Department of Electrical and Computer Engineering
Queen's University

# ELEC 374 Digital Systems Engineering
# Laboratory Project

Winter 2023

## Designing a Simple RISC Computer (Mini SRC): Phase 2

---

## 1. Objectives

The purpose of this project is to design, simulate, implement, and verify a simple RISC Computer (Mini SRC), consisting of a simple RISC processor, memory, and I/O. Phase 2 of this project consists of the design and Functional Simulation of the rest of the Mini SRC datapath. This includes the circuits associated with the "Select and Encode" logic, "Memory Subsystem", "CON FF" logic, and "Input/Output" ports, as well as load/store instructions, branch and jump instructions, and immediate instructions. You will add the necessary logic circuits to the Datapath circuitry built in Phase 1. You may want to read Section 4.4 of the Lab Reader to have a better understanding of this phase..

You are to simulate the Load and Store instructions *ld, ldi*, *and st,* to test the "Memory Subsystem" and the "Select and Encode" logic. Your simulations will also include the *addi, andi,* and *ori* ALU instructions, the conditional Branch Instructions *brzr*, *brnz*, *brmi,* and *brpl* in order to test the "CON FF" logic, the *jr* and *jal* instructions, the *mfhi* and *mflo* instructions as well as the *in* and *out* instructions to test the "Input/Output" ports. Design input can be done using an all HDL, or a mixed schematic/HDL approach. Testing will be done by Functional Simulation.

## 2. Preliminaries

### 2.1 The Memory Subsystem

As shown in Figure 1, the "Memory Subsystem" includes the Memory Address Register (MAR), the Memory Data Register (MDR), and the RAM Memory Component.

