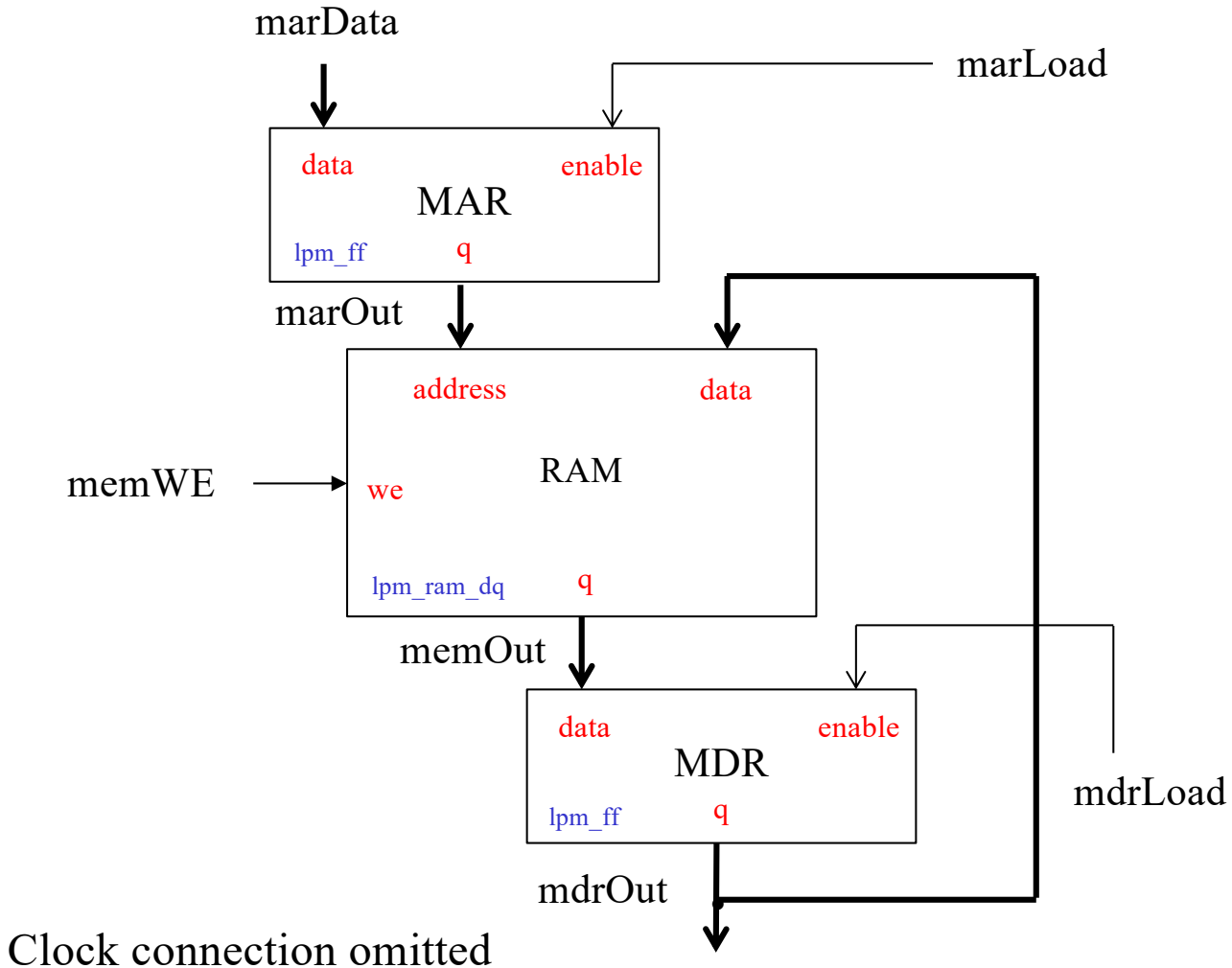
Experiment 7

- Microprogrammed CPU Design
- Implement with VHDL
- Structural approach using lpm components

Part 2 – Week 2 Lab

- Testing the RAM
  - Implement code to read and write data from the RAM using the sequencer.
- Use lpm_ram_dq for the RAM
- lpm_ram_dq has separate address and data ports
- Address of RAM is connected to output of MAR
- Data of RAM is connected to output of MDR
- Output of RAM is connected to the input of MDR

