HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

**SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING**



Digital Design Using VHDL

*Project: RISC*

|  |  |  |
| --- | --- | --- |
| **Advisor:** | Prof. Võ Lê Cường |  |
| **Students:** | Ngô Văn Cảnh | 20193206 |
|  | Nguyễn Danh Khuê | 20193224 |
|  | Phùng Mạnh Dũng | 20193210 |

Hà Nội, 11-2021

# Overview

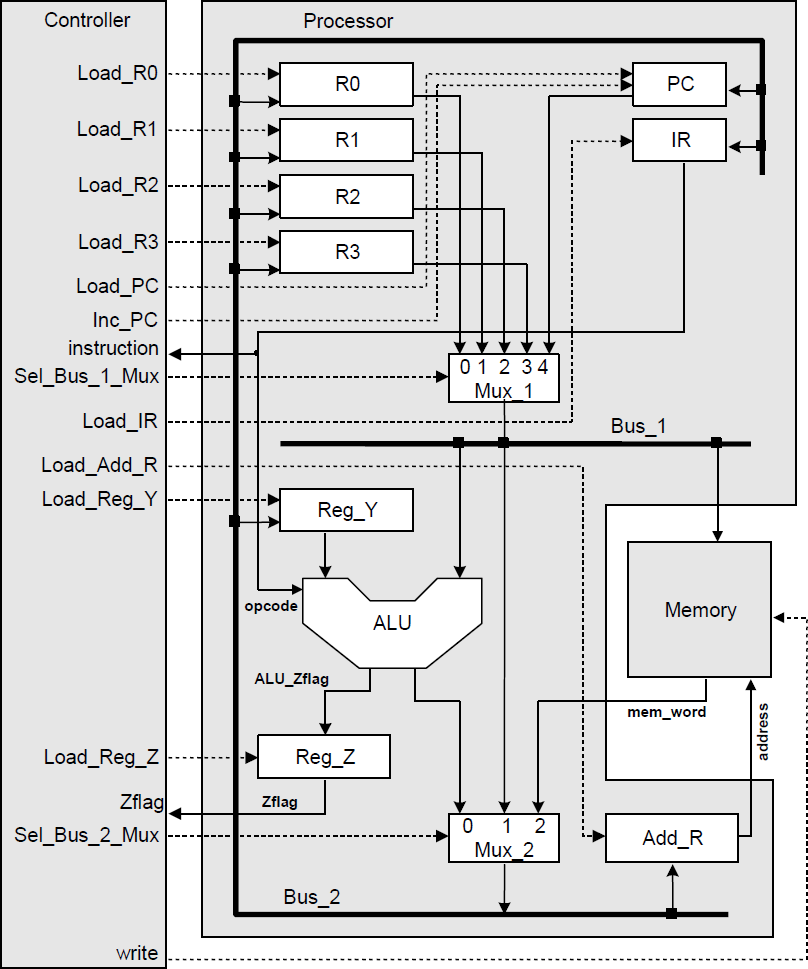
Reduced instruction-set computers (RISC) are designed to have a small set of instructions that execute in short clock cycles, with a small number of cycles per instruction. RISC machines are optimized to achieve efficient pipelining of their instruction streams. The machine also serves as a starting point for developing architectural variants and a more robust instruction set. Designers make high-level tradeoffs in selecting an architecture that serves an application. Once an architecture has been selected, a circuit that has sufficient performance (speed) must be synthesized. Hardware description languages (HDLs) play a key role in this process by modeling the system and serving as a descriptive medium that can be used by a synthesis tool.

# Hardware composition

RISC-SPM or Reduced Instruction Set Computer Store Program Machine consists of three functional units :

* 1. Processing Unit ( Processor )
  2. Controll Unit ( Controller )
  3. Memory Unit ( RAM )

The Overall Architecture of a simple RISC-SPM is shown below



Program instructions and data are stored in memory

Instructions are fetched from memory synchronously, decoded and executed to : Operate on data with ALU

Change the contents of storage registers

Change the content of the program counter (PC), instruction register (IR) and the address register (ADD\_R)

Change the content of memory

Retrieve data and instructions from memory

Control the movement of data on the system busses

The Program Counter (PC) contains the address of the next instruction to be executed The Instruction Register (IR) contains the instruction that currently being executed

The address register (Add\_R) contains the address of the memory location that will be addressed next by a read or write operation.

