# Simple Microprocessor Design

By Dr Hashim Ali

This application note gives an introduction to microprocessor architecture. The goal of the project is to build a 4-bit processor at logic level and then simulate the processor at layout level.

## 1. Introduction

This document introduces the basic concepts of microprocessor architecture in the simplest possible way with a custom instruction set. The design of the processor is very primitive, but already quite complex, as shown in Fig. 1.

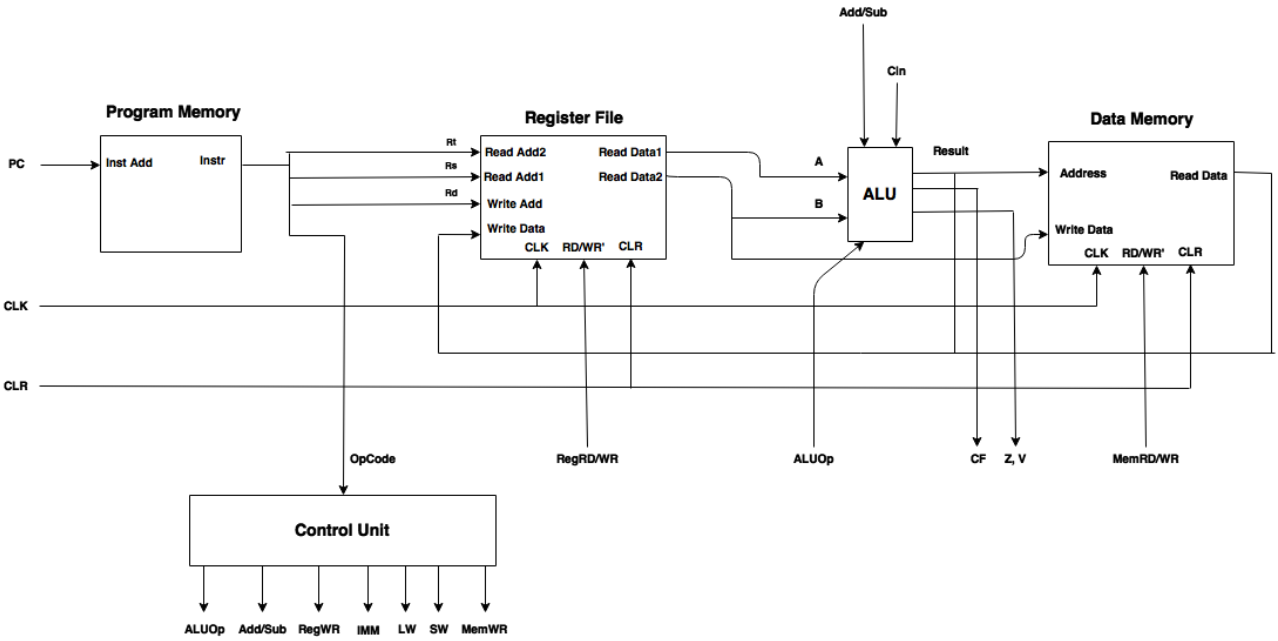



FIGURE 1. SIMPLE MICROPROCESSOR DESIGN

The goal of this assignment is to design a complete data-path and control unit of the simple microprocessor as shown in the above figure.

## 2. Design of Microprocessor

The design of simple processor architecture consists of:-

a) Instructions
- Memory reference instructions: LW (Load Word), SW (Store Word)
- Arithmetic-Logical instructions: AND, OR, ADD, SUB, SLT (Set Less Than)
- Control flow instructions: BEQ (Branch equal), J (Jump) [Not part of this project]

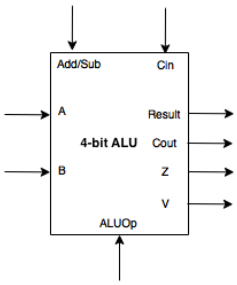
b) Generic Implementation
- Use Program Counter (PC) to supply instruction address
- Get the instruction from the Memory (also called as, Instruction memory or Program memory)
- Read Registers from Register File
- Use the instruction to decide exactly what to do

c) All instructions use the ALU after reading the registers.

