UM–SJTU Joint Institute

Introduction to Computer Organization
Ve370

Laboratory Report

Lab 4

Pipelined Processor

Name: Chunyu Wang Weiqing Xu Wenbo Yu

Date: November 9, 2022

1 Brief Description of Our Modeling and Implementation

Our basic idea is to modify the pipelined processor we have learned in class, as shown in the figure below:

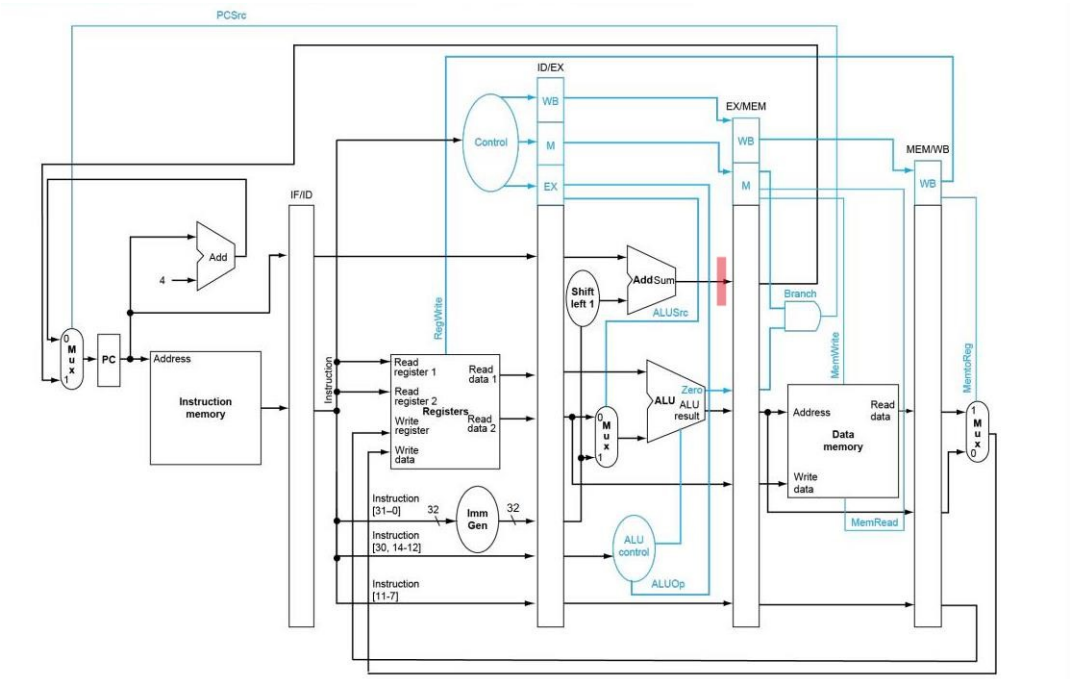

Figure 1: Caption

To support all the instructions listed in lab manual, we add:

1. a new control signal isJump generated by central control, 1 when jalr, otherwise 0.
2. a 2-to-1 mux in EX stage. The position of the mux is shown in the figure (red thick line). The mux takes the adder result (PC+Imm*2) and ALU result as input data, the jump signal from ID/EX register as selecting signal. The function of this mux is for jalr, since jalr need jump of PC but, as an I-type instruction, the destination can only be calculated by ALU.
3. 3 bit wire of funct3 to Data Memory, in order to know whether the load or save data is word or byte, or unsigned.
4. a new input of mux in WB stage, in order to save the next PC when jal and jalr.

For other detailed modification, we just need to set the control signal:

/	Branch	ALUSrc	ALUOp	MemWrite	MemRead	MemtoReg	Jump	RegWrite
I-type:load	0	1	00	0	1	01	0	1
I-type:Imm	0	1	11	0	0	00	0	1
S-type	0	1	00	1	0	00	0	0
B-type	1	0	01	0	0	00	0	0
jalr	1	1	00	0	0	10	1	1
J-type	1	1	00	0	0	10	0	1
R-type	0	0	10	0	0	00	0	1

ALUsel	ALU operation	instruction	Zero
0010	x1+x2	add,jal,jalr...	1
1000	x1-x2	sub,beq	(x1==x2)
1001	x1-x2	bne	!(x1==x2)
1100	x1-x2	blt	(x1<x2)
1110	x1-x2	bge	!(x1<x2)
0100	x1^x2	xor	/
0110	x1|x2	or	/
0111	x1&x2	and	/
0001	x1<<x2	sll	/
0101	x1>>x2	srl	/
1101	x1>>x2 (sign extension)	sra	/

2 Different Types of Instructions

2.1 addi

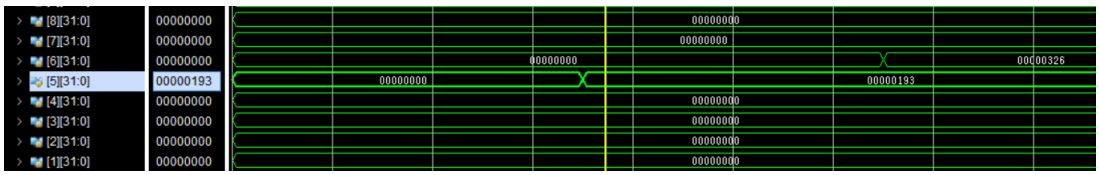

Figure 2: Result 1.

