Computer Organization Project 2

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

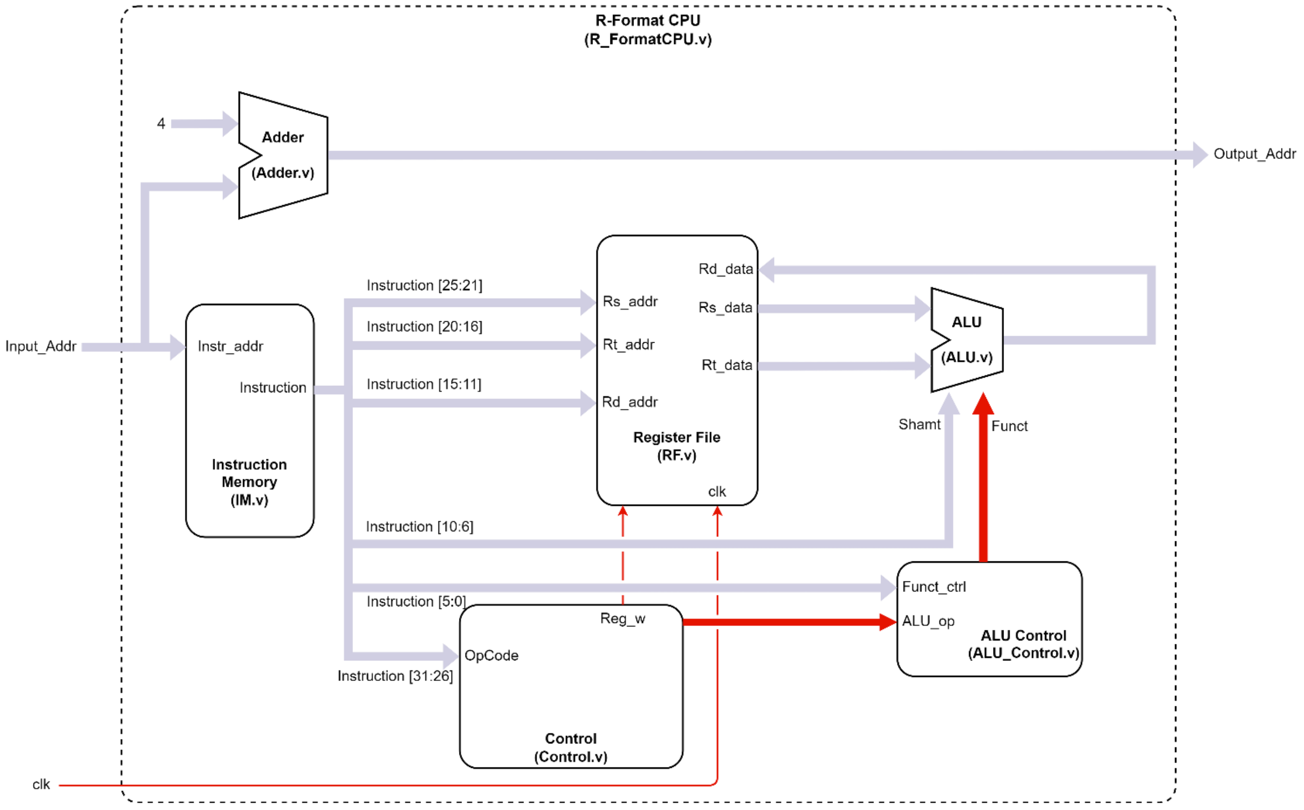


Figure 1：Architecture of a single cycle processor with R-format instructions

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
Shift left logical	Sll $Rd, $Rs, Shamt	$Rd = $Rs << Shamt	000000	100110	100001
Shift left logical variable	Sllv $Rd, $Rs, $Rt	$Rd = $Rs ≪ $Rt	000000	110110	110101

