同济大学计算机系

54 条 MIPS 单周期 CPU 实验报告

学　　号　2152118

姓　　名　史君宝

专　　业　计算机科学与技术

授课老师　张冬冬

一、实验内容

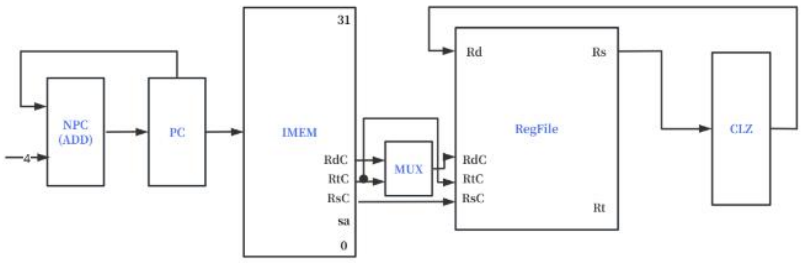
在本次 54 条 MIPS 单周期 CPU 实验中，将使用 Verilog HDL 编写 54 条 MIPS 指令，包括 R、I、J 型指令等其他指令，实现单周期 CPU 的设计、前仿真、后仿真和下板调试，并将结果上交就可以了。

二、数据通路

（实验步骤中要求用 logisim 画图的实验，在该部分给出 logisim 原理图，否则该部分在实验报告中不用写）

数据通路：

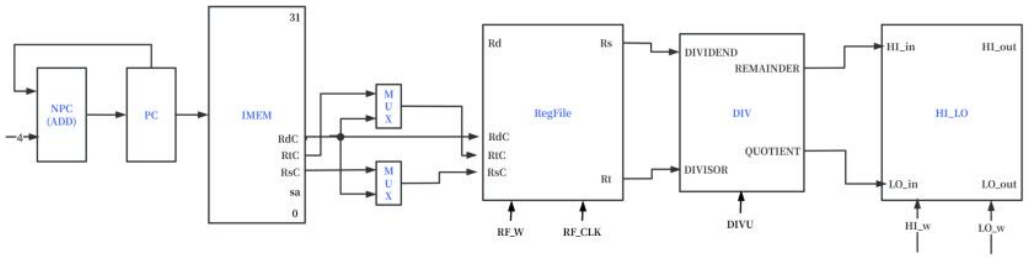
32.clz




```
temp ← 32
for i in 31 .. 0
   if GPR[rs]i = 1 then
      temp ← 31 - i
      break
   endif
endfor
GPR[rd] ← temp
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -> CLZ_in
CLZ_out -> Rd
```

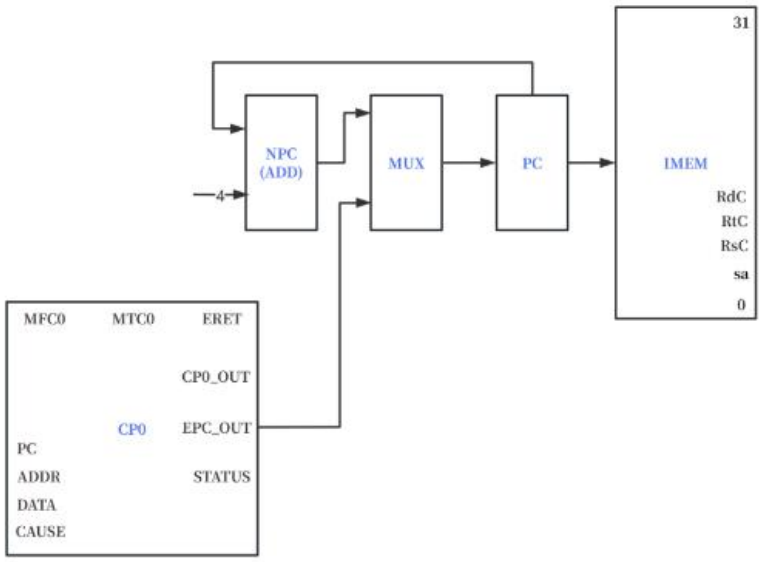
33.divu



$$
\begin{aligned}
q &\leftarrow (0 \,||\, GPR[rs]_{31..0}) \text{ div } (0 \,||\, GPR[rt]_{31..0}) \\
r &\leftarrow (0 \,||\, GPR[rs]_{31..0}) \text{ mod } (0 \,||\, GPR[rt]_{31..0}) \\
LO &\leftarrow \text{sign\_extend}(q_{31..0}) \\
HI &\leftarrow \text{sign\_extend}(r_{31..0})
\end{aligned}
$$

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -> DIVIDEND
Rt -> DIVISOR
Rs/Rt -> QUOTIENT
Rs%Rt -> REMAINDER
QUOTIENT -> HI_in
REMAINDER -> LO_in
```

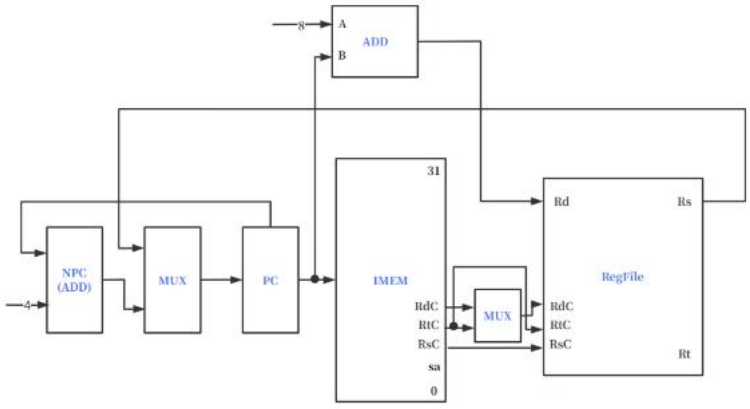
34. eret



```
if Status_ERL = 1 then
    temp ← ErrorEPC
    Status_ERL ← 0
else
    temp ← EPC
    Status_EXL ← 0
endif
if IsMIPS16Implemented() then
    PC ← temp_31..1 || 0
    ISAMode ← temp_0
else
    PC ← temp
endif
LLbit ← 0
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> MUX

