Introduction to the implementation:

Overall:
The pipelined processor can load with five different instructions at the same time, hence it contains five different periods controlled by the clock.

Detailed parts:
module PC_switch(PC_in, PC_out, clk);
Input: PC_in, clk; Output: PC_out
Take in the address of PC from WB period, and store it in the register. When the clock gives a tick, then push the address to the next stage.

module instruction_memory(PC, Inst);
Input: PC; Output: Inst
Receive the PC from the PC register and search for the instruction stored in the instruction memory in advance.

module PC_adder(in1, in2, out);
adder PC_adder(.in1(PC_cs), .in2(32'd4), .out(PC_plus4));
Add the PC address with "4", used when the instruction doesn't have PC jump.

module IF_ID(clk, current_PC, next_PC, Instruction, cPC, nPC, Inst);
Input: clk, current_PC, next_PC, Instruction; Output: cPC, nPC, Inst
The IF/ID register is used to store the data of the IF period and ready to be sent to ID period according to the clock tick.

module control(Inst, Branch, MemRead, MemtoReg, ALUop, MemWrite, ALUsrc, RegWrite, jump);
Input: Inst; Output: The others
The control unit takes in the instruction and assign the control signals to the specific elements.

module register(read_reg1, read_reg2, write_reg, write_data, read_data1, read_data2, reg_write);
Input: read_reg1, read_reg2, write_reg, write_data; Output: The others
The register search for the data stored in itself with the provided address and update the write_data according to the signal write_reg.

module ImmGen(ins,imm);
Input: ins; Output: imm
The Immediate Generator takes in the 12'b immediate and transfer it into a 32'b one.

module ID_EX(clk, RegWrite, MemtoReg, MemRead, MemWrite, Branch, ALUSrc, ALUOp, current_PC, next_PC, Readdata1, Readdata2, Immediate, function_code, rd,

jump, RegWrite_out, MemtoReg_out, MemRead_out, MemWrite_out, Branch_out, ALUSrc_out, ALUOp_out, current_PC_out, next_PC_out, Readdata1_out, Readdata2_out, Immediate_out, function_code_out, rd_out, jump_out);
The ID/EX register is used to store the data of the ID period and ready to be sent to EX period according to the clock tick.

module ALU_Control(Inst, ALUop, control);
Input: Inst, ALUop; Output: control
The ALU_Control takes in the Instruction and ALUop and gives out the control signal of ALU.

module TWO_MUX(in0, in1, select, out);
TWO_MUX ALU_Src_MUX
This mux selects the ALU's source from immediate or read_data2.

module ALU(in1, in2, control, zero, result);
ALU_Unit(.in1(ID_EX_ReadData1_out), .in2(ALU_in2), .control(ALU_Control_Signal), .zero(Zero), .result(ALU_Result));
The ALU takes in the data from register and ImmGen, then perform certain calculation according to the ALU_control signals.

module shift_left(in, out);
Perform multiplying by 2.

Branch_Adder(.in1(ID_EX_PC_out), .in2(Shifted_Immediate), .out(Branch_Target));
Add the PC with shifted immediate for PC jump.