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

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

<u>I/O Interface</u>

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

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

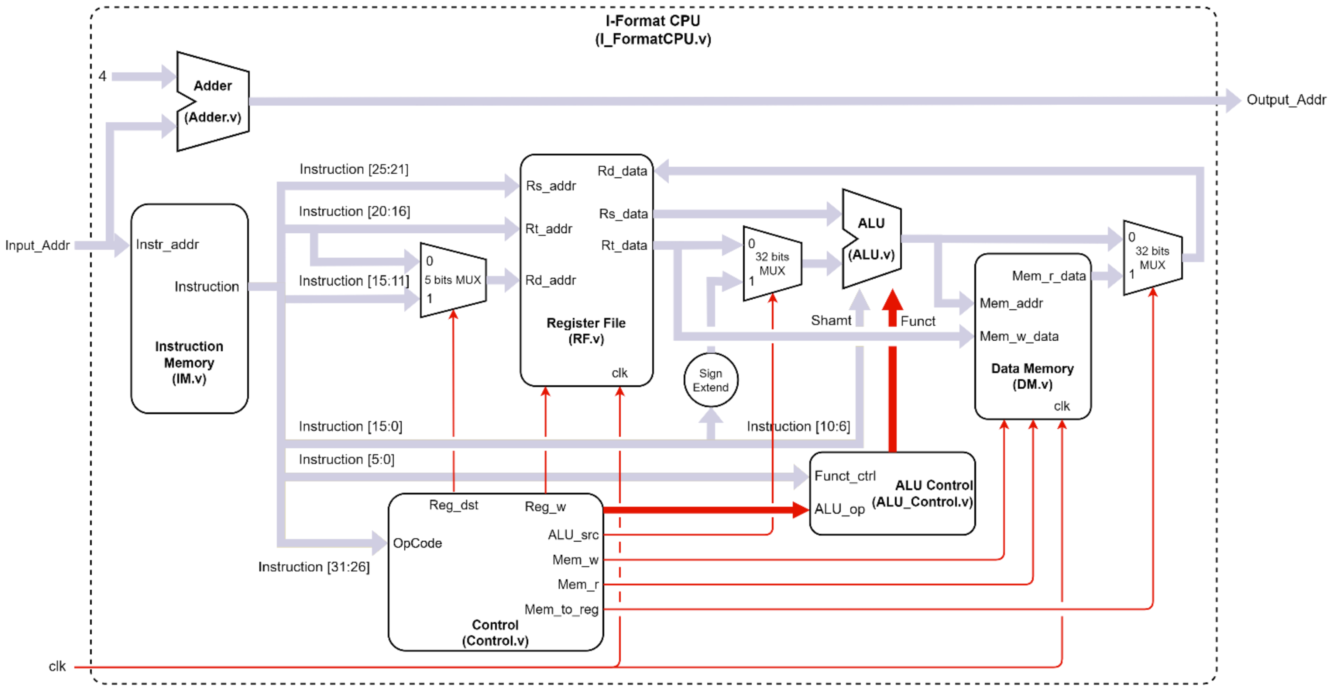


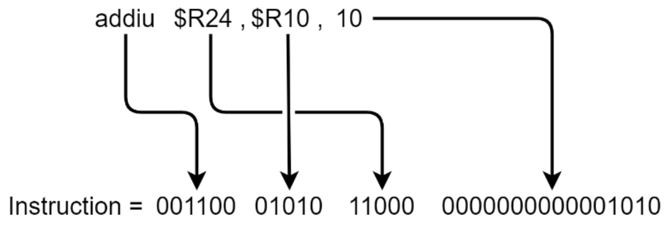
Figure 2：Architecture of a single cycle processor with R-format and I-format instructions

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
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
Set Less Than Imm	Slti $Rt, $Rs, Imm.	If $Rs < Imm. $Rt=1 Else $Rt=0	101010	101010

Note: When executing the I-format instruction, ALU_op is set as "00", "01", “11”. Then, ALU Control ignores the "Funct_ctrl", and triggers the ALU to perform "addition", "subtraction" or “set less than immediate value” and outputs the corresponding "Funct".