STATUS >> 5 -> STATUS
EPC_out -> MUX
MUX(EPC_out) -> PC
```

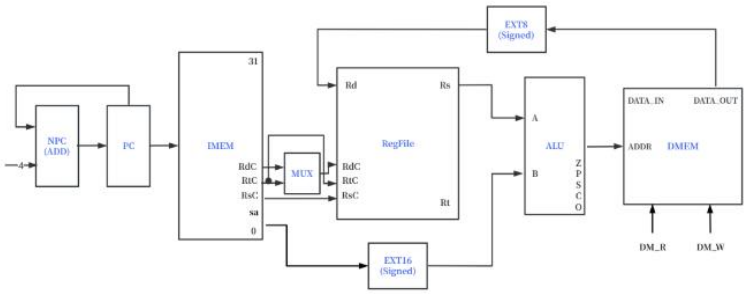
35. jalr




```
I:  temp ← GPR[rs]
    GPR[rd] ← PC + 8
I+1:if Config1_CA = 0 then
        PC ← temp
    else
        PC ← temp_GPRLEN-1..1 || 0
        ISAMode ← temp_0
    endif
```

```
PC -> Imem
PC -> ADD_A
ADD_A + ADD_B -> ADD_res

PC + 4 -> NPC
NPC -> MUX

Rs -> MUX
MUX(Rs) -> PC
ADD_res -> Rd
```

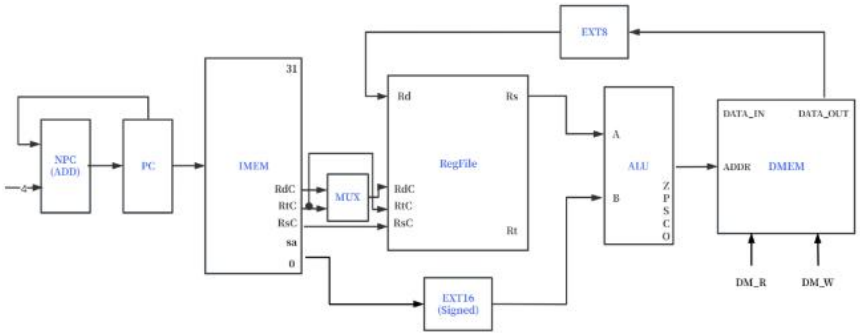
36. 1b




```
vAddr  ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
pAddr  ← pAddr_{PSIZE-1..2} || (pAddr_{1..0} xor ReverseEndian^2)
memword← LoadMemory (CCA, BYTE, pAddr, vAddr, DATA)
byte   ← vAddr_{1..0} xor BigEndianCPU^2
GPR[rt]← sign_extend(memword_{7+8*byte..8*byte})
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Dmem_out - Dmem_addr -> EXT_8_sign
EXT_8_sign_out -> Rd
```

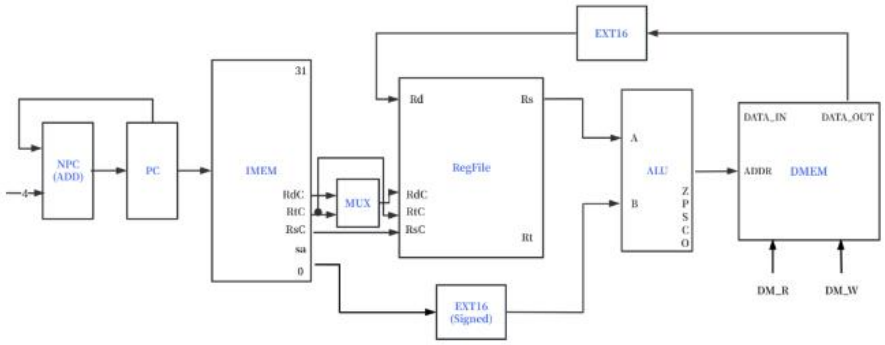
37. lbu




```
vAddr  ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
pAddr  ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor ReverseEndian^2)
memword← LoadMemory (CCA, BYTE, pAddr, vAddr, DATA)
byte   ← vAddr_1..0 xor BigEndianCPU^2
GPR[rt]← zero_extend(memword_7+8*byte..8*byte)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Dmem_out - Dmem_addr -> EXT_8_sign
EXT_8_sign_out -> Rd
```

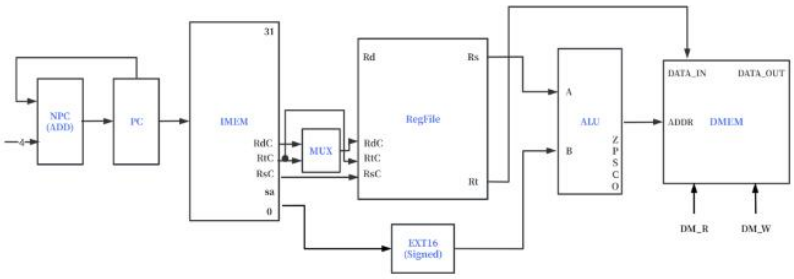
38. lhu



```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr_0 ≠ 0 then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (CCA, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr_1..0 xor (BigEndianCPU || 0)
GPR[rt] ← zero_extend(memword_15+8*byte..8*byte)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Dmem_out - Dmem_addr -> EXT_16
EXT_16_out -> Rd
```

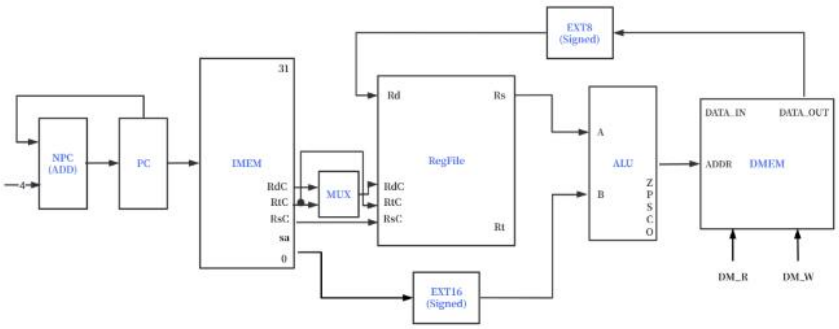
39. sb




