e.g., an ACV or illegal opcode exception).

**Your Job:** In the following programs, if an error is present, identify whether it is an assemble time or runtime error, and specify the line number of the instruction that causes the error. If there is no error in the program, check “No Error” and leave the line number blank. Since TRAP service routines are part of the operating system, we will assume today that they contain no errors.

```
1      .ORIG x3000
2      AND R0, R0, #0
3      LEA R1, #0
4      LDR R2, R0, R1
5      HALT
6      .END
```

- ☐ Assemble Time Error  
☐ Runtime Error  
☐ No Error

Line Number: \_\_\_\_\_

```
1      .ORIG x3000
2      LD R0, TEXT
3      OUT
4      HALT
5 TEXT  .STRINGZ "UT ECE"
6      .END
```

- ☐ Assemble Time Error  
☐ Runtime Error  
☐ No Error

Line Number: \_\_\_\_\_

```
1      .ORIG x3000
2      AND R0, R0, #0
3      .FILL xDDDD
4      HALT
5      .END
```

- ☐ Assemble Time Error  
☐ Runtime Error  
☐ No Error

Line Number: \_\_\_\_\_

Name: \_\_\_\_\_

**Problem 2.** (10 points):

The following LC-3 assembly language program operates on an array containing N elements. The number of elements N is contained in memory location x3200, and the base address of the array is contained in memory location x3201.

```

                                .ORIG x3000
                                LDI R0, N
                                BRz DONE
                                LDI R1, ARRAY
LOOP    LDR R2, R1, #0
                                BRzp SKIP
                                NOT R2, R2
                                ADD R2, R2, #1
                                STR R2, R1, #0
SKIP    ADD R1, R1, #1
                                ADD R0, R0, #-1
                                BRp LOOP
DONE    HALT

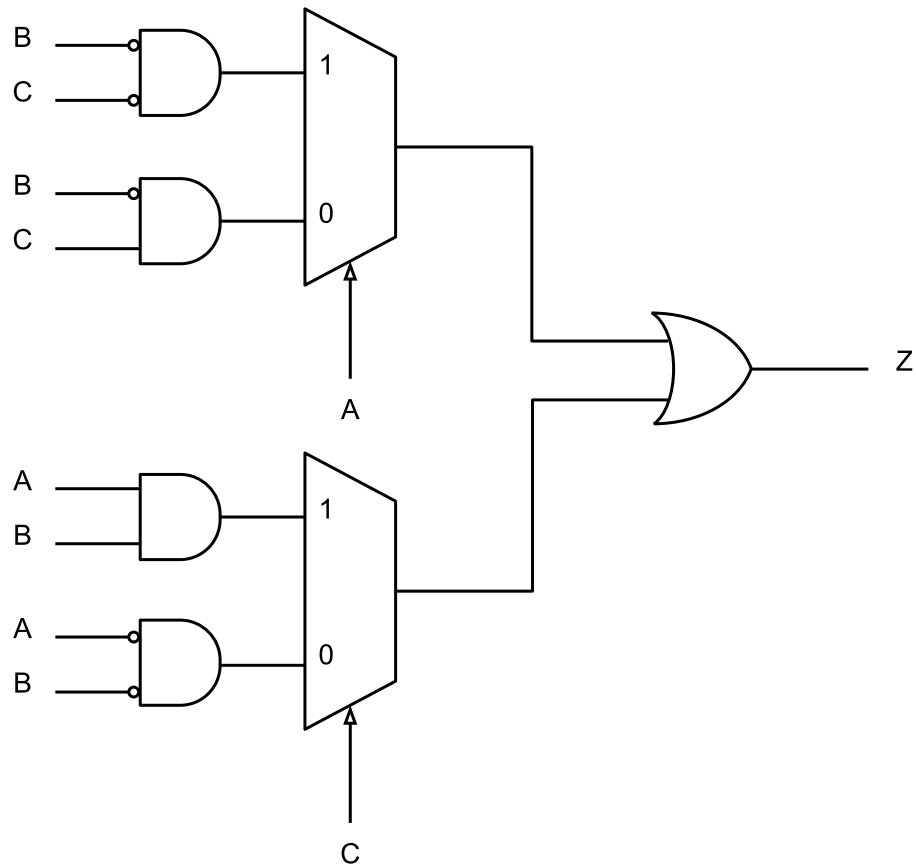
N       .FILL x3200
ARRAY   .FILL x3201
                                .END
```

What does this program do? Answer in 15 words or fewer.

Name: \_\_\_\_\_

**Problem 3.** (10 points):

A logic circuit is shown below. The logic circuit has three inputs: A, B, and C.