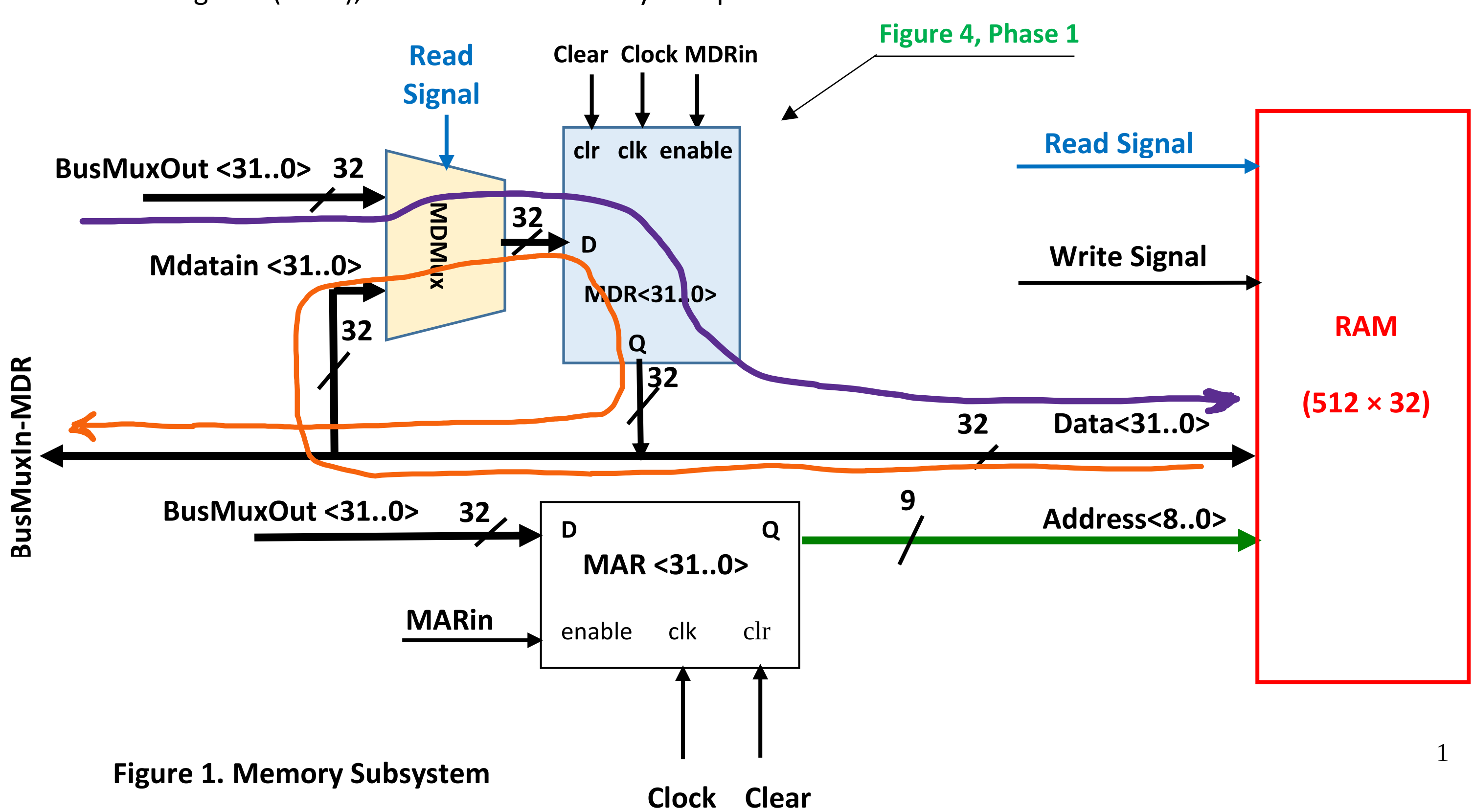


Figure 1. Memory Subsystem

You may choose a RAM (synchronous or asynchronous) from the Library and configure it, or you may write your own VHDL/Verilog code. The Read and Write control signals are the signals to the memory for Read and Write operations and may need to be latched if the memory is slow. These signals are generated by the Control Unit in Phase 3. A synchronous memory would work fine for this project. However, if the memory is asynchronous, then you may need to generate a Completion signal from the memory interface to let the Control Unit know when data becomes available.

## 2.2 The Select and Encode Logic

Figure 2 shows the block diagram for the "Select and Encode" logic. In Phase 1, in order to test the datapath, we used the *R0in – R15in* and *R0out – R15out* signals as external inputs. In this phase, we generate these signals internally by the "Select and Encode" logic.

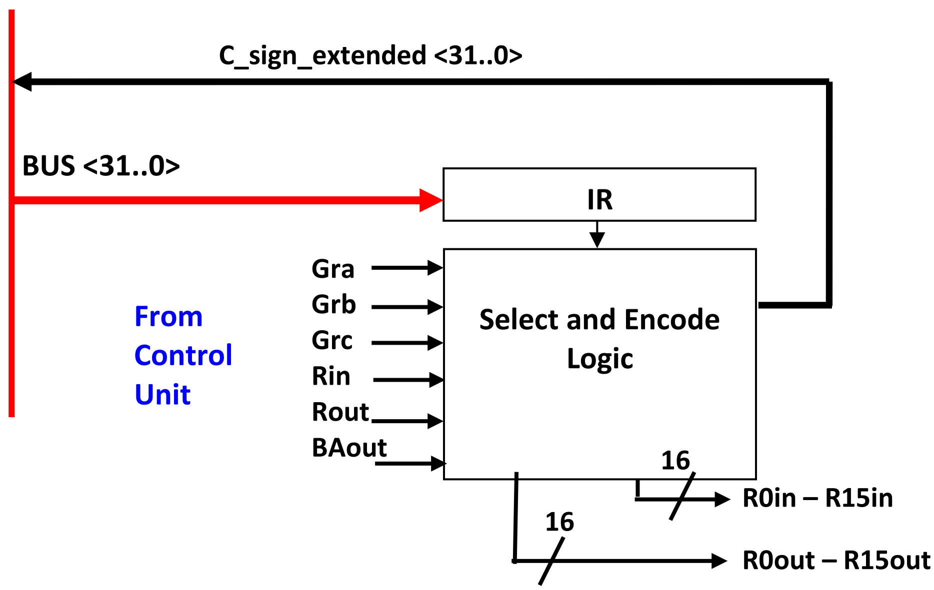


**Figure 2. Select and Encode block diagram**

As shown in Figure 2 and Figure 3, the "Select and Encode" logic accepts the *Gra, Grb, Grc, Rin, Rout*, and *BAout* signals as external inputs. In Phase 3, these signals will be generated internally by the Control Unit. The new control signals *R0in – R15in* and *R0out – R15out* signals are derived by selecting the appropriate 4-bit fields for Ra, Rb, and Rc in the IR register, using the *Gra, Grb, or Grc* control signal, and decoding them along with the *Rin, Rout*, and *BAout* control signals (you may want to consult the Instruction Formats in the CPU Specification document). The general version of this design for the SRC that has 32 general-purpose registers is shown in Figure 4.4 on page 148 of the Lab Reader. The logic needed for the case of only 16 registers in Mini SRC is shown in Figure 3.