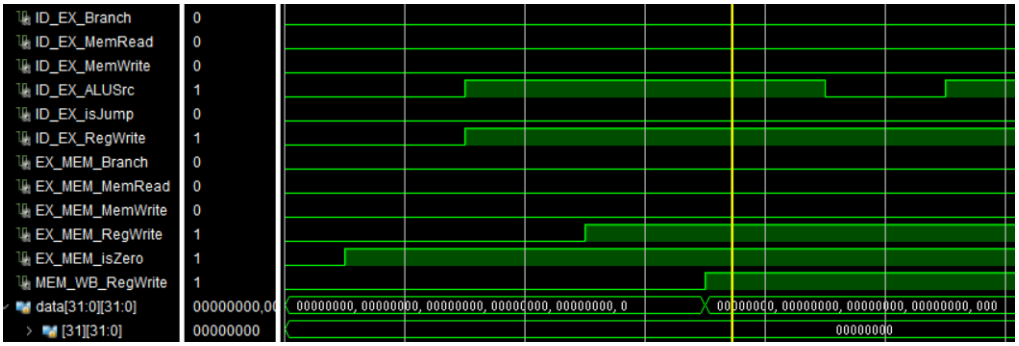

Figure 3: Result 2.

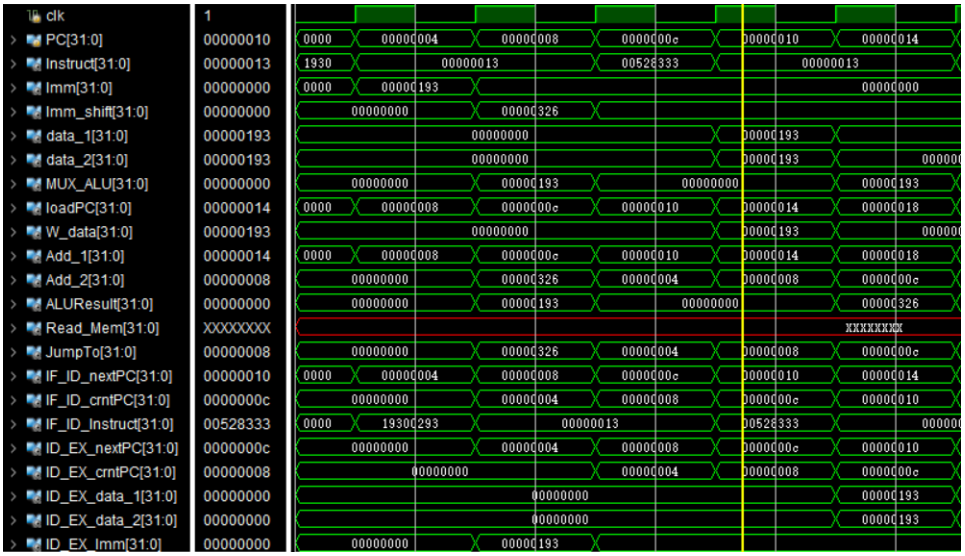

Figure 4: Result 3.

When PC = 0x0, it's addi. Control signals are generated and transmitted to pipeline registers. 0x193 is first recorded by pipeline register, and then being transmitted to ALU. The result would be recorded by register file after 5 clock cycles in total (note that in this case, the first clock cycle is clk=0).

