Computer Organization Project 2

Part I ：Implement a single cycle processor with R-format instructions

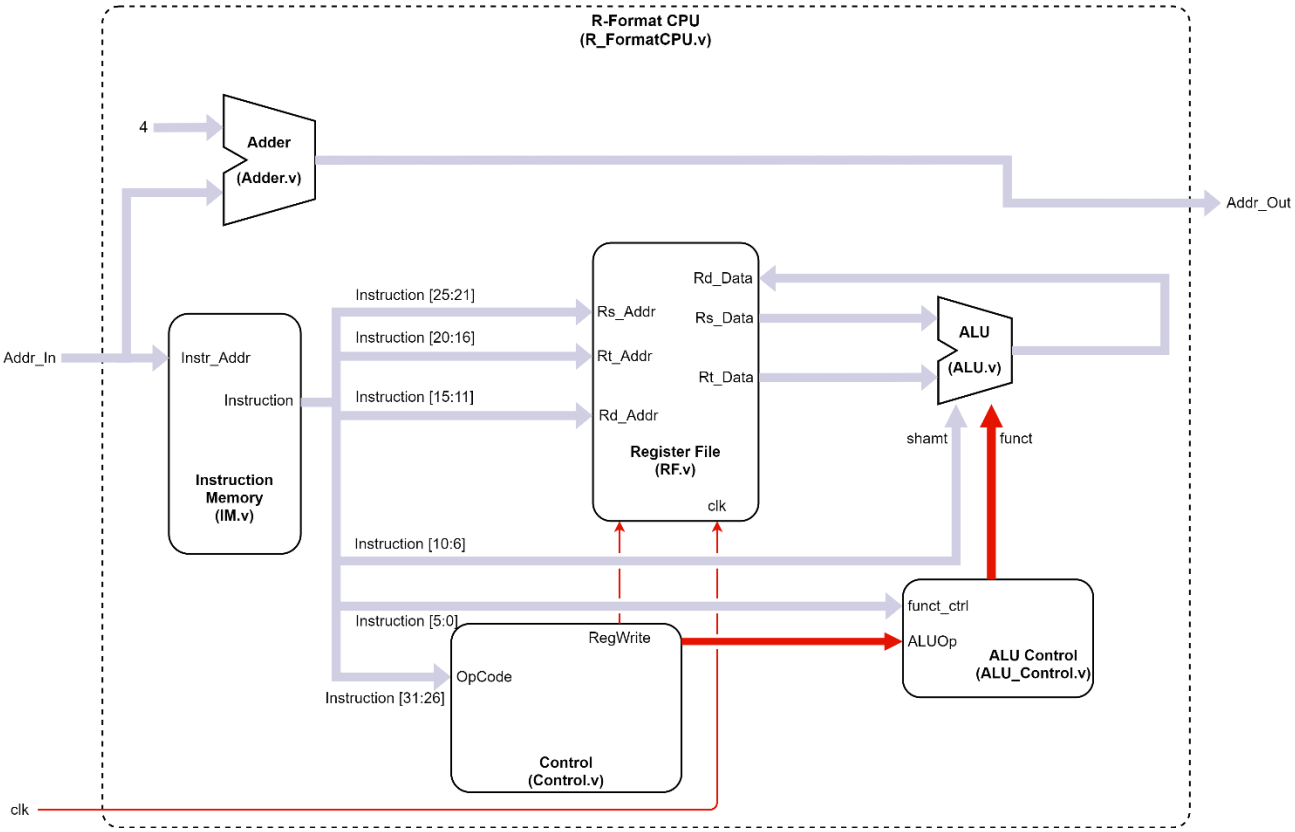


Figure 1：Architecture of a single cycle processor with R-format instructions

Implements a 32-bits processor and supports the following R-format instructions.

Instruction	Example	Meaning	OpCode	funct_ctrl	funct
Add unsigned	Addu $Rd, $Rs, $Rt	$Rd = $Rs + $Rt	000000	001011	001001
Sub unsigned	Subu $Rd, $Rs, $Rt	$Rd = $Rs – $Rt	000000	001101	001010
Or	Or $Rd, $Rs, $Rt	$Rd = $Rs | $Rt	000000	100101	010010
Shift right logical	Srl $Rd, $Rs, shamt	$Rd = $Rs >> shamt	000000	000010	100010

Note: Please refer to HW1 for the method of converting text instructions into 32-bits execution codes.