## Processing Unit

The processor includes *registers*, *buses*, *control lines*, and an *ALU* capable of performing arithmetic and logic operations on its operands depends on the opcode held in the instruction register.

#### Arithmetic Logic Unit

For the purposes of this example, the ALU has two operand datapaths, data\_1 and data\_2, and its instruction set is limited to only 4 instructions, that is :

##### Opcode Action

ADD Adds the datapaths to form data\_1 + data\_2 SUB Subtracts the datapaths to form data\_1 - data\_2

AND Takes the bitwise and of the datapaths data\_1 & data\_2 NOT Takes the bitwise Boolean complement of data\_1

module Arithmetic\_Logic\_Unit( output reg [7:0] ALU\_out, output ALU\_Zflag,

input [7:0] data\_1, data\_2, input [3:0] opcode

);

assign ALU\_Zflag = ~|ALU\_out; //reduction nor always @(opcode or data\_1 or data\_2) begin

case(opcode)

`ADD : ALU\_out = data\_2 + data\_1;

`SUB : ALU\_out = data\_2 - data\_1;

`AND : ALU\_out = data\_1 & data\_2;

`NOT : ALU\_out = ~ data\_1; default : ALU\_out = 0;

endcase

end endmodule

#### Register Unit

There are 9 registers in our design :

5 general-purpose registers R0, R1, R2, R3, Reg\_y ( 8-bit ) 3 special purpose register PC, IR, Add\_R ( 8-bit )

1 flag register Reg\_Z ( 1-bit )

All register has a load signal to store data, a clock signal (clk) to synchronize and a reset signal (rst) to erase data (all bits are set to 0)

The zero flag register Reg\_Z is a 1-bit register, so basically it is a D flip flop.

The Program Counter Register (PC) has an additional signal Inc\_PC to increase PC by 1 unit.

module Register\_Unit(output reg [7:0] data\_out, input [7:0] data\_in, input load, clk, rst);

always @(posedge clk or negedge rst) if (!rst)

data\_out <= 0; else if (load)

data\_out <= data\_in; endmodule

module D\_ff(output reg data\_out, input data\_in, load, clk, rst); always @(posedge clk or negedge rst)

if (!rst)

data\_out <= 0; else if (load)

data\_out <= data\_in; endmodule

module Program\_Counter(output reg [7:0] count, input [7:0] data\_in, input Load\_PC, Inc\_PC, clk, rst);

always @(posedge clk or negedge rst) if (!rst)

count <= 0; else if (Load\_PC)

count <= data\_in; else if (Inc\_PC)

count <= count+1; endmodule

#### Multiplexer

There are 2 multiplexers in the Processing Unit : Mux\_1 : it's a 5-1 multiplexer

Output : Bus\_1

Input : R0, R1, R2, R3, PC

Mux\_2 : it's a 3-1 multiplexer Output : Bus\_2

Input : ALU's output, Bus\_1

module Mux\_3\_1 (output reg [7:0] out, input [7:0] in0, in1, in2, input [1:0] sel); always @(in0 or in1 or in2 or sel)

case(sel)

2'b00 : out = in0; 2'b01 : out = in1; 2'b10 : out = in2; default : out = 'bx;

endcase endmodule

module Mux\_5\_1 (output reg [7:0] out, input [7:0] in0, in1, in2, in3, in4, input [2:0] sel);

always @(in0 or in1 or in2 or in3 or in4 or sel) case(sel)

3'b000 : out = in0; 3'b001 : out = in1; 3'b010 : out = in2; 3'b011 : out = in3; 3'b100 : out = in4; default : out = 'bx;

endcase endmodule

An instruction can be fetched from memory, placed on Bus\_2, and loaded into the instruction register. A word of data can be fetched from memory, and steered to a general-purpose register or to the operand register (Reg\_Y) prior to an operation of the ALU. The result of an ALU operation can be placed on Bus\_2, loaded into a register, and subsequently transferred to memory. A dedicated register (Reg\_Z) holds a flag indicating that the result of an ALU operation is 0.