d) In result, two types of functional units:
- Elements that operate on data values (Combinational)
  - Example: Program Memory (IMEM), ALU, Control Unit
- Elements that contain state (Sequential)

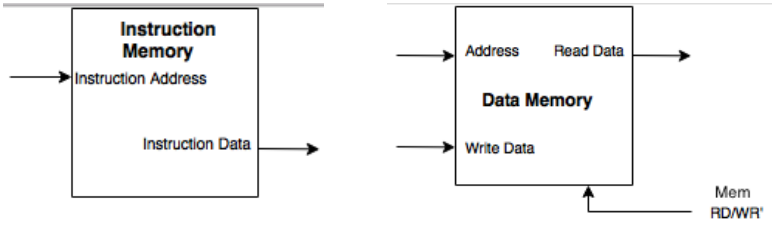
- Examples: Data Memory (DMEM), Register File, Program Counter

# 3. Building Blocks of Microprocessor



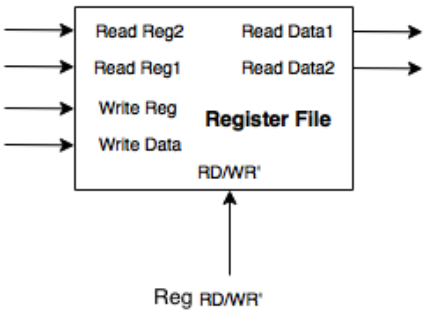

a) Arithmetic and Logic Unit
- An ALU will be composed of following units:-
  - Logical Unit: AND, OR
  - Arithmetic Unit: ADD, SUB
  - Comparator: SLT

b) Memory
- Memory to store instructions and data
- Instruction memory takes address and supplies instruction
- Data memory takes address and supply data for LW
- Data memory takes address and data and write into the memory for SW




c) Program Counter
- A Program Counter (PC) and its update mechanism.
- In this assignment, PC will start from 000 to 111.

d) Register File
- A Register File to include registers
  - It requires two operands and write a result back in register file
- Sometimes part of operands comes from the instructions
- Support of immediate class of instructions




The role of each block is described in table below.

| Block Name | Description | Size |
|---|---|---|
| **Program Counter (PC)** | The PC counts from 000 to 111. It monitors the address of the active instruction. Initially, the program counter is set to 000, so the microprocessor starts by the first instruction of the memory. | 3 bits |
| **Program Memory** | The program memory stores the program. Each program line has an 16 bit format instruction. The description of instructions is provided ahead. | $2^3$x16 bits |

| | | |
|---|---|---|
| **Register File** | The register file contains 4 auxiliary registers. Each register is of 4 bits. Upon request (RegRd/WR=1), the registers data will be available to ALU. In order to save the ALU result, the RegRD/WR will be 0. | $2^2$x4 bits |
| **Arithmetic Logic Unit (ALU)** | The ALU performs the operations; AND, OR, AND, OR, SLT.<br>The appropriate operation will be selected using ALUOp, Add/Sub, Cin.<br>The ALU generates Flags; CF (Carry Out), Z (Zero), V (Overflow). | 4 bits |
| **Data Memory** | The data memory stores the data, needed for program instructions.<br>In order to read data (using LW instr.) from the data memory, the ALU will generate the address (WriteAdd). So, the size of the data memory is dependent on the ALU.<br>4—bit ALU means $2^4$ = 16 different addresses can be accessed.<br>The data memory will transfer the data (ReadData) to Register File where each register is of 4—bit.<br>In order to write data (using SW instr.) to the data memory, the ALU will generate address (WriteAdd) and register file will provide data (WriteData). | $2^4$x4 bits |

The details of control signals as shown in Fig. 1 is provided by the instruction (stored in Program Memory), which plays a fundamental role in the control of the microprocessor.

