Lab Report

Experiment:	**Multi-cycle CPU Lab**
Department：	**Electrical Engineering**
Name:	**Xinyue Ma 20170765**
	Pengfei Gao 20206017

I INTRODUCTION

The central processing unit (CPU), as the computing and control core of a computer system, is the final execution unit for information processing and program operation. In this lab, we design a Multi-cycle CPU based on Verilog.

RISC-V is an open instruction set architecture (ISA) based on Reduced Instruction Set Computing (RISC). We aim to implement over 30 instructions of RISC-V in this lab. Compared with single-cycle CPU, multi-cycle CPU takes multiple cycles rather than a single cycle, different instructions may require different number of clock cycles, thus multi-cycle CPU has better performance.

II DESIGN

A CPU is composed of control units and data units. Data path perform data processing operations, registers and buses. Control path control the data path of the processor, it determines how to respond to the instruction that is fetched to the processor. In this lab, memory model, register file and clock signal is given, thus we just should implement other parts of CPU.

To implement the multi-cycle CPU, we use Finite State Machine (FSM) to divide the whole process into five stages: IF(START), ID, EX, WB, MEM, thus we need to set two 3-bit state variable (state and next state) to represent state transitions.

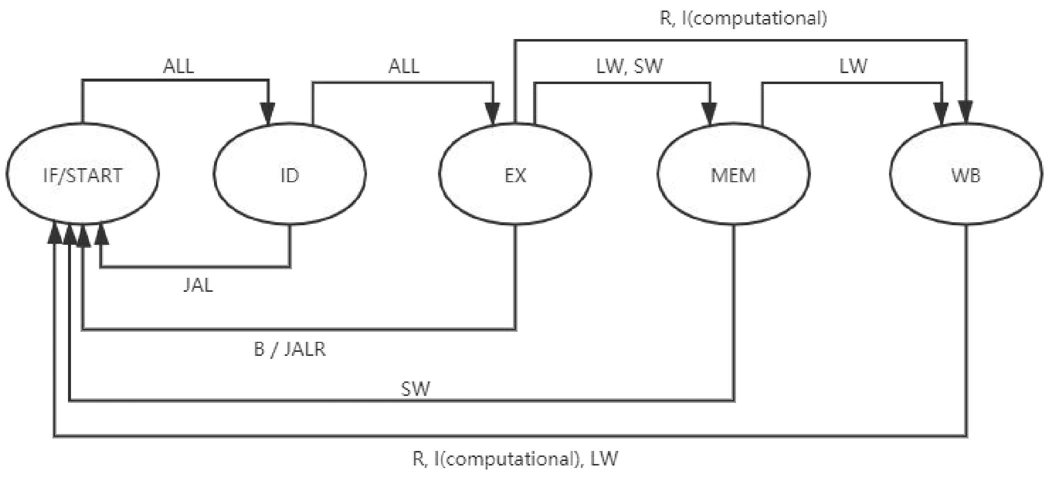


Figure 2.2 FSM of multi-cycle CPU

In IF stage, according to the instruction address in the program counter (PC), we take the instruction out from memory, and PC will automatically increase to gain the next instruction address.

In ID stage, we analyze and decode the instruction obtained in the fetch instruction stage, determine the specific operation that this instruction needs to complete, and generate the corresponding operation control signal which can be used to drive various operations in the execution state.

In EXE stage, the instruction action will be executed concretely according to the operation control signal.

In MEM stage, only LW, SW instructions can reach this stage, we write the data to the storage location specified by the data address in memory, or retrieves the data from the storage location.

Finally, in WB stage, the result of an instruction execution or data from the access memory will be written back to the appropriate destination register, and we set IF as the next stage.

State	R-type	I-type: computational	LW	SW	JAL	JALR	B
IF	ID	ID	ID	ID	ID	ID	ID
ID	EXE	EXE	EXE	EXE	IF	EXE	EXE
EXE	WB	WB	MEM	MEM		IF	IF
MEM			WB	IF			
WB	IF	IF	IF				

Figure 2.3 Transition table of Multi-cycle CPU

III IMPLEMENTATION

After getting the instruction address in PC, CPU will perform one stage in every single clock cycle according to the FSM figure. When one whole process finishes, state will be reset to IF stage to process next instruction. Memory is divided into instruction part(4KB) and data part(16KB). PC will be set as 0x000 initially and the initial value of stack pointer is 0xF00. When the instruction sequences is “0x00c00093” and “0x00008067”, we halt the program.

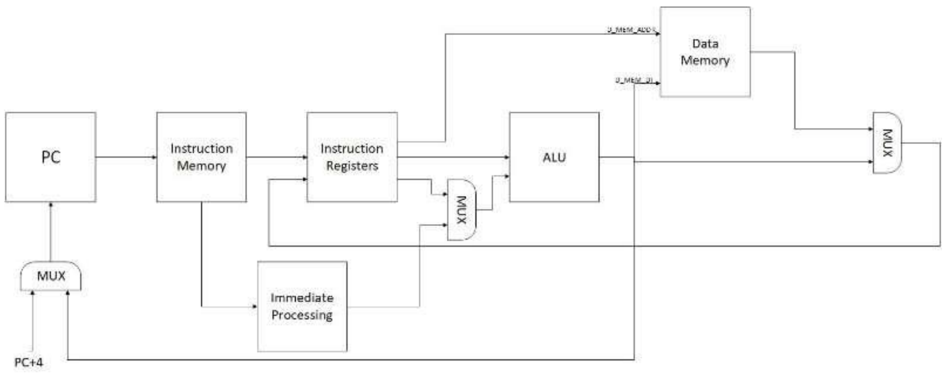
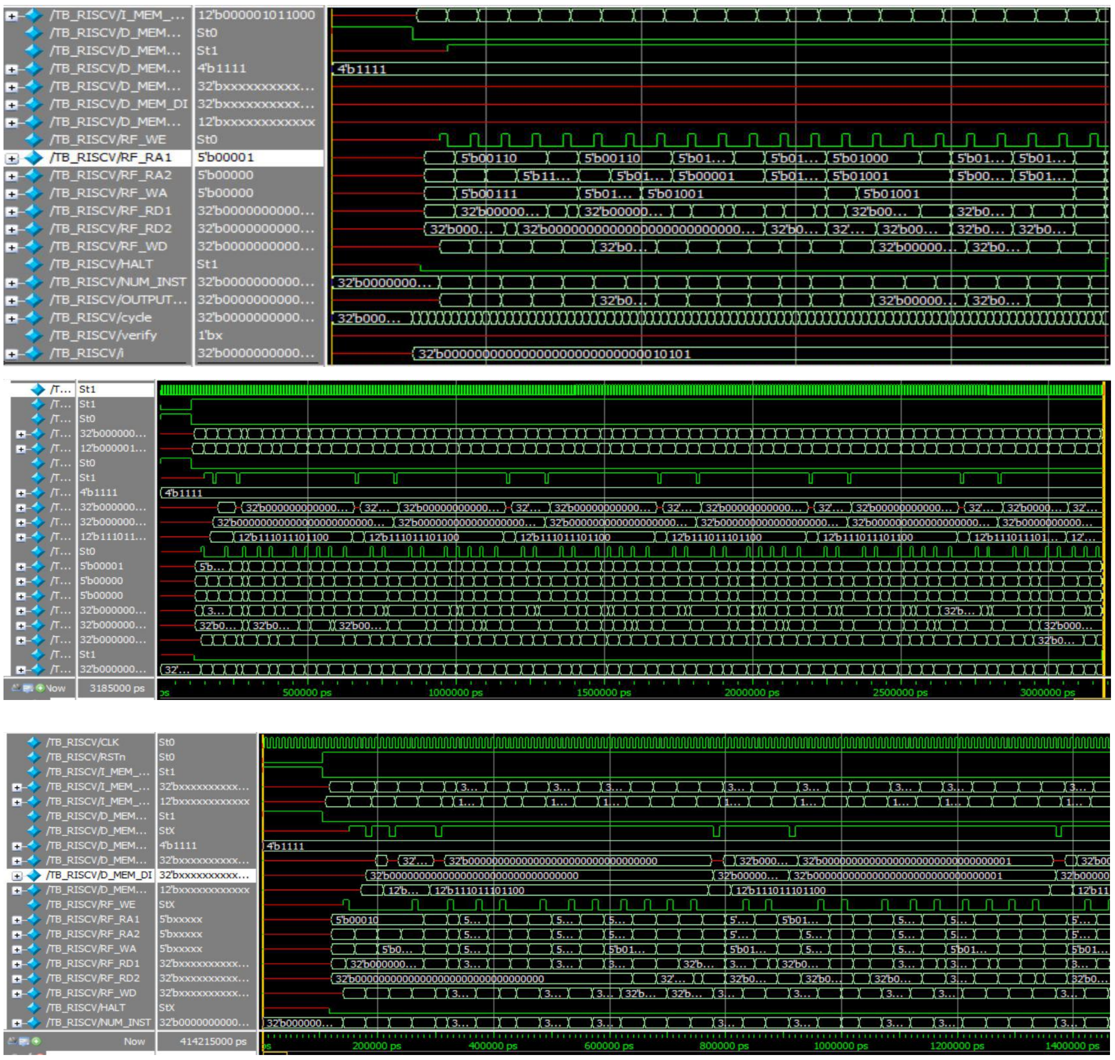


Figure 3.1 Overall design figure

IV EVALUATION

After finishing the code, we use the given Testbench files to evaluate the effectiveness. We used ModelSim to simulate our project, and according to the wave figure, all tests are pass.

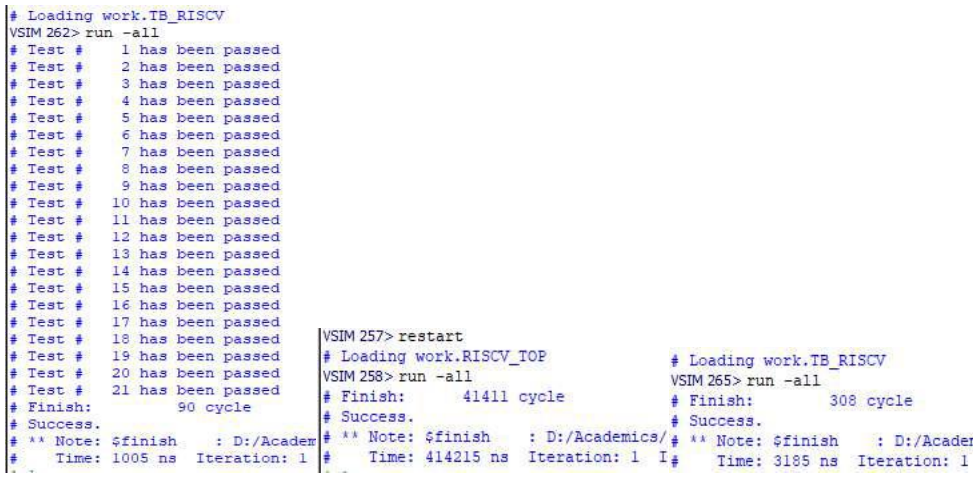


```
/TB_RISCV/I_MEM_...    12'b000001011000
/TB_RISCV/D_MEM...     St0
/TB_RISCV/D_MEM...     St1
/TB_RISCV/D_MEM...     4'b1111
/TB_RISCV/D_MEM...     32'bxxxxxxxxxxx...
/TB_RISCV/D_MEM_DI     32'bxxxxxxxxxxx...
/TB_RISCV/D_MEM...     12'bxxxxxxxxxxxx
/TB_RISCV/RF_WE        St0
/TB_RISCV/RF_RA1       5'b00001
/TB_RISCV/RF_RA2       5'b00000
/TB_RISCV/RF_WA        5'b00000
/TB_RISCV/RF_RD1       32'b0000000000...
/TB_RISCV/RF_RD2       32'b0000000000...
/TB_RISCV/RF_WD        32'b0000000000...
/TB_RISCV/HALT         St1
/TB_RISCV/NUM_INST     32'b0000000000...
/TB_RISCV/OUTPUT...    32'b0000000000...
/TB_RISCV/cycle        32'b0000000000...
/TB_RISCV/verify       1'bx
/TB_RISCV/i            32'b0000000000...
```

```
Now  3185000 ps
```

```
/TB_RISCV/CLK          St0
/TB_RISCV/RSTn         St0
/TB_RISCV/I_MEM_...    St1
/TB_RISCV/I_MEM_...    32'bxxxxxxxxxxx...
/TB_RISCV/I_MEM_...    12'bxxxxxxxxxxxx
/TB_RISCV/D_MEM...     St1
/TB_RISCV/D_MEM...     StX
/TB_RISCV/D_MEM...     4'b1111
/TB_RISCV/D_MEM...     32'bxxxxxxxxxxx...
/TB_RISCV/D_MEM_DI     32'bxxxxxxxxxxx...
/TB_RISCV/D_MEM...     12'bxxxxxxxxxxxx
/TB_RISCV/RF_WE        StX
/TB_RISCV/RF_RA1       5'bxxxxx
/TB_RISCV/RF_RA2       5'bxxxxx
/TB_RISCV/RF_WA        5'bxxxxx
/TB_RISCV/RF_RD1       32'bxxxxxxxxxxx...
/TB_RISCV/RF_RD2       32'bxxxxxxxxxxx...
/TB_RISCV/RF_WD        32'bxxxxxxxxxxx...
/TB_RISCV/HALT         StX
/TB_RISCV/NUM_INST     32'b0000000000...
Now  414215000 ps
```

Figure 5.1 Waveform of the Test(Partial)



```
# Loading work.TB_RISCV
VSIM 262> run -all
# Test #     1 has been passed
# Test #     2 has been passed
# Test #     3 has been passed
# Test #     4 has been passed
# Test #     5 has been passed
# Test #     6 has been passed
# Test #     7 has been passed
# Test #     8 has been passed
# Test #     9 has been passed
# Test #    10 has been passed
# Test #    11 has been passed
# Test #    12 has been passed
# Test #    13 has been passed
# Test #    14 has been passed
# Test #    15 has been passed
# Test #    16 has been passed
# Test #    17 has been passed
# Test #    18 has been passed
# Test #    19 has been passed
# Test #    20 has been passed
# Test #    21 has been passed
# Finish:          90 cycle
# Success.
# ** Note: $finish    : D:/Academ
#    Time: 1005 ns  Iteration: 1
```

```
VSIM 257> restart
# Loading work.RISCV_TOP
VSIM 258> run -all
# Finish:       41411 cycle
# Success.
# ** Note: $finish    : D:/Academics/
#    Time: 414215 ns  Iteration: 1  I
```

```
# Loading work.TB_RISCV
VSIM 265> run -all
# Finish:         308 cycle
# Success.
# ** Note: $finish    : D:/Acade
#    Time: 3185 ns  Iteration: 1
```

Figure 5.2 Results of the Test

V DISCUSSION

Due to the fact that we had mid-term exam last week, the time is a little tight. Because the CPU is synchronous, so we just put all things into one module instead of dividing them into several modules like lab 3.

VI CONCLUSION

In this lab, we make a VISC-V Multi-Cycle CPU based on Verilog. After simulation, every given test was passed. Thus, the lab achieves the expected result successfully.