## Control Unit

#### Functions of the control unit :

* + 1. Determine when to load registers
    2. Select the path of data through the multiplexers
    3. Determine when data should be written to memory
    4. Control the three-state busses in the architecture.

#### Control Signals :

##### Control Signal Action

**Control Signal Action**

Load\_Add\_R Loads the Address Register

Load\_PC Loads Bus\_2 to the Program Counter Load\_IR Loads Bus\_2 to the Instruction Register Inc\_PC Increments the Program Counter

Sel\_Bus\_1\_Mux Selects among the Program Counter, R0, R1, R2, and R3 to drive Bus\_1 Sel\_Bus\_2\_Mux Selects among ALU\_out, Bus\_1, and memory to drive Bus\_2

Load\_R0 Loads general purpose register R0 Load\_R1 Loads general purpose register R1 Load\_R2 Loads general purpose register R2 Load\_R3 Loads general purpose register R3 Load\_Reg\_Y Loads Bus\_2 to the register Reg\_Y

Load\_Reg\_Z Stores the zero Flag of ALU in register Reg\_Z write Loads Bus\_1 into the memory

#### Instruction Set

A machine language program consists of a stored sequence of 8-bit words (bytes). The format of an instruction of RISC\_SPM can be long or short, depending on the operation :

Short instructions : requires 1 byte of memory to specifies 4-bit opcode, 2-bit source register address, a 2-bit destination register address.

Long instruction : requires 2 bytes of memory. The first word of a long instruction contains a 4-bit opcode. The remaining 4 bits of the word can be used to specify addresses of a pair of source and destination registers, depending on the instruction. The second word contains the address of the memory word that holds an operand required by the instruction

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | **Dst** | **Src** |  |
| 0 0 1 0  **Opcode** | 0 1  **Dst** | 1 0  **Src** | **Address** |
| 0 1 1 0 | x x | 1 0 | 0 0 0 1 1 1 0 1 |

The instruction mnemonics and their actions are listed below.

##### Short instruction

**Action**

NOP No operation is performed; all registers retain their values. The addresses of the source and destination register are don't-cares, they have no effect.

##### Short instruction

**Action**

ADD Adds the contents of the source and destination registers and stores the result into the destination register.

SUB Subtracts the content of the source register from the destination register and stores the result into the destination register.

AND Forms the bitwise and of the contents of the source and destination registers and stores the result into the destination register.

NOT Forms the bitwise complement of the content of the source register and stores the result into the destination register.

HALT Halts execution until reset

##### Long instruction

**Action**

RD Reads a word from the location specified by the second byte and loads the result into the destination register. The source register bits are don't-cares.

WR Writes the contents of the source register to the word in memory specified by the address held in the second byte. The destination register bits are don't-cares.

Branches the activity flow by loading the program counter with the word at the address

BR specified by the second byte of the instruction. The source and destination bits are don't- cares.

BRZ Branches if the zero flag register is asserted. The RISC\_SPM instruction set is summarized below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ins** | **Opcode** | **Dst** | **Src** | **Action** |
| NOP | 0000 | xx | xx | none |
| ADD | 0001 | dst | src | dst <= src + dst |
| SUB | 0010 | dst | src | dst <= dst - src |
| AND | 0011 | dst | src | dst <= src && dst |
| NOT | 0100 | dst | src | dst <= ~ src |
| RD | 0101 | dst | xx | dst <= memory [Add\_R] |
| WR | 0110 | xx | src | memory[Add\_R] < = src |
| BR | 0111 | xx | xx | PC <= memory[Add\_R] |
| BRZ | 1000 | xx | xx | PC <= memory[Add\_R] |
| HALT | 1111 | xx | xx | Halts execution until reset (Finish programm) |

#### Controller States

Three phases of operation: fetch, decode, and execute.

Fetching : Retrieves an instruction from memory (2 clock cycles)

Decoding : Decodes the instruction, manipulates datapaths ,and loads registers (1 clock cycle) Execution : Generates the results of the instruction (0, 1 or 2 clock cycles)

ALU operations

Update storage registers Update program counter (PC)

Update the instruction register (IR) Update the address register (ADD\_R) Update memory

Control datapaths

##### State Action

idle State entered after reset is asserted. No action.