```
vAddr      ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, STORE)
pAddr      ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor ReverseEndian^2)
bytesel    ← vAddr_1..0 xor BigEndianCPU^2
dataword   ← GPR[rt]_31-8*bytesel..0 || 0^8*bytesel
StoreMemory (CCA, BYTE, dataword, pAddr, vAddr, DATA)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Rt[7:0] - Dmem_w -> Dmem_in
```

40. sh

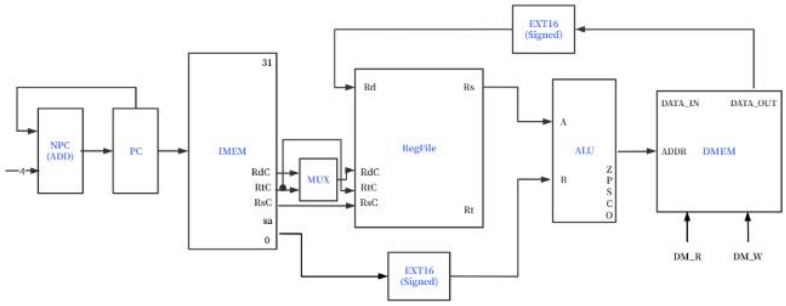


-

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr0 ≠ 0 then
   SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddrPSIZE-1..2 || (pAddr1..0 xor (ReverseEndian || 0))
bytesel← vAddr1..0 xor (BigEndianCPU || 0)
dataword← GPR[rt]31-8*bytesel..0 || 08*bytesel
StoreMemory (CCA, HALFWORD, dataword, pAddr, vAddr, DATA)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Rt[15:0] - Dmem_w -> Dmem_in
```

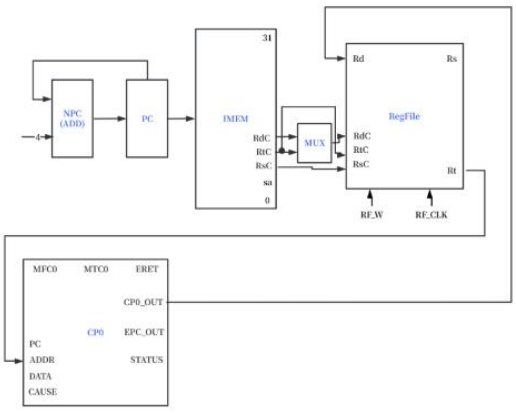
41. lh




```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr0 ≠ 0 then
   SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE-1..2 || (pAddr1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (CCA, HALFWORD, pAddr, vAddr, DATA)
byte   ← vAddr1..0 xor (BigEndianCPU || 0)
GPR[rt] ← sign_extend(memword15+8*byte..8*byte)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Imem[15:0] -> EXT_16_sign
Rs -> A
EXT_16_sign_out -> B

result -> Dmem_addr
Dmem_out - Dmem_r -> EXT_16_sign
EXT_16_sign_out -> Dmem_in
```

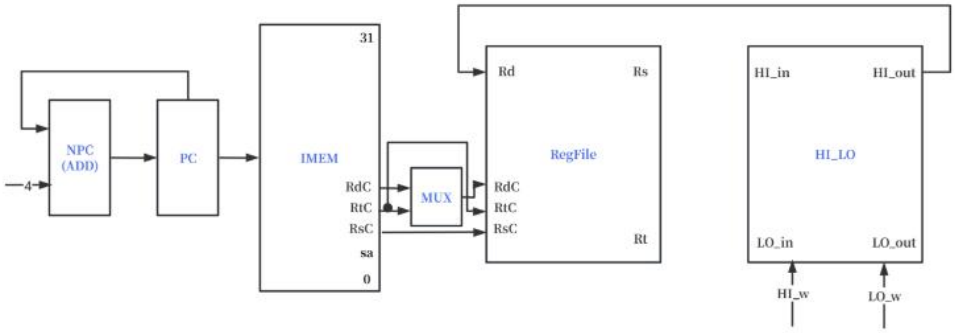
42. mfc0



```
data ← CPR[0,rd,sel]
GPR[rt] ← data
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rt -> CP0_addr
CP0_out -> Rd
```

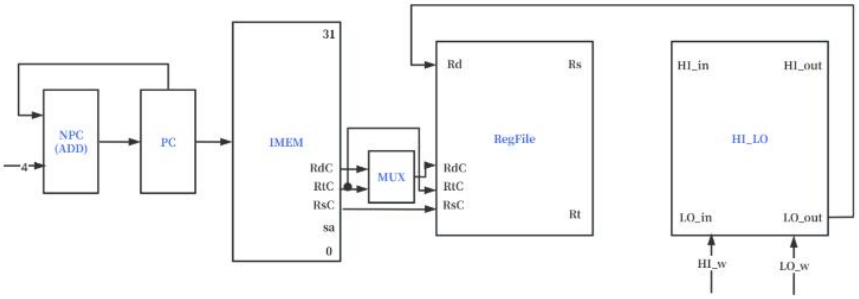
43. mfhi




```
GPR[rd] ← HI
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

HI_out -> Rd
```

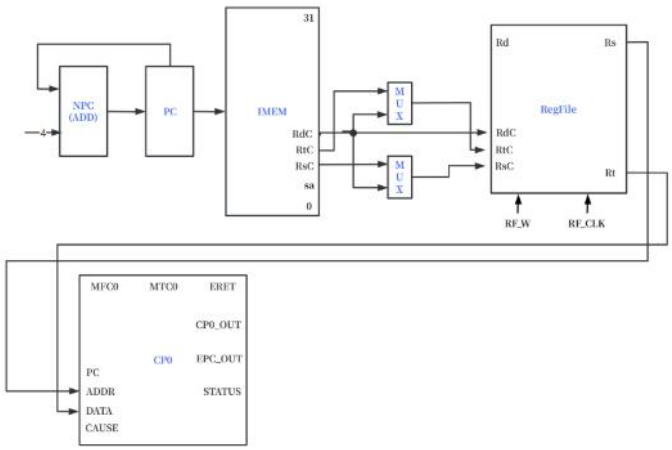
44. mflo



```
GPR[rd] ← LO
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

LO_out -> Rd
```

