

# CS150: Computer Organization and Architecture

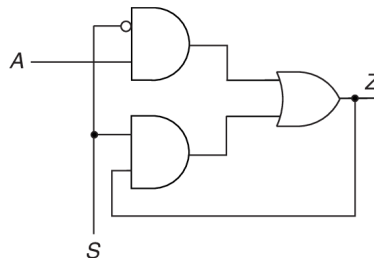
## Final Exam

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

No books, notes, or electronic devices of any kind are to be used. This exam contains 10 questions and is worth 320 points.

1. (10 pts) A certain combinational circuit has two inputs. The values of the two inputs for the last five cycles were **01**, **10**, **11**, **00**, and **01**. The values of the two inputs are again **10**. What is the effect that the previous five inputs have had on the current output?

2. (10 pts) Consider the circuit below.



- What is the value of Z in this circuit when the input S is 0?
- If the input S is 0 and is then switched to 1, how will the value of Z be effected after the switch?

3. (50 pts) Build an assembly language subroutine named `SetLowerNibble` that will take an argument on the stack and set the lower 4 bits of the value to be ones without affecting the upper four bits of the value. For example, if the value of the argument on the stack is 11010011 when your subroutine is called, the result should be 11011111 when your subroutine is finished. Your subroutine will place the computed result onto the stack in order to return it to the caller. Your subroutine must be placed at address 0x24 in program memory and must use only instructions that are contained in the CS150 AVR instruction subset. The following code shows how your subroutine will be used:

```
...  
push R23          ; push R23 value onto stack  
call SetLowerNibble ; set bits [3..0] of the value  
pop R31           ; put result into R31  
...
```

4. (20 pts) Consider the following Norfair screenshot:

The screenshot shows the Norfair simulator interface. The main window displays assembly code with line numbers 24 to 50. Line 49, `brbs 0, StayPuft`, is highlighted in green. The right panel shows the 'Norfair: Registers' and 'Norfair: Memory' windows.

Norfair: Registers		Norfair: Memory	
Name	Value	Address	Value
PC	0x0018	0x8E7	0x00
SPH	0x08	0x8E8	0x00
SPL	0xFD	0x8E9	0x00
SREG	0x14	0x8EA	0x00
R0	0xFF	0x8EB	0x00
R1	0x00	0x8EC	0x00
R2	0x32	0x8ED	0x00
R3	0x4A	0x8EE	0x00
R4	0x03	0x8EF	0x00
R5	0x6C	0x8F0	0x00
R6	0x05	0x8F1	0x00
R7	0xD2	0x8F2	0x00
R8	0x00	0x8F3	0x00
R9	0x29	0x8F4	0x00
R10	0x73	0x8F5	0x00
R11	0x88	0x8F6	0x00
R12	0x00	0x8F7	0x00
R13	0x19	0x8F8	0x00
R14	0x73	0x8F9	0x00
R15	0xFD	0x8FA	0x00
R16	0xB1	0x8FB	0x00
R17	0x33	0x8FC	0x00
R18	0x80	0x8FD	0x00
R19	0x81	0x8FE	0x00
R20	0x94	0x8FF	0x02

The 'Messages' window at the bottom shows: Asserting RESET... ok. Simulation started. Asserting RESET... ok.

After the processor fetches and executes the BRBS instruction that is highlighted on line 49 of the editor, what will be the value of the following items?

PC: \_\_\_\_\_

SREG: \_\_\_\_\_

SPL: \_\_\_\_\_

Memory Location 0x8FF: \_\_\_\_\_

5. (20 pts) Consider the following Norfair screenshot:

The screenshot shows the Norfair simulator interface. The main window displays assembly code with line numbers 19 to 45. Line 28, `rcall Find`, is highlighted in green. To the right, the 'Norfair: Registers' panel shows the PC register at 0x0013 and the SPL register at 0xFC. The 'Norfair: Memory' panel shows memory locations from 0x8E7 to 0x8FF, with values mostly 0x00 and 0x14 at 0x8FF. The Messages panel at the bottom shows simulation status.

Norfair: Registers		Norfair: Memory	
Name	Value	Address	Value
PC	0x0013	0x8E7	0x00
SPH	0x08	0x8E8	0x00
SPL	0xFC	0x8E9	0x00
SREG	0x21	0x8EA	0x00
R0	0x00	0x8EB	0x00
R1	0x00	0x8EC	0x00
R2	0x00	0x8ED	0x00
R3	0x00	0x8EE	0x00
R4	0x00	0x8EF	0x00
R5	0x00	0x8F0	0x00
R6	0x00	0x8F1	0x00
R7	0x00	0x8F2	0x00
R8	0x00	0x8F3	0x00
R9	0x00	0x8F4	0x00
R10	0x00	0x8F5	0x00
R11	0x00	0x8F6	0x00
R12	0x00	0x8F7	0x00
R13	0x00	0x8F8	0x00
R14	0x00	0x8F9	0x00
R15	0x00	0x8FA	0x00
R16	0xA5	0x8FB	0x00
R17	0x00	0x8FC	0x00
R18	0x00	0x8FD	0x00
R19	0x00	0x8FE	0x00
R20	0x00	0x8FF	0x14

Messages:

- Asserting RESET... ok.
- Simulation started.
- Simulation halted.
- Loading Context... ok.

