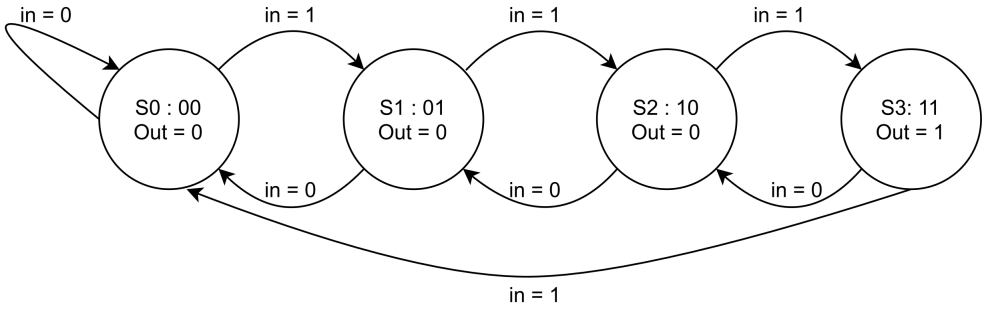
EECS 151/251A Homework 4

Due 11:59pm Monday, October 4th, 2021

Submit your answers on Gradscope.

1 More fun with FSMs

(a) Consider the following state transition diagram of a given FSM. Fill in the missing values in the following state transition table (each letter should be filled in with a one or two digit binary number). Note that CS is short for current state, and NS is short for next state. The state encodings are labeled in each state bubble (eg. S0: 00).



Is this a Moore or a Mealy FSM? [ ]

in	CS	NS	Out
0	00	**a**	**i**
1	00	**b**	**j**
0	01	**c**	**k**
1	01	**d**	**l**
0	10	**e**	**m**
1	10	**f**	**n**
0	11	**g**	**o**
1	11	**h**	**p**

(b) Below is a Verilog implemenetation of this FSM. Fill in the blanks such that it operates according to the state transition diagram.

```
module fsm (
    input clk,
    input reset,
    input in,
    output reg out
);

parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
parameter S3 = 2'b11;

reg [1:0] current_state;
reg [1:0] next_state;

always @(*) begin
    out = _____a_____;
    case (state)
        S0 : begin
            if (in) next_state = S1;
            else next_state = _____b_____;
           end
        S1 : begin
            if (in) next_state = _____c_____;
            else next_state = S0;
           end
        S2 : begin
            if (in) next_state = S3;
            else next_state = _____d_____;
           end
        S3 : begin
            out = _____e_____;
            if (in) next_state = S0;
            else next_state = S2;
           end
        default: begin
            next_state = S0;
        end
    endcase
end

always @(posedge clk) begin
    if (reset) begin
        current_state <= S0;
```

```
45      else begin
46          current_state <= _____f_____;
47      end
48  end
49
50  endmodule
```

(c) Below are a set of K-maps that may be used to optimize the next state and output logic for the FSM. Fill in the missing values (denoted by letters). Note that next state, NS, in the state transition table is made up of NS={NS1,NS0}. Then, minimize the logic from the K-maps and write the minimized expressions below.

NS1:

	00	01	11	10
0	**a**	0	1	0
1	0	**b**	**c**	**d**

NS0:

	00	01	11	10
0	0	**b**	0	**d**
1	**a**	0	**c**	1

Out:

	00	01	11	10
0	0	0	1	0
1	0	0	1	0

NS1: NS0: Out:

(d) You are now allowed to change the state parameter encodings for this FSM. What would you assign the state encodings to be such that the next state logic can be minimized further? Show your work with a K-map or otherwise. Write the state parameter assignments and the minimized expressions for NS1, NS0, and Out below. Note that the state assignments should remain as two bits.

S0: S1: S2: S3:

NS1: NS0: Out:

2 FSM Complexity - 251 Only, Optional for 151

(a) For an 8 state FSM, how many unique state encoding assignments exist, assuming that the state register holds 8 bits? How many are there if the states are 1-hot encoded?

Number of unique encodings: ☐

Number of unique encodings if 1-hot encoded: ☐

(b) How many unique FSMs of 1 inputs, 1 output, and 3 states exist?

Number of unique FSMs: ☐

3 RISCV Intro

In addition to the RISCV lectures, we recommend looking through the RISCV ISA Spec directly. Chapter 2 focuses on the RV32I Base Integer Instruction Set, which may be particularly useful.

(a) Write the opcode for each of the following RISCV instructions.

(i) `sra`

Opcode:

(ii) `andi`

Opcode:

(iii) `beq`

Opcode:

(b) Write the five classic stages of the RISCV datapath (abbreviations are ok).

Stage 1:

Stage 2:

Stage 3:

Stage 4:

Stage 5:

(c) Write down the values of the specified registers after the following programs have run. Show your work by annotating what happens/changes after each instruction. Note that some instructions are pseudo-instructions, such as `li` for load immediate. Refer to Table 25.2 in the RISC-V spec for a list of pseudo-instructions and their base implementations. You may assume that all memory locations are initialized to 0.

(i)

```
li x0, 10
li x1, 52
li x2, 12
add x2, x1, x2
sub x2, x2, x0
ori x1, x2, 1
```

x0 = [ ], x1 = [ ], x2 = [ ]

(ii)

```
li x1, 0xcafe
li x2, 0xf
li x3, 0xe80d
li x4, 0x28c
sh x1, 2(x0)
sll x2, x2, x4
sb x3, 0(x0)
lw x3, 0(x0)
or x2, x3, x2
```

x1 = [ ], x2 = [ ], x3 = [ ], x4 = [ ]

(iii)

```
      li x1, 9
      li x2, 12
      li x4, 0x8068
more: sltu x3, x1, x2
      add x1, x1, x3
      sra x4, x4, x3
      bge x1, x2, done
      j more
done: nop
```

x1 = [ ], x2 = [ ], x3 = [ ], x4 = [ ]

4 RISCV Pipeline

(a) Label the following statements as True or False.

(i) The ALU is a fully combinational block.

(ii) Structural hazards can always be solved by adding more hardware.

(iii) Stalls from data hazards can never be reduced by adding more hardware.

(b) We want to implement a basic 32b single-read, single-write register file with 8 registers. The `x0` register should retain the same properties as it does in RV32I. The register file should only be writable when `wrEn` is high. Fill in the blanks in the following Verilog for the register file.

```
module regfile (
    input clk,
    input wrEn,
    input [2:0] addrA,
    input [2:0] addrD,
    input [31:0] dataD,
    output reg [31:0] dataA,
);

reg [ _____a_____:0] rf [0:_____b_____];

always @(posedge clk) begin
    if (addrD == 0) begin
        rf[addrD] <= _____c_____;
    end else if (wrEn) begin
        rf[addrD] <= _____d_____;
    end
end

always @(*) begin
    dataA = _____e_____;
end
endmodule
```