MIPS 微处理器设计实验报告

——电信提高 2101 班杨筠松 U202115980

一、实验任务

本次实验中全部采用 Verilog 硬件描述语言设计实现简单指令集 MIPS 微处理器，要求指令存储器在时钟上升沿读出指令，指令指针的修改、寄存器文件写入、数据存储器数据写入都在时钟下降沿完成。完成完整设计代码输入、各模块完整功能仿真，整体仿真，验证所有指令执行情况。

且假定所有通用寄存器复位时取值都为各自寄存器编号乘以 4；PC 寄存器初始值为 0；数据存储器和指令存储器容量大小为 32*32，且地址都从 0 开始，指令存储器初始化时装载测试 MIPS 汇编程序的机器指令，数据存储器所有存储单元的初始值为其对应地址的取值。需要注意的是数据存储器的地址呈现以下规则：都是 4 的整数倍。

仿真以下 MIPS 汇编语言程序段的执行流程：

```
main:
    add $4,$2,$3
    lw $4,4($2)
    sw $5,8($2)
    sub $2,$4,$3
    or $2,$4,$3
    and $2,$4,$3
    slt $2,$4,$3
    beq $3,$3,equ
    lw $2,0($3)
equ:
    beq $3,$4,exit
    sw $2,0($3)

exit:
    j main
```

各小组还应实现扩展的指令:bne，bltz，bgez.并且完成后初始化寄存器的值各种情况依次执行,如下指令以完成对扩展指令的验证

```
main:
    bne $t1 $t1 label1
    bne $t1 $t0 label1
label2:
    bltz $t1, label3
label1:
    bltz $t0, label2
label3:
    bgez $t0, label2
    bgez $t0, label4
    bne $t1, $t1, label1
label4:
    bgez $0, label4
```

二、实验目的

1) 熟悉 MIPS 指令和 MIPS 指令的执行过程
2) 掌握 MIPS 微处理器各组件构成和执行逻辑
3) 掌握 MARS 导出机器码及 vivado 中 verilog 程序设计

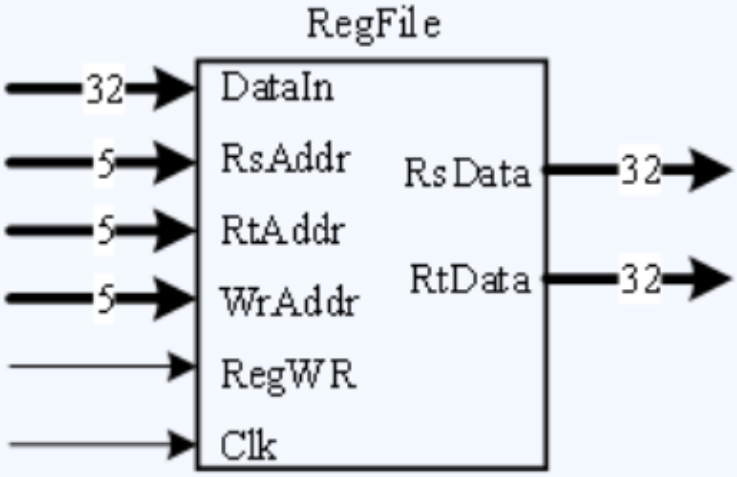
三、实验环境

1) Window 11 操作系统
2) 编辑工具:vscode, vivado 2018.3,sublime text3
3) MIPS 模拟器：MARS

四、设计思路

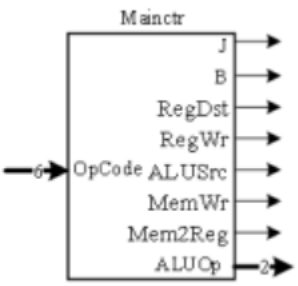
按照下列各图依次实现 ALU、RegFile、MainCtr、CPUMips、Controller 的设计，以及 MIPS 微处理器的顶层文件设计

同步输入\异步输出寄存器文件RegFile，容量为32*32位，且编号为0的寄存器取值恒为0。内部存储变量名为‘regs’