The *BAout* (base address) signal, when asserted, gates 0's onto the bus if R0 is selected (see the revisions to R0 circuitry in Figure 5 of this document) in the Load and Store instructions; otherwise, it will put the contents of one of the selected registers R1 – R15 onto the bus.

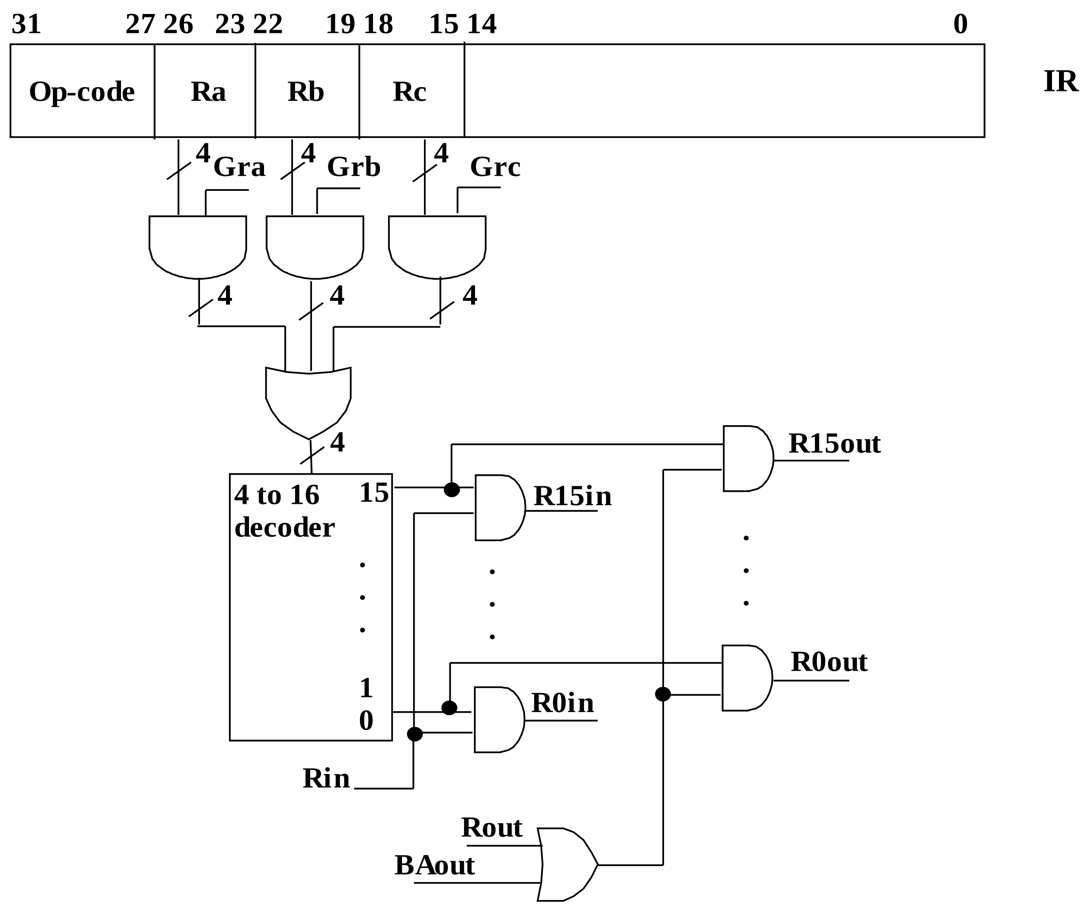


**Figure 3. “Select and Encode” logic to generate R0in-R15in and R0out-R15out**

To support 2’s complement numbers in Load/Store (*ld, ldi,* and *st*) instructions as well as ALU (*addi, andi,* and *ori*) instructions, the constant C in the IR needs to be sign-extended to 32 bit. The logic needed in Mini SRC is shown in Figure 4 (similar to Figure 4.5 on page 150 of the Lab Reader). The sign-extension is done by fanning out the msb of the appropriate field (IR<18>) to all the higher-order bits.