Load the Add\_R with the contents of the PC. (Note: PC is initialized to the starting address 00H by

fet1

the reset action.) The state is entered at the first active clock after reset is de-asserted, and is revisited after a NOP instruction is decoded.

fet2 Load the IR with the word addressed by the Add\_R, and increment the PC to point to the next location in memory, in anticipation of the next instruction or data fetch.

dec Decode the IR and assert signals to control datapaths and register transfers.

exe Execute the ALU operation for a single-byte instruction, conditionally assert the zero flag, and load the destination register.

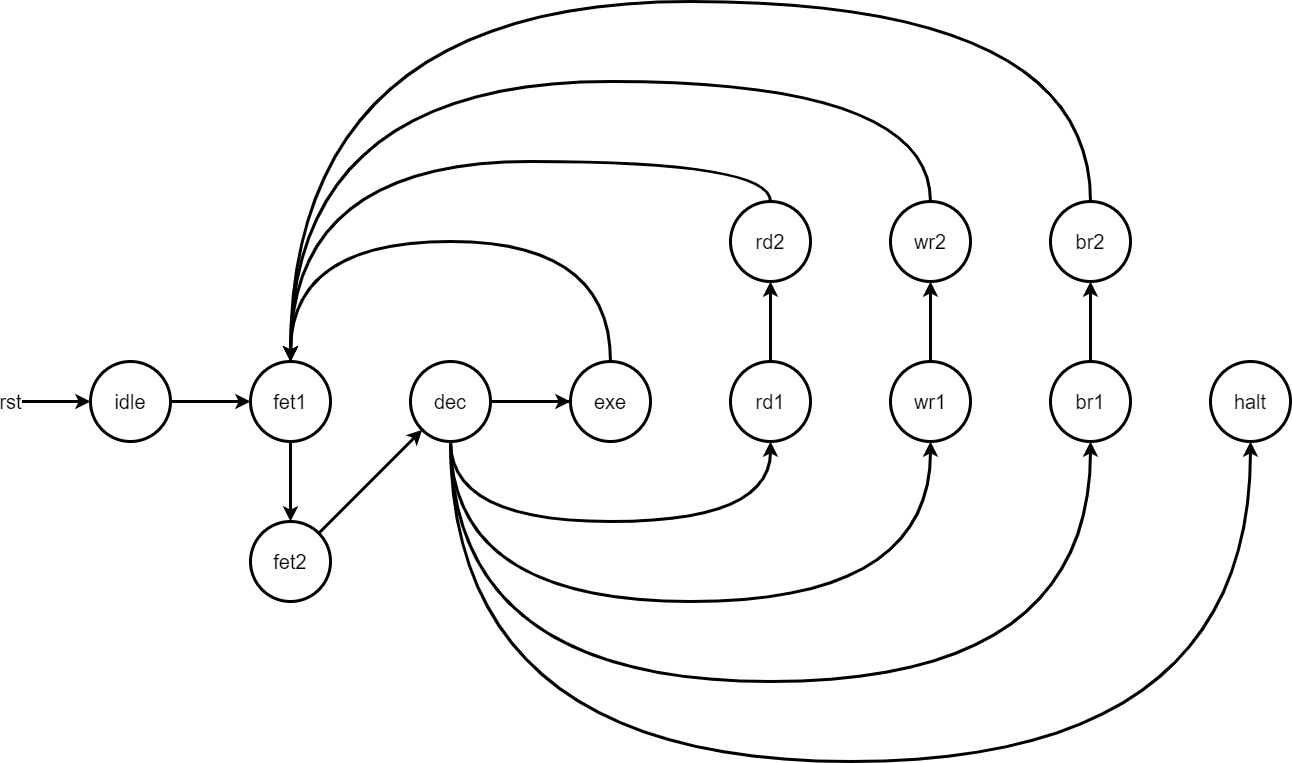
rd1 Load the Add\_R with the second byte of a RD instruction, and increment the PC.

rd2 Load the destination register with the memory word addressed by the byte loaded in rd1. wr1 Load the Add\_R with the second byte of a WR instruction, and increment the PC.