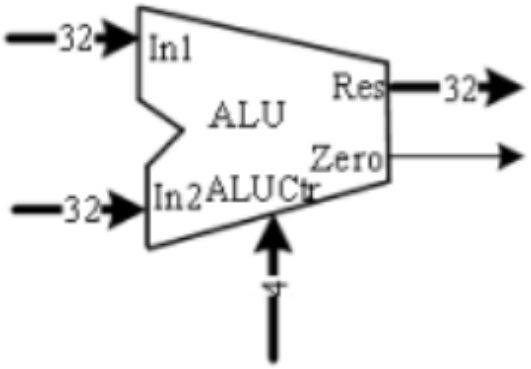
主控制器MainCtr将操作码译码产生各模块控制信号 其中输入输出真值表如下： Opcode J B RegDst RegWr ALUSrc MemWr Mem2Reg ALUOp 0000000 0 0 1 1 0 0 0 10 100011 0 0 0 1 1 0 1 00 101011 0 0 x 0 1 1 x 00 000100 0 1 x 0 0 0 x 01 000010 1 x x 0 x 0 x xx 默认情况所有输出信号都为0.

Opcode	J	B	RegDst	RegWr	ALUSrc	MemWr	Mem2Reg	ALUOp
0000000	0	0	1	1	0	0	0	10
100011	0	0	0	1	1	0	1	00
101011	0	0	x	0	1	1	x	00
000100	0	1	x	0	0	0	x	01
000010	1	x	x	0	x	0	x	xx

