

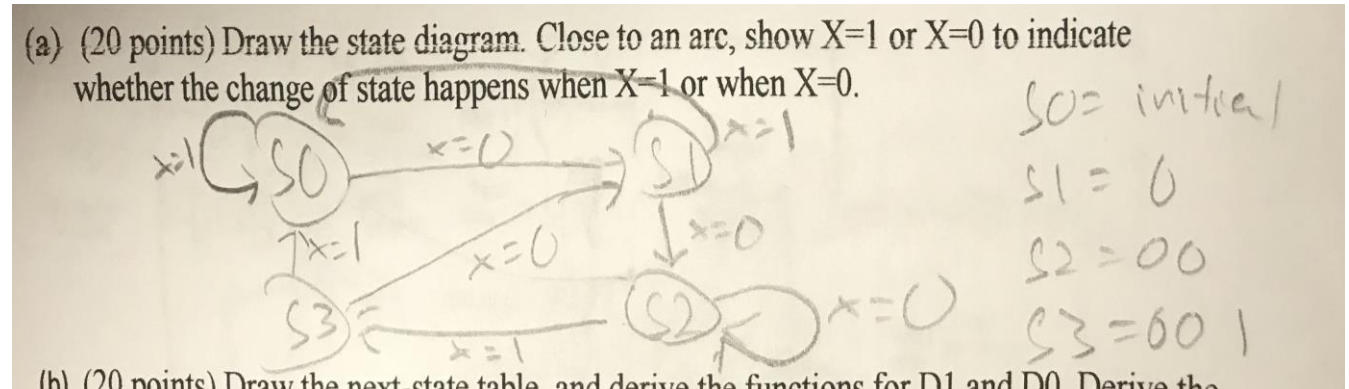
Assignment #6 – Digital Logic Design II – Sequential Logic

CDA 3100, Computer Organization I

Due: 12/8/18, please submit an electronic version on Canvas.

TROY ALLEN

Problem 1 (50 points) Design a circuit that has two inputs, clk and X, and produces one output O. X may change every clock cycle, and the change happens at the falling edge. The circuit samples the input at every rising edge of the clock. If the input is 1, consider as read a 1, else read a 0. O is 1 (for one clock cycle, from positive edge to positive edge) if the last three bits read are 001, where 1 is the most recent bit.



(b) (20 points) Draw the next-state table, and derive the functions for D1 and D0. Derive the output function.

S0 = 00, S1 = 01, S2 = 10, S3 = 11

$D1 = (Q1 \& \sim Q0) \mid (\sim Q1 \& Q0 \& \sim X)$ $D0 = (\sim Q1 \& \sim Q0 \& \sim X) \mid (Q1 \& \sim Q0 \& X) \mid (Q1 \& Q0 \& \sim X)$ $O = Q1 \& Q2$

Q1	Q0	X	D1	D0
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	1
1	1	1	0	0

(c) (10 points) Complete the Verilog module for this circuit. Use the following code as a template.

```

module HW6P2 (clk, X, O);
    input clk, X;
    output O;

    wire D1, D0, Q1, Q0, Q1bar, Q0bar;

    assign D0 = (~Q1&~Q0&~X) | (Q1&~Q0&X) | (Q1&Q0&~X);
    Dff1 C0 (D0, clk, Q0, Q0bar);

    assign D1 = (Q1&~Q0) | (~Q1&Q0&~X);

    Dff1 C1 (D1, clk, Q1, Q1bar);

    assign O = Q1&Q2;

endmodule

```

Problem 2 (50 points) Design a MIPS processor supporting **only** the **R-type** and the **AAA** **rs, rt, a** instruction. The AAA **rs, rt, a** is an **I-type** instruction that does the following:

If rs equal to a (after sign extension), the next PC will be the data memory value at address rt ; otherwise, rt will be the data memory value at address rt .

For this problem, assume the `opcode` of R-type is 000000 and the `opcode` of AAA is 100000.

- (30 points) Please design the datapath of this processor, add 2-1 MUX when necessary. Please show clearly the indices of the bits near the wires. Please give clear index to the 2-1 MUX starting from 1.
- (20 points) Please determine the values of `RegWrite` and the control signals of the 2-1 MUX by filling in the table. In case of “don’t care”, write down “X.” Assume other control signals have been generated correctly.
- (Extra 10 points) Please write down the logic functions of `RegWrite` and the control signals of the 2-1 MUX. Certain bits in the instruction can be denoted, for example, as `instruct[31]`. The ALU zero output can be denoted simply as “zero.” **When deriving the logic functions, if the value of a signal is “don’t care” under a certain condition, assume it should be 0.**

The diagram illustrates a processor architecture with the following components and connections:

- PC (Program Counter):** Receives a 4-bit value from a multiplexer and outputs a 32-bit address to **Instruction memory**.
- Instruction memory:** Provides a 32-bit **Instruction [31:0]** to the **RegWrite** block.
- RegWrite:** A central control block that manages register operations. It contains:
 - Read register 1:** Outputs **Read data 1** (32-bit) to the **ALU**.
 - Read register 2:** Outputs **Read data 2** (32-bit) to the **ALU**.
 - Write register:** Receives a 5-bit register index from the instruction.
 - Write data:** Receives a 32-bit data value from the **ALU result** and outputs it to **Data memory**.
- Registers:** A set of 32 registers. The diagram shows three specific registers:
 - Register 1 (index 1) containing the value 25.
 - Register 2 (index 2) containing the value 16.
 - Register 3 (index 3) containing the value 17.
- ALU (Arithmetic Logic Unit):** Takes two 32-bit inputs and performs an operation. It outputs a 32-bit **ALU result** and a **Zero** flag.
 - One input is **Read data 1** from Register 1.
 - The other input is the result of a 32-bit addition: $[25:16] + [17:1]$.
- Add:** A 4-bit adder that takes a 4-bit constant and a 4-bit input from the **PC** to produce the next PC value.
- Data memory:** Receives a 32-bit **Address** and **Write data** from the **RegWrite** block. It outputs a 32-bit **Read data** back to the **RegWrite** block.
- Handwritten annotations:**
 - A multiplexer at the top left selects between a 4-bit constant and the PC output.
 - At the bottom, a 32-bit value $[15:0]$ is shown, which is the result of adding 16 to 17 (32).

c)

```
MUXCtrl1 = Instruct[31] & zero
```

```
MUXCtrl2 = Instruct[31] & zero
```

MUXCtrl3 = Instruct[31] | zero

```
MUXCtrl4 = ~Instruct[31] & zero
```