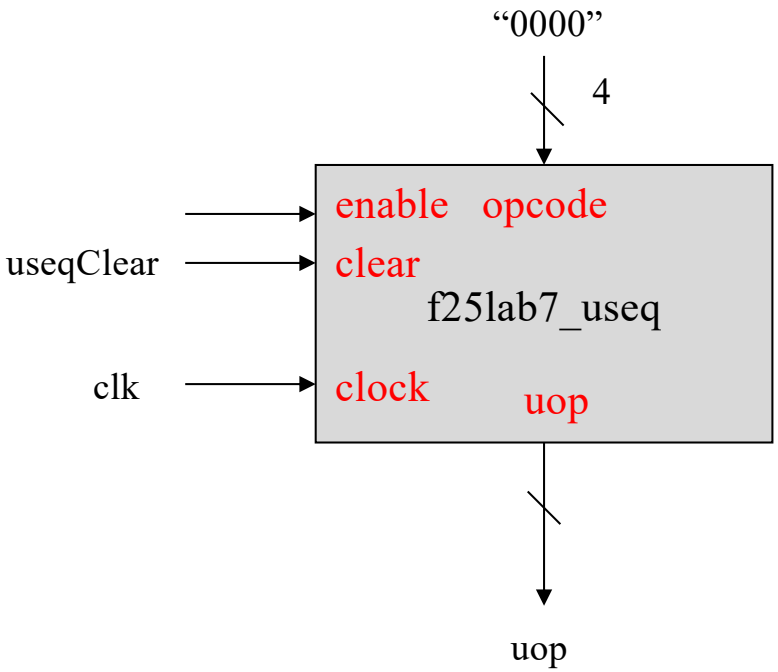
RAM Connection



MAR ⟵ address
MDR ⟵M[MAR]
M[MAR] ⟵MDR

Memory will always output the content of the location indicated by the address port

Part 2 - Microsequencer



Part 2

Create a project for lab7 with following code:

```
library ieee; use ieee.std_logic_1164.all;
library lpm; use lpm.lpm_components.all;
entity lab7Part2 is
port (clk, clear: in std_logic;
      useqEnOut: out std_logic;
      marData: in std_logic_vector(7 downto 0); -- connected to switches, input to MAR
      marSegHi, marSegLo, mdrSegHi, mdrSegLo: std_logic_vector(0 to 6);  -- seven segment display
      ctrlSignals: out std_logic_vector(0 to 7)); -- control signals
end lab7Part2;
architecture structural of lab7 is
-- add component declaration of f25lab7_useq
-- add component declaration for seven-segment decoder

  signal useqEnable, useqClear: std_logic;
  signal uop: std_logic_vector(0 to 7);
  signal marLoad, mdrLoad, memWE: std_logic;
begin
  delay: lpm_counter generic map(lpm_width=>2) port map(clock=>clk, cout=>useqEnable);
  useqClear <= clear;
  useqEnOut <= useqEnable;
  labelG: for i in 0 to 7 generate
    ctrlSignals(i) <= uop(i) and useqEnable;
  end generate;
  marLoad <= uop(0) and useqEnable;
  mdrLoad <= uop(1) and useqEnable;
  memWE <= uop(2) and useqEnable;

  -- add port map statement of f25lab7_useq, use the uROm Test file in the next slide
  -- add mar, mdr registers and ram, seven-segment displays

end structural;
```

Part 2 – uROM Test File

```
-- uROM Test File Part 2
-- 8-bit control signals

WIDTH=17;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
  [0..FF]: 00000000000000000;
--      uop
--    01234567M01234567
  00: 00000000000000001; -- control signal (cs) = 00000000, next address (na) = 0x01
  01: 10000001000000010; -- cs = 10000001, na = 0x02, Load MAR from sw, MAR = 0x54
  02: 01000010000000011; -- cs = 01000010, na = 0x03, MDR <- M[MAR], MDR = 0xFC
  03: 10000011000000100; -- cs = 10000011, na = 0x04, Load MAR from sw, MAR = 0x55
  04: 01000100000000101; -- cs = 01000100, na = 0x05, MDR <- M[MAR], MDR = 0xDD
  05: 10000101000000110; -- cs = 10000101, na = 0x06, Load MAR from sw, MAR = 0x56
  06: 00100110000000111; -- cs = 00100110, na = 0x07, M[MAR] <- MDR
  07: 10000111000001000; -- cs = 10000111, na = 0x08, Load MAR from sw, MAR = 0X54
  08: 01001000000001001; -- cs = 01001000, na = 0x09, MDR <- M[MAR], MDR = 0xFC
  09: 10001001000001010; -- cs = 10001001, na = 0x0A, Load MAR from sw, MAR = 0x56
  0A: 01001010000000000; -- cs = 01001010, na = 0x00, MDR <- M[MAR], MDR = 0xDD
END;
```