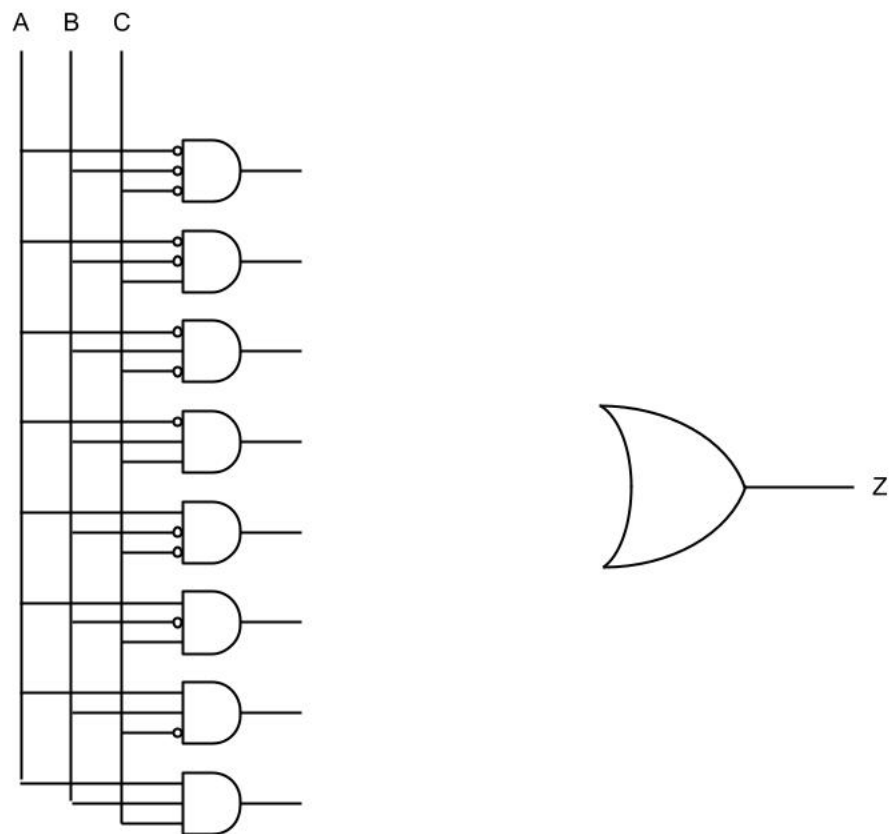
**PROBLEM CONTINUES ON NEXT PAGE**

Name: \_\_\_\_\_

**Part a.** (5 points): Complete the truth table so that it reflects the behavior of the logic circuit.

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

**Part b.** (5 points): Implement the logic circuit using the PLA below.

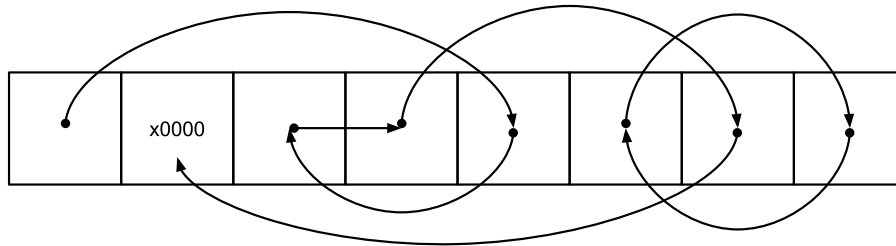


Name: \_\_\_\_\_

**Problem 4.** (10 points):

In class, we learned that a *pointer* is simply a memory address. For example, we say “R1 is a pointer” when the 16-bit value contained in R1 represents a memory address. We *dereference a pointer* when we load the contents of the memory address pointed to by a pointer. We’ve seen this multiple times in class. For example, with linked lists. Each node contained a pointer to the next node. We dereferenced the pointer when we wanted to move on to the next node.

Consider an array of pointers, each of which points to a different element of the array. Assume that this array always contains a single element that is the null pointer x0000. Also, assume that if you start at the first pointer in the array, you will always reach the null pointer x0000. An example is shown below.



Below is a program written in LC-3 assembly language that counts the number of pointers we need to dereference until we hit the null pointer, starting from the first pointer in the array. The LC-3 program assumes that the array starts at memory location x3200 and stores the answer in the memory location labelled RESULT. Given the example above, the LC-3 program would store a 5 in the memory location labelled RESULT. Note that some instructions are missing.

**Your Job:** Fill in the missing instructions.

```

                                .ORIG x3000
                                AND  R0, R0, #0
                                LDI  R1, ARRAY
LOOP                            BRz  DONE
                                
                                
                                BRnzp LOOP
DONE                            ST  R0, RESULT
                                HALT
ARRAY                          .FILL x3200
RESULT                        .BLKW #1
                                .END
```