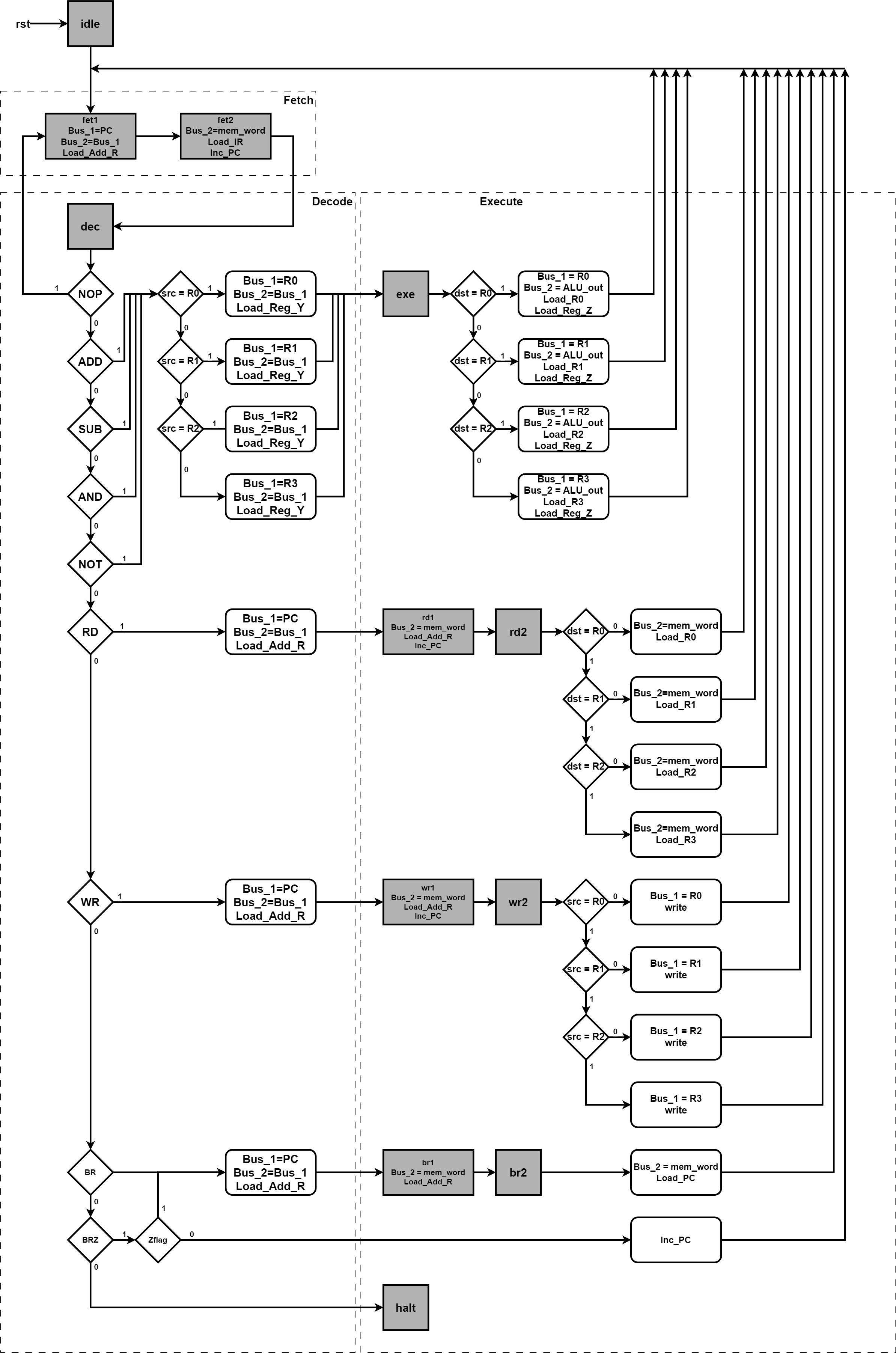
wr2 Load the source register with the memory word addressed by the byte loaded in wr1. br1 Load the Add\_R with the second byte of a BR instruction, and increment the PC.

br2 Load the PC with the memory word addressed by the byte loaded in br1. halt Default state to trap failure to decode a valid instruction.