control signal

mux select

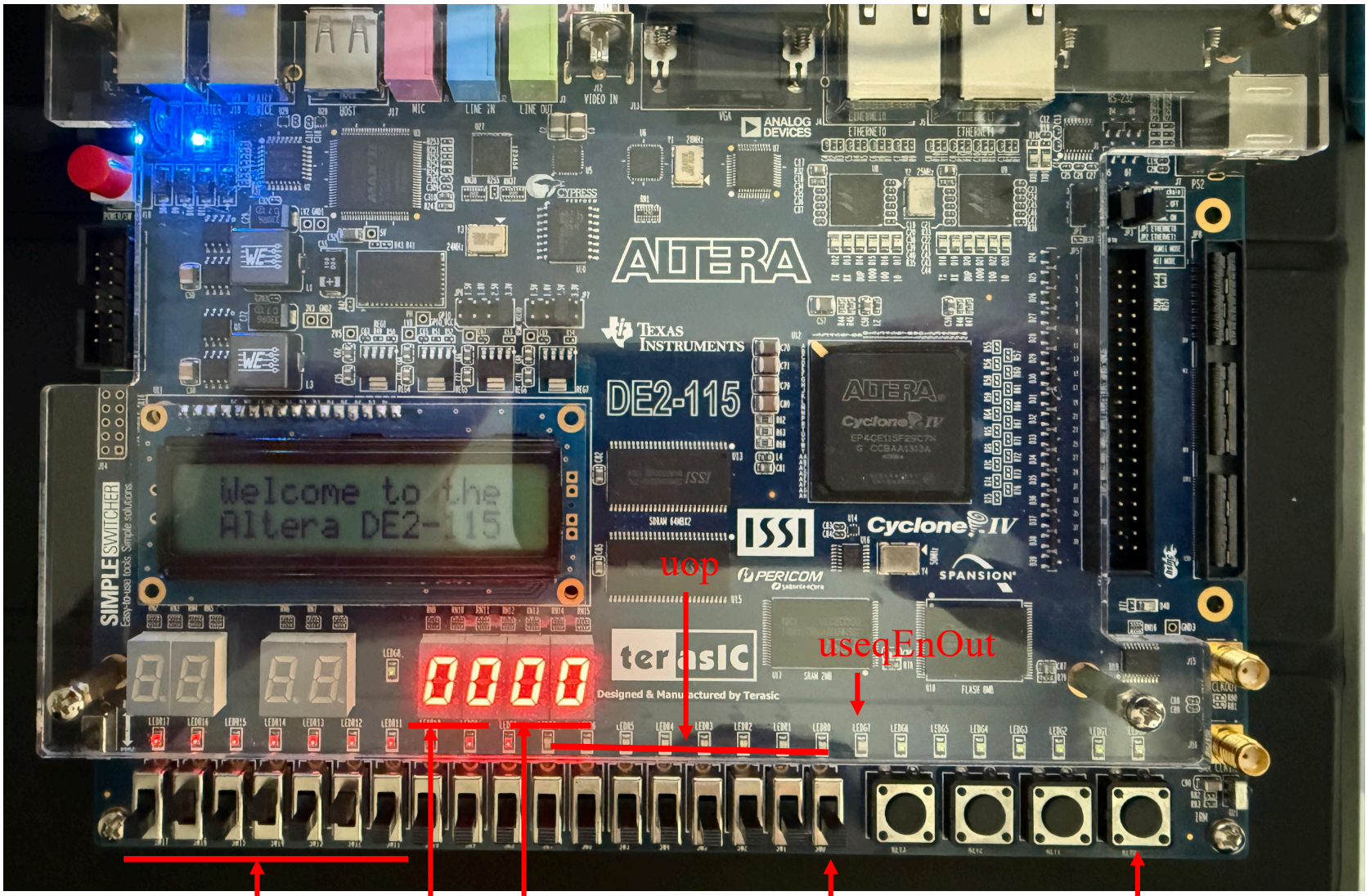
next address

Part 2 – RAM File

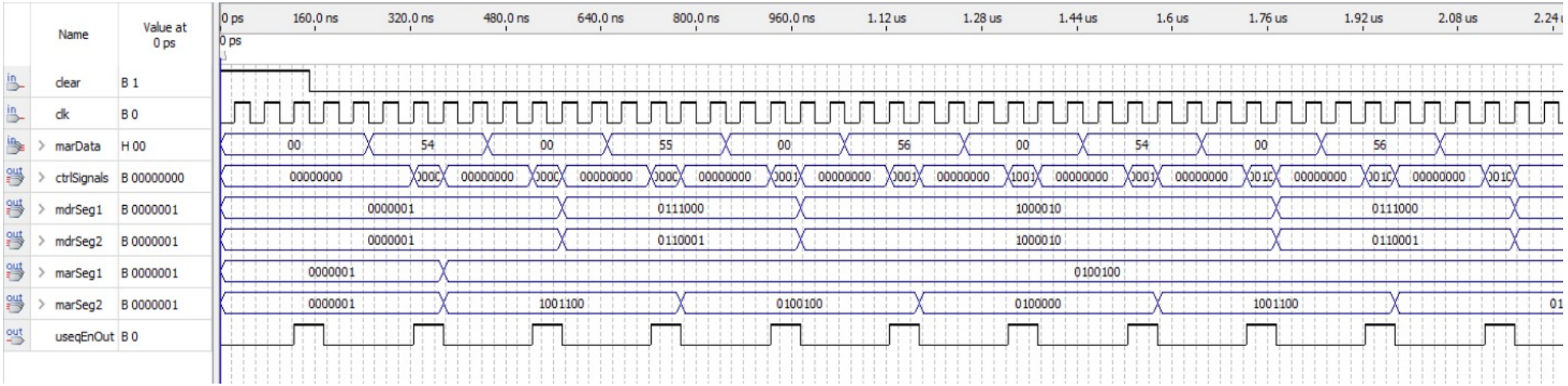
```
-- RAM Test File Part 2
--

WIDTH=8;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
  [0..FF]: 0;
  54: FC;
  55: DD;
  56: AA;
END;
```


uop
useqEnOut
MAR data
MDR
MAR
clear
clk

- Assign
  - clk = key3, clear = sw0, useqEnOut = ledg7
  - ctrlSignals(0) = ledr7, ctrlSignals(1) = ledr6, …, ctrlSignals(7) = ledr0
  - marData(7-0) = sw17-sw10
  - marSegLo = hex0, marSegHi = hex1
  - mdrSegLo = hex2, mdrSegHi = hex3
- You should see