| Control Signal | Description |
|---|---|
| **ALUOp** | ALUOp (3-bits) selects the appropriate operation of ALU. |
| **Add/Sub , Cin** | Add/Sub=0, Cin=0 is for Addition operation and Add/Sub=1, Cin=1 for subtraction. |
| **RegRd/WR'** | RegRD/WR'=1 to read data from Register File. RegRD/WR'=0 to write data to Register File. |
| **MemRD/WR'** | MemRD/WR'=1 to read data from Data Memory. MemRD/WR'=0 to write data to Data Memory. |
| **IMM** | IMM=1 to supply immediate data directly to ALU. IMM should be 1, during Load (LW) instruction to supply memory Offset. |
| **LW** | LW=1 to load data from Data Memory, otherwise LW=0. |
| **SW** | SW=1 to store data to Data Memory, otherwise SW=0. |

# 4. Instructions Format

The simple microprocessor uses 16-bit instructions that are stored in the Program Memory. There are three types of instructions:-

### a) Register to Register (R-R) Instructions

- Instructions
  - AND, OR, ADD, SUB, SLT
- Process
  - Read source operands from Registers
  - Execute operation (AND, ADD etc) in ALU
  - Write result back to Registers

| 15:13 | 12:8 | 7:6 | 5:4 | 3:2 | 1:0 |
|---|---|---|---|---|---|
| — | OpCode | Rd | Rs | Rt | — |

**b) Immediate Instructions**

- Instructions
  - ANDi, ORi, ADDi, SUBi
- Process
  - **Read** first source operand from **Registers**
  - **Read** second source operand from **Instruction**
  - **Execute** operation (Or, ADD etc) in ALU
  - **Write** result back to **Registers**

| 15:13 | 12:8 | 7:6 | 5:4 | 3:2 | 1:0 |
|---|---|---|---|---|---|
| — | OpCode | Rd | Rs | Immediate | |

**c) Memory Reference Instructions**

**I. LW (Load Word) Instruction**

- Process
  - **Read** first source operand from **Registers**
  - **Read** second source operand from **Instruction**
  - **Execute** operation (ADD) in ALU to calculate word address
  - **Read** word from **Data Memory**
  - **Write** result back to **Registers**

| 15:13 | 12:8 | 7:6 | 5:4 | 3:2 | 1:0 |
|---|---|---|---|---|---|
| — | OpCode | Rd | Rs | Immediate | |

**II. SW (Store Word) Instruction**

- Process
  - **Read** first source operand from **Registers**
  - **Read** second source operand from **Instruction**
  - **Read** data from **Registers**
  - **Execute** operation (ADD) in ALU to calculate word address
  - **Write** word to **Data Memory**

| 15:13 | 12:8 | 7:6 | 5:4 | 3:2 | 1:0 |
|---|---|---|---|---|---|
| — | OpCode | Rd | Rs | Immediate | |

Assume that all registers initially stores the numeric value 0, and ***Rd***, ***Rs***, ***Rt*** can refer to any register within Register File.

The microprocessor you have to design will have the following types of instructions:

| **Instruction** | **OpCode** | **Type** | **Operation** |
|---|---|---|---|
| AND Rd, Rs, Rt | 0000 | R | Rd <— Rs AND Rt |
| OR Rd, Rs, Rt | 0001 | R | Rd <— Rs OR Rt |
| ADD Rd, Rs, Rt | 0010 | R | Rd <— Rs + Rt |
| SUB Rd, Rs, Rt | 0011 | R | Rd <— Rs - Rt |
| ANDi Rd, Rs, IMM | 1000 | I | Rd <— Rs + IMM |

| | | | |
|---|---|---|---|
| ADDi Rd, Rs, IMM | 1010 | I | Rd <— Rs + IMM |
| LW Rd, IMM(Rs) | 1110 | I | Rd <— DMEM[Rs + IMM] |
| SW Rd, IMM(Rs) | 1111 | I | DMEM[Rs + IMM] <— Rd |

# 5. Circuit Elements

## I. The ALU Design

The core of the processor - all the actual computations are performed here. As shown in the instruction set, operations such as addition, subtraction and logical operations are all done in this unit. Also, the output of the ALU is used as the address for certain memory related operations.