After the processor fetches and executes the RCALL instruction that is highlighted on line 28 of the editor, what will be the value of the following items?

PC: \_\_\_\_\_

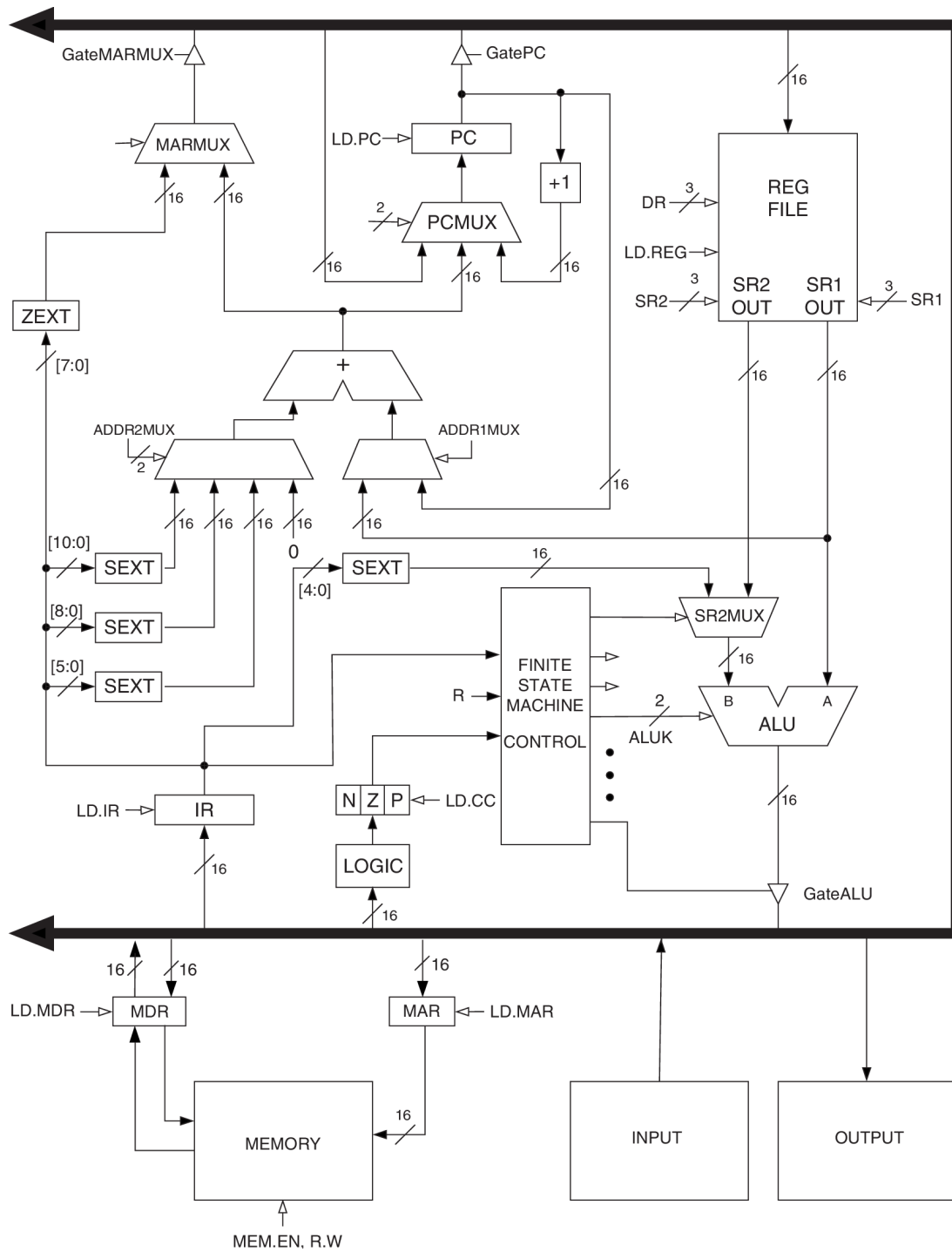
SPL: \_\_\_\_\_

Memory Location 0x8FC: \_\_\_\_\_

Memory Location 0x8FF: \_\_\_\_\_

6. (50 pts) Build an assembly language subroutine named `SuperfyR0` that will find the largest of the values in the registers R0, R1, R2, and R3 and copy that value into R0. For example, if R0 contains 28, R1 contains 15, R2 contains 3, and R4 contains 77 when your subroutine is called, when your subroutine is finished R0 will contain 77, R1 will contain 15, R2 will contain 3, and R4 will contain 77. Your subroutine must be placed at address 0x30 in program memory and must use only instructions that are contained in the CS150 AVR instruction subset. Your subroutine cannot use any data memory locations or any registers other than R0, R1, R2, and R3 to solve this problem.