Note: When executing the R-format instruction, ALUOp is set as "10". Then, the ALU Control recognizes the "funct_ctrl" and converts the corresponding ALU function code "funct".

I/O Interface

```
module R_FormatCPU (
        output  wire    [31:0]  Addr_Out,
        input   wire    [31:0]  Addr_In,
        input   wire            clk
);
```

Part II ： Implement a single cycle processor with R-format and I-format instructions

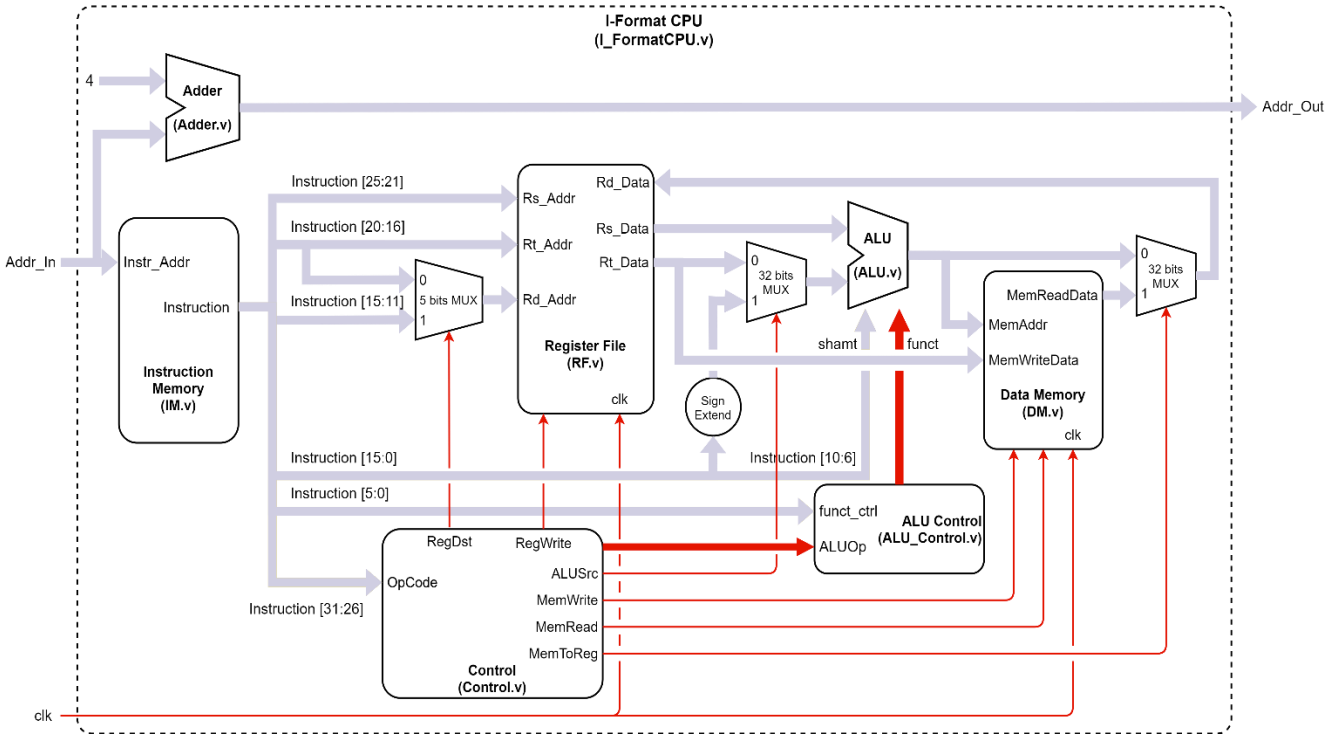


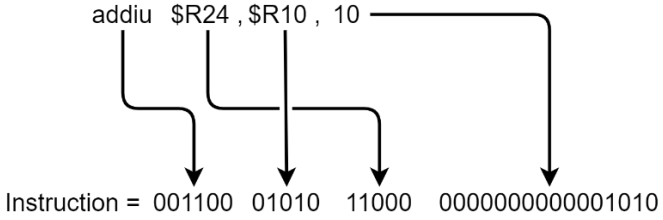
Figure 2：Architecture of a single cycle processor with R-format and I-format instructions

Implement a 32-bits processor that supports the R-format of the previous part and supports the following I-format instructions.