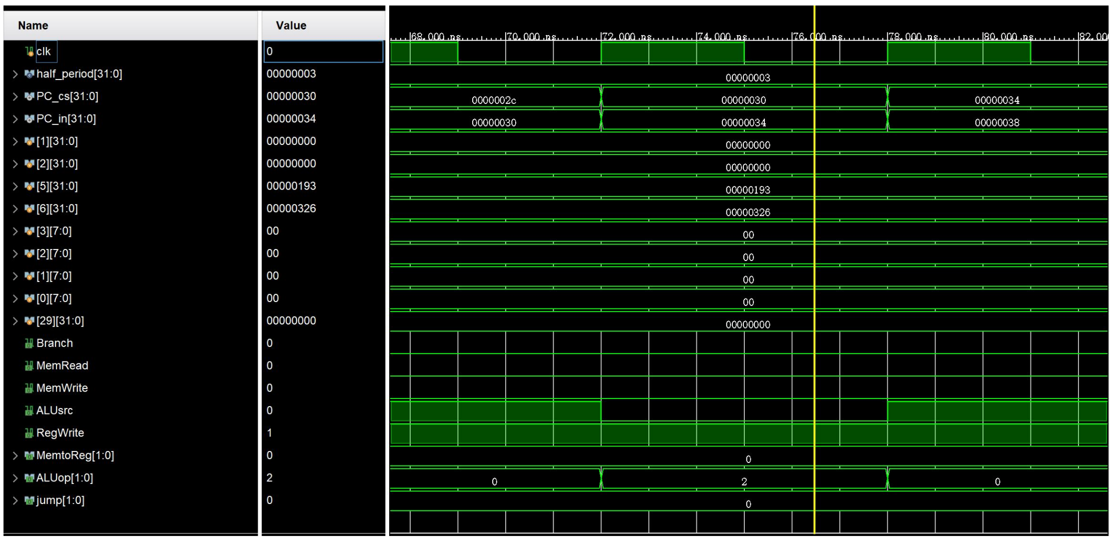
module EX_MEM(clk, RegWrite, MemtoReg, MemRead, MemWrite, Branch, Zero, PC_sum, next_PC, ALU_result, Readdata2, funct3, rd, RegWrite_out, MemtoReg_out, MemRead_out, MemWrite_out, Branch_out, Zero_out, PC_sum_out, next_PC_out, ALU_result_out, Readdata2_out, funct3_out, rd_out);
The EX/MEM register is used to store the data of the EX period and ready to be sent to MEM period according to the clock tick.

module data_memory(MemRead, MemWrite, inst, addr, WriteData, ReadData);
Input: MemRead, MemWrite, inst, addr, WriteData; Output: ReadData
The Data Memory serves as data store for using. It read one address and is controlled by the MemWrite&MemRead to decide whether read or write is performed.

module MEM_WB(clk, RegWrite, MemtoReg, next_PC, Readdata, ALU_result, rd, Branch_target, RegWrite_out, MemtoReg_out, next_PC_out, Readdata_out, ALU_result_out, rd_out, Branch_target_out);
The MEM/WB register is used to store the data of the MEM period and ready to be sent to WB period according to the clock tick.

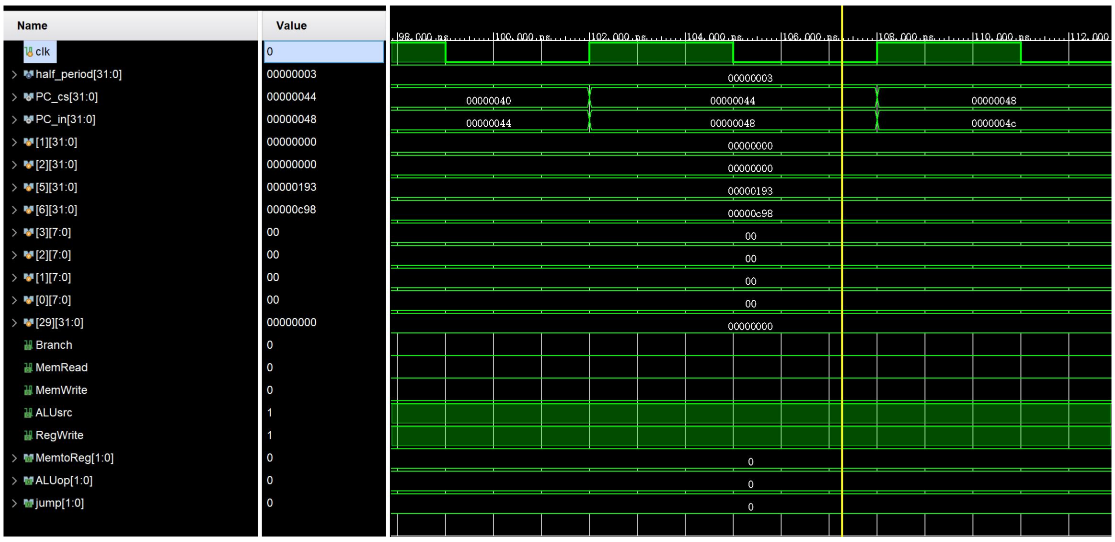
Simulation Results:

addi t0 x0 0x193: t0 = x5 = 0x193



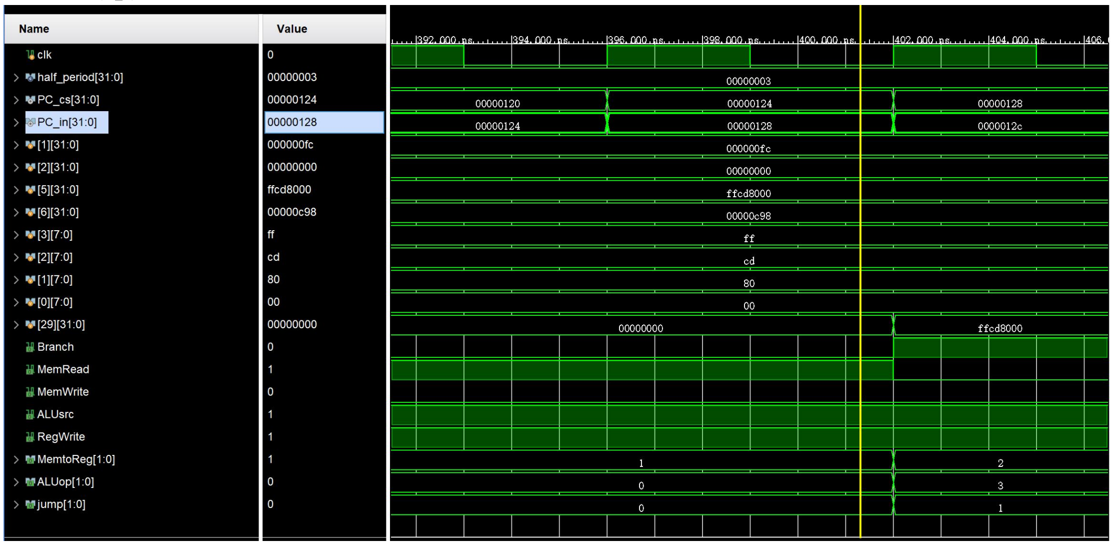
Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	00000030
PC_in[31:0]	00000034
[1][31:0]	00000000
[2][31:0]	00000000
[5][31:0]	00000193
[6][31:0]	00000326
[3][7:0]	00
[2][7:0]	00
[1][7:0]	00
[0][7:0]	00
[29][31:0]	00000000
Branch	0
MemRead	0
MemWrite	0
ALUsrc	0
RegWrite	1
MemtoReg[1:0]	0
ALUop[1:0]	2
jump[1:0]	0