As mentioned above, the ALU is able to execute the following operations:

| Instruction | Inputs | | | | | | Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | ALUOp (Select Lines) | | Add/ Sub | Cin | Result | Cout (Carry Out) | Flags | |
| | | | A1 | A0 | | | | | Z (Zero) | V (Overflow) |
| AND | | | 0 | 0 | 0 | 0 | | | | |
| OR | | | 0 | 1 | 0 | 0 | | | | |
| ADD | | | 1 | 0 | 0 | x | | | | |
| SUB | | | 1 | 0 | 1 | 1 | | | | |
| SLT | | | 1 | 1 | 1 | 0 | | | | |

The simple way to design multiple inputs ALU is to first design a 1-bit ALU with above instructions as shown in the Fig. 2.
The combinational circuit block diagram of the 4-bit ALU design using 1-bit ALU is shown in Fig. 3. and block diagram of 4-bit ALU is shown in Fig. 4.
**Rebuild the similar ALU in Logisim and verify its working.**

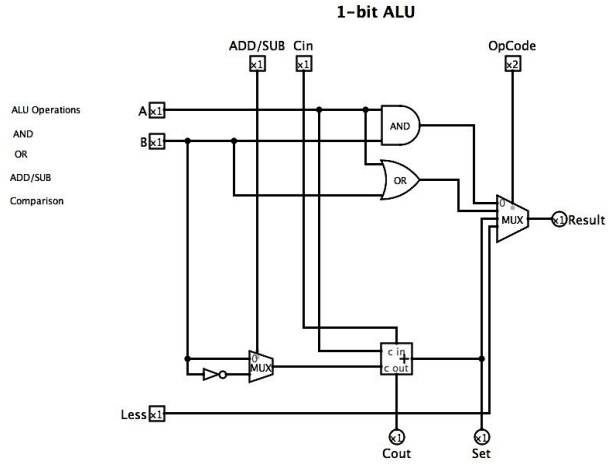



FIGURE 2. COMBINATIONAL CIRCUIT DIAGRAM OF 1-BIT ALU

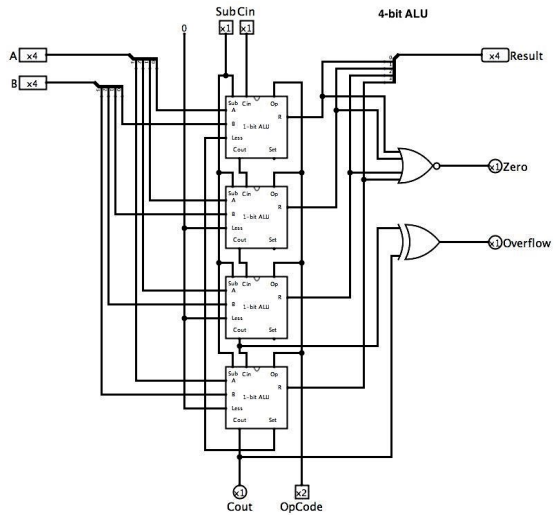



FIGURE 3. COMBINATIONAL CIRCUIT DIAGRAM OF 4-BIT ALU

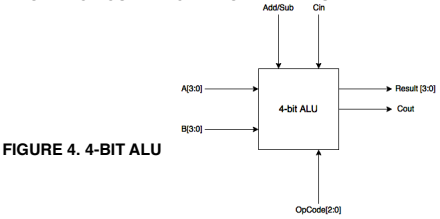



FIGURE 4. 4-BIT ALU

## II. Register File

This is a set of 4 registers, each storing an 4 bit value. There are 2 output values Rs and Rt, whose values are selected based on the instruction word, as in the instruction definitions given above by giving control signal (RegRD/WR' = 1). There is one port by which data can be written into one of the registers Rd, but make sure to give an control signal (RegRD/WR' = 0) to control whether or not to update the value. The clear signal CLR is to clear the Register File.