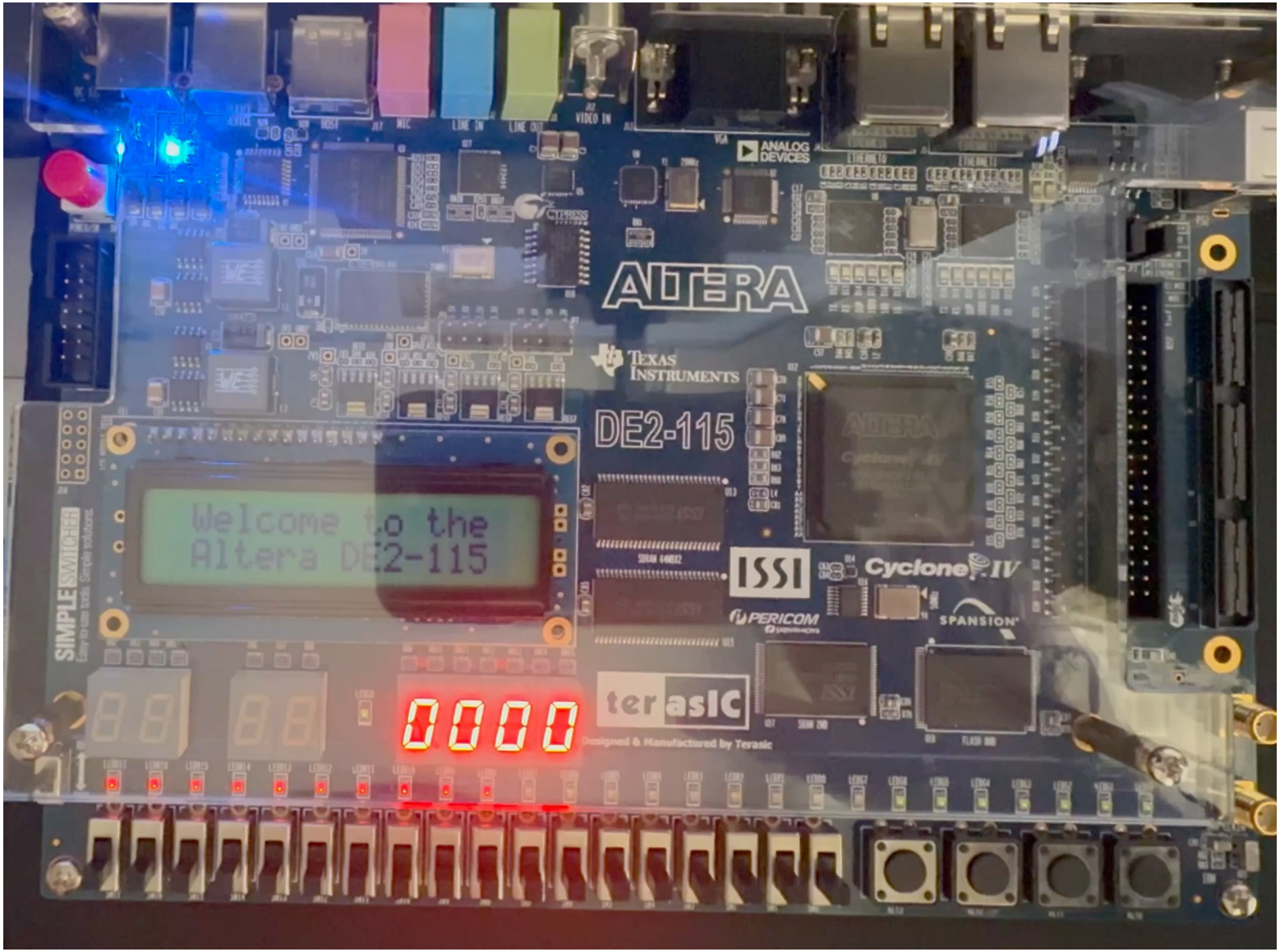
Part 2 – uROM Test Results

- uop(0) = marLoad, uop(1) = drLoad, uop(2) = memWE
- uop(0-2) = “100” – load switch value (marData) into MAI
- uop(0-2) = ”010” – MDR ⟵ M[MAR], memory read
- uop(0-2) = ”001” – M[MAR] ⟵ MDR, memory write

```
--      uop
--    01234567M01234567
  00: 00000000000000001; -- control signal (cs) = 000
  01: 10000001000000010; -- cs = 10000001, na = 0x02,
  02: 01000010000000011; -- cs = 01000010, na = 0x03,
  03: 10000011000000100; -- cs = 10000011, na = 0x04,
  04: 01000100000000101; -- cs = 01000100, na = 0x05,
  05: 10000101000000110; -- cs = 10000101, na = 0x06,
  06: 00100110000000111; -- cs = 00100110, na = 0x07,
  07: 10000111000001000; -- cs = 10000111, na = 0x08,
  08: 01001000000001001; -- cs = 01001000, na = 0x09,
  09: 10001001000001010; -- cs = 10001001, na = 0x0A,
  0A: 01001010000000000; -- cs = 01001010, na = 0x00,
```

The operation of the uROM test file:

uROM address	Micro operations	With RAM test file and simulation file	
01	MAR ⟵ switch values (0x54)	MAR = 0x54	
02	MDR ⟵ M[MAR]	MDR = 0xFC	Read data from RAM location 0x54
03	MAR ⟵ switch values (0x55)	MAR = 0x55	
04	MDR ⟵ M[MAR]	MDR = 0xDD	Read data from RAM location 0x55
05	MAR ⟵ switch values (0x56)	MAR = 0x56	
06	M[MAR] ⟵ MDR	MDR = 0xDD	Write 0xDD into RAM location 0x56
07	MAR ⟵ switch values (0x54)	MAR = 0x54	Read data from RAM location 0x54
08	M[MAR] ⟵ MDR	MDR = 0xFC	
09	MAR ⟵ switch values (0x56)	MAR = 0x56	Read data from RAM location 0x56
0A	M[MAR] ⟵ MDR	MDR = 0xDD	


ALTERA
Texas Instruments
DE2-115
Welcome to the
Altera DE2-115
ISSI
Cyclone IV
PERICOM
SPANSION
terasIC
0000