#### State transition diagram



* 1. ASM chart



## Memory Unit

For simplicity, the memory unit of the machine is modeled as an array of D flip-flops that form a **256 bytes**

RAM.

module Memory\_Unit( output [7:0] data\_out,

input [7:0] data\_in, address, input write, clk

);

reg [7:0] memory [255:0]; //256 bytes ram always @(posedge clk) begin

if(write)

memory[address] <= data\_in;

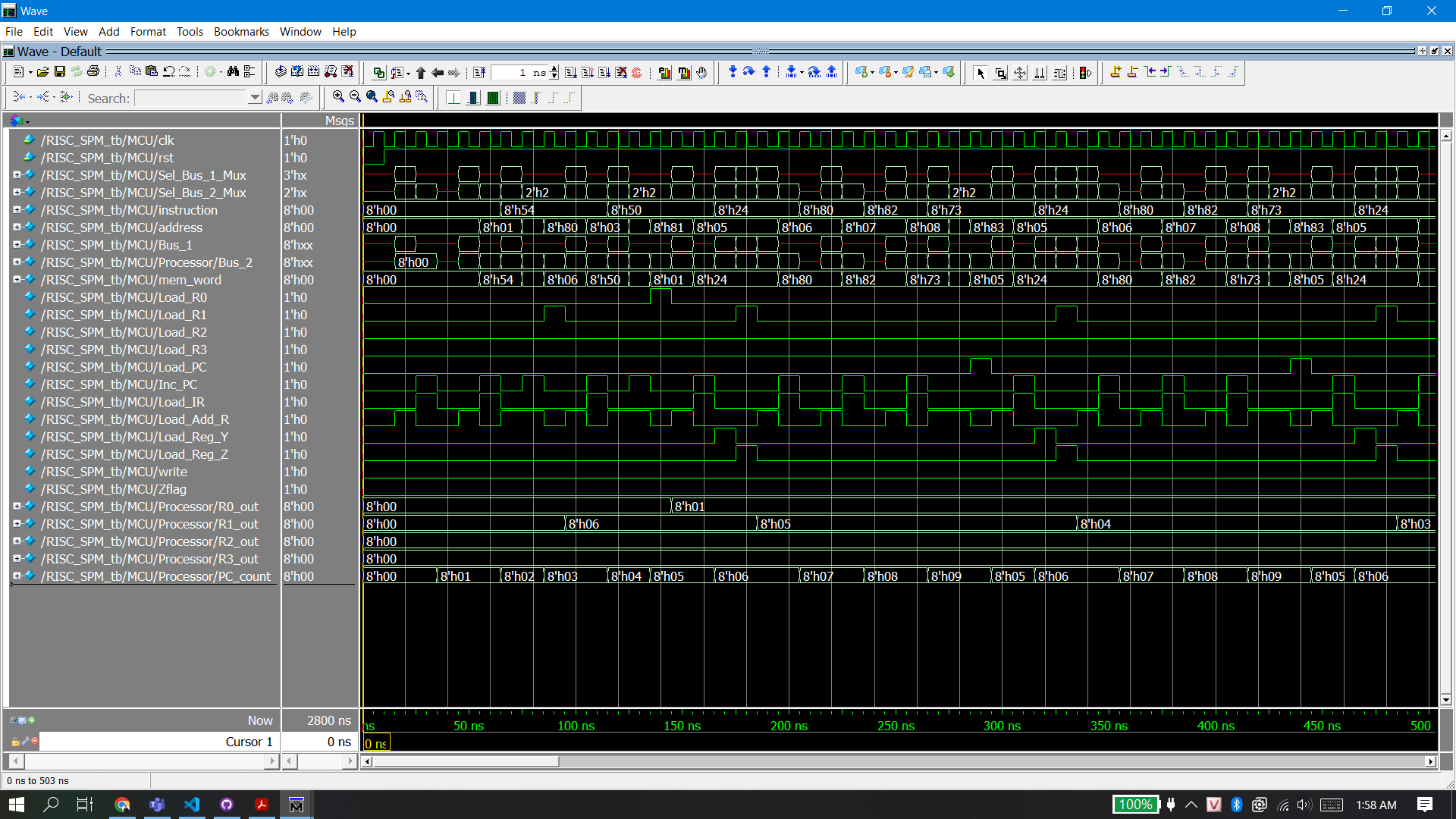
end

assign data\_out = memory[address]; endmodule

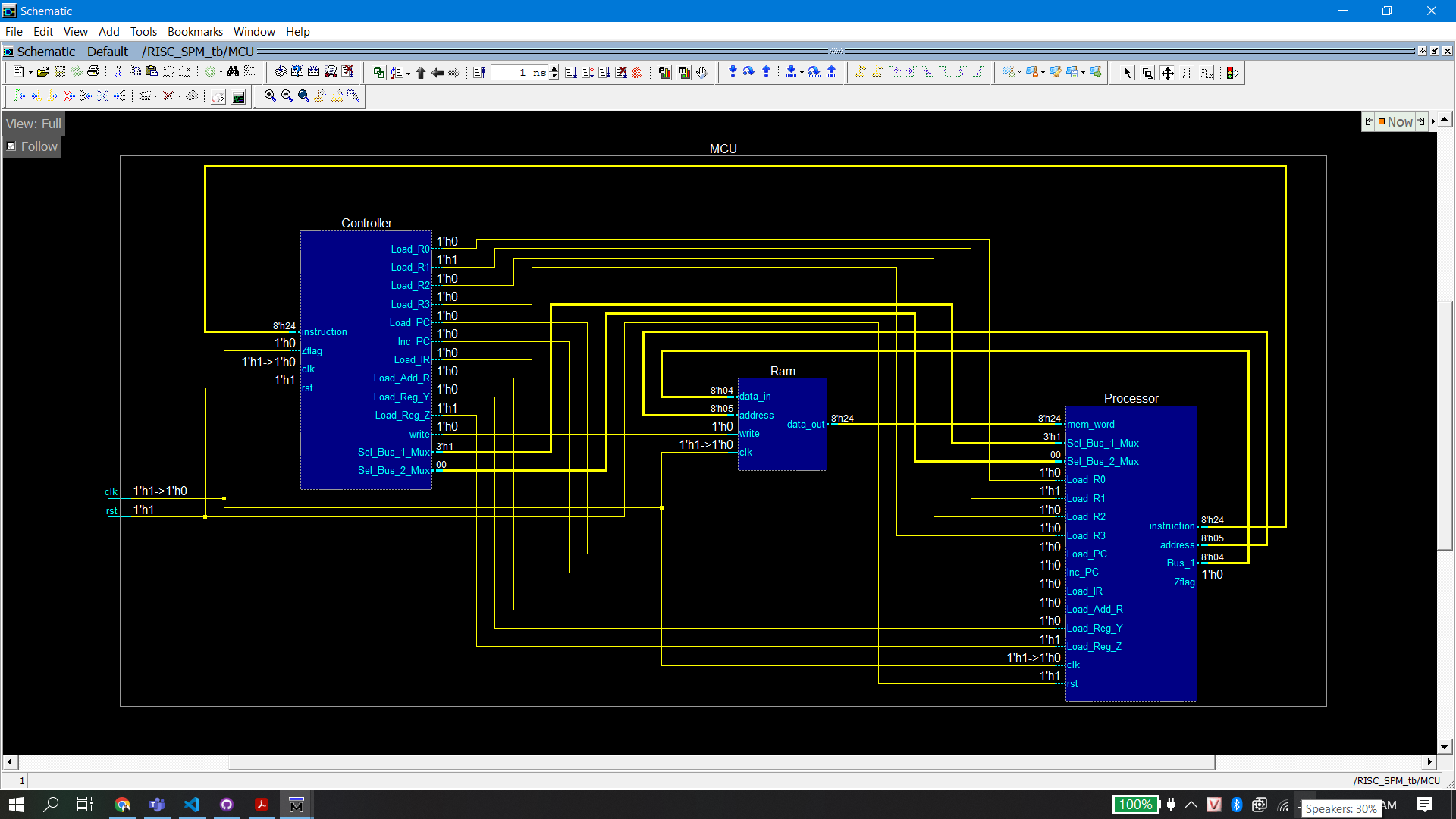
# Design verification

To ensure the working of the machine, each module has it own testbench : Memory Unit, Control Unit, Register Unit, Arithmetic Logic Unit.

### 1. Waveform



2.Schematic



### 3.Memory Data