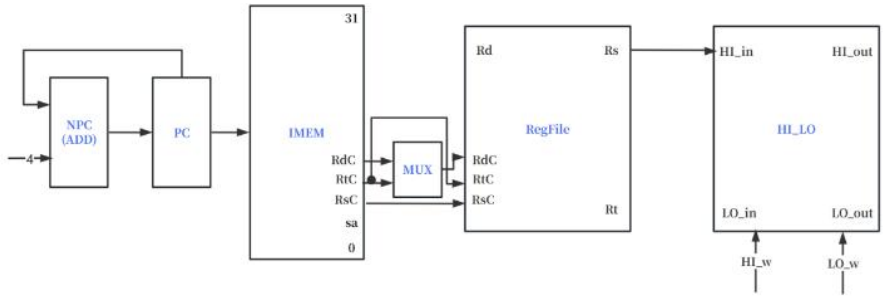
45. mtc0




```
CPR[0,rd,sel] ← data
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rt -> CP0_addr
Rs -> CP0_data
```

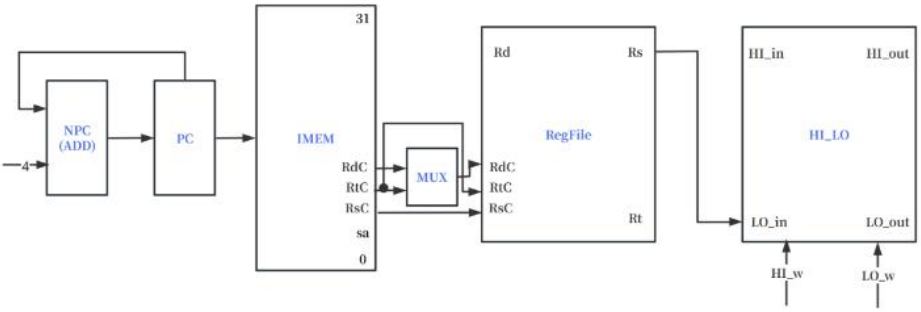
46. mthi




```
HI ← GPR[rs]
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -HI_w -> HI_in
```

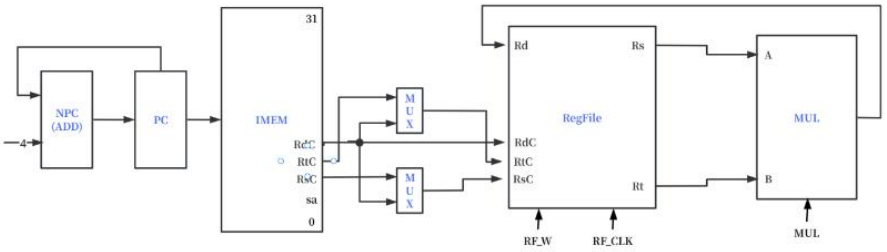
47.mtlo




```
LO ← GPR[rs]
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -LO_w -> LO_in_in
```

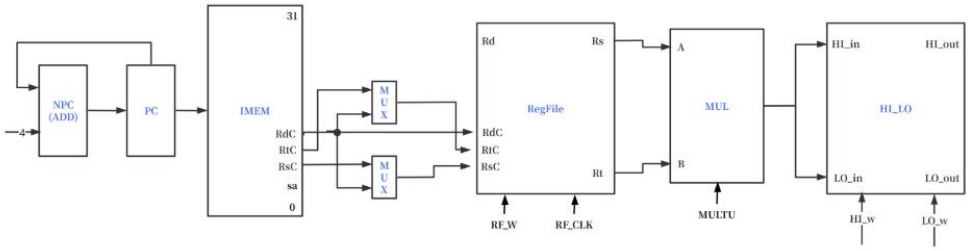
48.mul




```
temp <- GPR[rs] * GPR[rt]
GPR[rd] <- temp31..0
HI <- UNPREDICTABLE
LO <- UNPREDICTABLE
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -> MUL_A
Rt -> MUL_B
MUL_A * MUL_B -> MUL_res
MUL_res[31:0] -> Rd
```

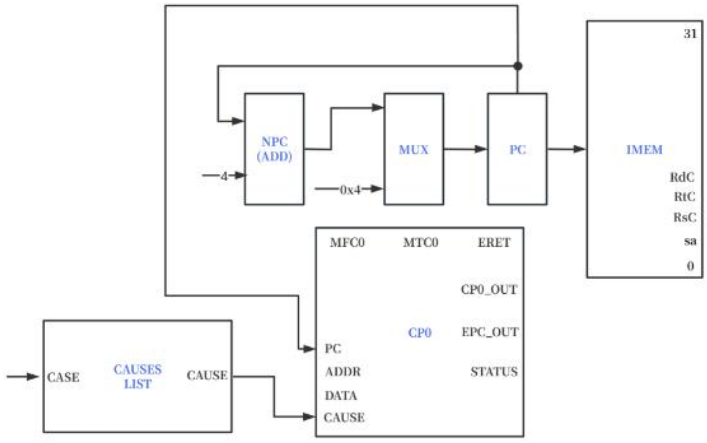
49.multu




```
prod← (0 || GPR[rs]31..0) × (0 || GPR[rt]31..0)
LO ← prod31..0
HI ← prod63..32
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -> MUL_A
Rt -> MUL_B
MUL_A * MUL_B -> MUL_res
MUL_res[31:0] -> LO_in
MUL_res[63:32] -> HI_in
```

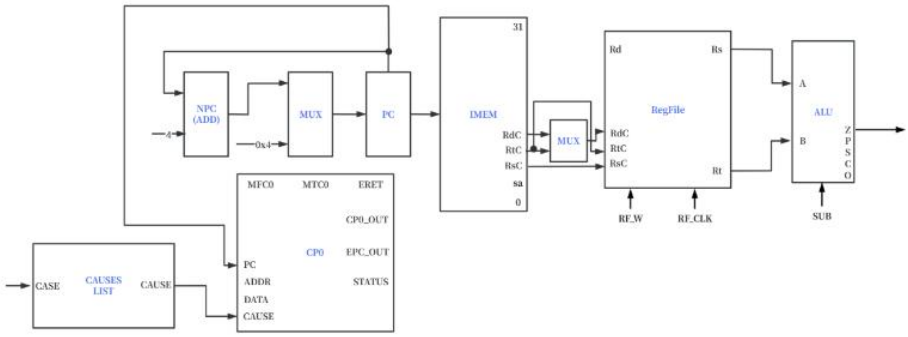
50. syscall




```
SignalException(SystemCall)
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> MUX

PC -> CP0
CAUSES - LIST_CAUSE -> CP0_CAUSE
STATUS << 5 -> STATUS

0x4 -> MUX
MUX(0x4) -> PC
```