In Logisim, D-FlipFlops can be used to create Register File, as shown in the Fig. 5. **Rebuild the similar Register File in Logisim and verify its working.**

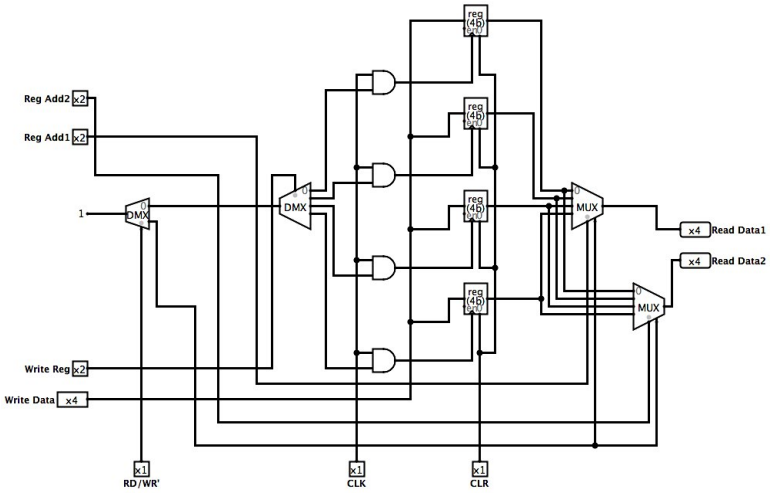



FIGURE 5. SEQUENTIAL CIRCUIT DIAGRAM OF REGISTER FILE

## III. Program/Instruction Memory (IMEM)

This can be a combinational unit - it takes just the address bus (3 bit value) as input, and gives out a 16-bit value that is the instruction to be decoded. The address is provided by the Program Counter (PC).

**Build the Program Memory using ROM component in Logisim and verify its working.**

## IV. Data Memory (DMEM)

This needs to be a sequential / clocked unit. At any given cycle, you are either reading from it or writing to it. In case of a LW or SW instruction, the address is either immediate or the output of the ALU. The data output of the DMEM will always go into the Register File, where it gets stored into a register selected by Rd.

The Data Memory circuit diagram is shown in Fig. 6. **Rebuild the similar Data Memory in Logisim using RAM component and verify its working.**

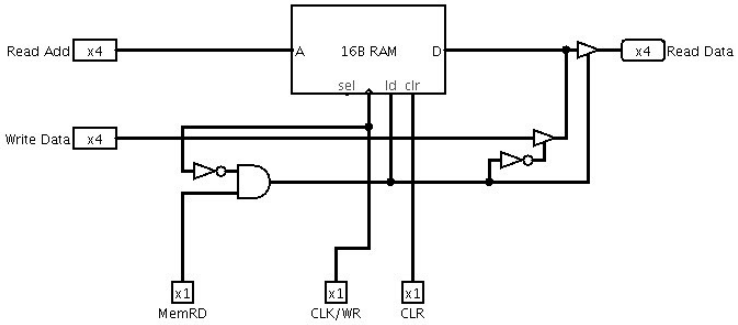



FIGURE 6. SEQUENTIAL CIRCUIT DIAGRAM OF DATA MEMORY

## V. Program Counter (PC)

3 bit register. Under normal operations, will always increment by 1 on every clock cycle, to access the next instruction.

The Program Counter circuit diagram is shown in Fig. 6. **Rebuild the similar Program Counter in Logisim and verify its working.**

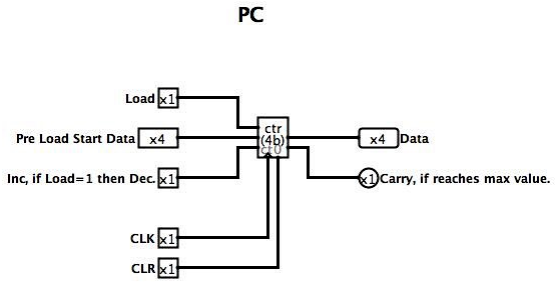



FIGURE 7. SEQUENTIAL CIRCUIT DIAGRAM OF PROGRAM COUNTER

## VI. Control Unit Design

The unit that actually makes the entire processor work as expected. The input to this is the instruction word, and the output is a set of control signals that decide, for example, whether the register file is to updated, what operation is to be done by the ALU, whether memory read or write is required etc.