Convert Instruction to Binary



I/O Interface

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

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

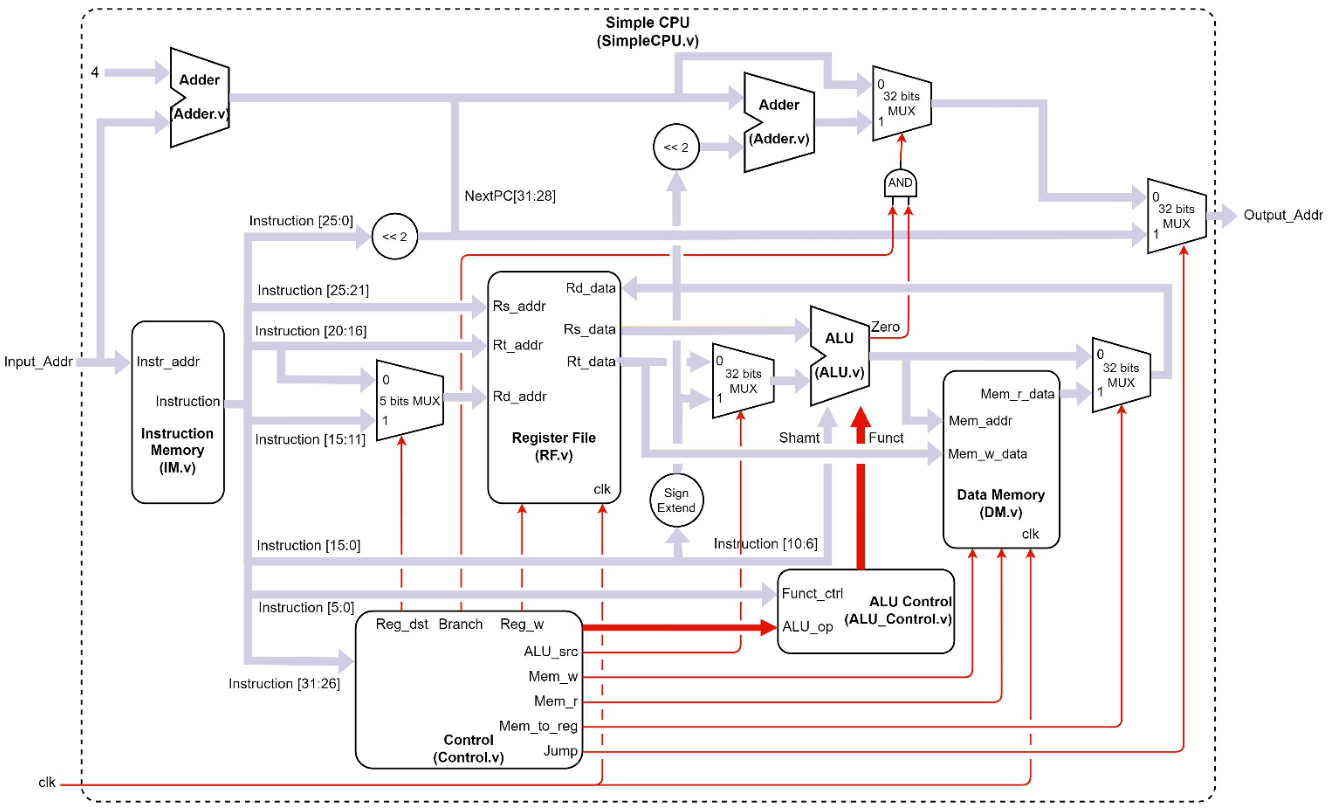


Figure 3：Architecture of a single cycle processor with branch and jump instructions

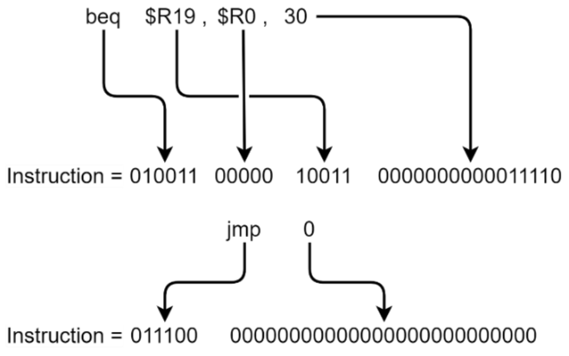
Implement a 32-bits processor that supports the first two parts of R-format and I-format, and supports the following branch and jump instructions.