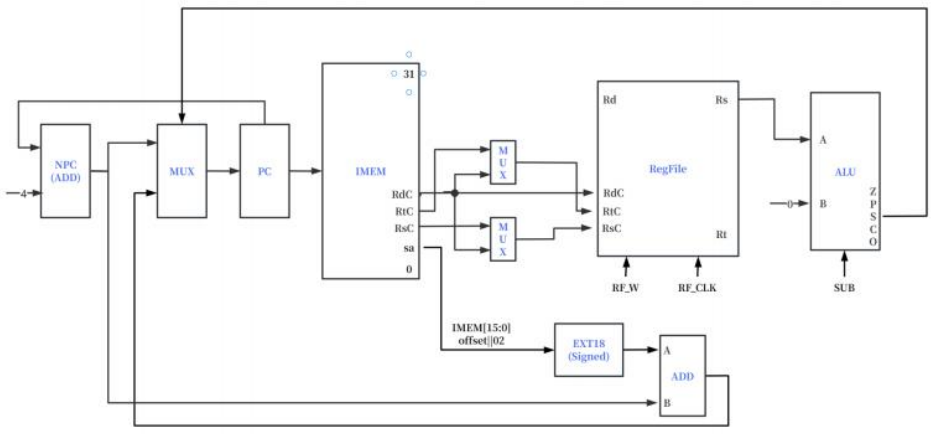
51. teq




```
if GPR[rs] = GPR[rt] then
    SignalException(Trap)
endif
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> MUX
0x4 -> MUX

Rs -> A
Rt -> B
Zero -> Rs - Rt == 0

if Zero:
    PC -> CP0
    CAUSES - LIST_CAUSE -> CP0_CAUSE
    STATUS << 5 -> STATUS
    MUX(0x4) -> PC

else:
    MUX(PC) -> PC
```

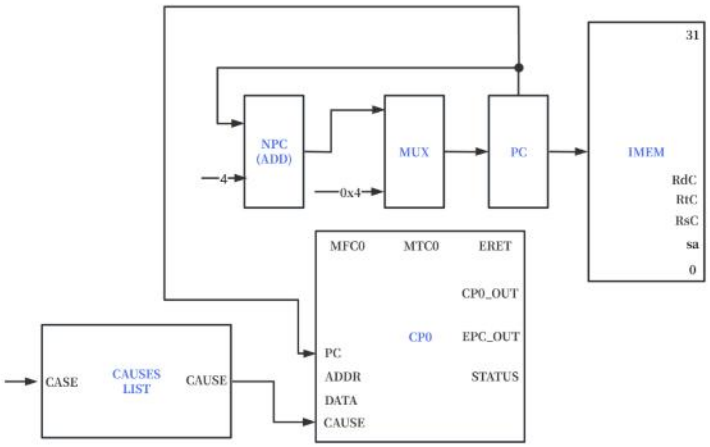
52. bgez




```
I:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] > 0^GPRLEN
I+1:    if condition then
            PC ← PC + target_offset
        endif
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> MUX

Rs -> A
0 -> B
Rs - 0 -> result

Imem[15:0] || 00 -> EXT_18_sign
EXT_18_sign_out -> ADD_A
NPC -> ADD_B
ADD_A + ADD_B -> ADD_res

if S < 0:
    MUX(NPC) -> PC
else:
    MUX(ADD_res) -> PC
```

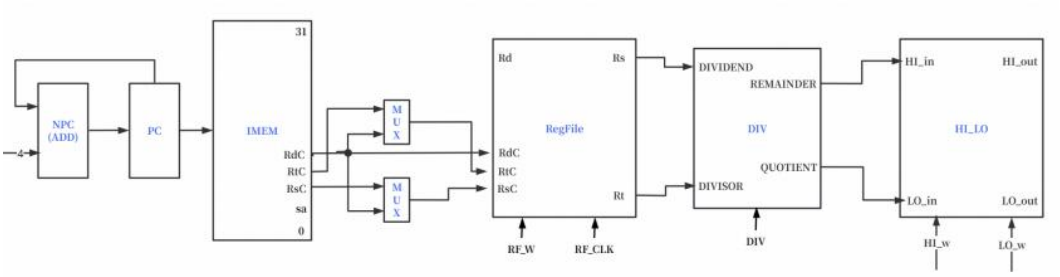
53. break



SignalException(Breakpoint)

```
PC -> Imem
PC + 4 -> NPC
NPC -> MUX

PC -> CP0
CAUSES - LIST_CAUSE -> CP0_CAUSE
STATUS << 5 -> STATUS

0x4 -> MUX
MUX(0x4) -> PC
```

54.div




```
q  ← GPR[rs]31..0 div GPR[rt]31..0
LO ← q
r  ← GPR[rs]31..0 mod GPR[rt]31..0
HI ← r
```

```
PC -> Imem
PC + 4 -> NPC
NPC -> PC

Rs -> DIVIDEND
Rt -> DIVISOR
Rs/Rt -> QUOTIENT
Rs%Rt -> REMAINDER
QUOTIENT -> HI_in
REMAINDER -> LO_in
```

三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的 verilog 代码）

sccomp_dataflow.v 文件

```
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc,
```

```
output [31:0] result
);

wire [31:0] IMEM_addr_in;
wire [31:0] IMEM_inst_out;

wire DMEM_ena;
wire DMEM_write;
wire DMEM_read;
wire [31:0] DMEM_addr_in;
wire [31:0] DMEM_addr_in_real;
wire [31:0] DMEM_data_in;
wire [31:0] DMEM_data_out;
wire [31:0] pc_out ;

assign IMEM_addr_in = pc_out - 32'h00400000;

assign DMEM_addr_in_real = (DMEM_addr_in_real -  32'h10010000)/4;
    assign pc = pc_out;
assign inst =  IMEM_inst_out;


IMEM Imem(
    .IMEM_addr_in(IMEM_addr_in[12:2]),
    .IMEM_inst_out(IMEM_inst_out)
    );

DMEM Dmem(
    .DMEM_clk(clk_in),
    .DMEM_ena(DMEM_ena),
    .DMEM_write(DMEM_write),
    .DMEM_read(DMEM_read),
    .DMEM_addr_in(DMEM_addr_in_real[10:0]),
    .DMEM_data_in(DMEM_data_in),
    .DMEM_data_out(DMEM_data_out)
    );

CPU single_cpu(
    .CPU_clk(clk_in),
    .CPU_inst_in(IMEM_inst_out),
    .CPU_ena(1'b1),
    .CPU_rst(reset),
    .DMEM_ena(DMEM_ena),
    .DMEM_write(DMEM_write),
```

```
        .DMEM_read(DMEM_read),
        .DMEM_addr_in(DMEM_addr_in),
        .DMEM_data_in(DMEM_data_in),
        .DMEM_data_out(DMEM_data_out),
        .pc_out(pc_out),

        .result(result)
        );

endmodule
```