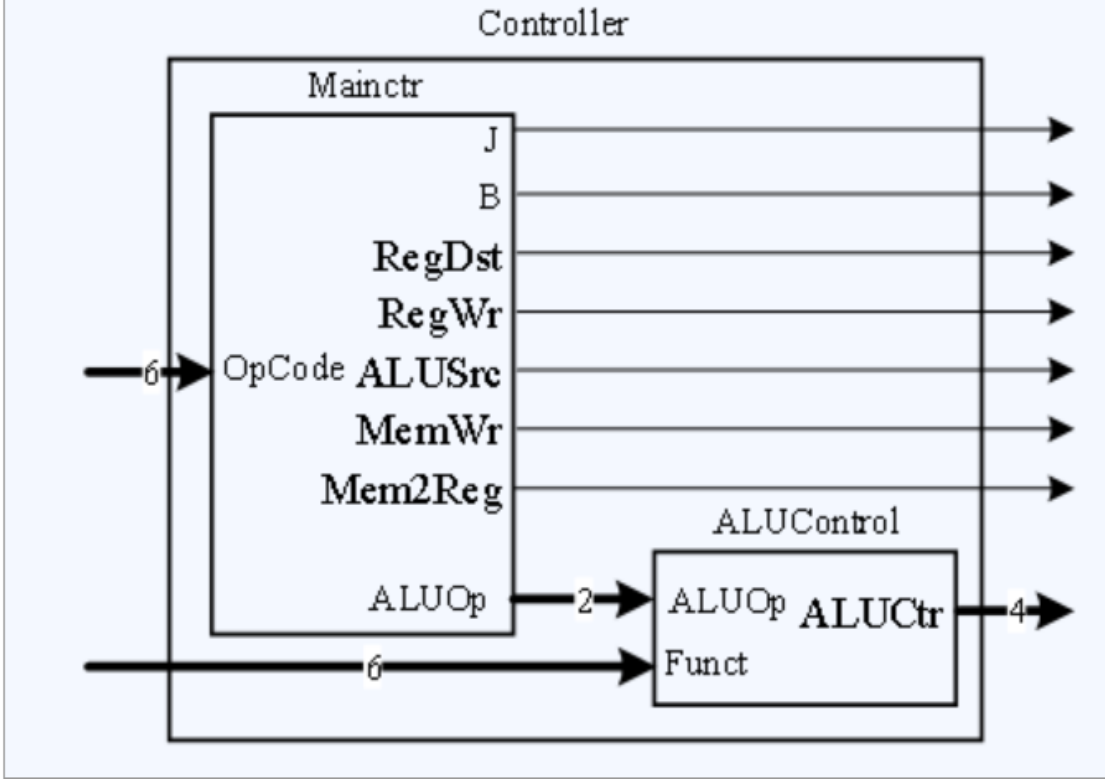


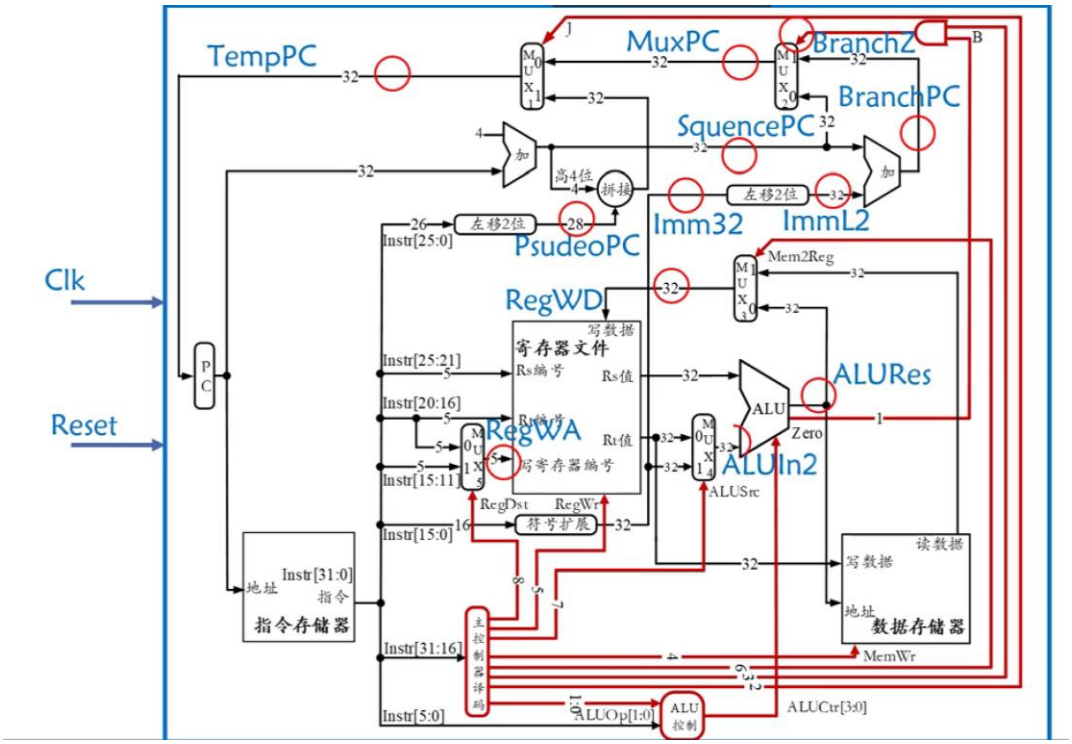
32位算术逻辑运算单元ALU 其中ALUCtr信号与执行的运算关系如下， 运算类型 ALUCtr编码 加 0010 减 0110 小于设置 0111 与 0000 或 0001 默认情况下Res取值为0.

运算类型	ALUCtr编码
加	0010
减	0110
小于设置	0111
与	0000
或	0001



控制器Controller采用层次化设计思想，引用子模块： 主控制器MainCtr (OpCode, J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp)、 ALU控制器ALUControl (ALUOp, Funct, ALUCtr) 实现对指令操作码和功能码译码产生数据通路所需的所有控制信号





而对于扩展电路中，为了完成想要支持的指令集，考虑将 ALUOp 信号进行扩展为三位，这一举动将会改变 MainCtr、ALUControl 和 ALU 文件，而对外信号 zero 保持不变，多种情况**复用 zero 标志位**即可，即无论 ALU 内部怎么操作，保证 zero 信号对外时，高电平表示此情况可以跳转，低电平表示此情况不可跳转，

具体说明如下：

- R 指令：1001_000_10 ----> 1001_000_100
- sw lw 指令：xxxx_xxx_00 ---->xxxx_xxx_000（其中 x 并不表示高阻态）
- jmp 指令：xxx0_001_xx --->xxx0_001_xxx （其中 x 表示高阻态）
- beq 指令 x0x0_010_01 ---> x0x0_010_010 --> 0110
- bnq 指令 x0x0_010_011 ---> 0100
- bltz 和指令 x0x0_010_110 --- > 1110
- bgez 指令 x0x0_010_111 ----> 1111

而观察具体机器码可以发现，如下举例（依次是 bne，bltz，bgez）：

$$15290002 \rightarrow 000101\_01001\_01001\_0000000000000010$$

$$05200001 \rightarrow 000001\_01001\_00000\_0000000000000001$$

$$05210001 \rightarrow 000001\_01001\_00001\_0000000000000001$$

可以发现 bne 的 op 段码为 000101，而 bltz 和 bgez 都是 Op 段码为 000001,唯一区别在于 Rt 段不同，从而只需要将 Rst 组线接入到 MainCtr 即可进行判断，即在 MainCtr 中进行译码成不同情况，传递给 ALU 做相应计算即可