Instruction	Example	Meaning	OpCode	funct
Add.imm.unsigned	Addiu $Rt, $Rs, Imm.	$Rt = $Rs + Imm.	001100	001001
Sub.imm.unsigned	Subiu $Rt, $Rs, Imm.	$Rt = $Rs – Imm.	001101	001010
Store word	Sw $Rt, Imm. ($Rs)	Mem.[$Rs+Imm.] =$Rt	010000	001001
Load word	Lw $Rt, Imm. ($Rs)	$Rt =Mem.[$Rs+Imm.]	010001	001001

Note: When executing the I-format instruction, ALUOp is set as "00" or "01". Then, ALU Control ignores the "funct_ctrl", and triggers the ALU to perform "addition" or "subtraction" and outputs the corresponding "funct".

Convert Instruction to Binary



I/O Interface

```
module I_FormatCPU (
        output  wire    [31:0] Addr_Out,
        input   wire    [31:0] Addr_In,
        input   wire           clk
);
```

Part III ： Implement a single cycle processor with branch and jump instructions

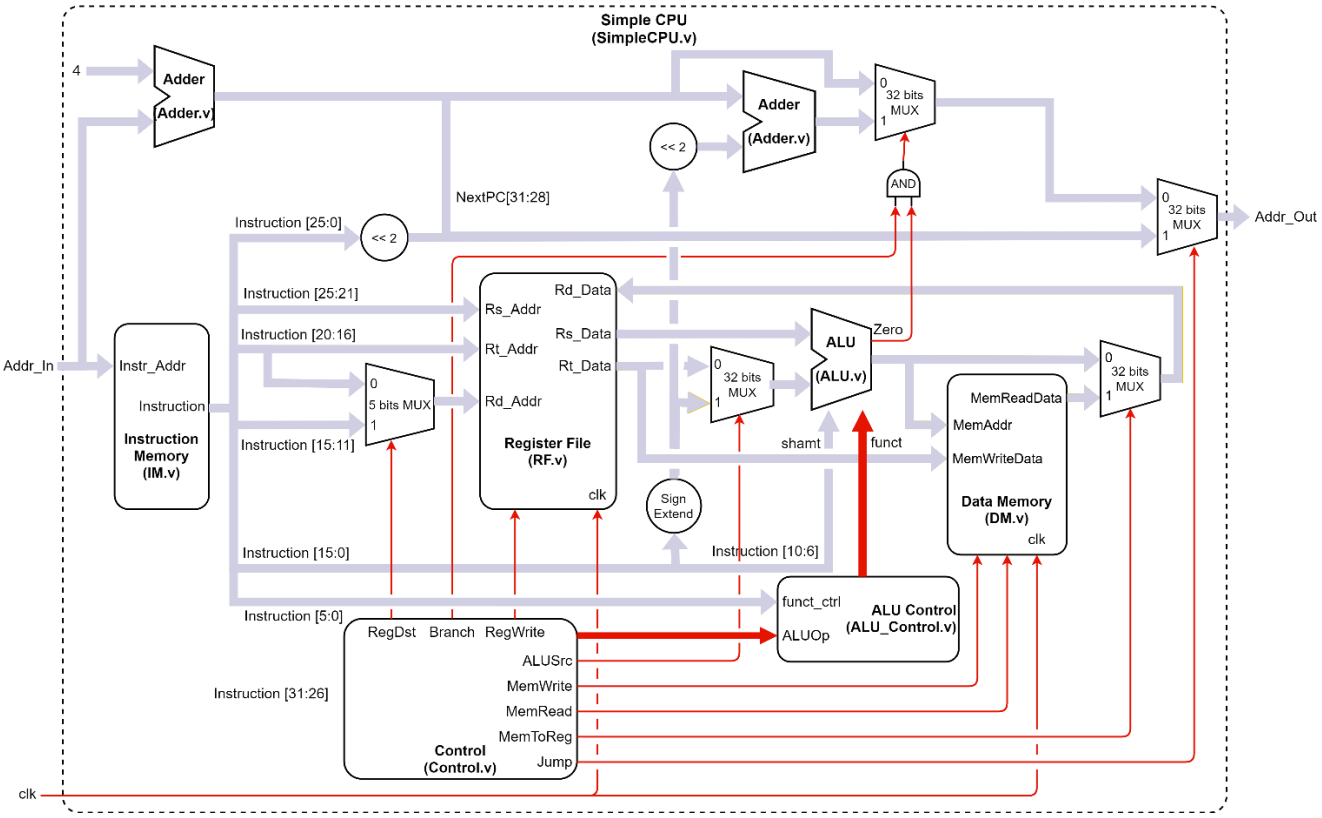


Figure 3：Architecture of a single cycle processor with branch and jump instructions

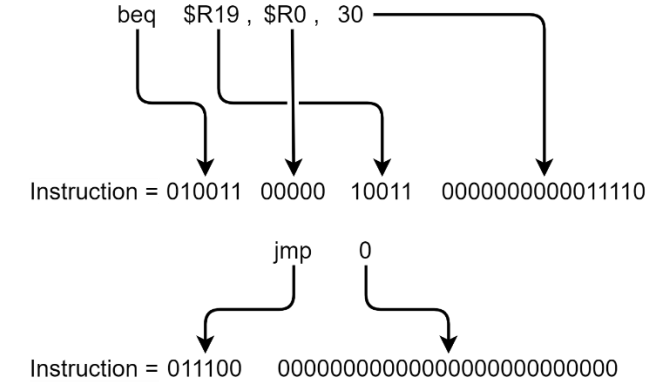
Implement a 32-bits processor that supports the first two parts of R-format and I-format, and supports the following branch and jump instructions.

Instruction	Example	Meaning	OpCode	funct
Branch on equal	Beq $Rs, $Rt, Imm	if ($Rs ≡ $Rt) Addr_Out = Addr_In + 4 + Imm.* 4	010011	001010
Jump	J Imm.	Addr_Out = NextPC[31:28] | Imm.* 4	011100	001010

Note: When executing the branch instruction, ALUOp is set as "01". Then, ALU Control ignores the "funct_ctrl", triggers the ALU to perform "subtraction" and outputs the corresponding "funct".

Note: NextPC = Addr_In + 4; " | " is OR operation.

Convert Instruction to Binary



I/O Interface

```
module SimpleCPU (
        output  wire    [31:0] Addr_Out,
        input   wire    [31:0] Addr_In,
        input   wire           clk
```