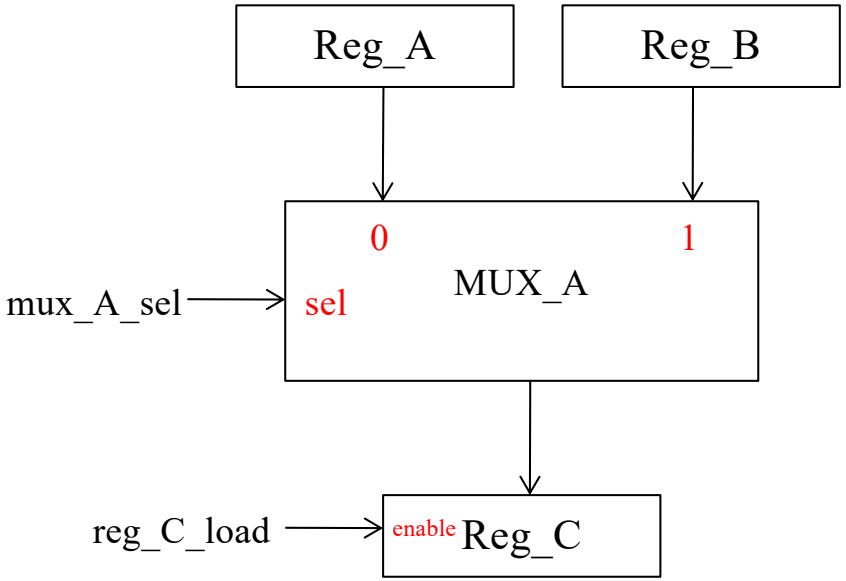
Assignments

- Week 2. Testing the RAM
  - Implement code to read and write data from the RAM using the sequencer.
- Develop a block diagram for the CPU design. Submit your design at the end of the lab. You can use tools like draw.io or visio

Block Diagram Design

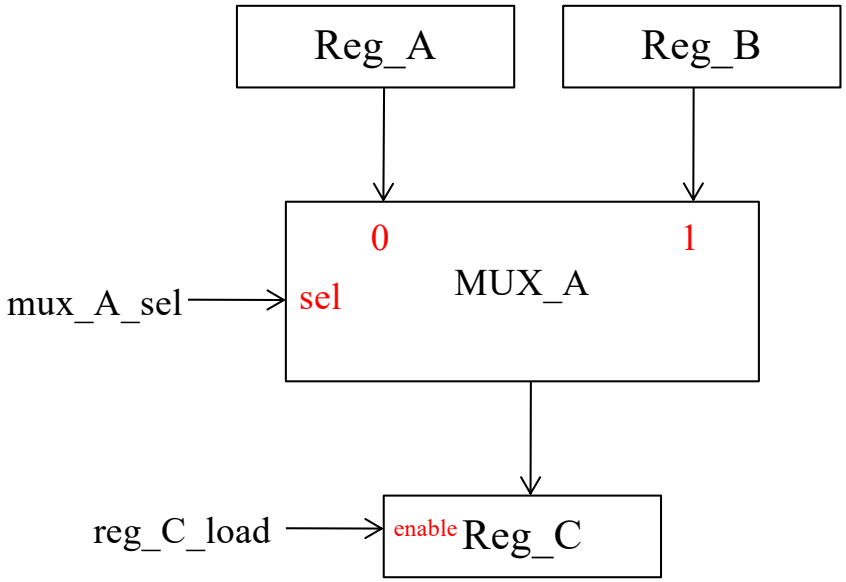
- Look at the set of RTL statements and group them according to destination