IMEM.v 文件

```
module IMEM(
    input [10:0] IMEM_addr_in,
    output [31:0] IMEM_inst_out
    );

    dist_mem_gen_0 IMEM_inst_coe(    //实例化 IP 核，导入指令的 coe 文件
        .a(IMEM_addr_in),
        .spo(IMEM_inst_out)
        );

endmodule
```

DMEM.v 文件

```
module DMEM(
    input DMEM_clk,
    input DMEM_ena,
    input DMEM_write,
    input DMEM_read,
    input [10:0] DMEM_addr_in,
    input [31:0] DMEM_data_in,
    output [31:0] DMEM_data_out
    );
```

```
    reg [31:0] DMEM_reg [1024:0] ;

    assign DMEM_data_out = (DMEM_ena && DMEM_read && !DMEM_write)?
DMEM_reg[DMEM_addr_in] : 32'bz ;

    always @(posedge DMEM_clk)
    begin
        if(DMEM_ena && DMEM_write && !DMEM_read)
        begin
            DMEM_reg[DMEM_addr_in] <= DMEM_data_in ;
        end
    end

endmodule
```

CPU.v 文件

```
module CPU(
    input CPU_clk,
    input CPU_rst,
    input CPU_ena,
    input [31:0] CPU_inst_in,
    input [31:0] DMEM_data_out,
    output DMEM_ena,
    output DMEM_write,
    output DMEM_read,
    output [31:0] DMEM_addr_in,
    output [31:0] DMEM_data_in,
    output [31:0] pc_out,

    output [31:0] result
    );

    wire add_ena, addu_ena,
          sub_ena, subu_ena,
          and_ena, or_ena, xor_ena, nor_ena,
          slt_ena, sltu_ena,
          sll_ena, srl_ena, sra_ena, sllv_ena, srlv_ena, srav_ena, //
          jr_ena,
```

```
        addi_ena, addiu_ena,
        andi_ena, ori_ena, xori_ena,
        lw_ena, sw_ena,
        beq_ena, bne_ena,
        slti_ena, sltiu_ena,
        lui_ena,
        j_ena, jal_ena;

    // ALU
    wire [31:0] A,B;


    wire N, Z, V, C;
    // wire [31:0] result;
    wire [4:0] alu_choose;

    // RegFile
    wire RF_write ;
    wire [31:0] RF_Rd_in ;
    wire [31:0] RF_Rs_out ;
    wire [31:0] RF_Rt_out ;

    // PC
    wire [31:0] pc_in ;

    /* NPC  路 */
    wire [31:0] NPC;
    assign NPC = pc_out + 4 ;

    //DMEM
    assign DMEM_ena  = (DMEM_read || DMEM_write)? 1'b1 : 1'b0 ;
    assign DMEM_addr_in = result ;
    assign DMEM_data_in = RF_Rt_out ;



    assign add_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100000)? 1'b1 : 1'b0 ;
    assign addu_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100001)? 1'b1 : 1'b0 ;
    assign sub_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100010)? 1'b1 : 1'b0 ;
    assign subu_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100011)? 1'b1 : 1'b0 ;
```

```
    assign and_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100100)? 1'b1 : 1'b0 ;
    assign or_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100101)? 1'b1 : 1'b0 ;
    assign xor_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100110)? 1'b1 : 1'b0 ;
    assign nor_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b100111)? 1'b1 : 1'b0 ;
    assign slt_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b101010)? 1'b1 : 1'b0 ;
    assign sltu_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b101011)? 1'b1 : 1'b0 ;
    assign sll_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b0)? 1'b1 : 1'b0 ;
    assign srl_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b000010)? 1'b1 : 1'b0 ;
    assign sra_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b000011)? 1'b1 : 1'b0 ;
    assign sllv_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b000100)? 1'b1 : 1'b0 ;
    assign srlv_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b000110)? 1'b1 : 1'b0 ;
    assign srav_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b000111)? 1'b1 : 1'b0 ;
    assign jr_ena = (CPU_inst_in[31:26] == 6'b0 && CPU_inst_in[5:0] ==
6'b001000)? 1'b1 : 1'b0 ;


    assign addi_ena = (CPU_inst_in[31:26] == 6'b001000)? 1'b1 :1'b0 ;
    assign addiu_ena = (CPU_inst_in[31:26] == 6'b001001)? 1'b1 :1'b0 ;
    assign andi_ena = (CPU_inst_in[31:26] == 6'b001100)? 1'b1 :1'b0 ;
    assign ori_ena = (CPU_inst_in[31:26] == 6'b001101)? 1'b1 :1'b0 ;
    assign xori_ena = (CPU_inst_in[31:26] == 6'b001110)? 1'b1 :1'b0 ;
    assign lw_ena = (CPU_inst_in[31:26] == 6'b100011)? 1'b1 :1'b0 ;
    assign sw_ena = (CPU_inst_in[31:26] == 6'b101011)? 1'b1 :1'b0 ;
    assign beq_ena = (CPU_inst_in[31:26] == 6'b000100)? 1'b1 :1'b0 ;
    assign bne_ena = (CPU_inst_in[31:26] == 6'b000101)? 1'b1 :1'b0 ;
    assign slti_ena = (CPU_inst_in[31:26] == 6'b001010)? 1'b1 :1'b0 ;
    assign sltiu_ena = (CPU_inst_in[31:26] == 6'b001011)? 1'b1 :1'b0 ;
    assign lui_ena = (CPU_inst_in[31:26] == 6'b001111)? 1'b1 :1'b0 ;


    assign j_ena = (CPU_inst_in[31:26] == 6'b000010)? 1'b1 :1'b0 ;
    assign jal_ena = (CPU_inst_in[31:26] == 6'b000011)? 1'b1 :1'b0 ;
```

```
    assign alu_choose[4] = (jr_ena || addi_ena || addiu_ena || andi_ena
|| ori_ena || xori_ena || lw_ena || sw_ena ||
                    beq_ena || bne_ena || slti_ena || sltiu_ena ||
lui_ena || j_ena || jal_ena)? 1'b1 :1'b0 ;

    assign alu_choose[3] = (slt_ena || sltu_ena || sll_ena || srl_ena ||
sra_ena || sllv_ena || srlv_ena || srav_ena ||
                    beq_ena || bne_ena || slti_ena || sltiu_ena ||
lui_ena || j_ena || jal_ena)? 1'b1 :1'b0 ;

    assign alu_choose[2] = (and_ena || or_ena || xor_ena || nor_ena ||
sra_ena || sllv_ena || srlv_ena || srav_ena ||
                    ori_ena || xori_ena || lw_ena || sw_ena || lui_ena
|| j_ena || jal_ena)? 1'b1 :1'b0 ;

    assign alu_choose[1] = (sub_ena || subu_ena || xor_ena || nor_ena ||
sll_ena || srl_ena || srlv_ena || srav_ena ||
                    addiu_ena || andi_ena || lw_ena || sw_ena ||
slti_ena || sltiu_ena || jal_ena)? 1'b1 :1'b0 ;

    assign alu_choose[0] = (addu_ena || subu_ena || or_ena || nor_ena ||
sltu_ena || srl_ena || sllv_ena || srav_ena||
                    addi_ena || andi_ena || xori_ena || sw_ena ||
bne_ena || sltiu_ena || j_ena)? 1'b1 :1'b0 ;



    reg [4:0] RsC, RtC, RdC;
    reg [4:0] shamt;
    reg [15:0] immediate;