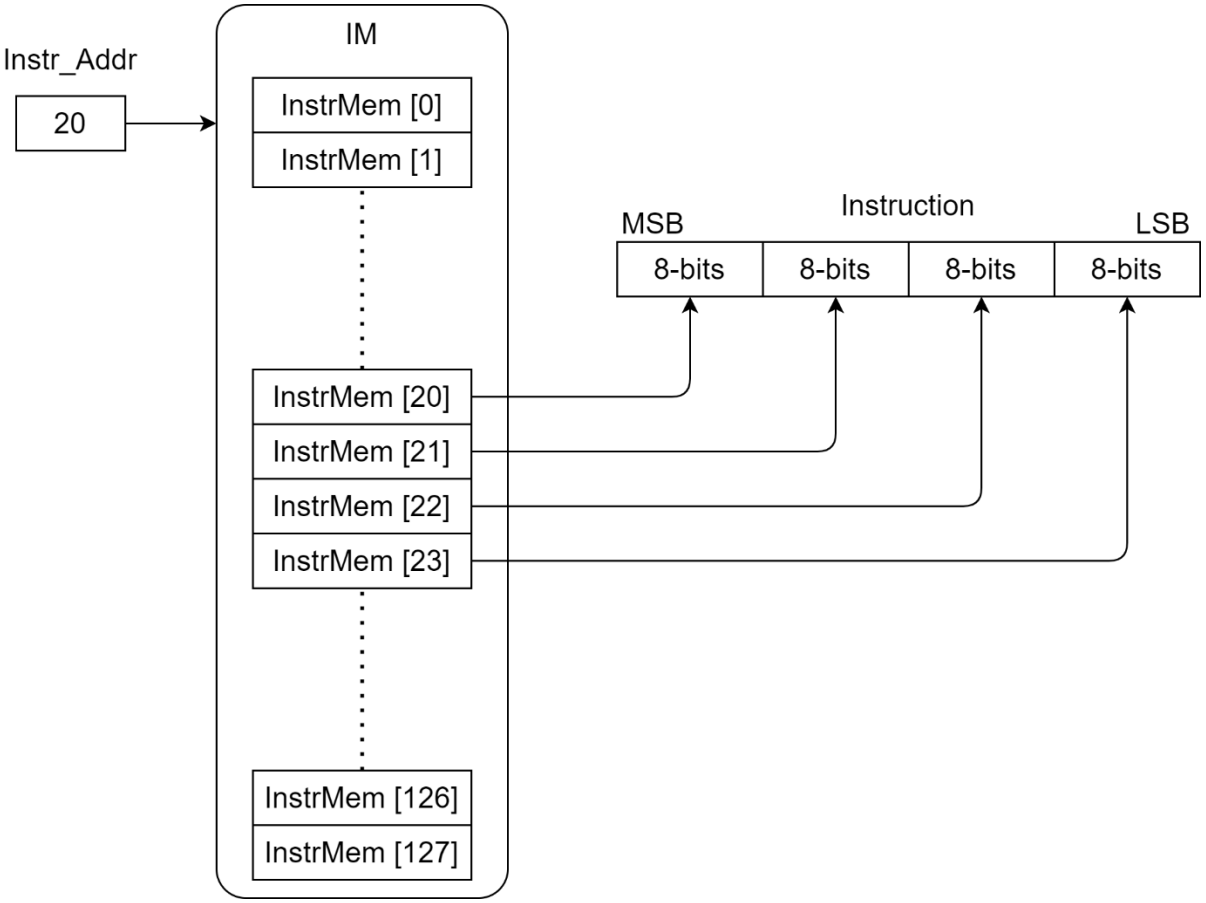
Register File

Different from HW1, the RF in this PA support dual-bus parallel read, and one write bus. The bit width and number of registers are the same as that in HW1, with 32 registers having a width of 32- bits. Its initial value is set by "/testbench/RF.dat".

