Lab 2: RISC-V single-cycle core architecture

1 Processor Datapath

The datapath is a diagram that describes how signals inside the core propagate. They are broken down into 5 main sections, **Program Counter**, **Program Memory**, **Decoder**, **Register Memory**, **ALU**, and **Data Memory**.

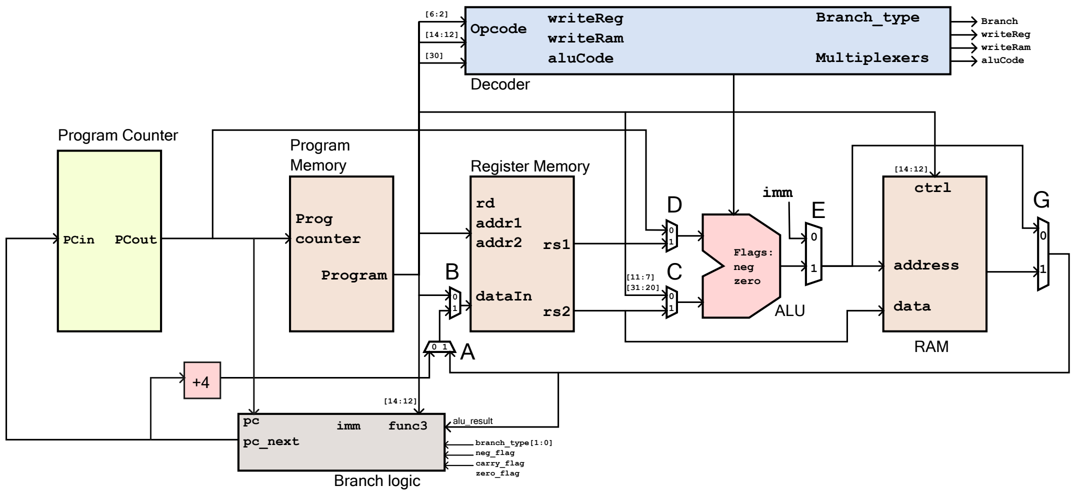

Figure 1: Single Cycle RISC-V Data-path

Component	**Description**
Program Counter	This is responsible for updating the program counter and works closely with the branch logic unit.
Program Memory	A series of 8-bit registers. The binaries for the program are loaded here. Takes in the program counter and returns the instruction.
Register Memory	Stores the 32 core registers of the RISC-V core described in *lab 1*
Decoder	Takes in the opcode of the instruction and outputs control signals for the modules and multiplexers, which guide data through the core in accordance with the instructions specification.
ALU	Takes in 2 32-bit values and performs an operation depending on the control signal given by the decoder.
Data Memory	Made up of many 8-bit registers, this holds the memory registers (RAM), where data can be loaded from and stored to in any location by the core.
Branch unit	Using control signals from the ALU, this unit sends control signals to the program counter to control how branching should operate.
Multiplexers	They can be configured in certain ways to route data through the CPU in a flexible way as to make data flow through the core.

Table 1: Overview of RISC-V core modules

2 Exercises

There are 3 difficulties to this lab

Level 1: Introduction to CPU architecture and
For those who are new to Processor design and architecture, this is tutorial based lab running through the implementation of the Datapath in *Figure 1*.

Level 2: Partial implementation of the RISC-V instruction set.

Level 3: Custom single-cycle CPU

- Using the RV32I specification, create a working single-cycle CPU supporting the instructions in *Appendix 1*.
- A custom datapath diagram must be submitted before the deadline and presented to show thoughful cosiderations of the implementation challenges and feasibility.

For compatibility with tests, the program_memory module should look like the following:

```
module program_memory(
  input Clock,
  input [31:0] PC,
  output logic [31:0] instruction
);

  logic [7:0] Program_memory [1023:0];

  initial begin
    program_memory = $readmemh("./program.hex", Program_memory); // Loads external
program.hex file into memory
  end

  always_comb begin
    instruction[0:7]   = Program_memory[PC];
    instruction[8:15]  = Program_memory[PC + 4];
    instruction[16:23] = Program_memory[PC + 8];
    instruction[24:31] = Program_memory[PC + 12];
  end
endmodule
```