    reg [25:0] j_address;


    always @(*)
    begin
        RsC <= (add_ena || addu_ena || sub_ena || subu_ena || and_ena ||
or_ena || xor_ena || nor_ena || slt_ena || sltu_ena || sllv_ena || srlv_ena
||
                srav_ena || jr_ena || addi_ena || addiu_ena || andi_ena
|| ori_ena || xori_ena || lw_ena || sw_ena || beq_ena || bne_ena ||
slti_ena ||
                sltiu_ena)? CPU_inst_in[25:21] : 5'bz ;
    end
```

```
    always @(*)
    begin
        RtC <= (add_ena || addu_ena || sub_ena || subu_ena || and_ena ||
or_ena || xor_ena || nor_ena || slt_ena || sltu_ena || sll_ena || srl_ena
||
                sra_ena || sllv_ena || srlv_ena || srav_ena || sw_ena ||
beq_ena || bne_ena )? CPU_inst_in[20:16] : 5'bz ;
    end

    always @(*)
    begin
        RdC <= (add_ena || addu_ena || sub_ena || subu_ena || and_ena ||
or_ena || xor_ena || nor_ena || slt_ena || sltu_ena || sll_ena || srl_ena
||
                sra_ena  ||  sllv_ena  ||  srlv_ena  ||  srav_ena)?
CPU_inst_in[15:11] : (( addi_ena || addiu_ena || andi_ena || ori_ena ||
xori_ena ||
                lw_ena  ||  slti_ena  ||  sltiu_ena  ||  lui_ena)?
CPU_inst_in[20:16] : (jal_ena? 5'b11111 : 5'bz));
    end

    always @(*)
    begin
        shamt <= (sll_ena || srl_ena || sra_ena)? CPU_inst_in[10:6] :
5'bz ;
    end

    always @(*)
    begin
        immediate <= (addi_ena || addiu_ena || andi_ena || ori_ena ||
xori_ena || lw_ena || sw_ena || beq_ena || bne_ena || slti_ena || sltiu_ena
||
                      lui_ena)? CPU_inst_in[15:0] : 16'bz ;
    end

    always @(*)
    begin
        j_address <= (j_ena || jal_ena)? CPU_inst_in[25:0] : 26'bz ;
    end



    wire [31:0] EXT_1_out ;
```

```
    wire [31:0] EXT_2_out ;
    wire [31:0] EXT_3_out ;
    wire [31:0] EXT_4_out ;
    wire [31:0] EXT_5_out ;
    wire [31:0] EXT_6_out ;

    wire [4:0] EXT_ena ;


    assign EXT_ena[0] = (slt_ena || sltu_ena || slti_ena || sltiu_ena)?
1'b1 : 1'b0 ;
    assign EXT_ena[1] = (sll_ena || srl_ena || sra_ena)? 1'b1 : 1'b0 ;
    assign EXT_ena[2] = (addi_ena || addiu_ena || lw_ena || sw_ena ||
slti_ena || sltiu_ena)? 1'b1 : 1'b0 ;
    assign EXT_ena[3] = (andi_ena || ori_ena || xori_ena || lui_ena)?
1'b1 : 1'b0 ;
    assign EXT_ena[4] = (beq_ena || bne_ena)? 1'b1 : 1'b0 ;


    assign EXT_1_out = (slt_ena || sltu_ena || slti_ena || sltiu_ena)?
{31'b0 , result[0]} : 32'bz ;
    assign EXT_2_out = (sll_ena || srl_ena || sra_ena)? {27'b0, shamt} :
32'bz ;
    assign EXT_3_out = (addi_ena || addiu_ena || lw_ena || sw_ena ||
slti_ena || sltiu_ena)? {{16{immediate[15]}} , immediate} : 32'bz ;
    assign EXT_4_out = (andi_ena || ori_ena || xori_ena || lui_ena)?
{16'b0 , immediate} : 32'bz ;
    assign EXT_5_out = (beq_ena || bne_ena)? {{14{immediate[15]}} ,
immediate , 2'b0} : 32'bz ;



    wire GET_ena ;
    wire [31:0] GET_out ;

    assign GET_ena = (j_ena || jal_ena)? 1'b1 : 1'b0 ;
    assign GET_out = (GET_ena)? {pc_out[31:28] , j_address , 2'b0} :
32'bz ;


    assign DMEM_read = (lw_ena)? 1'b1 : 1'b0 ;
    assign DMEM_write = (sw_ena) ? 1'b1 : 1'b0 ;
```

```
    wire MUX_A_ena ;
    wire MUX_B_ena ;
    wire MUX_PC_ena ;


    wire [31:0] MUX_A_out ;
    wire [31:0] MUX_B_out ;
    wire [31:0] MUX_PC_out ;

    assign pc_in = MUX_PC_out ;

    assign MUX_A_ena = (sll_ena || srl_ena || sra_ena)? 1'b1 : 1'b0 ;
    assign MUX_A_out = (MUX_A_ena)? EXT_2_out : ((sllv_ena || srlv_ena
|| srav_ena)? {27'b0 , RF_Rs_out[4:0]} :  RF_Rs_out);

    assign MUX_B_ena = (addi_ena || addiu_ena || andi_ena || ori_ena ||
xori_ena || lw_ena || sw_ena ||
                        slti_ena || sltiu_ena || lui_ena)? 1'b1 : 1'b0 ;
    assign MUX_B_out = (MUX_B_ena)? ((andi_ena || ori_ena || xori_ena ||
lui_ena)? EXT_4_out : EXT_3_out) : RF_Rt_out ;

    assign A = MUX_A_out ;
    assign B = MUX_B_out ;


    assign MUX_PC_ena = (jr_ena || beq_ena || bne_ena || j_ena || jal_ena)?
1'b1 : 1'b0 ;
    assign beq_bne_ena = beq_ena ? 1'b1 : ((bne_ena)? 1'b0 : 1'b1);
    assign MUX_PC_out = (MUX_PC_ena)? ((j_ena || jal_ena)? GET_out :
((jr_ena)? RF_Rs_out : (beq_bne_ena)? NPC : NPC + EXT_5_out)): NPC ;


    wire MUX_1_ena ;
    wire MUX_2_ena ;
    wire MUX_3_ena ;

    wire [31:0] MUX_1_out ;
    wire [31:0] MUX_2_out ;
    wire [31:0] MUX_3_out ;


    assign MUX_1_ena = (add_ena || addu_ena || sub_ena || subu_ena ||
and_ena || or_ena || xor_ena || nor_ena ||
```

```
                        sll_ena || srl_ena || sra_ena || sllv_ena ||
srlv_ena || srav_ena || lui_ena || addi_ena ||
                        addiu_ena || andi_ena || ori_ena || xori_ena)?
1'b1 : 1'b0 ;

    assign MUX_2_ena = (jal_ena)? 1'b1 : 1'b0 ;

    assign MUX_3_ena = (!lw_ena)? 1'b1 : 1'b0 ;


    assign MUX_1_out = (MUX_1_ena)? result : EXT_1_out ;

    assign MUX_2_out = (MUX_2_ena)? pc_out + 8 : MUX_3_out ;

    assign MUX_3_out = (MUX_3_ena)? MUX_1_out : DMEM_data_out ;

    assign RF_Rd_in =  MUX_2_out ;


    assign RF_write = (jr_ena || sw_ena || beq_ena || bne_ena || j_ena) ?
1'b0 : 1'b1 ;



    ALU Alu(
        .A(A),
        .B(B),
        .alu_choose(alu_choose),
        .result(result),
        .N(N),
        .Z(Z),
        .V(V),
        .C(C)
        );

    RegFile Regfile(
        .RF_clk(CPU_clk),          //时  锟脚猴拷
        .RF_rst(CPU_rst),          // 锟脚猴拷
        .RF_ena(CPU_ena),              //使  锟脚猴拷
        .RF_write(RF_write),       //锟侥硫达拷     写  锟脚猴拷
        .RsC(RsC),                 //Rs  址
        .RtC(RtC),                 //Rt  址
        .RdC(RdC),                 //Rd  址
        .RF_Rd_in(RF_Rd_in),       //Rd 写
```

```
        .RF_Rs_out(RF_Rs_out),    //Rs
        .RF_Rt_out(RF_Rt_out)     //Rt
        );

    PC Pc(
        .pc_clk(CPU_clk),
        .pc_rst(CPU_rst),
        .pc_ena(CPU_ena),
        .pc_in(pc_in),
        .pc_out(pc_out)
        );


endmodule
```