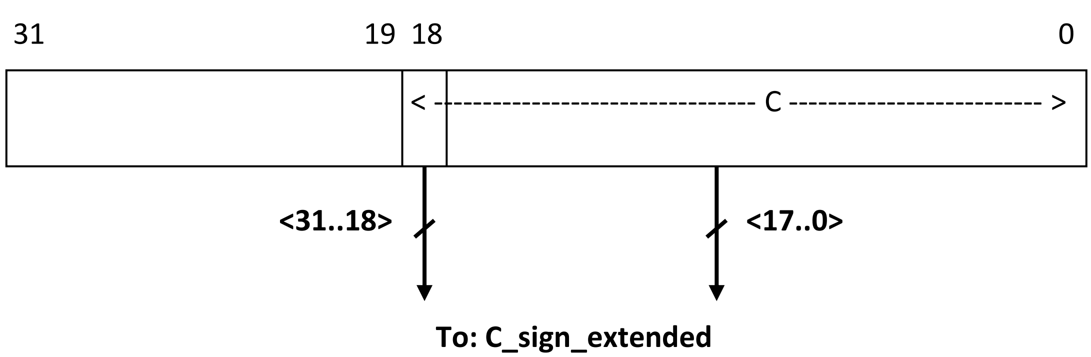


**Figure 4. Sign extension of constant C**

## 2.3 Revision to Register R0

The specification of the Load and Store instructions (*ld, ldi, st*) in Mini SRC suggests that depending on the chosen register Rb, the effective address/data is either the constant C (when Rb = R0) or the constant C plus the contents of the specified Rb register (when Rb ≠ R0). As discussed earlier, the *BAout* signal gates 0's onto the bus if R0 is selected, or it gates the selected register's contents if one of the registers R1 – R15 is selected. To support the Load and Store instructions, the register R0 circuitry will then need to be revised as shown in Figure 5.

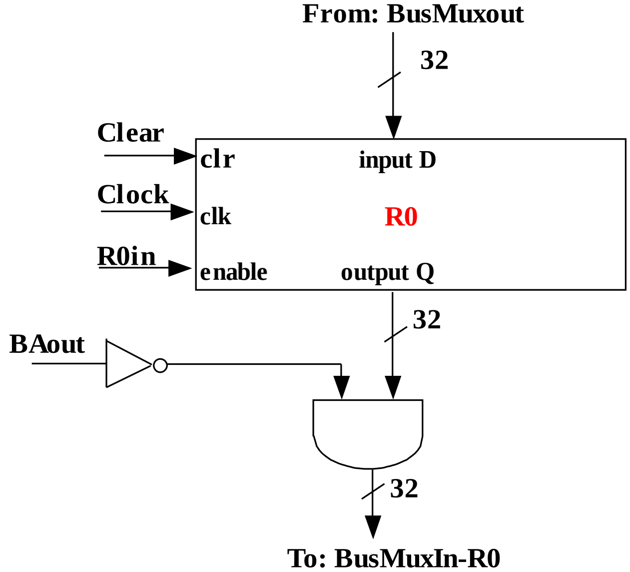


**Figure 5. Revised register R0**

## 2.4 The "CON FF" Logic

Figure 6 shows the block diagram for the "CON FF" logic. The "CON FF" logic is used to determine whether the correct condition has been met to cause branching to take place in a Conditional Branch instruction.