7. (10 pts). Consider the diagram below that depicts a stored-program computer. Does this computer implement the Harvard model or the Von Neumann model? Please justify your answer.



8. (50 pts). Build an assembly language subroutine named `IsEightTendie` that will determine if the value in R8 is evenly divisible by 8. If the value in R8 is evenly divisible by 8, your subroutine should place the value 0xFF on the stack, and if the value in R8 is not evenly divisible by 8, your subroutine should place the value 0x00 on the stack. Your subroutine must be placed at address 0x26 in program memory and must use only instructions that are contained in the CS150 AVR instruction subset. You can use any registers that you want, but the value of all registers must be preserved for the caller. The following code shows how your subroutine will be used:

```
...  
call IsEightTendie ; check to see if R8 has what we're after  
pop R1             ; put result into R1  
...
```

9. (50 pts). Build an assembly language subroutine named `Normalize` that will take an argument on the stack and transform it as per the following description. If the value of the argument is odd, your subroutine should just return the original argument on the stack. If the value of the argument is greater than the value in `R0`, your subroutine should subtract the value in `R1` from the value of the argument and return the new value of the argument by placing it on the stack. If the value of the argument is less than the value in `R1`, your subroutine should add the value of `R0` to the value of the argument and return the new value of the argument by placing it on the stack. Your subroutine must be placed at address `0x20` in program memory and must use only instructions that are contained in the CS150 AVR instruction subset. The following code shows how your subroutine will be used:

```
...  
push R30          ; push R30 value onto stack  
call Normalize    ; Normalize the value  
pop R30           ; put result into R30  
...
```



10. (50 pts). Build an assembly language subroutine named `ExportR1Bits` that copy bits from R1 into other registers. Your subroutine will copy bits 7, 5, and 0 of R1 into R31, and will copy bits 6, 3, and 1 of R1 into R30, and will copy bits 4 and 2 of R1 into R29. The bits found in R1 must occupy the same places in the destination register as they did in R1. A diagram of where each bit in R1 must go is shown below:

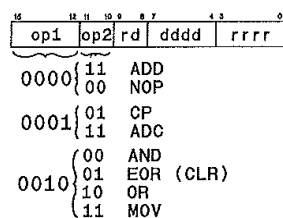
**R1** [ R31 R30 R31 R29 R30 R29 R30 R31 ]

For example, suppose that R1 contains the value 11111111 and that R29, R30, and R31 all contain 0 when your function is called. When your function finishes, R31 will contain 10100001, R30 will contain 01001010, and R29 will contain 00010100. As another example, suppose that R1 contains the value 10011101 and that R29, R30, and R31 all contain 0 when your function is called. When your function finishes R31 will contain the value 10000001, R30 will contain the value 00001000, and R29 will contain the value 00010100. Your subroutine must use only instructions that are contained in the CS150 AVR instruction subset. The next page is blank if you need more space to work.

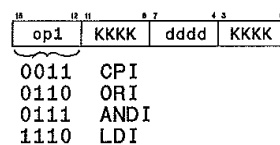
This page intentionally left blank.

## AVR Instruction Subset

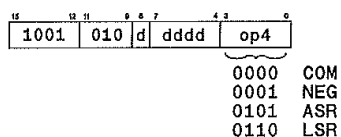
### ALU Instructions



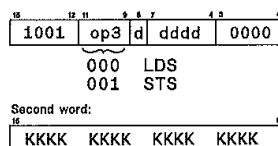
### Immediate Instructions



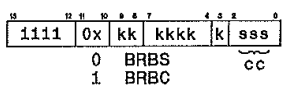
### Unary Logical Instructions



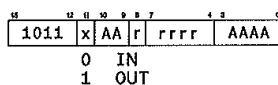
### Load/Store Instructions



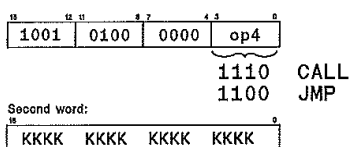
### Branch Instructions



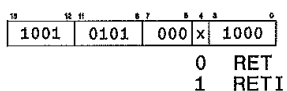
### Input/Output Instructions



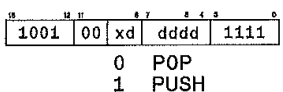
### Call/Jump Instructions



### Return Instructions



### Stack Instructions



### Relative Jump Instructions