2.2 add

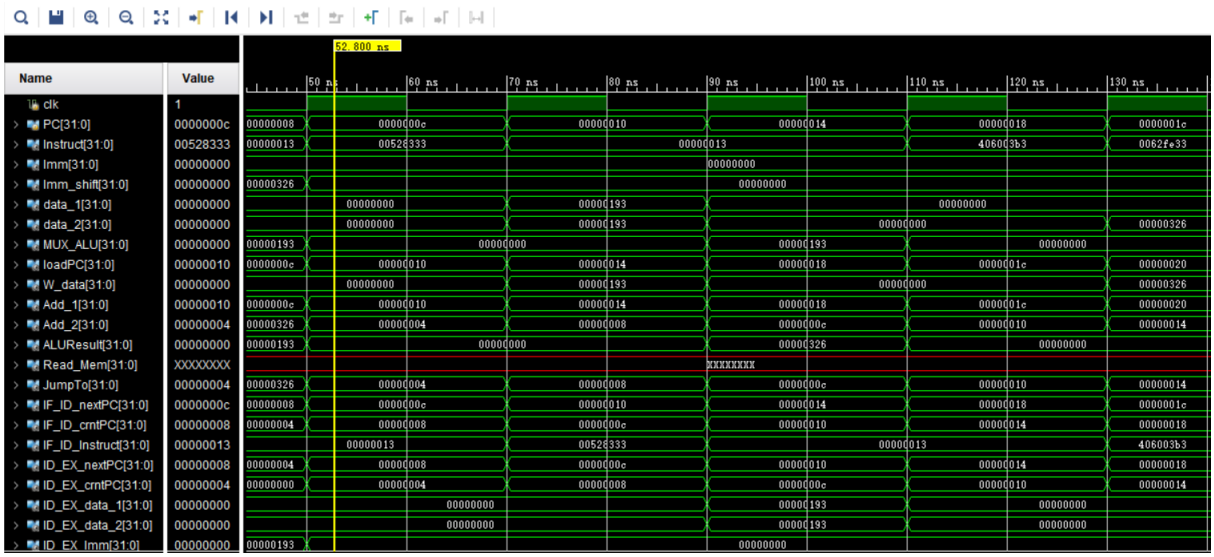

Figure 5: Result 1.

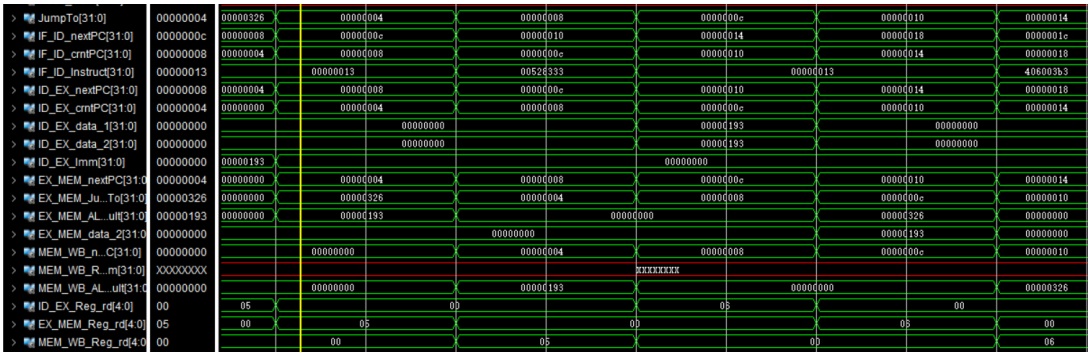

Figure 6: Result 2.

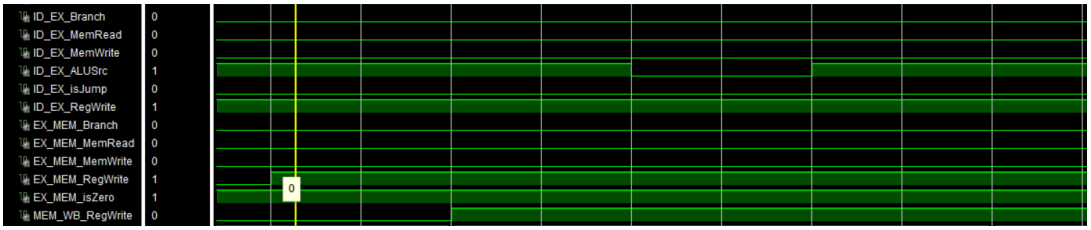

Figure 7: Result 3.

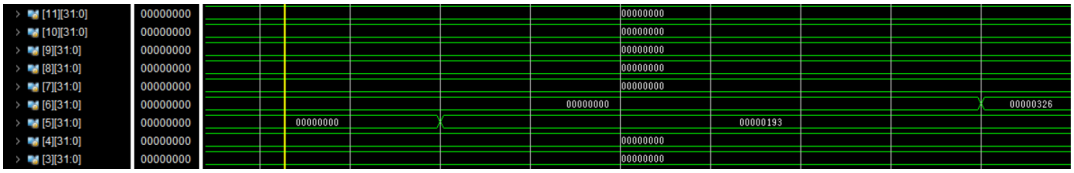

Figure 8: Result 4.

When PC = 0xc, it's add. Control signals go into ID/EX pipeline register. Rs1 and Rs2 are given, and the contents would go through ID/EX pipeline register and be added in ALU. The sum would be recorded by register file in x6.

2.3 sw

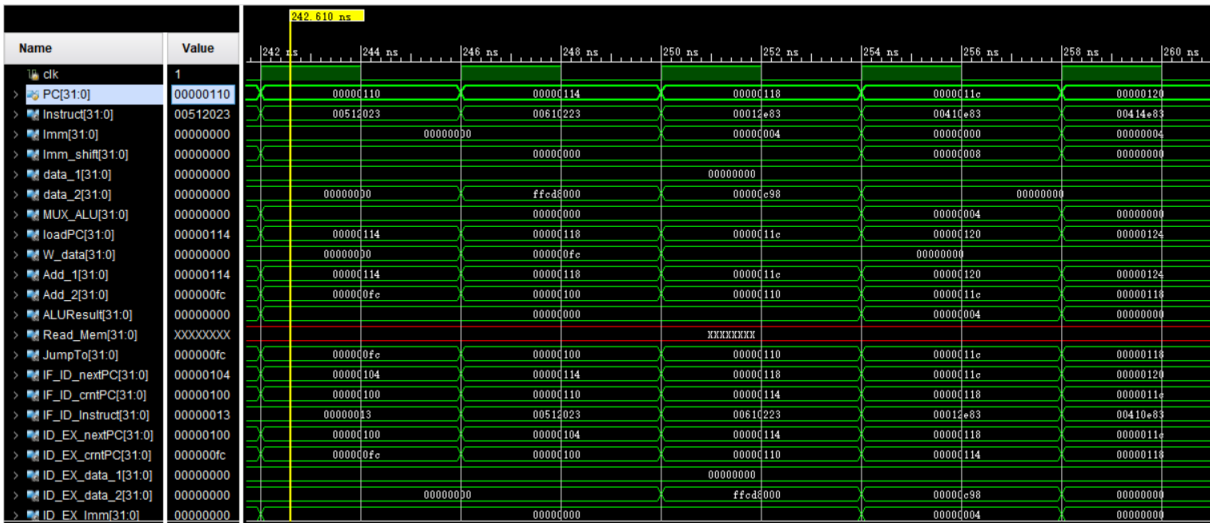

Figure 9: Result 1.

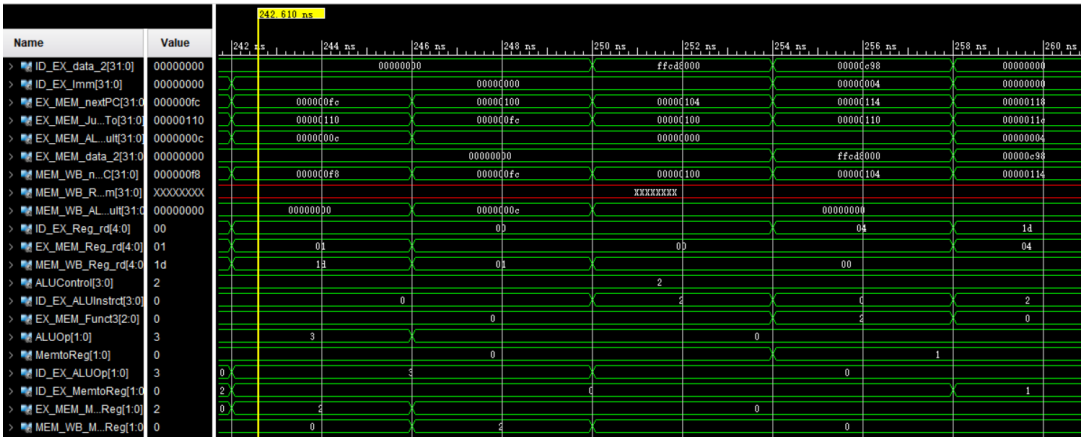

Figure 10: Result 2.

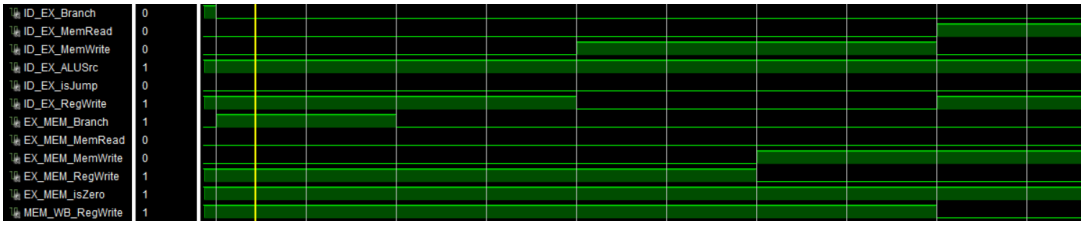

Figure 11: Result 3.

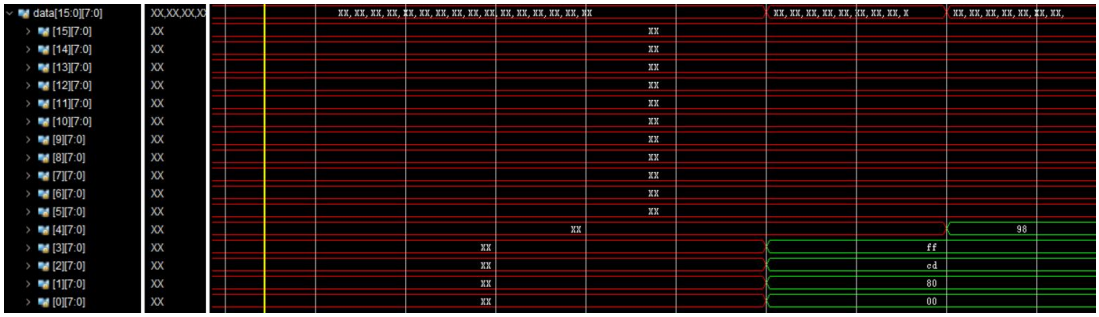

Figure 12: Result 4.

When PC = 0x110, it's sw. It reads data in x5 and x2, and then save the data of x5 into the address indicated by x2. In total, it takes only 4 clock cycles since the result doesn't need to be stored in register file.

2.4 lw

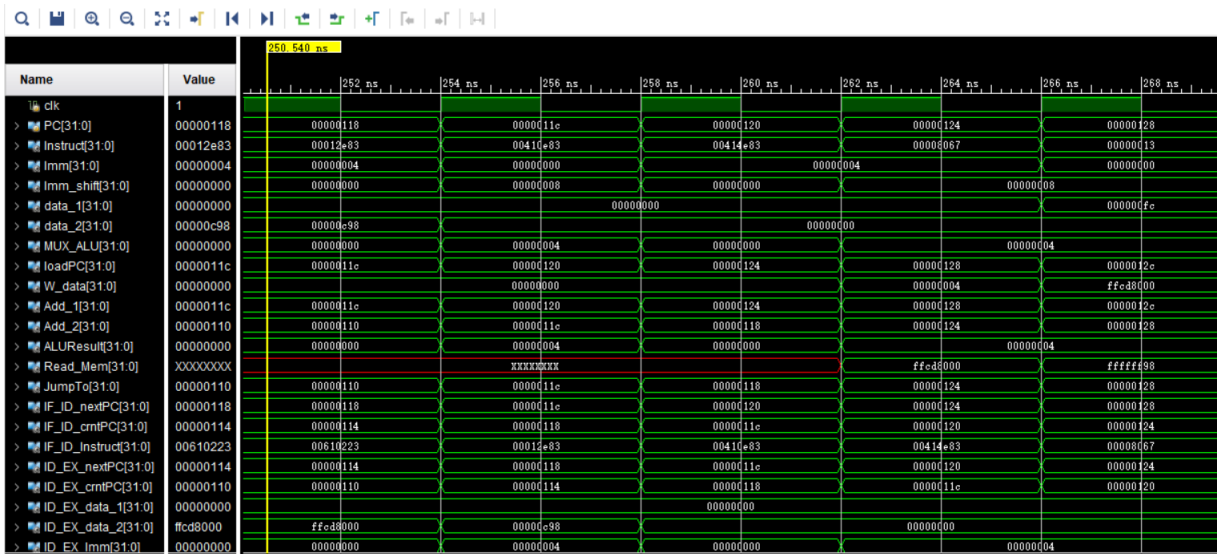

Figure 13: Result 1.

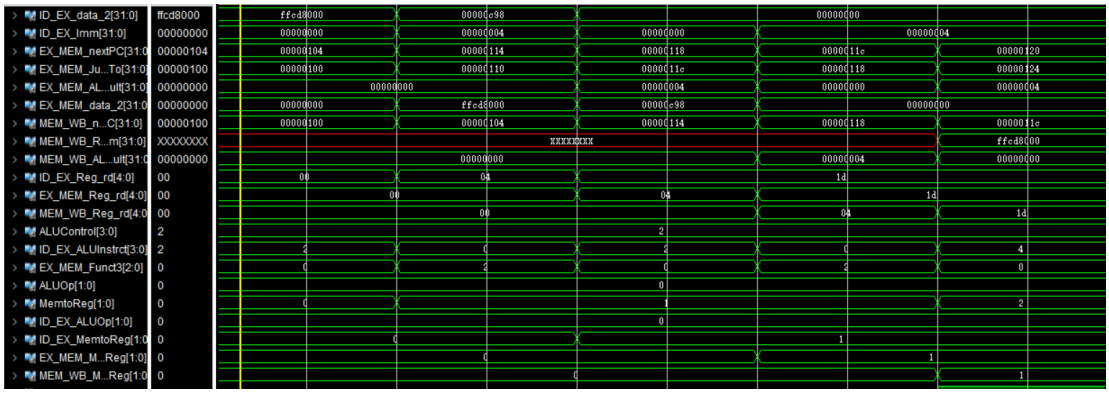

Figure 14: Result 2.

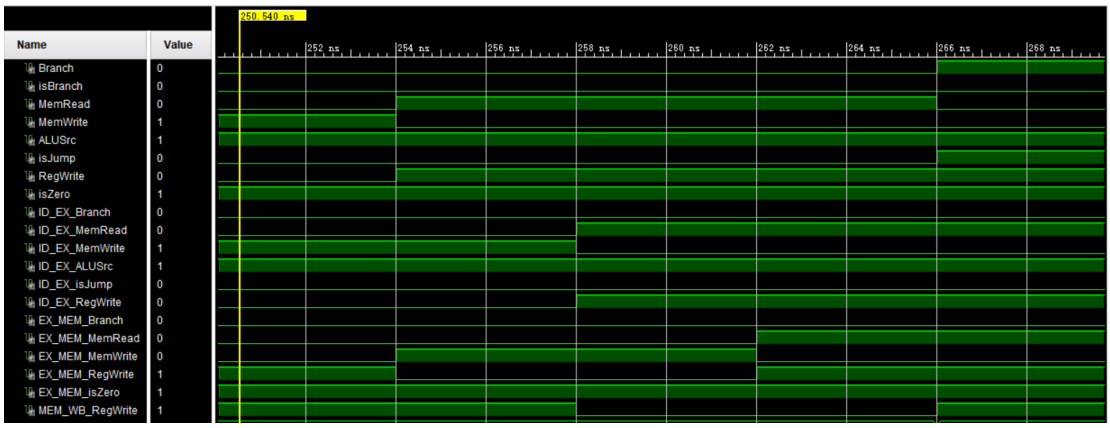

Figure 15: Result 3.

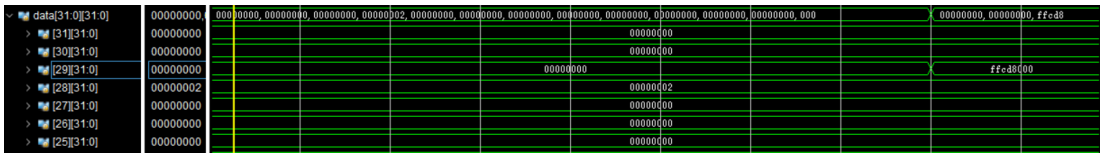

Figure 16: Result 4.

When PC = 0x118, it's lw. It reads the address in x2, and update it into x29. This instruction takes 5 clock cycles since the data read from DataMemory need to be recorded into register file.

2.5 beq

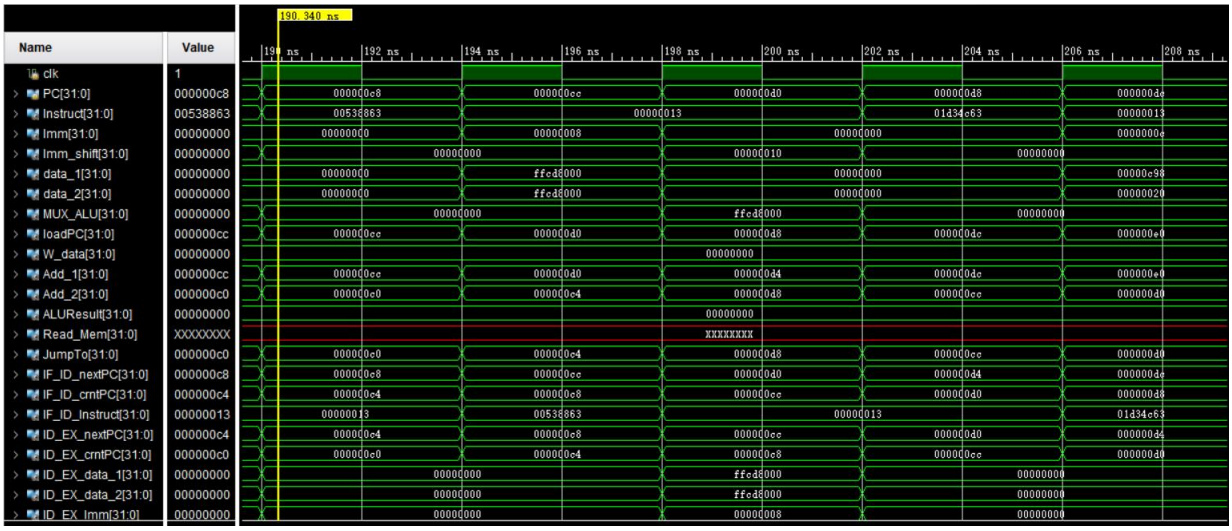

Figure 17: Result 1.

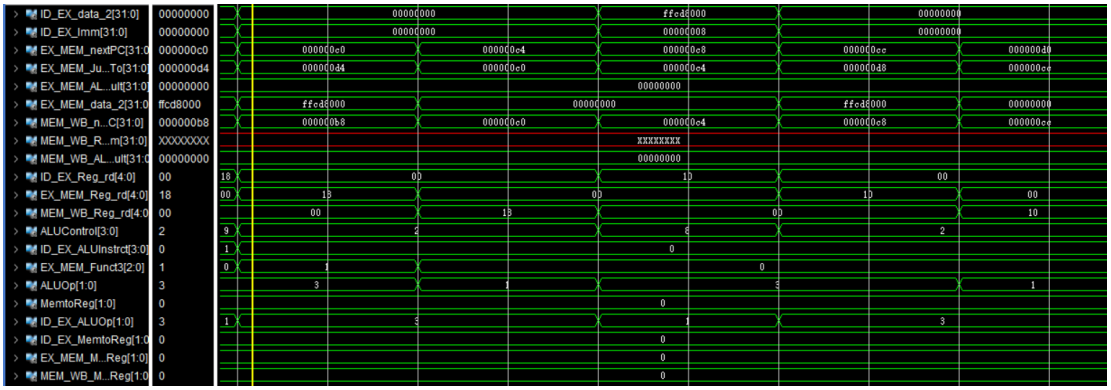

Figure 18: Result 2.