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

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

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

Convert Instruction to Binary



beq $R19 , $R0 , 30
Instruction = 010011 00000 10011 0000000000011110
jmp 0
Instruction = 011100 00000000000000000000000000

I/O Interface

```
module SimpleCPU (
output  wire    [31:0] Output_Addr,
input   wire    [31:0] Input_Addr,
input   wire              clk
);
```

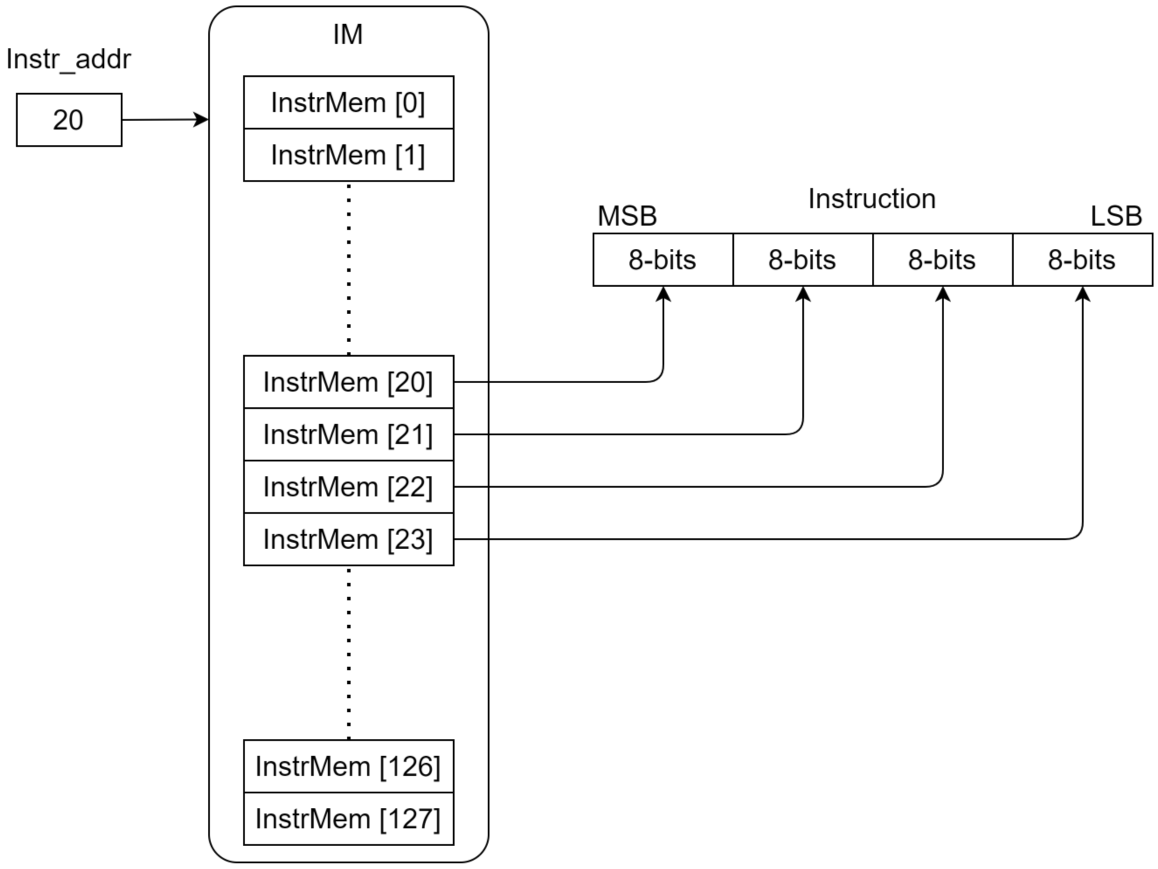
<u>Register File</u>

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".