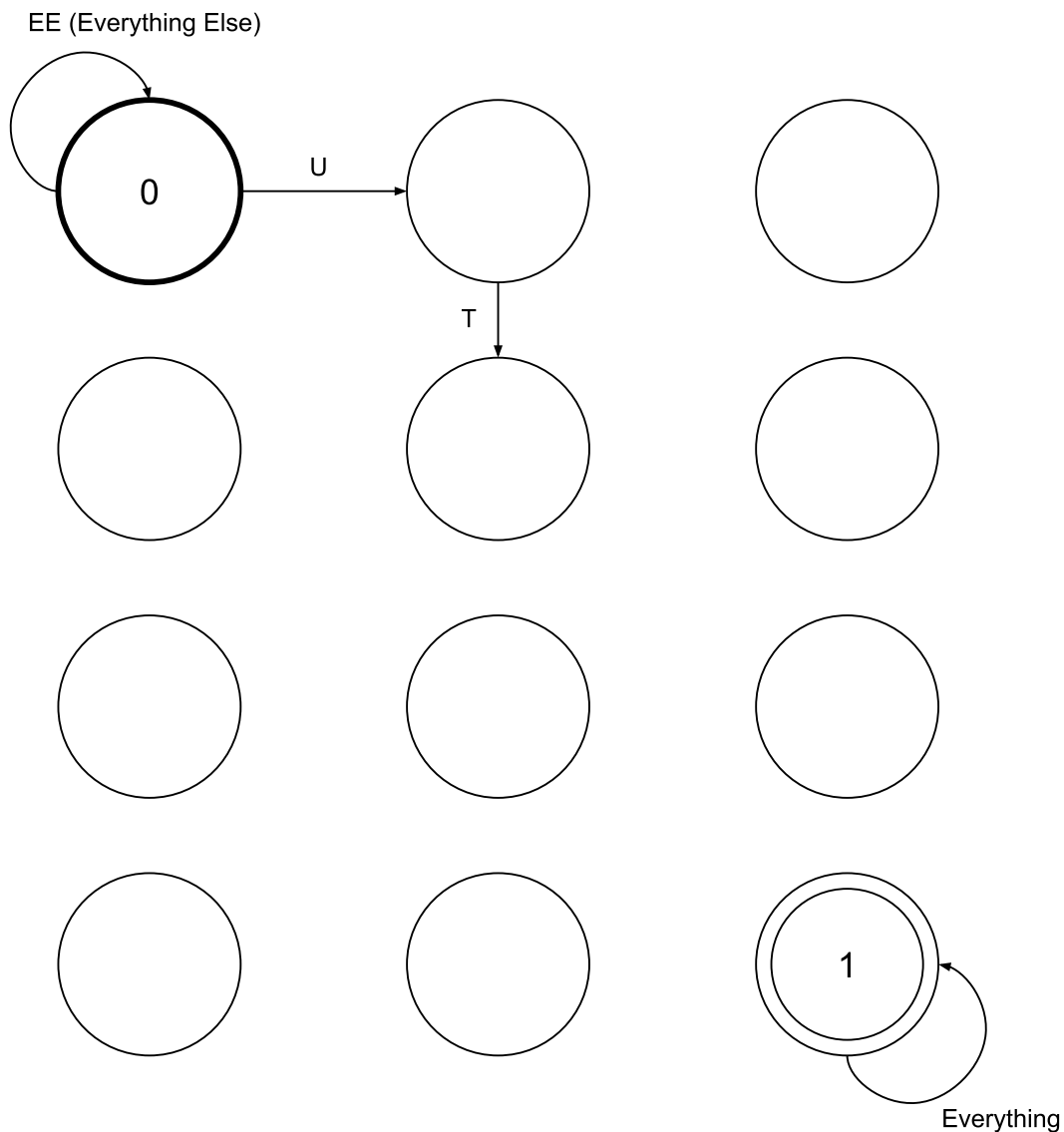
Name: \_\_\_\_\_

**Problem 5.** (10 points):

In this problem, we wish to design a finite state machine that detects whether or not the character string UTECE is present in a sequence of characters.

We will input the sequence of characters to the finite state machine, one character at a time. The finite state machine will output a 0 every cycle until it detects the sequence UTECE. If it never detects UTECE, it will never output a 1. If the finite state machine detects the sequence UTECE, it will output a 1 every cycle thereafter.

**Your Job:** Complete the finite state machine. We have provided twelve states. Use as many as you need. For each state, you must show the transition for every possible input. Luckily, only a small number of inputs will produce meaningful transitions. The rest can be combined into an “Everything Else” input (“EE”, for short). For example, from the initial state, the only relevant transition is if the input is “U” – all the other inputs can be combined into an “Everything Else” transition. Finally, we have identified the final state (the double-circle) if UTECE is detected.



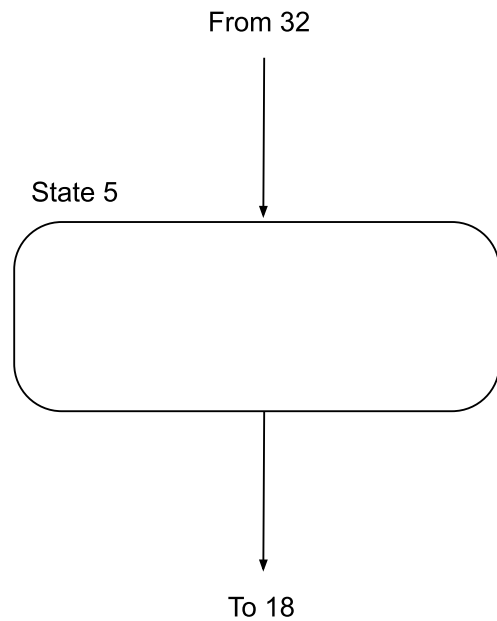
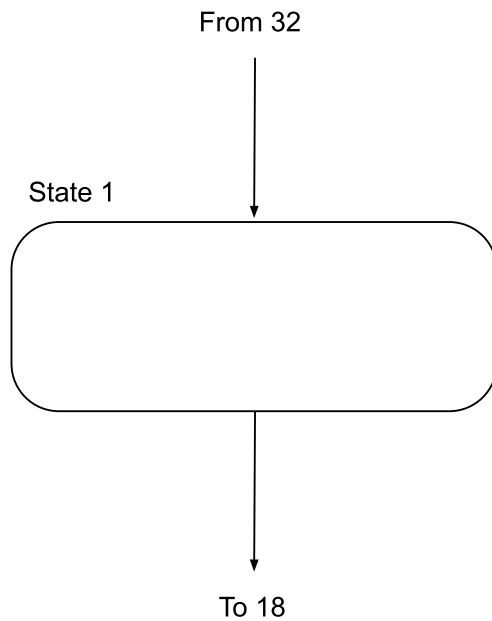
Name: \_\_\_\_\_