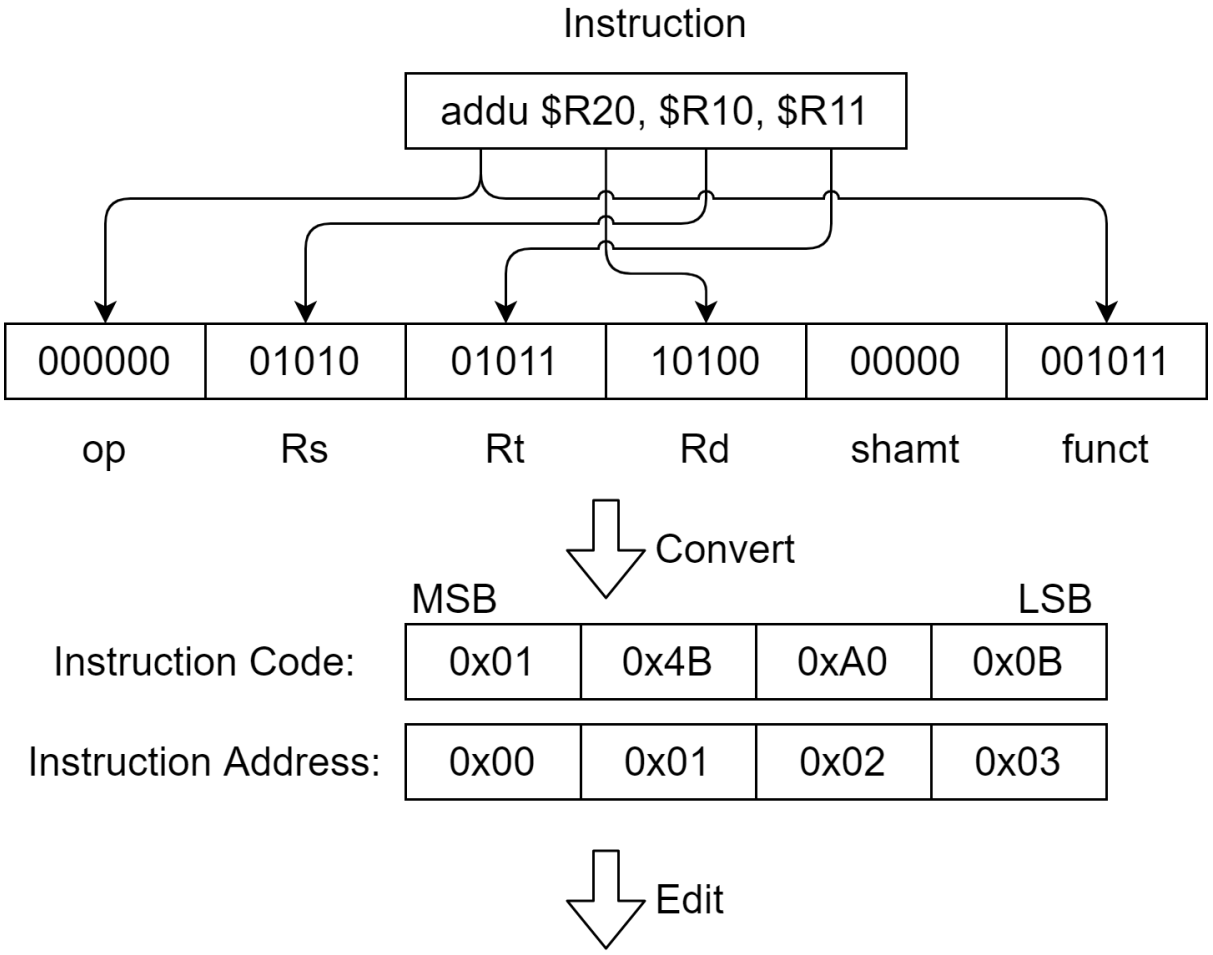
Instruction Memory

It consists of 128x8 bits memory with Big-endian. Its initial value is set by "/testbench/IM.dat".

a. Instruction reading



b. Command setting



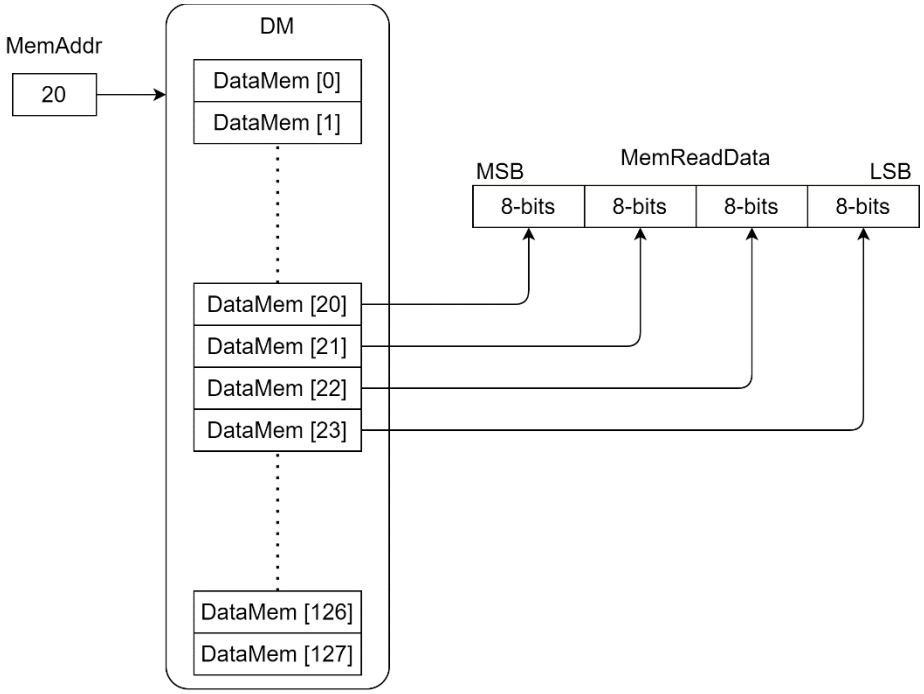
IM.dat:

```
// Instruction Memory in Hex
01      // Addr = 0x00
4B      // Addr = 0x01
A0      // Addr = 0x02
0B      // Addr = 0x03
01      // Addr = 0x04
AC      // Addr = 0x05
A8      // Addr = 0x06
0D      // Addr = 0x07
02      // Addr = 0x08
32      // Addr = 0x09
B0      // Addr = 0x0A
25      // Addr = 0x0B
01      // Addr = 0x0C
C0      // Addr = 0x0D
BA      // Addr = 0x0E
82      // Addr = 0x0F
FF      // Addr = 0x10
```