One way to implement this is to make it a combinational block that will generate the following signals:

| Instruction | Type | Input | Outputs | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OpCode | ALUOp | | Add/ Sub | Cin | RegRD/ WR' | IMM | LW | SW | MemRD/ WR' |
| | | | A1 | A0 | | | | | | | |
| AND | R | 0000 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| OR | R | 0001 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ADD | R | 0010 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| SUB | R | 0011 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| SLT | R | 0100 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| NOP | — | 0101 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| — | — | 0110 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| — | — | 0111 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ANDi | I | 1000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| ORi | I | 1001 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| ADDi | I | 1010 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| SUBi | I | 1011 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| — | — | 1100 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| — | — | 1101 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

| Instruction | Type | Input | Outputs | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OpCode | ALUOp | | Add/ Sub | Cin | RegRD/ WR' | IMM | LW | SW | MemRD/ WR' |
| | | | A1 | A0 | | | | | | | |
| LW | I | 1110 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| SW | I | 1111 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

Design a combinational circuit of the Control Unit by applying K-Maps or any technique and build a circuit using Logisim.

# Logisim

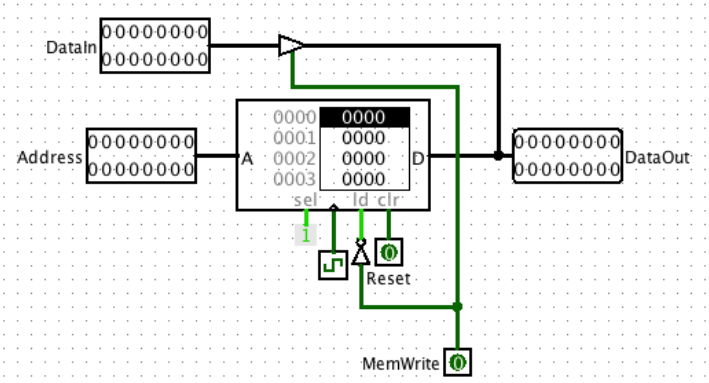
It is strongly recommended that you download and run Logisim on your local machine while developing your processor. As you've probably discovered in lab, Logisim can quickly overwhelm the instructional machines. Though Logisim is relatively stable, it is still recommended that you save often and also make backup copies of your .circ files early and often. The official version of Logisim we will be using is available on the course webpage.

If you are having trouble with Logisim: **RESTART IT and RELOAD** your circuit! Don't waste your time chasing a bug that is not your fault. However, if restarting doesn't solve the problem, it is more likely that the bug is a flaw in your project. Please post to the newsgroup about any crazy bugs that you find and we will investigate.

Do NOT copy and paste from different Logisim windows.
Logisim has been known to have trouble with this in the past. So try to avoid it.

# RAM Modules

Logisim RAM modules can be found in the built-in memory library. To add the library to your project, select "Project/Load Library/Built-in Library..." and select the Memory module.




Because the RAM module doesn't look like the idealized memory we saw in lecture, you may feel confused about where to begin. The picture above shows a good way to wire up a circuit to use RAM. Here are a few things to know before you get started.

- "sel" determines whether or not the RAM module is active. We will probably not run into any cases where we need to turn our RAM off, so you can wire a constant 1 to this.
- "A" chooses which address will be accessed.
- The clock input provides synchronization for memory writes. Be sure to use the same clock here as you do for your reg file.

• "ld" determines whether we are reading or writing to memory. If "out" is high, then "D" will be driven with the contents of memory at address "A".

• "clr" will instantly set all contents of memory to 0 if high. You should wire a manual switch so you can clear out memory whenever you want to restart a test.

• "D" acts as both data in and data out for this module. This means you have to be careful not to drive this line from two conflicting sources, which in this case are DataIn and the output of the memory. You can solve this by using a controlled buffer (aka a tri-state buffer) on the "D" port of the RAM module. By wiring logic to the "out" port and the valve port of the controlled buffer together so that they are always opposite values (as in the picture above), we can prevent conflicts between data being driven in and the contents of memory coming out.

• The "poke" tool can be used to modify the contents of the memory. You can also use right-click --> Load Image... to load an image from a file.

The best way to learn how these work is simply to play with them. You can also refer to Logisim documentation on RAM modules here.

Use a RAM module for Data memory. Your instruction memory is a ROM in the harness.
Hint: you might want to connect the wire attached to the the D port of the RAM module to your processor's "Data Mem Data" output, the one attached to the A port to your "Data Mem Address" output, and the MemWrite control wire to your "Data Mem Write" output.