**Problem 6.** (20 points):

A student filling in the control store signals needed for each state of the LC-3 made some mistakes with state 1 and state 5. The result is shown in the table below. Some control signals are correct, some are incorrect. Assume all control signals not shown are correct.

	LD.REG	LD.CC	LD.PC	GateALU	PCMUX		ALUK	
State 1	1	1	1	1	0	1	0	0
State 5	1	0	0	1	0	0	0	1

**Part a.** (5 points): Fill in states 1 and 5, given the control signals specified in the above table.



**PROBLEM CONTINUES ON NEXT PAGE**



Name: \_\_\_\_\_

**Part b.** (15 points): Assume the following program is loaded in the memory of an LC-3 machine with the control signals specified above and starts executing at x3000.

```

                                .ORIG x3000
0                               LEA R0, NUM1
1                               LDR R0, R0, #0
2                               ADD R0, R0, #4
3                               ADD R0, R0, #3
4                               ADD R0, R0, #2
5      LOC                     ADD R0, R0, #1
6                               AND R0, R0, #0
7                               BRnp DONE
8                               LD R0, NUM2
9                               BRp LOC
A      DONE                   HALT
B      NUM1                   .FILL x3002
C      NUM2                   .FILL x3009
                                .END
```

**Your Job:** In the table below, identify the line number of each instruction executed, and write the contents of R0 at the end of execution of each instruction. Assume that R0 initially contains x0000. Use as many rows in the table as you need.

	Line Number	Value in R0
1st Instruction		
2nd Instruction		
3rd Instruction		
4th Instruction		
5th Instruction		
6th Instruction		
7th Instruction		
8th Instruction		
9th Instruction		
10th Instruction		

Name: \_\_\_\_\_

**Problem 7.** (20 points):

The subroutine SETN (shown on the next page) searches a 16-bit bit vector, starting with bit 0, looking for the first occurrence of N consecutive 0s, and sets those N bits to 1. R0 contains the address of the bit vector, R1 contains the number of bits N. The subroutine returns in R2 the bit number of the first bit that is set. If SETN completes this task successfully, it returns a 0 in R5. If SETN cannot find N consecutive 0s, it fails, and returns a 1 in R5. In the case SETN fails, R2 contains garbage.

For example, if R0 = x4000, M[x4000] = 0000110000110100, and R1 = 3, SETN will set M[x4000] to 0000110111110100, R2 to 6, and R5 to 0.

Four subroutines are provided to help you write SETN. (All of them may not be necessary.)

**Subroutine One: INIT.** INIT clears the bit vector pointed to by R0. For example, if R0 = x4000, M[x4000] = xEB0A, INIT will set M[x4000] to x0000.

**Subroutine Two: SET.** SET sets the bit specified by R1 in the bit vector pointed to by R0. For example, if R0 = x4000, M[x4000] = x0000, and R1 = 4, SET will set M[x4000] to x0010.

**Subroutine Three: CLEAR.** CLEAR clears the bit specified by R1 in the bit vector pointed to by R0. For example, if R0 = x4000, M[x4000] = x7A03, and R1 = 0, CLEAR will set M[x4000] to x7A02.

**Subroutine Four: EXAMINE** EXAMINE sets R2 to the value (0 or 1) of the bit specified by R1 in the bit vector pointed to by R0. For example, if R0 = x4000, M[x4000] = x8000, and R1 = 15, EXAMINE will set R2 to 1.

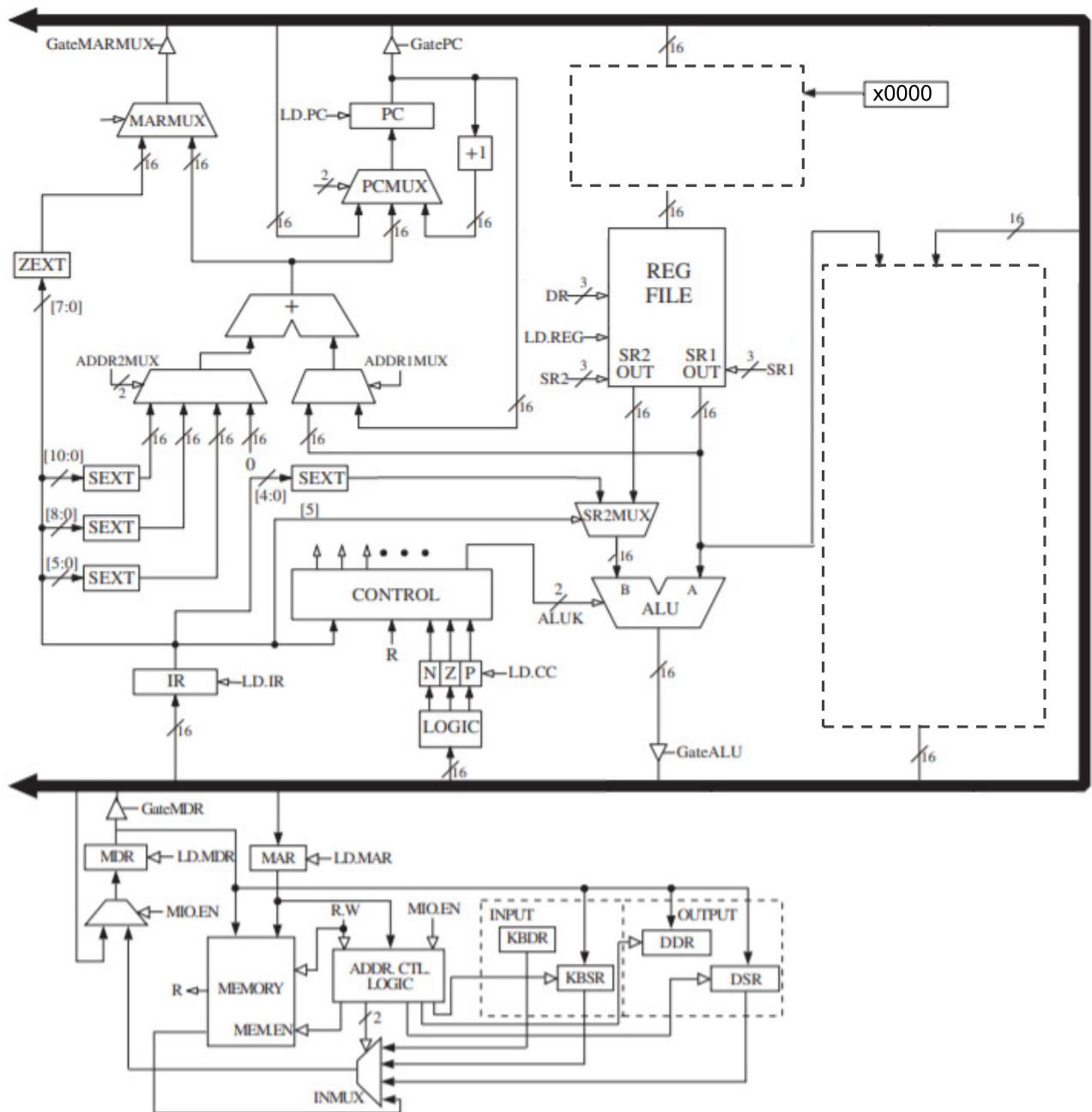
**Your Job:** Fill in the missing instructions of the subroutine SETN shown on the next page.

**PROBLEM CONTINUES ON NEXT PAGE**

SETN	AND R5, R5, #0
	ST R3, SAVER3
	ST R4, SAVER4
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	ST R1, N
	AND R1, R1, #0
	LD R3, N
LOOP1	BRz FOUND
	ADD R4, R1, #-16
	BRz FAILURE
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	ADD R2, R2, #0
	BRp RESET_N
	ADD R3, R3, #-1
	BR LOOP1
RESET_N	LD R3, N
	BR LOOP1
FOUND	ADD R1, R1, #-1
	LD R3, N
LOOP2	BRz DONE
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	ADD R3, R3, #-1
	BR LOOP2
FAILURE	ADD R5, R5, #1
DONE	ADD R2, R1, #1
	LD R3, SAVER3
	LD R4, SAVER4
	LD R1, N
	<div style="border: 1px solid black; height: 20px; width: 100%;"></div>
	RET
N	.BLKW #1
LINKAGE	.BLKW #1
SAVER3	.BLKW #1
SAVER4	.BLKW #1



**Part b.** (10 points): Complete the data path to implement MUL by adding the necessary structures and control signals in the spaces within the dotted lines.



Name: \_\_\_\_\_

**Problem 9.** (20 points):

A user program executing on an LC-3 computer takes 283 clock cycles to execute. The table below identifies nine of the 283 clock cycles during the execution of the program. Your job is to identify which state the computer is in during each of the nine clock cycles and what the contents of the PC, PSR, MAR, and MDR are at the END of each of the nine clock cycles. Assume memory accesses take 5 clock cycles. Assume that user programs run at PL0. Note that a part of this problem is to figure out the starting address of the user program.

In case you forgot (or don't have it on your three extra sheets), PSR[15] = 0 means privileged mode, and PSR[15] = 1 means user mode. PSR[10:8] specify the priority level of the program. PSR[2:0] specify the condition codes.

**For counting clock cycles, use the complete state machine (i.e., the one that includes the states that test for ACV exceptions).**

Cycle	State	PC	MAR	MDR	PSR
1	18			x0000	
8				xE1FE	
12					x0302
33			x01AB		
41			x1A30	x103F	
51				x8000	
68		x3002			
76				x6000	
80					

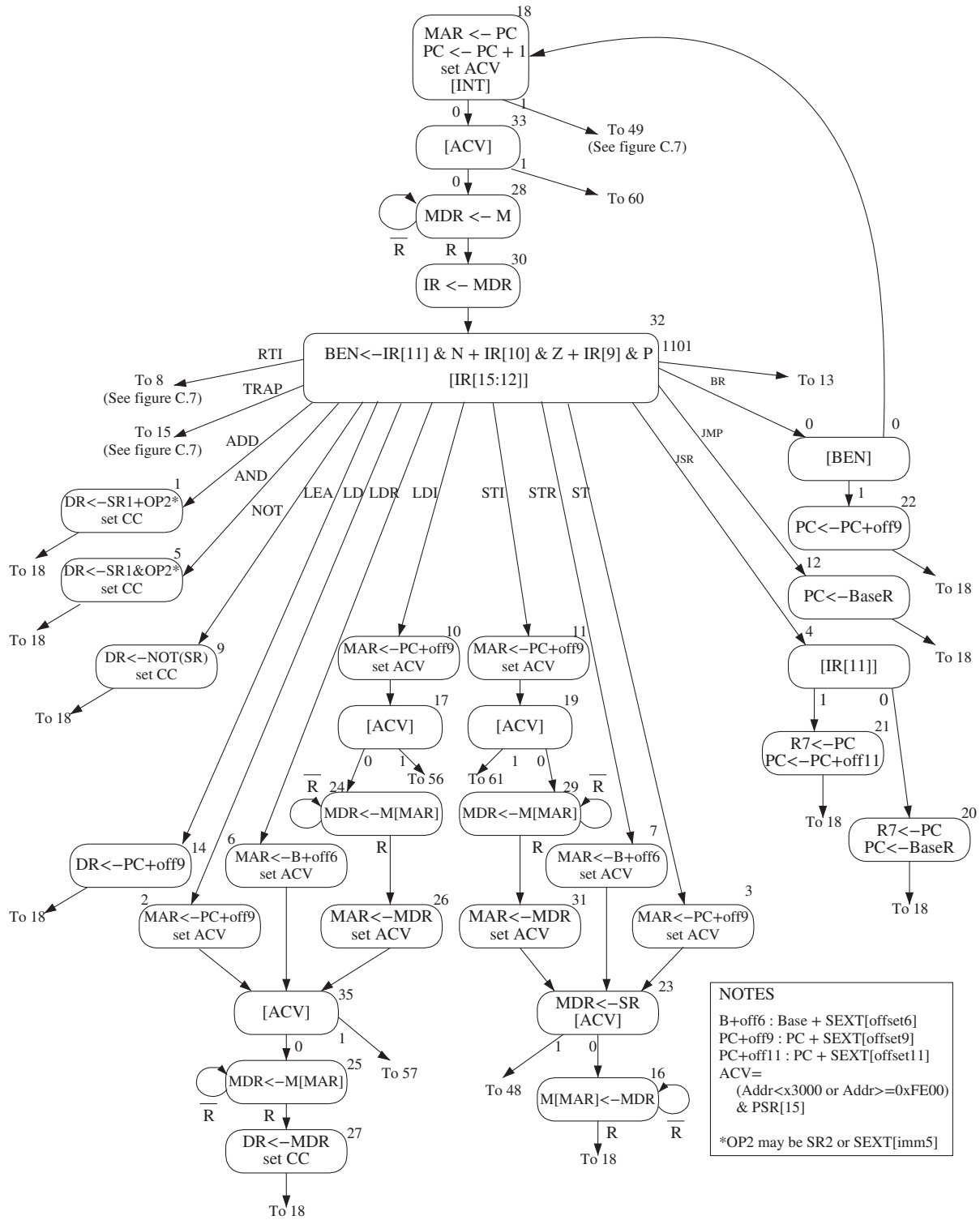


Figure C.2 A state machine for the LC-3.

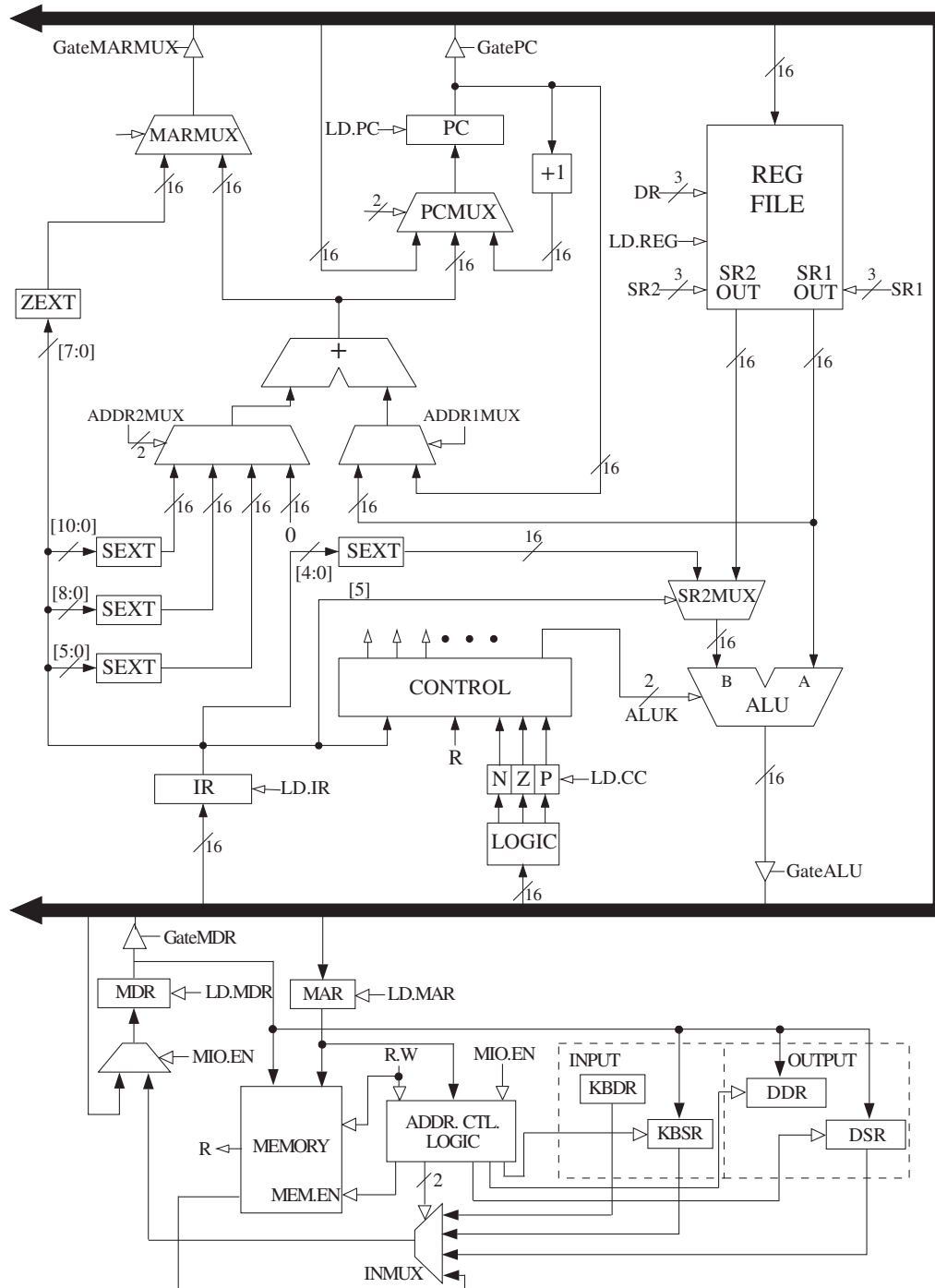


Figure C.3 The LC-3 data path.



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD <sup>+</sup>	0001				DR			SR1			0	00		SR2		
ADD <sup>+</sup>	0001				DR			SR1			1	imm5				
AND <sup>+</sup>	0101				DR			SR1			0	00		SR2		
AND <sup>+</sup>	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					
LD <sup>+</sup>	0010				DR			PCoffset9								
LDI <sup>+</sup>	1010				DR			PCoffset9								
LDR <sup>+</sup>	0110				DR			BaseR			offset6					
LEA	1110				DR			PCoffset9								
NOT <sup>+</sup>	1001				DR			SR			111111					
RET	1100				000			111			000000					
RTI	1000				000000000000											
ST	0011				SR			PCoffset9								
STI	1011				SR			PCoffset9								
STR	0111				SR			BaseR			offset6					
TRAP	1111				0000			trapvect8								
reserved	1101															

Figure A.2 Format of the entire LC-3 instruction set. Note: + indicates instructions that modify condition codes

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.