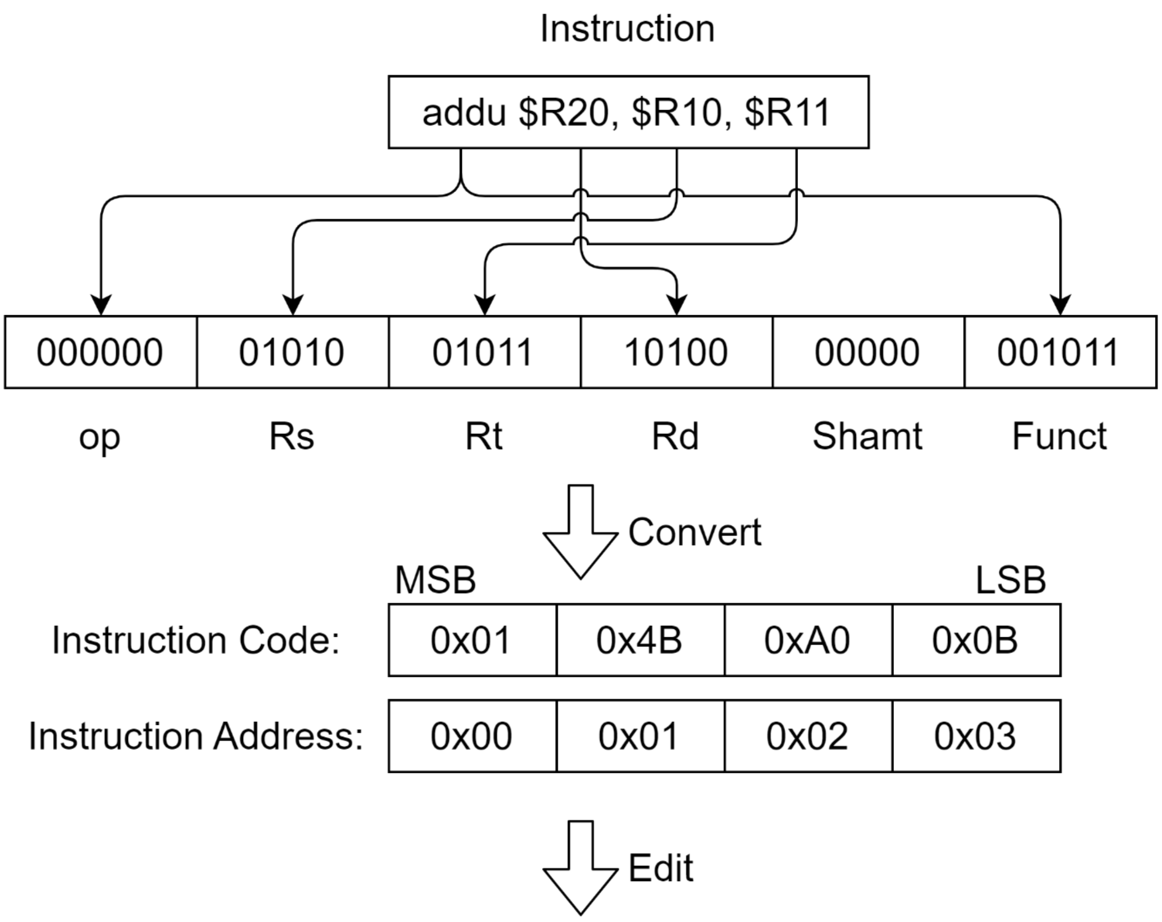
<u>Instruction Memory</u>

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

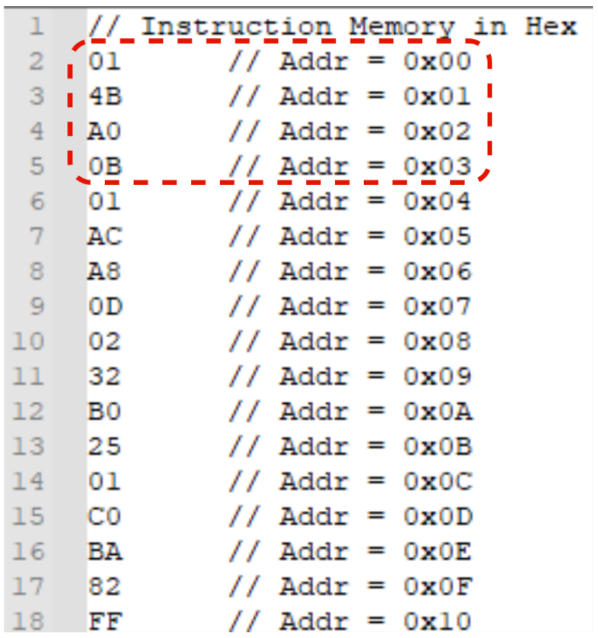
a. Instruction reading



b. Command setting



IM.dat:

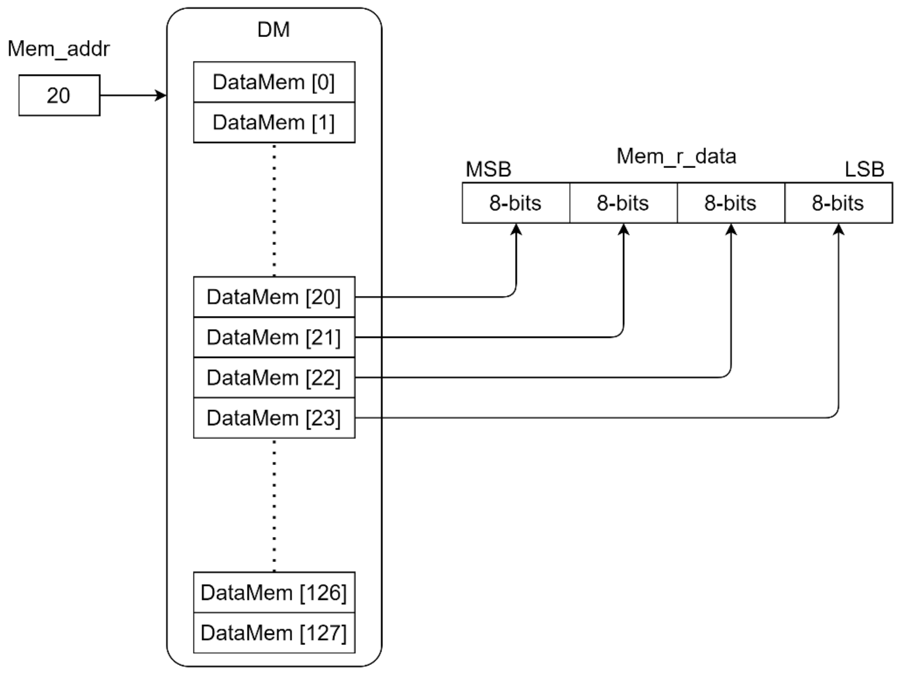


```
// Instruction Memory in Hex
01        // Addr = 0x00
4B        // Addr = 0x01
A0        // Addr = 0x02
0B        // Addr = 0x03
01        // Addr = 0x04
AC        // Addr = 0x05
A8        // Addr = 0x06
0D        // Addr = 0x07
02        // Addr = 0x08
32        // Addr = 0x09
B0        // Addr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

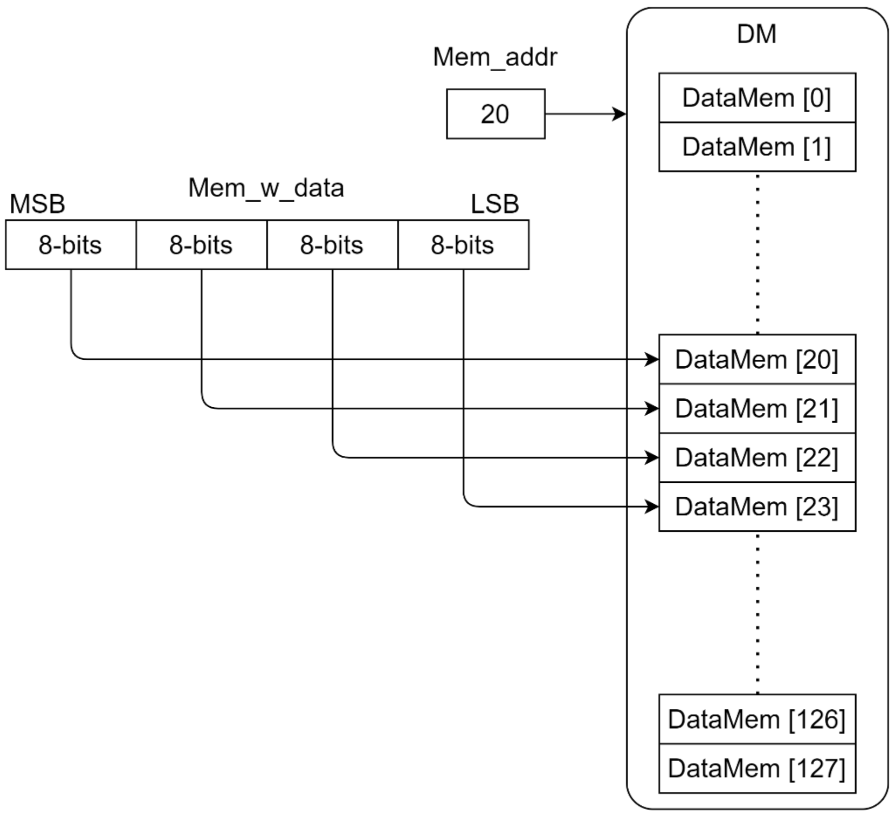
Data Memory

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

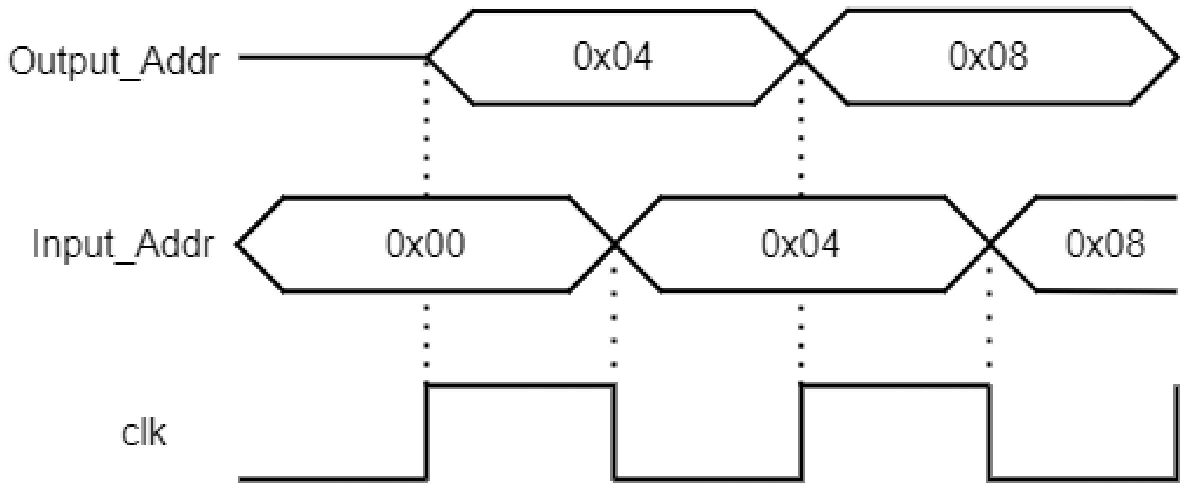
Execute Testbench ("tb_R_FormatCPU.v", "tb_I_FormatCPU.v", "tb_SimpleCPU.v") to initializes Instruction Memory, Register File, and Data Memory, respectively, according to "/testbench/IM.v", "/testbench/RF.v", "/testbench/ DM.v" (except Part I).

b. Clock

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

c. Addressing and Termination