1. Reg_C ⟵ Reg_A
2. Reg_C ⟵ Reg_B



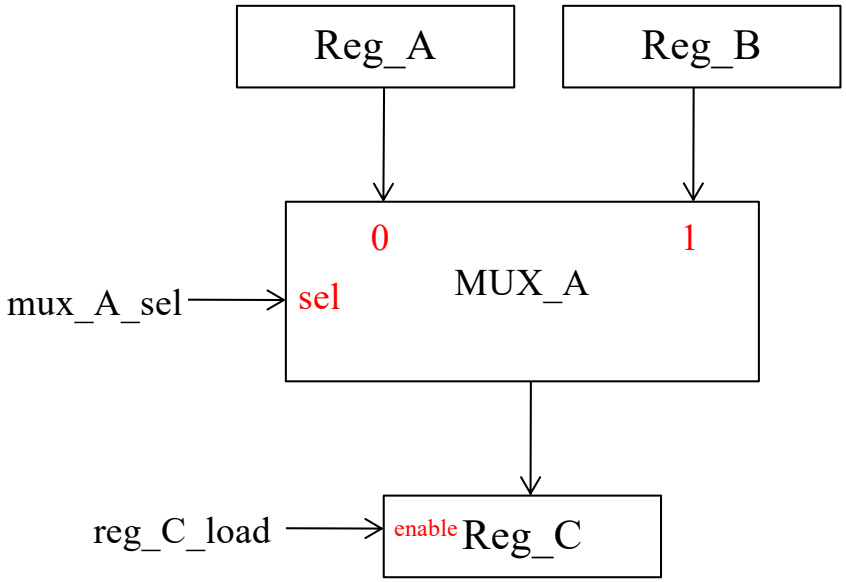
Control Signals

- mux_A_sel
- reg_C_load



Microcode

- Reg_C ⟵ Reg_A
  - mux_A_sel = 0
  - reg_C_load = 1
- Reg_C ⟵ Reg_B
  - mux_A_sel = 1
  - reg_C_load = 1