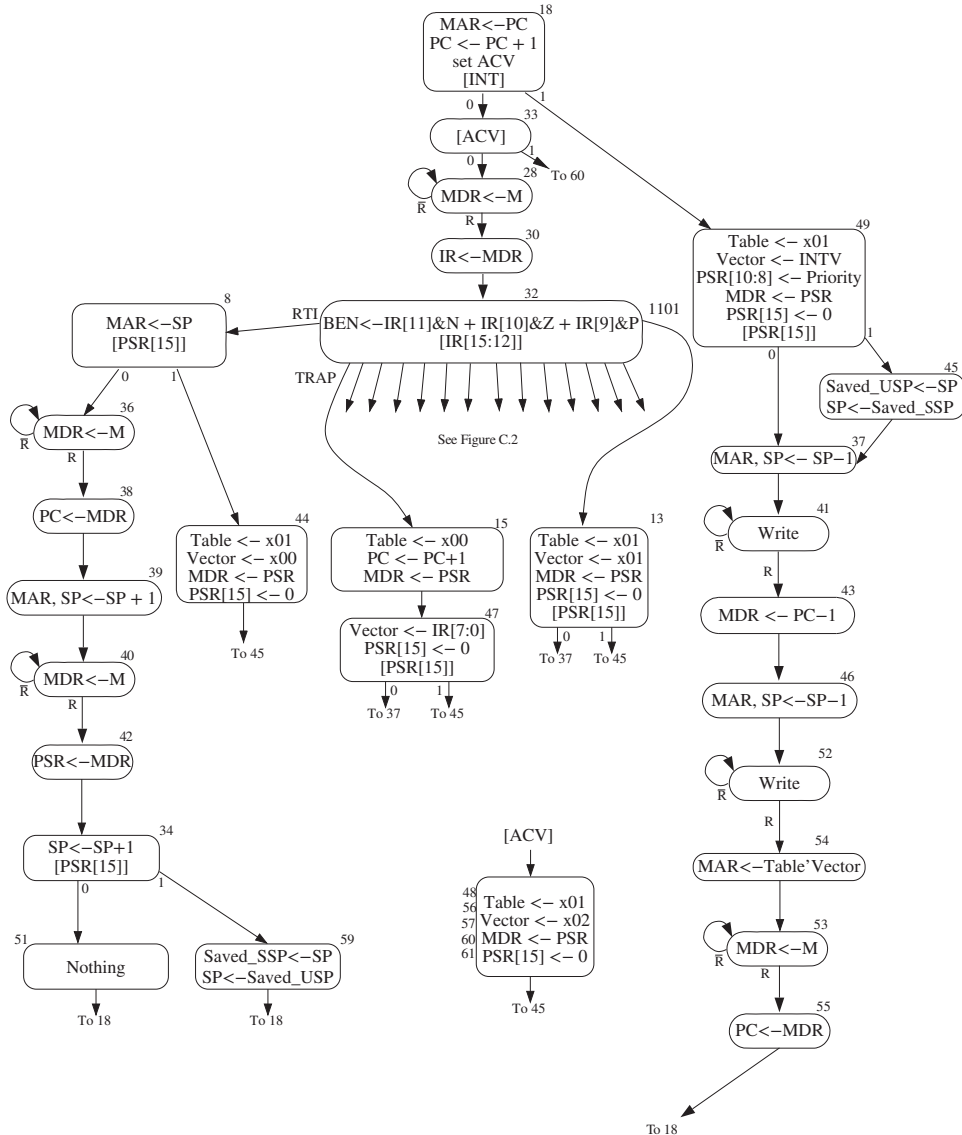


Figure C.7 LC-3 state machine showing interrupt control.

**Table C.1 Data Path Control Signals**

Signal Name	Signal Values
LD.MAR/1:	NO, LOAD
LD.MDR/1:	NO, LOAD
LD.IR/1:	NO, LOAD
LD.BEN/1:	NO, LOAD
LD.REG/1:	NO, LOAD
LD.CC/1:	NO, LOAD
LD.PC/1:	NO, LOAD
LD.Priv/1:	NO, LOAD
LD.Priority/1:	NO, LOAD
LD.SavedSSP/1:	NO, LOAD
LD.SavedUSP/1:	NO, LOAD
LD.ACV/1:	NO, LOAD
LD.Vector/1:	NO, LOAD
GatePC/1:	NO, YES
GateMDR/1:	NO, YES
GateALU/1:	NO, YES
GateMARMUX/1:	NO, YES
GateVector/1:	NO, YES
GatePC-1/1:	NO, YES
GatePSR/1:	NO, YES
GateSP/1:	NO, YES
PCMUX/2:	PC+1 (00) ;select pc+1 BUS (01) ;select value from bus ADDER (10) ;select output of address adder
DRMUX/2:	11.9 (00) ;destination IR[11:9] R7 (01) ;destination R7 SP (10) ;destination R6
SR1MUX/2:	11.9 (00) ;source IR[11:9] 8.6 (01) ;source IR[8:6] SP (10) ;source R6
ADDR1MUX/1:	PC (0), BaseR (1)
ADDR2MUX/2:	ZERO (00) ;select the value zero offset6 (01) ;select SEXT[IR[5:0]] PCoffset9 (10) ;select SEXT[IR[8:0]] PCoffset11 (11) ;select SEXT[IR[10:0]]
SPMUX/2:	SP+1 (00) ;select stack pointer+1 SP-1 (01) ;select stack pointer-1 Saved SSP (10) ;select saved Supervisor Stack Pointer Saved USP (11) ;select saved User Stack Pointer
MARMUX/1:	7.0 (0) ;select ZEXT[IR[7:0]] ADDER (1) ;select output of address adder
TableMUX/1:	x00 (0), x01 (1)
VectorMUX/2:	INTV (00) Priv.exception (01) Opc.exception (10) ACV.exception (11)
PSRMUX/1:	individual settings, BUS
ALUK/2:	ADD (00), AND (01), NOT (10), PASSA (11)
MIO.EN/1:	NO, YES
R.W/1:	RD, WR
Set.Priv/1:	0 ;Supervisor mode 1 ;User mode

**Table A.3** Trap Service Routines

Trap Vector	Assembler Name	Description
x20	GETC	Read a single character from the keyboard. The character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x21	OUT	Write a character in R0[7:0] to the console display.
x22	PUTS	Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. Writing terminates with the occurrence of x0000 in a memory location.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console monitor, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x24	PUTSP	Write a string of ASCII characters to the console. The characters are contained in consecutive memory locations, two characters per memory location, starting with the address specified in R0. The ASCII code contained in bits [7:0] of a memory location is written to the console first. Then the ASCII code contained in bits [15:8] of that memory location is written to the console. (A character string consisting of an odd number of characters to be written will have x00 in bits [15:8] of the memory location containing the last character to be written.) Writing terminates with the occurrence of x0000 in a memory location.
x25	HALT	Halt execution and print a message on the console.