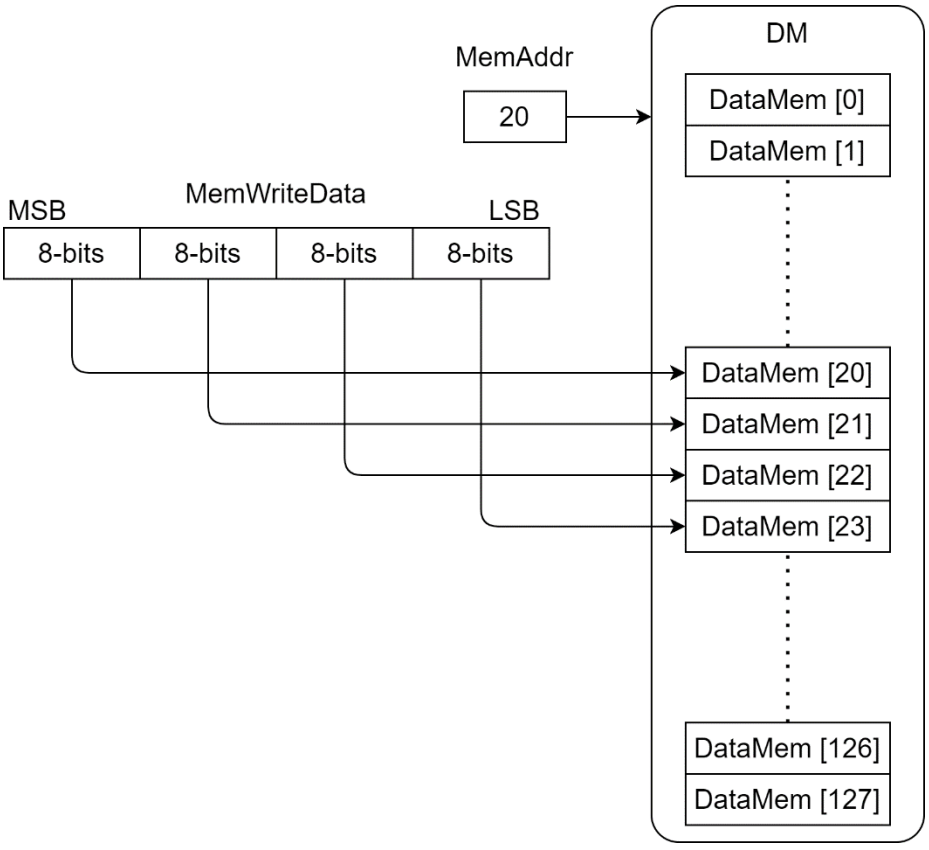
Data Memory

It consists of 128x8 bits memory with Big-endian. Its initial value is set by "/testbench/DM.dat".