Testbench initializes Input_Addr signal to 0, and assigns Output_Addr to Input_Addr before each positive edge of clk. When Input_Addr 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 Output_Addr fails or the program is in an infinite loop, please terminate the simulation and determine the problem manually.

<u>Submission</u>

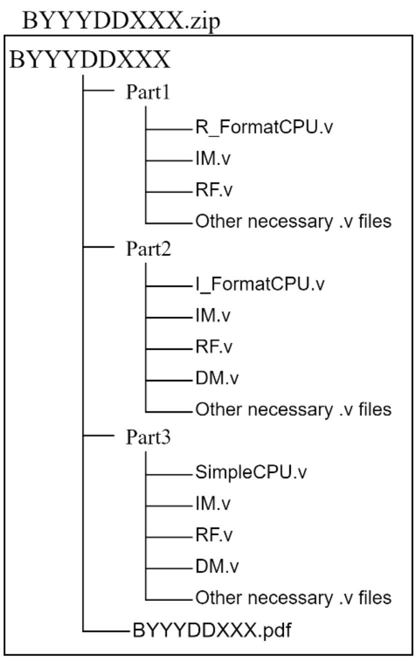
Report (BYYYDDXXX.pdf) :

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.

※ Convert the report to PDF and name it the student ID - "BYYYDDXXX.pdf".

Compressed files (BYYYDDXXX.zip):

- Report (BYYYDDXXX.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



- Note: Please ensure that all your program files and PDF files are directly placed in the zipped file, rather than being wrapped in a single folder.

Score :

1. Main program: Part I (32%), Part II (32%), Part III (16%). All programs are tested by an external testbench.
2. Screenshots of each module, and describe the process and method (10%).
3. The execution results of the sample programs in each part, screenshots, and explanations (10%).
4. No plagiarism.

Submission time: Upload to Moodle before 13:00 on 113/5/9