五、实验代码及注释

```
`timescale 1ns / 1ps
module RegFile (RsAddr, RtAddr, WrAddr, DataIn, RegWR, Clk, Reset, RsData, RtData);
// codes
  input [31: 0] DataIn;
  input [4: 0] RsAddr, RtAddr, WrAddr;
  input RegWR, Clk, Reset;
  output wire [31: 0] RsData, RtData;

  reg [31: 0] regs[31:0];
  assign RsData = (RsAddr == 5'b0)? 32'b0: regs[RsAddr];
  assign RtData = (RtAddr == 5'b0)? 32'b0: regs[RtAddr];
  integer i;

  always @(posedge Clk)begin
  if (!Reset & RegWR)
     regs[WrAddr] = DataIn;
  else if (Reset)
     for (i = 0; i < 32; i = i + 1)
     begin
         if (i == 8) regs[i] = 32'h0;
         else if (i == 9) regs[i] = 32'b0;
         else regs[i] = i;
     end
  end
endmodule
```

```
`timescale 1ns / 1ps
module ALUControl (ALUOp, Funct, ALUCtr);
// codes
  input [2: 0] ALUOp;
  input [5: 0] Funct;
  output reg [3: 0] ALUCtr;

  always @(*) begin
  case (ALUOp)
    3'b000:
    ALUCtr <= 4'b0010;
    3'b010:
    ALUCtr <= 4'b0110;            // beq
    3'b011:
      ALUCtr <= 4'b0100;
    3'b110:
      ALUCtr <= 4'b1110;
    3'b111:
      ALUCtr <= 4'b1111;
    3'b100:
    begin
      case (Funct)
      6'b100000:
        ALUCtr <= 4'b0010;
      6'b100010:
        ALUCtr <= 4'b0110;
      6'b100100:
        ALUCtr <= 4'b0000;
      6'b100101:
        ALUCtr <= 4'b0001;
      6'b101010:
        ALUCtr <= 4'b0111;
      endcase
    end
  endcase
  end

endmodule
```

```
`timescale 1ns / 1ps
module ALU (In1, In2, ALUCtr, Res, Zero);
// codes
  input signed [31: 0] In1, In2;
  input [3: 0] ALUCtr;
  output reg [31: 0] Res;
  output reg Zero;

  always @(In1 or In2 or ALUCtr) begin
  case (ALUCtr)
    4'b0010: //add
    begin
      Res = In1 + In2;
      Zero = 0;
    end
    4'b0110: //sub
    begin
    Res = In1 - In2;
    Zero = (Res == 0)? 1 : 0;
    end
    4'b0100:
    begin
      Res = In1 - In2;
      Zero = (Res != 0)? 1: 0;
    end
    4'b1110:
    begin
      Zero = (In1 < 0) ? 1 : 0;
    end
    4'b1111:
    begin
      Zero = (In1 >= 0)? 1 : 0;
    end
    4'b0000: //and
    begin
    Res = In1 & In2;
    Zero = 0;
      end
    4'b0001: //or
    begin
    Res = In1 | In2;
    Zero = 0;
    end
    4'b0111: //slt
    begin
      Res = (In1 < In2)?1:0;
      Zero = 0;
    end
    default:
    begin
        Zero = 0;
      Res = 32'b0;
    end
  endcase
  end
endmodule
```