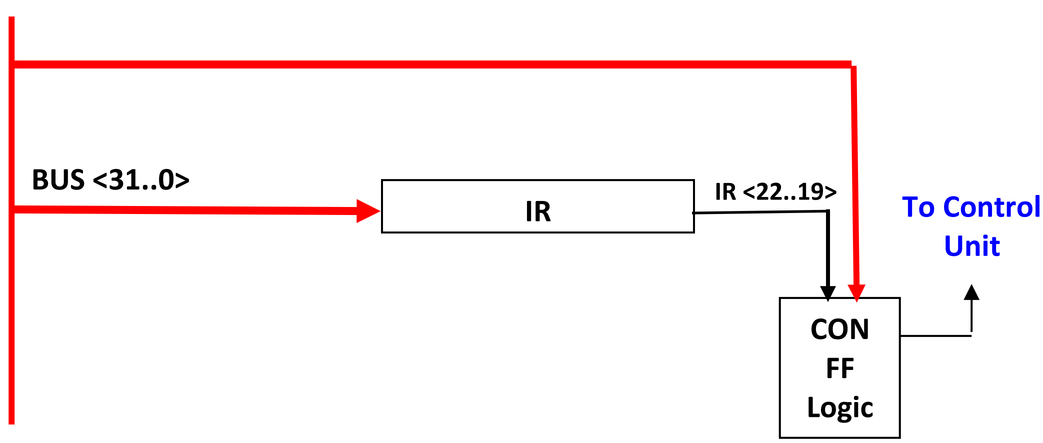


**Figure 6. CON FF logic**

- Conditional Branch Instructions
  brzr, brnz, brmi, brpl

  As described in the Mini SRC specification, the branch instruction has the following format:

| 31 27 | 26 23 | 22 19 | 18 0 |
|---|---|---|---|
| Op-code | Ra | C2 | C |

Branch PC ← PC + 1 + C (sign-extended) if R[Ra] meets the condition

| C2 field: | | | |
|---|---|---|---|
| | --00: branch if zero | brzr | Ra, C |
| | --01: branch if nonzero | brnz | Ra, C |
| | --10: branch if positive | brpl | Ra, C |
| | --11: branch if negative | brmi | Ra, C |

The signals needed to control branching instructions are derived from the numerical value in register Ra, and from the branching condition in the C2 field, in IR[22..19]. For SRC, the required logic is shown in Figure 4.10 on page 158 of the Lab Reader. We simplify this in Mini SRC, as shown in Figure 7. Note that the CONin signal (connected to the enable input of the CON FF) will be generated by the Control Unit in Phase 3.

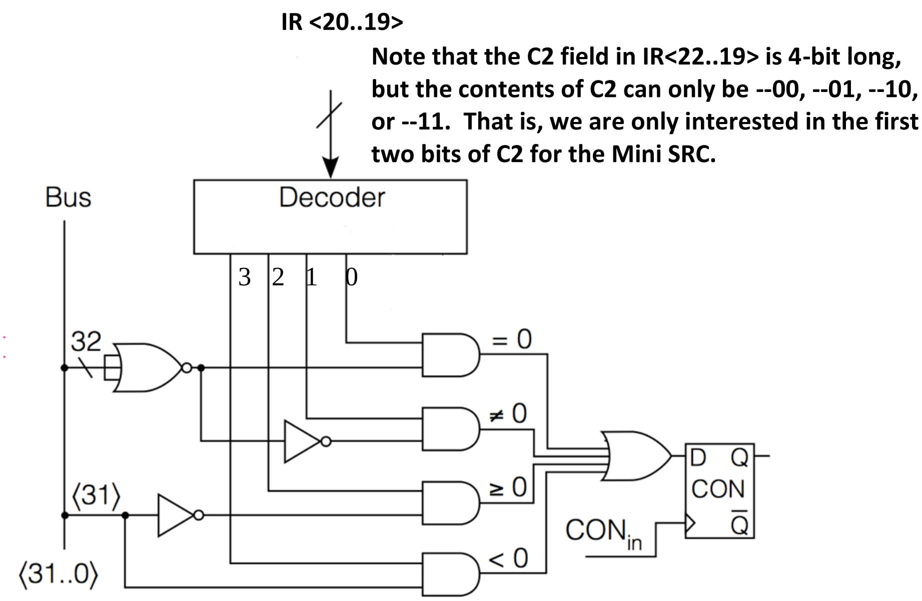


**Figure 7. Computation of the conditional value CON in the CON FF Logic**

### 2.5 Input, Output ports

The Input and Output Ports are shown in Figure 8. As for the input device, the device may have a strobe signal to indicate when the data is available. It is up to you if you want to support such an input device with a strobe signal. Depending on your design, you may (or may not) need the *Strobe* or the *Clock* signal.