add t1 t0 t0: t1 = 2*t0 = 0x326



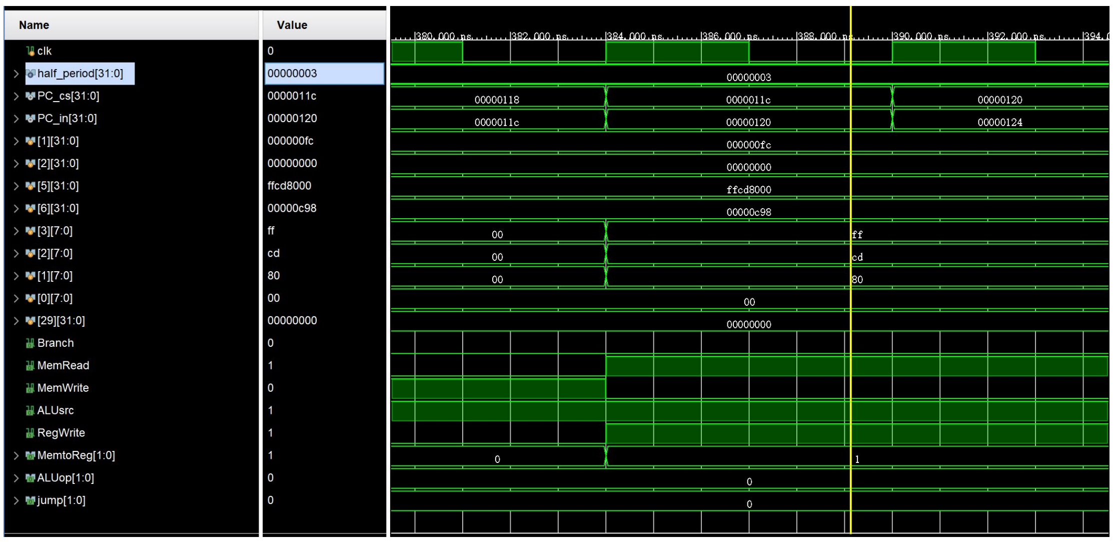
Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	00000044
PC_in[31:0]	00000048
[1][31:0]	00000000
[2][31:0]	00000000
[5][31:0]	00000193
[6][31:0]	00000c98
[3][7:0]	00
[2][7:0]	00
[1][7:0]	00
[0][7:0]	00
[29][31:0]	00000000
Branch	0
MemRead	0
MemWrite	0
ALUsrc	1
RegWrite	1
MemtoReg[1:0]	0
ALUop[1:0]	0
jump[1:0]	0

lw t4 0(sp): t4 = ffcd8000



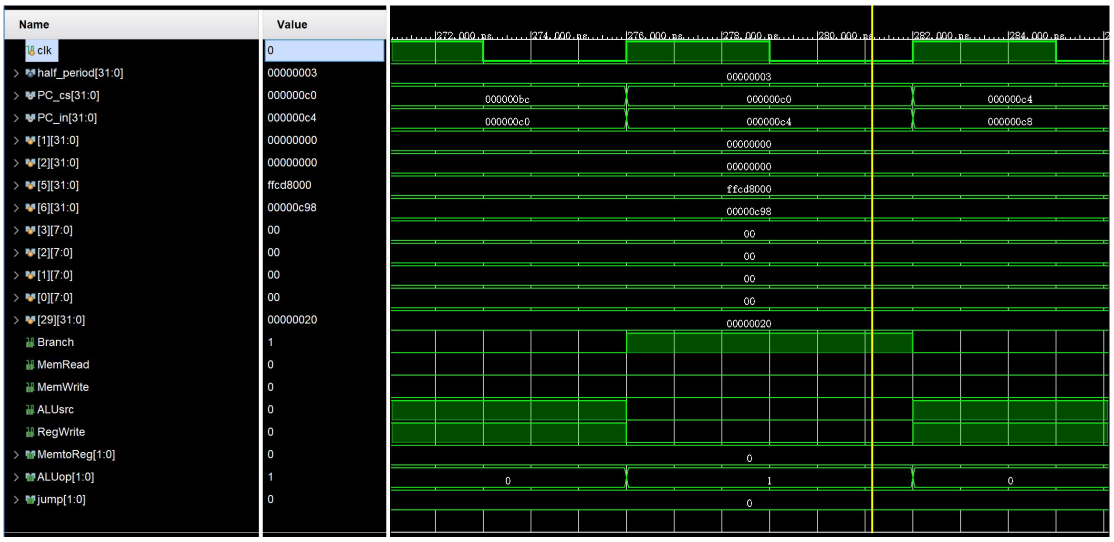
Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	00000124
PC_in[31:0]	00000128
[1][31:0]	000000fc
[2][31:0]	00000000
[5][31:0]	ffcd8000
[6][31:0]	00000c98
[3][7:0]	ff
[2][7:0]	cd
[1][7:0]	80
[0][7:0]	00
[29][31:0]	00000000
Branch	0
MemRead	1
MemWrite	0
ALUsrc	1
RegWrite	1
MemtoReg[1:0]	1
ALUop[1:0]	0
jump[1:0]	0