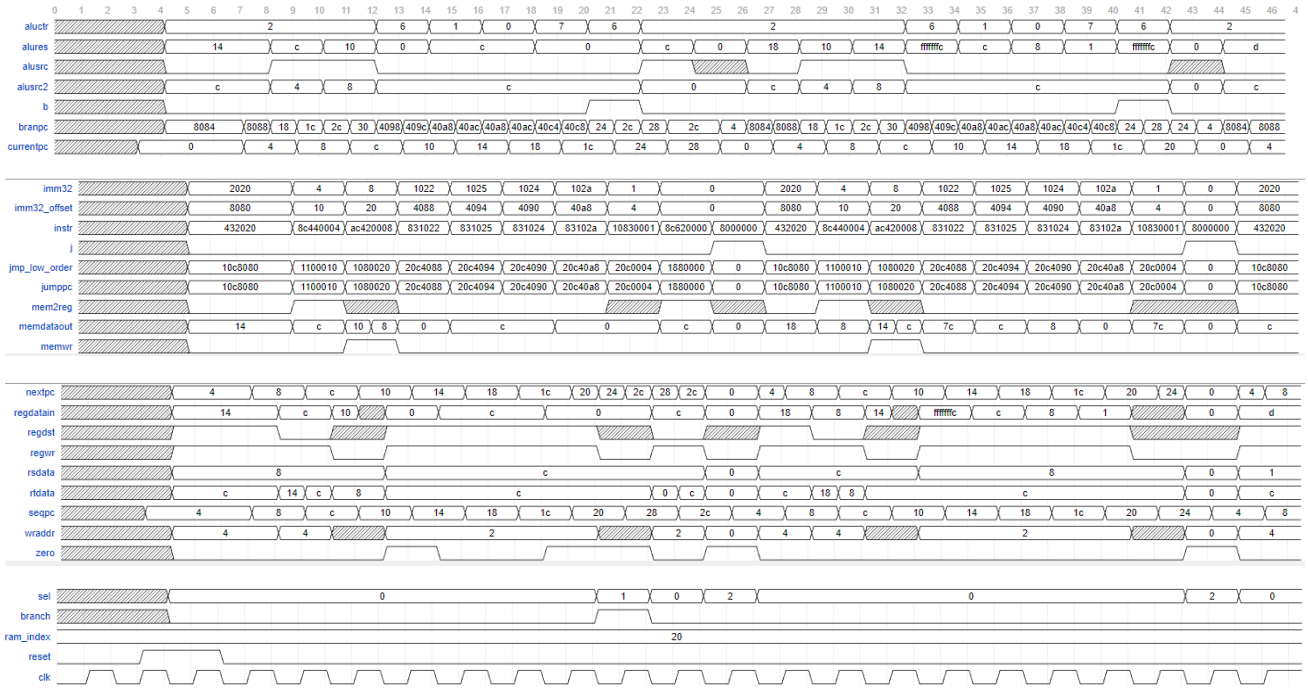
```
module MainCtr (OpCode, RtFlag, J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp);
// codes
  input [5: 0] OpCode;
  input [4: 0] RtFlag;
  output J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;
  output [2: 0] ALUOp;

  reg [9: 0] outregs;

  always @(*) begin
  case (OpCode)
    6'b000000: outregs <= 10'b1001_000_100;     //R
    6'b100011: outregs <= 10'b0111_000_000;     //lw
    6'b101011: outregs <= 10'bx1x0_100_000;     //sw
    6'b000100: outregs <= 10'bx0x0_010_010;     //beq
    6'b000010: outregs <= 10'bxxx0_001_xxx;     //jmp
    // extention parts
    6'b000101: outregs <= 10'bx0x0_010_011;     //bne
    6'b000001: begin
        if (RtFlag == 5'b00000) outregs <= 10'bx0x0_010_110;
        else outregs <= 10'bx0x0_010_111;
    end
    default: outregs <= 10'b0000_000_000;
  endcase
  end
  assign {RegDst, ALUSrc, Mem2Reg, RegWr, MemWr, B, J, ALUOp} = outregs;
endmodule
```

```
module Controller (OpCode, RtFlag, Funct, J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUCtr);
// codes
  input [5: 0] Funct;
  input [4: 0] RtFlag;
  output [3: 0] ALUCtr;
  input [5: 0] OpCode;
  output J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;
  wire [2: 0] ALUOp;

  MainCtr MainCtr(
  .OpCode (OpCode),
  .RtFlag (RtFlag),
  .J  (J),
  .B  (B),
  .RegDst (RegDst),
  .RegWr  (RegWr),
  .ALUSrc (ALUSrc),
  .MemWr  (MemWr),
  .Mem2Reg  (Mem2Reg),
  .ALUOp  (ALUOp)
  );

  ALUControl ALUControl(
  ALUOp, Funct, ALUCtr
  );
```

```
module CPUMips(
    input Clk,
    input Reset,
    output t
);
    wire [31:0] TempPC, MuxPC, JumpPC, BranchPC, SquencePC, Imm32, ImmL2, RegWD, RsData, RtData, ALUIn2, ALURes, MemRD, Instr;
    wire [4:0] RegWA;
    wire [27:0] PsudeoPC;
    wire BranchZ, J, B, Zero, RegDst, RegWr, ALUSrc, MemWR, Mem2Reg;
    wire [1:0] ALUOp;
    wire [3:0] ALUCtr;
    reg [31:0] PC;


    assign PsudeoPC = {Instr[25:0], 2'b00};
    assign JumpPC = {SquencePC[31:28], PsudeoPC};
    assign SquencePC = PC + 4;
    assign BranchPC = ImmL2 + SquencePC;
    assign MuxPC = BranchZ?BranchPC:SquencePC;
    assign TempPC = J?JumpPC:MuxPC;
    assign BranchZ = B&Zero;

    assign ImmL2 = {Imm32[29:0], 2'b00};
    assign Imm32 = {Instr[15]?16'hffff:16'h0, Instr[15:0]};
    assign ALUIn2 = ALUSrc?Imm32:RtData;

    assign RegWA = RegDst?Instr[15:11]:Instr[20:16];
    assign RegWD = Mem2Reg?MemRD:ALURes;

    ALU UnitALU (RsData, ALUIn2, ALUCtr, ALURes, Zero);
    dramIP Unitdram(~Clk, MemWR, ALURes[6:2], RtData, MemRD);
    iromIP Unitirom(~Clk, PC[6:2], Instr);
    RegFile UnitRegFile(Instr[25:21], Instr[20:16], RegWA, RegWD, RegWr, ~Clk, Reset, RsData, RtData);
    Controller UnitController(Instr[31:26], Instr[20:16], Instr[5:0], J, B, RegDst, RegWr, ALUSrc, MemWR, Mem2Reg, ALUCtr);

    always @(posedge Clk) begin
        if (Reset)
            PC <= 0;
        else
            PC <= TempPC;
    end

endmodule
```

六、实验步骤、结果展示与说明

首先对各模块进行仿真，检验其能准确完成基本工作，最终的 MIPS 顶层仿真如下所示，由结果显示可知，MIPS 顶层行为级别仿真已达到要求。

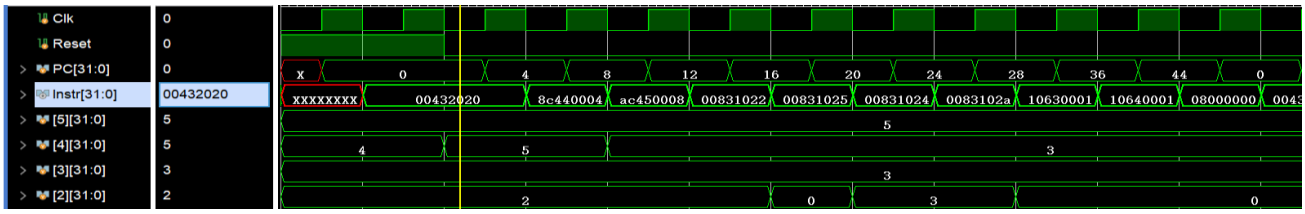


对电路进行综合，并且将需要导出的机器码导出，并且生成.coe 文件，导入到 IP 盒中，coe 文件如下所示：

```
memory_initialization_radix=16;
memory_initialization_vector=00432020 8c440004 ac450008 00831022 00831025 00831024
0083102a 10630001 8c620000 10640001 ac620000 08000000;
```

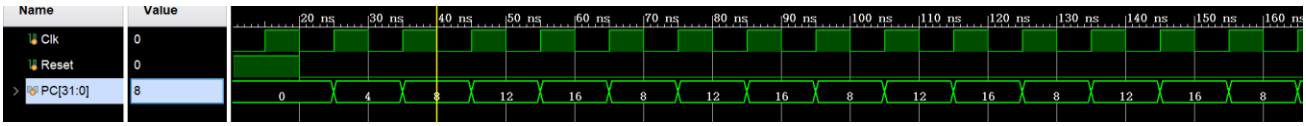
```
memory_initialization_radix=16;
memory_initialization_vector=2409ffff 24080001 15290002 15280001 05200001 0500fffe
0521fffd 05010001 1529fffc 0401ffff;
```

观察最终程序执行结果如下所示：

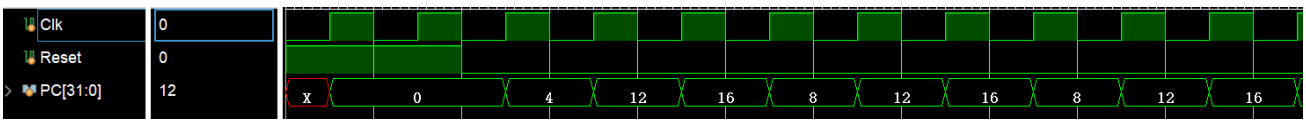


扩展实验中：

当$t0 = 0，$t1 = 0 时，结果如下所示，可以验证 bgez，bne 的正确性



当$t0 = 8，$t1 = 0 时，结果如下所示：可以验证 bltz 的正确性：



七、实验总结

本次实验中完成了 MIPS 顶层仿真，对于 MIPS 指令的执行过程有了清晰的理解，加强了对于系统的理解。