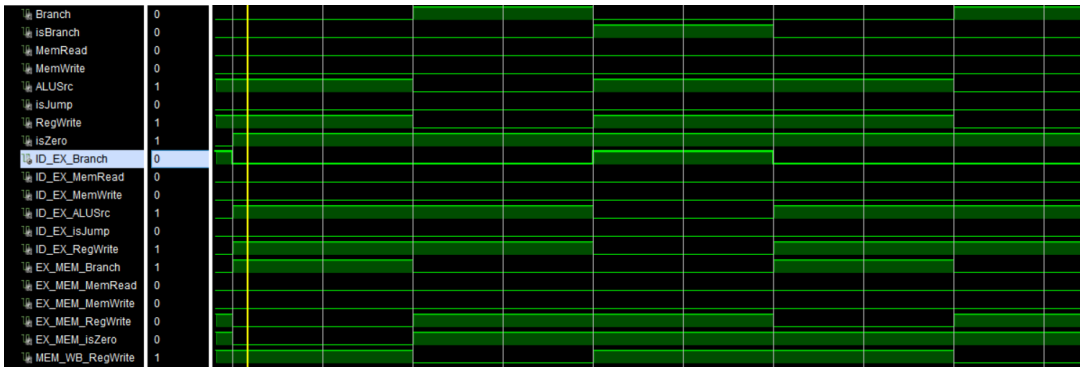

Figure 19: Result 3.

When PC = 0xc8, it's beq. It reads data in x7 and x5, and compare in ALU. They are equal, so branch to a new PC address created by PC+immediate. In total, it takes 4 clock cycles.

2.6 jal

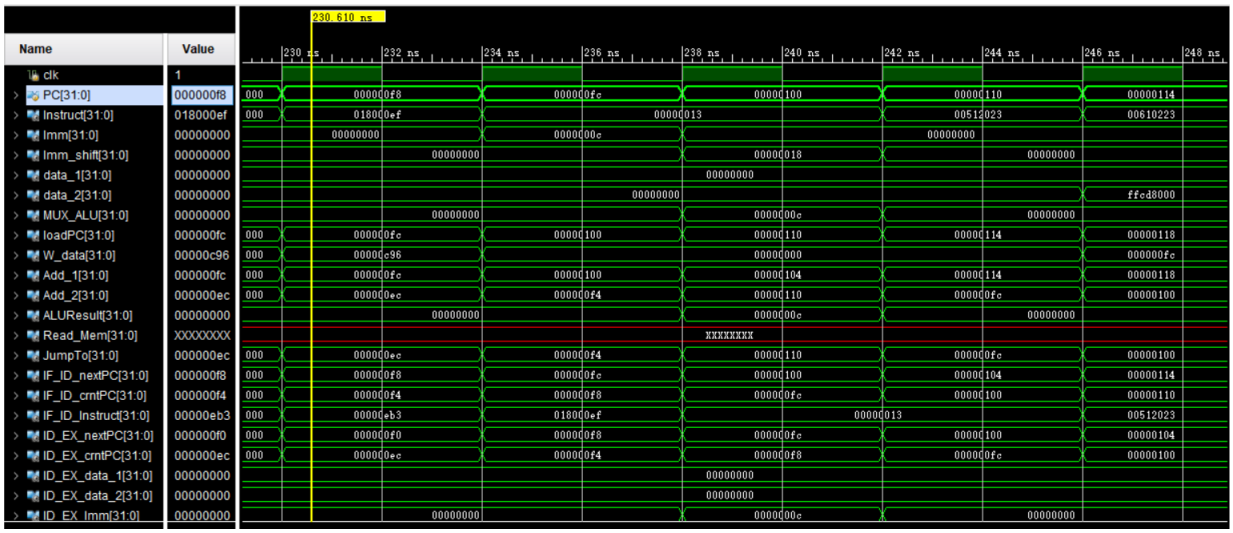

Figure 20: Result 1.

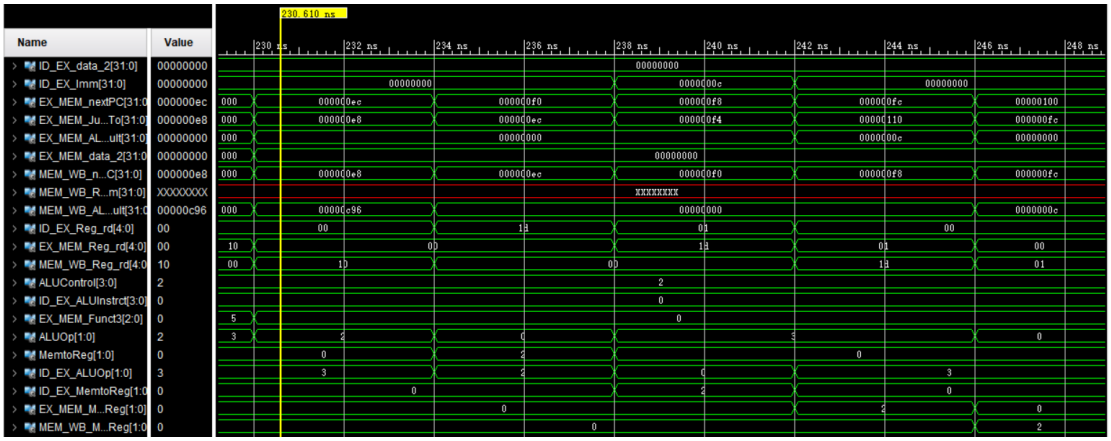

Figure 21: Result 2.

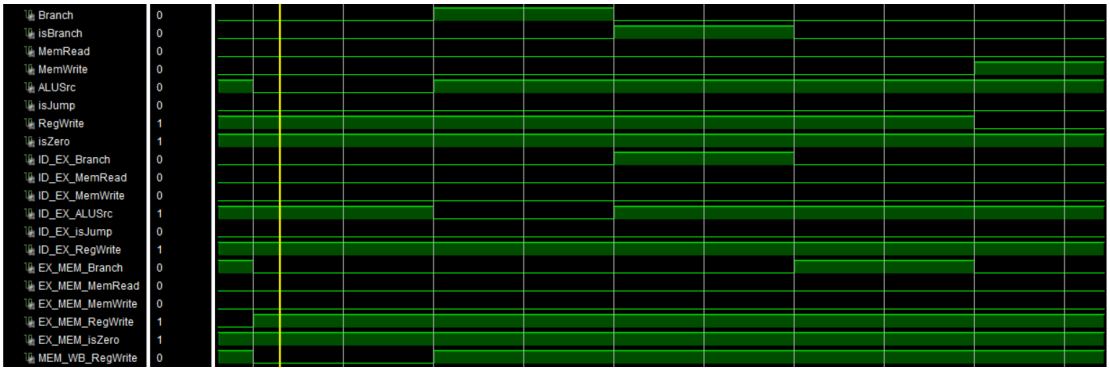

Figure 22: Result 3.

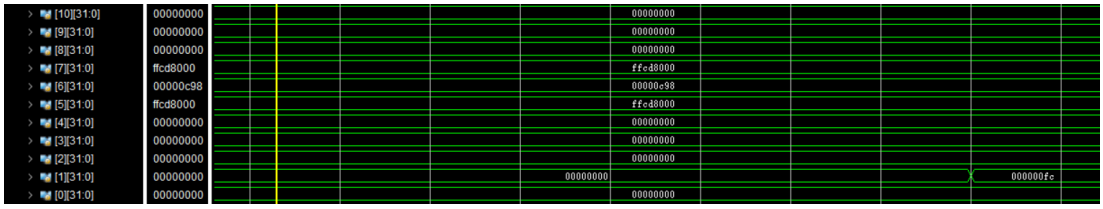

Figure 23: Result 4.

When PC = 0xf8, it's jal. It saves the return address in x1, and this process takes 5 clock cycles. However, jumping to the target address only takes 4 clock cycles.

3 Schematic

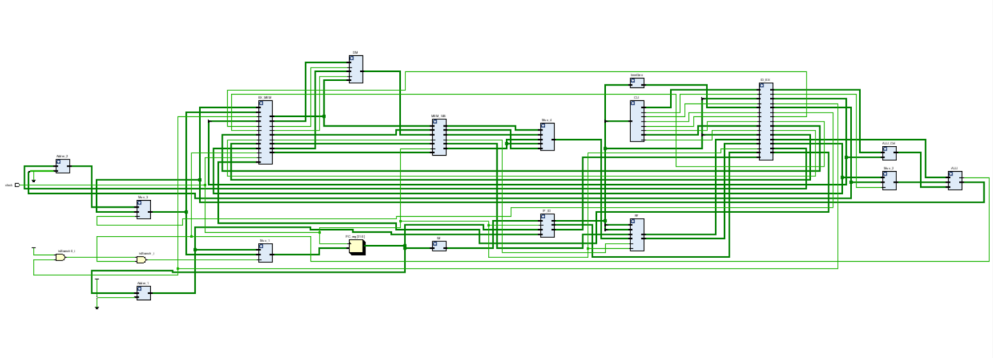

Figure 24: Schematic.