sw t0 0(sp): mem[0] = t0 = ffcd8000



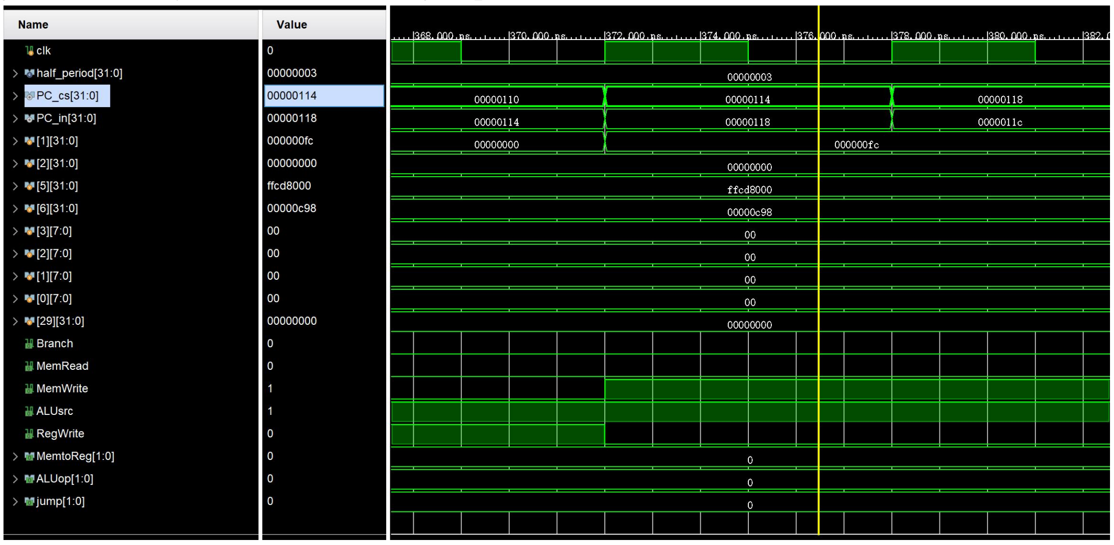
Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	0000011c
PC_in[31:0]	00000120
[1][31:0]	000000fc
[2][31:0]	00000000
[5][31:0]	ffcd8000
[6][31:0]	00000c98
[3][7:0]	ff
[2][7:0]	cd
[1][7:0]	80
[0][7:0]	00
[29][31:0]	00000000
Branch	0
MemRead	1
MemWrite	0
ALUsrc	1
RegWrite	1
MemtoReg[1:0]	1
ALUop[1:0]	0
jump[1:0]	0

beq t2 x0 error1: t2 != 0, no jump



Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	000000c0
PC_in[31:0]	000000c4
[1][31:0]	00000000
[2][31:0]	00000000
[5][31:0]	ffcd8000
[6][31:0]	00000c98
[3][7:0]	00
[2][7:0]	00
[1][7:0]	00
[0][7:0]	00
[29][31:0]	00000020
Branch	1
MemRead	0
MemWrite	0
ALUsrc	0
RegWrite	0
MemtoReg[1:0]	0
ALUop[1:0]	1
jump[1:0]	0

jal x1 memory_test: x1<=PC+4, jump to test



Name	Value
clk	0
half_period[31:0]	00000003
PC_cs[31:0]	00000114
PC_in[31:0]	00000118
[1][31:0]	000000fc
[2][31:0]	00000000
[5][31:0]	ffcd8000
[6][31:0]	00000c98
[3][7:0]	00
[2][7:0]	00
[1][7:0]	00
[0][7:0]	00
[29][31:0]	00000000
Branch	0
MemRead	0
MemWrite	1
ALUsrc	1
RegWrite	0
MemtoReg[1:0]	0
ALUop[1:0]	0
jump[1:0]	0

Schematic: