CSE 2441 **DIGITAL LOGIC DESIGN** **FALL 2021**

TERM DESIGN PROJECT ADDENDUM

Program Load: Before testing your TRISC design and realization, you must load the following program in RAM.

0: 0F, *1*: 61, *2*: 62, *3*: 1E, *4*: 74, *5*: 0E, *6*: 66, *7*: 89, *8*: 88, *9*: 69, *A*: 2E, *B*: 7B, *C*: 6C, *D*: 88, *E:* EE, *F*: FF

Loading the program requires you to incorporate a triscRAM loader in your project as illustrated in Figure 1. TRISC organization is shown in Figure 2. Note that the RAMs shown in the two figures are the same.

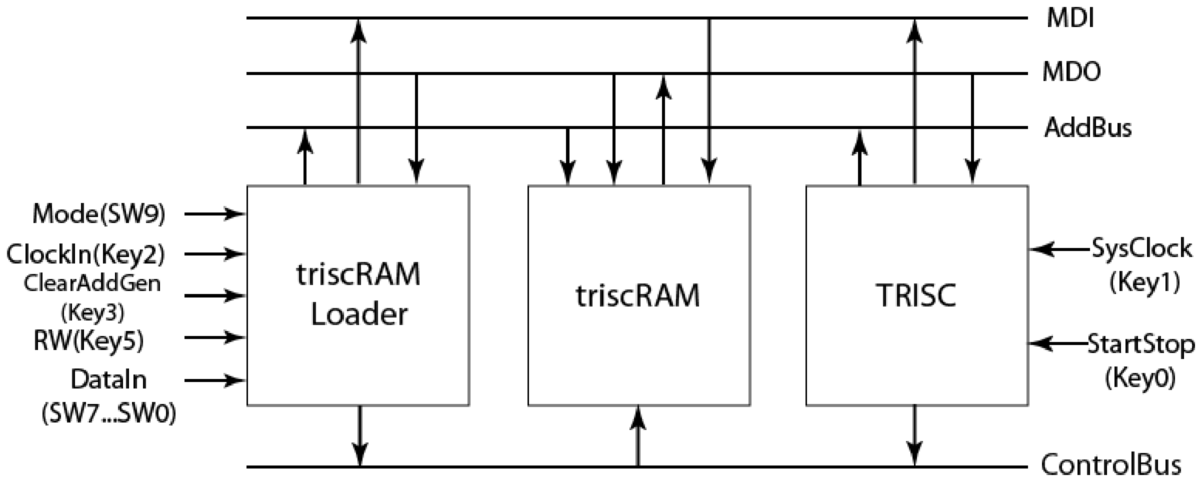

Figure 1 – triscRAM Loader (TRISC mode: SW9=0, Loader mode: SW9=1)

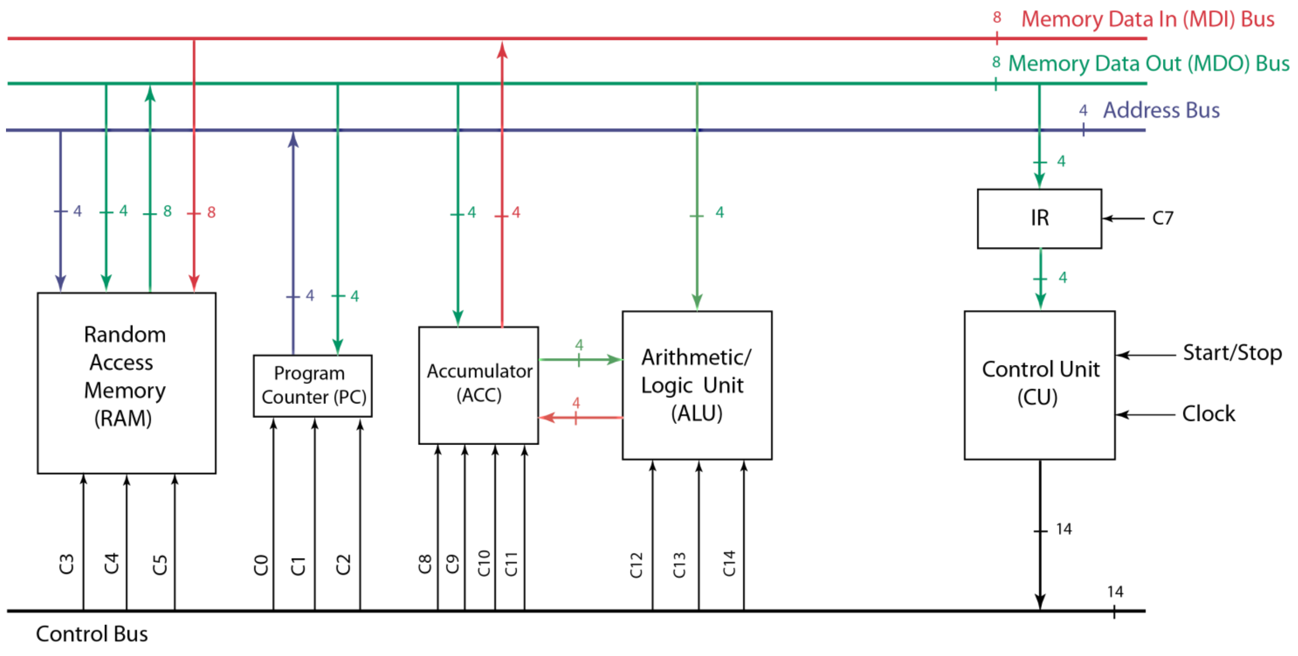

Figure 2 – TRISC for INC, CLR, LDA, STA, ADD, and JMP.

The following code must be included in your TRISC code in order incorporate the loader.

```
------------------------------------------------------------------------------------------------------------------------------------------
input Mode, ClockIn, ClearAddGen, RW,          //Mode = SW9, ClockIn = Key2, ClearAddGen = Key3, RW = Key5
input [7:0]    DataIn,                         //DataIn = {SW7,SW6,SW5,SW4,SW3,SW2,SW1,SW0}

wire [3:0]      AddIn, AddGen;
wire RAMin, RAMwrite, toggle;
wire [7:0]      RAMdata;

assign AddIn = Mode == 1'b0 ? RAMadd : AddGen;
assign RAMin = Mode == 1'b0 ? SysClock*c4 : ClockIn;
assign RAMdata = Mode == 1'b0 ? MDI : DataIn;
assign RAMwrite = Mode == 1'b0 ? c5 : ~RW;

OnOffToggle DivideX2
(
        .OnOff(ClockIn) ,                      // input  OnOff_sig
        .IN(1'b1) ,                            // input  IN_sig
        .OUT(toggle)                           // output  OUT_sig
);
BinUp AddressGen
(
        .inc(toggle) ,                         // input  inc_sig
        .clear(ClearAddGen) ,                  // input  clear_sig
        .load(1'b1) ,                          // input  load_sig
        .D(4'b0) ,                             // input  [N-1:0] D_sig
        .Q(AddGen)                             // output [N-1:0] Q_sig
);
triscRAM        RAM
(
        .address ( AddIn ),
        .clock ( ~RAMin ),
        .data ( RAMdata ),
        .wren ( RAMwrite ),
        .q ( MDO )
);
------------------------------------------------------------------------------------------------------------------------------------------
```

Load Procedure: Perform the following steps to load the program after you have incorporated the loader code.

1. Set Mode (SW9) = 1 to load RAM, = 0 to operate TRISC
2. Press and release ClearAddGen (Key3) to clear RAM address
3. Enter data on SW7…SW0
4. Press and Hold RW (Key5) to enable write to RAM
5. Press and release ClockIn (Key2) to write (Key5 pressed) or read (Key5 released) RAM

Note that the above is similar to the RAM read and write procedures you used in Lab 11.

Program Execution: Perform the following steps to execute the program you entered above.

1. Place SW9 in the off position (SW9=0). This disables the triscRAM loader.
2. Push and release Key0: This initializes the Program Counter (PC) and other registers.
3. Repeatedly push and release Key 1. This clocks the Control Unit (CU) which generates the sequence of control signals to executed the sequence of instructions contained in the program. Several clock signals are necessary to completely execute the program.

Program Results: The accumulator (ACC) should contain the following values after each instruction executes.

0: F, *1*: 0, *2*: 1, *3*: 1, *4*: 0, *5*: 1, *6*: 2, *7*: 2, *8*: 1, *9*: 3, *A*: 4, *B*: 0, *C*: 1, *D*: 1, *E*: NA, *F*: NA

Note that the instructions do not execute in order due to the jump (JMP) instructions. The program should end when the JMP instruction at memory location 8 is executed. How will that instruction behave?

Control Signal Definitions (Active High)

C0 – *Clear* Program Counter (PC)

C1 – *Load* PC

C2 – *Increment* PC

C3 – *Select* Memory Address Register (MAR) source (C3=1: PC, C3=0: MDO)

C4 – *Execute* RAM READ/WRITE cycle

C5 – *Enable* RAM WRITE cycle

C7 – *Load* Instruction Register (IR)

C8 – *Clear* Accumulator (ACC)

C9 – *Increment* ACC

C10 – *Select* ACC source (C10=0: ALU, C10=1: MDO)

C11 – *Load* ACC

C12, C13 – Arithmetic Logic Unit (ALU) Function Code (00: ADD, 10: SUB, 01: AND, 11: XOR)

C14 – *Load* Buffer Register (BR)

C15 – *Load* Flag Register (FR)