a. Data reading



b. Data writing



Testbench Description

a. Initialize

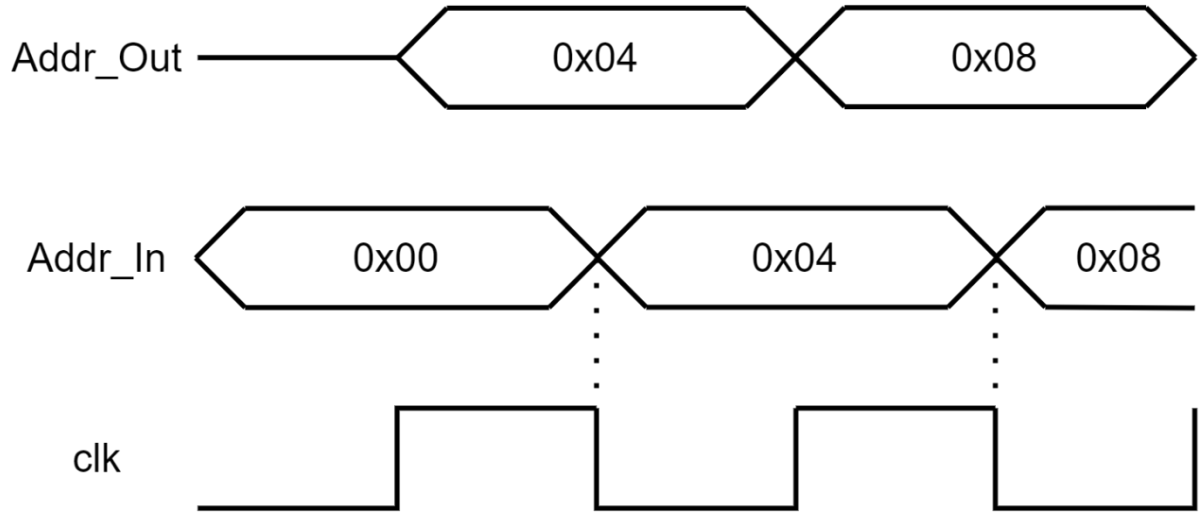
Execute Testbench ("tb_R_FormatCPU.v", "tb_I_FormatCPU.v", "tb_SimpleCPU.v") to initializes Instruction Memory, Register File, and Data Memory, respectively, according to "/testbench/IM.v", "/testbench/RF.v", "/testbench/ DM.v" (except Part I).

b. Clock

Generate a periodic clock (clk) to drive the CPU module in the testbench.

c. Addressing and Termination

Testbench initializes Addr_In signal to 0, and assigns Addr_Out to Addr_In before each positive edge of clk. When Addr_In is greater than or equal to the maximum address space of the current job instruction, Testbench ends the execution and outputs the current register and memory content ("/testbench/RF.out", "/testbench/DM.out"). The following figure shows the basic waveform of Testbench action:



Note: When the system Addr_Out fails or the program is in an infinite loop, please terminate the simulation and determine the problem manually.

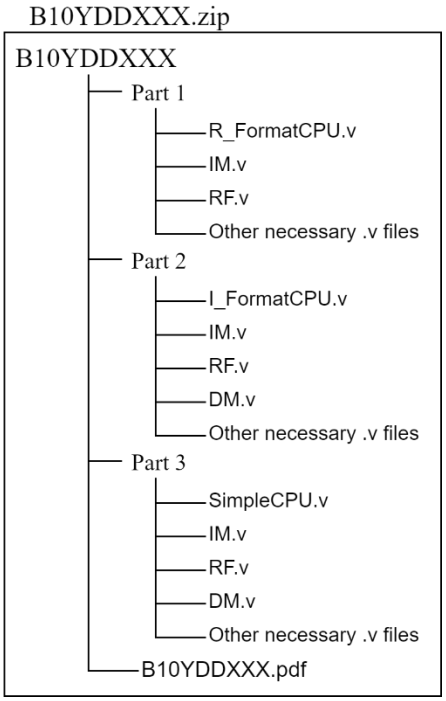
Submission

Report (B10YDDXXX.pdf) :

a. Cover
b. Screenshots and descriptions of each module code.
c. Screenshots and descriptions of the execution results ("/testbench/RF.out", "/testbench/DM.out") of the sample programs ("/testbench/IM.dat") in each part
d. Describes the custom test program and analyzes its results in each part.
e. Conclusion and insights.
f. Export it as a PDF format and name the file by the student ID - "B10YDDXXX.pdf ".

Compressed files (B10YDDXXX.zip):

- Report (B10YDDXXX.pdf)
- Part I
    a. R_FormatCPU.v
    b. IM.v
    c. RF.v
    d. Other necessary .v files
- Part II
    a. I_FormatCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. Other necessary .v files
- Part III
    a. SimpleCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. Other necessary .v files



Score :

1. Main program: Part I (30%), Part II (30%), Part III (10%). All programs are tested by an external testbench.
2. Screenshots of each program, and describe the process and method (10%).
3. The execution results of the sample programs in each part, screenshots, and explanations (10%).
4. Each part customizes the test program and explains and analyzes its results (5%).
5. Format/Conclusion/Completeness (5%).
6. No plagiarism.

Submission time: Upload to Moodle before 13:00 on 111/04/28