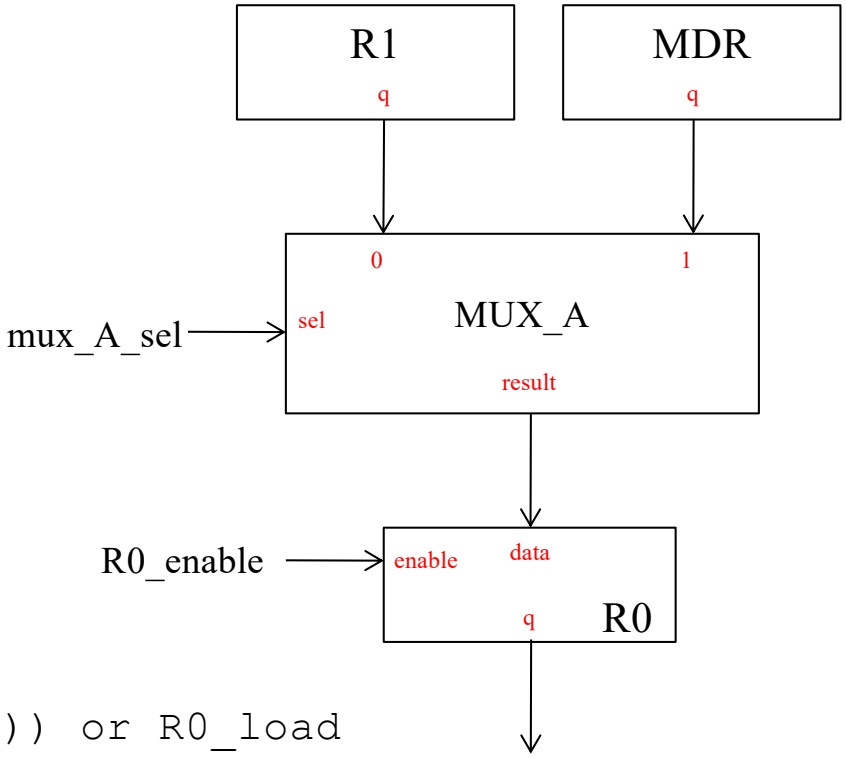


R0 ⟵ MDR

R0 ⟵ R1

R0 ⟵ R1 if IR(0) = 0

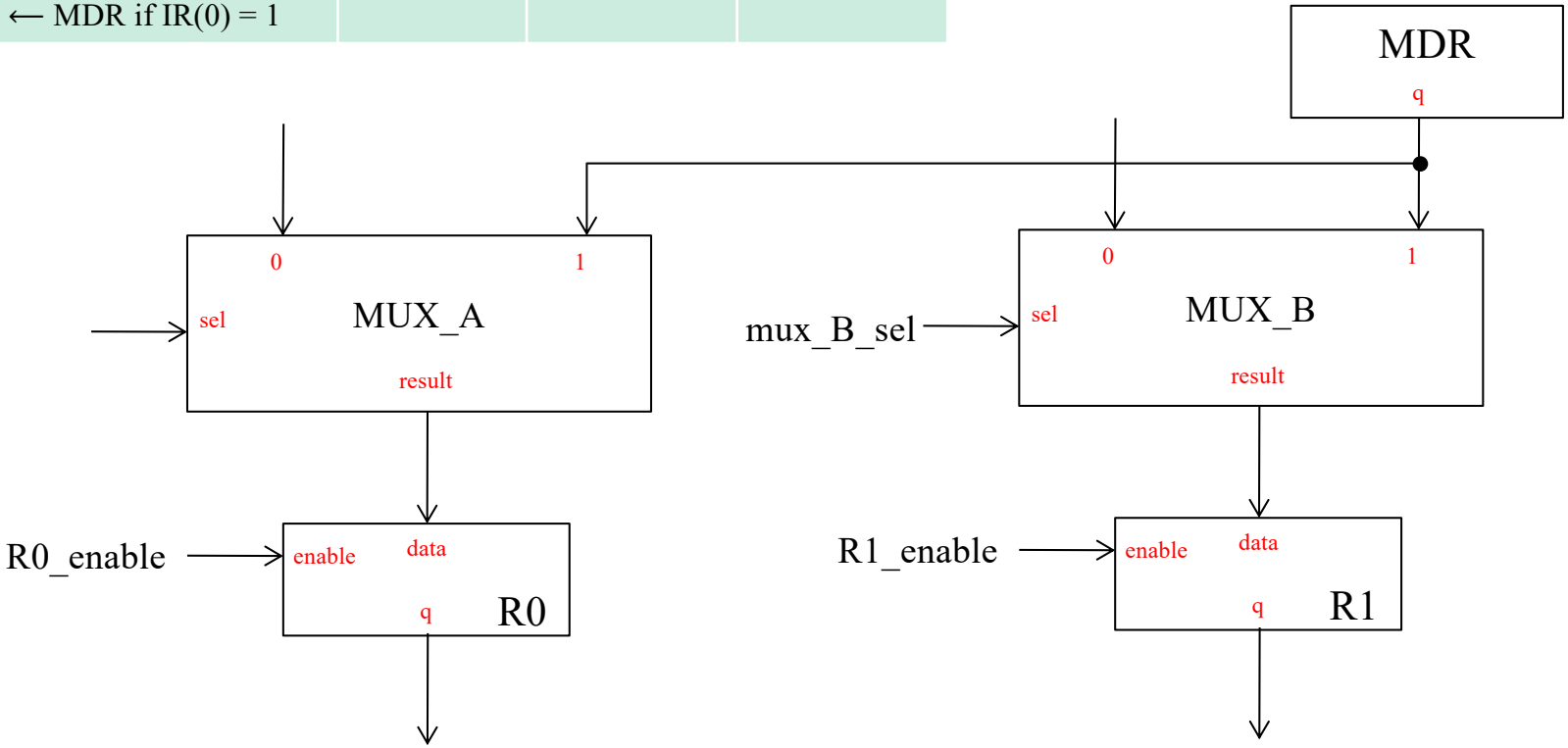
	mux_A_sel	R0_cond	R0_load
R0 ⟵ MDR	1	0	1
R0 ⟵ R1	0	0	1
R0 ⟵ R1 if IR(0) = 0	0	1	0




```
R0_enable = (R0_cond and not IR(0)) or R0_load
```

R0 ← MDR if IR(0) = 0, R1 ← MDR if IR(0) = 1

	R01_cond	mux_A_sel	mux_A_sel
R0 ← MDR if IR(0) = 0, R1 ← MDR if IR(0) = 1	1	1	1

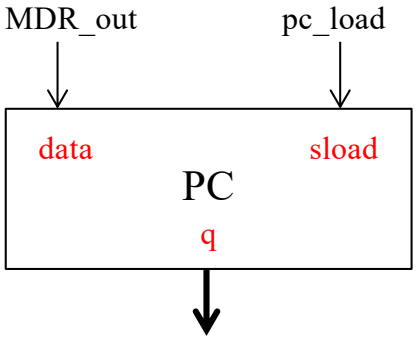



```
R0_enable = (R01_cond and not IR(0)) or …
R1_enable = (R01_cond and IR(0)) or …
```

Conditional Jump – JUMPZ X

1. To make a “Jump”, you need to load the jump address into PC (by setting the sload of PC to ‘1’)
2. For the conditional jump, the sload of PC should be conditioned on Z
3. The uROM should have a JUMPZ and JUMP control signal to indicate which instruction is active

```
pc_load <= (JUMPZ and Z) or JUMP
```



JUMP X

JUMP1: MAR ⟵ PC
JUMP2: MDR ⟵ M[MAR]
JUMP3: PC ⟵ MDR

JUMPZ X

JUMPZ1: MAR ⟵ PC
JUMPZ2: MDR ⟵ M[MAR], PC ⟵ PC+1
JUMPZ3: PC ⟵ MDR if Z = 1