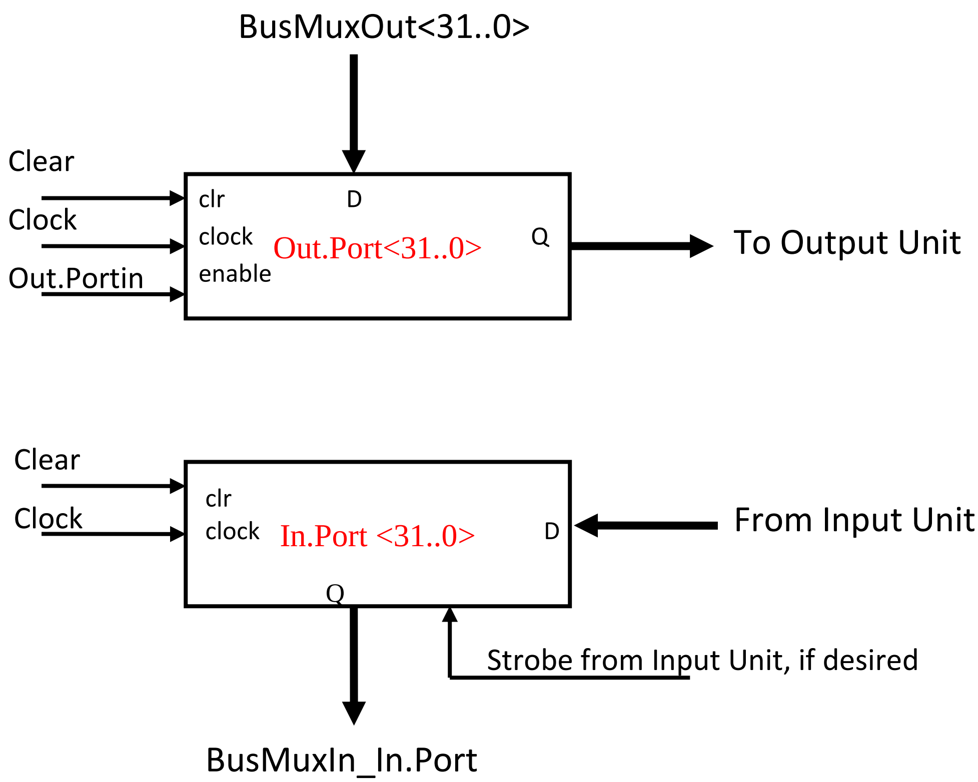


**Figure 8. Input and Output ports**

## 3. Lab Procedure

Add the necessary logic discussed in Section 2, in that order, to your Datapath in Phase 1. Then, using the following control sequences, test your logic circuits for the *ld, ldi, st, addi, andi, ori, brzr, brnz, brmi, brpl, jr, jal, mfhi, mflo, out,* and *in* instructions.

### 3.1 Load Instructions – ld and ldi:

In order to test your RAM and the memory interface logic, functionally simulate the *ld* and *ldi* instructions by using the following Control Sequences (depending on your memory subsystem, you may come up with similar Control Sequences). As in Phase 1, cycles T0, T1, and T2 are used for the instruction fetch. The selection of a register or the value 0 is affected by the BAout control signal. Sign extension is accomplished by the Cout control signal.
Mdatain[31..0] has been provided in T1 and T6 for clarity and should not be regarded as input signal in your simulations anymore. Instead, the memory data should now come directly from your RAM memory output data.

### Control Sequence: ld

| **Step** | **Control Sequence** |
|---|---|
| T0 | PCout, MARin, IncPC, Zin |
| T1 | Zlowout, PCin, Read, Mdatain[31..0], MDRin |
| T2 | MDRout, IRin |
| T3 | Grb, BAout, Yin |
| T4 | Cout, ADD, Zin |
| T5 | Zlowout, MARin |
| T6 | Read, Mdatain[31..0], MDRin |
| T7 | MDRout, Gra, Rin |

### Control Sequence: ldi

| **Step** | **Control Sequence** |
|---|---|
| T0-T2 | Same as before for "Instruction Fetch" |
| T3 | Grb, BAout, Yin |
| T4 | Cout, ADD, Zin |
| T5 | Zlowout, Gra, Rin |

Demonstrate to the TA the proper operation of your memory interface logic and RAM by simulating the following instructions. You will need to preload your registers with some known values, and the contents of the RAM memory using a Memory Initialization file (MIF). Alternatively, you may initialize the memory using an HDL code. Note that $ sign means that the number is a hexadecimal number.

| | | |
|---|---|---|
| Case 1: | ld | R1, $75 |
| Case 2: | ld | R0, $45(R1) |
| Case 3: | ldi | R1, $75 |
| Case 4: | ldi | R0, $45(R1) |

## 3.2 Store Instruction - st:

In order to test your RAM and the memory interface logic, functionally simulate the *st* instruction for the following cases. Devise your own Control Sequence, inferred from the control sequence for the *ld* instruction.

In your simulations, show that the memory location with the address $90 for Case 1, and (R4) + $90 for Case 2, is loaded with the value $67 in R4, respectively. Thus, for verification purposes, you may need to read back the contents of these memory locations. Demonstrate your simulations to one of the TAs in the Lab.

| | | |
|---|---|---|
| Case 1: | st | $90, R4 |
| Case 2: | st | $90(R4), R4 |

## 3.3 ALU Immediate Instructions – addi, andi, ori:

Demonstrate the functionality of your "Add immediate" instruction by simulating the Control Sequence for *addi R2, R3, -3* instruction, as follows:

**Control Sequence: addi**

| Step | Control Sequence |
|---|---|
| T0-T2 | Same as before for "Instruction Fetch" |
| T3 | Grb, Rout, Yin |
| T4 | Cout, ADD, Zin |
| T5 | Zlowout, Gra, Rin |

Demonstrate the operation of your "AND immediate" and "OR immediate" instructions by simulating the Control Sequence for *andi R2, R3, $25* and *ori R2, R3, $25* instructions, respectively. The Control Sequences are the same as the one for *addi* instruction except for using the AND/OR control signal in T4 instead of the ADD signal.

## 3.4 Branch instructions – brzr, brnz, brpl, brmi:

In order to test the "CON FF" logic, functionally simulate the *brzr, brnz, brpl*, and *brmi* instructions by using the following Control Sequence.

**Control Sequence: Branch**

| Step | Control Sequence | |
|---|---|---|
| T0-T2 | Same as before for "Instruction Fetch" | |
| T3 | Gra, Rout, CONin | |
| T4 | PCout, Yin | |
| T5 | Cout, ADD, Zin | |
| T6 | Zlowout, CON ➔ PCin | // IF CON FF = 1, THEN<br>PCin ← PC + 1 + C (sign-extended)<br>// See if there is any way of doing this better |

Verify your implementation by simulating the following Conditional Branch instructions:

| | | |
|---|---|---|
| Case 1: | brzr | R6, 25 |
| Case 2: | brnz | R6, 25 |
| Case 3: | brpl | R6, 25 |
| Case 4: | brmi | R6, 25 |

Preload the register R6 and the PC, so the branch will be "taken" or "not taken". Demonstrate your simulation to one of the TAs in the Lab.

### 3.5 Jump Instructions – jr, jal:

Demonstrate the functionality of *jr* instruction by simulating the Control Sequence for *jr R2* instruction. Preload the register involved.

**Control Sequence: jr**

| **Step** | **Control Sequence** |
|---|---|
| T0-T2 | Same as before for "Instruction Fetch" |
| T3 | Gra, Rout, PCin |

Derive the control sequence for *jal* instruction and demonstrate its functionality by simulating the Control Sequence for *jal R2* instruction. Preload the register involved.

### 3.6 Special Instructions - mfhi and mflo:

Derive the control sequences for mfhi and mflo instructions and demonstrate their functionality to one of the TAs by simulating the Control Sequences for *mfhi R4 and mflo R6* instructions. Preload the registers involved.

### 3.7 Input/output Instructions – in, out:

Demonstrate the functionality of your Output Port logic by simulating the control sequence for *out R2* instruction. Preload the register involved.

**Control Sequence: out**

| **Step** | **Control Sequence** |
|---|---|
| T0-T2 | Same as before for "Instruction Fetch" |
| T3 | Gra, Rout, Out.Port |

In order to test your Input Port logic, functionally simulate the *in R3* instruction. Demonstrate your simulations to one of the TAs in the Lab.

## 4. Report:

Upload your Phase 2 report (one per group) to onQ by 11:59pm on the day of your Phase 2 demo. Phase 2 report consists of:

- Your VHDL/Verilog code, and schematic (if any)
- Your testbenches (if they are similar, include one testbench and discuss the differences for the other cases).
- Printout of the contents of memory
- Functional simulation runs for all the instructions in Phase 2 (Sections 3.1 to 3.7)