ALU.v 文件

```
module ALU(
    input [31:0] A,
    input [31:0] B,
    input [4:0] alu_choose,
    output reg [31:0] result,
    output N,
    output Z,
    output V,
    output C
    );

    // 设置四个标志位
    assign N = result[31];
    assign Z = (result == 32'b0)? 1 : 0 ;
    assign V = result[32];
    assign C = result[32];

    //ALU 分流表 所有指令都包含在其中
    parameter ADD  = 5'b00000;
    parameter ADDU = 5'b00001;
    parameter SUB  = 5'b00010;
    parameter SUBU = 5'b00011;
    parameter AND  = 5'b00100;
```

```
parameter OR   = 5'b00101;
parameter XOR  = 5'b00110;
parameter NOR  = 5'b00111;
parameter SLT  = 5'b01000;
parameter SLTU = 5'b01001;
parameter SLL  = 5'b01010;
parameter SRL  = 5'b01011;
parameter SRA  = 5'b01100;
parameter SLLV = 5'b01101;
parameter SRLV = 5'b01110;
parameter SRAV = 5'b01111;
parameter JR   = 5'b10000;
parameter ADDI = 5'b10001;
parameter ADDIU= 5'b10010;
parameter ANDI = 5'b10011;
parameter ORI  = 5'b10100;
parameter XORI = 5'b10101;
parameter LW   = 5'b10110;
parameter SW   = 5'b10111;
parameter BEQ  = 5'b11000;
parameter BNE  = 5'b11001;
parameter SLTI = 5'b11010;
parameter SLTIU= 5'b11011;
parameter LUI  = 5'b11100;
parameter J    = 5'b11101;
parameter JAL  = 5'b11110;

//实现无符号数 有符号化
wire signed [31:0] signA , signB;
assign signA = A;
assign signB = B;

//实现 ALU 的分流
always @(*)
begin
    case(alu_choose)
        ADD :
            begin
                result <= signA + signB ;
            end
        ADDU :
            begin
                result <= A + B ;
            end
```

```
SUB :
    begin
        result <= signA - signB ;
    end
SUBU :
    begin
        result <= A - B ;
    end
AND :
    begin
        result <= A & B ;
    end
OR :
    begin
        result <= A | B ;
    end
XOR :
    begin
        result <= A ^ B ;
    end
NOR :
    begin
        result <= ~(A | B) ;
    end
SLT :
    begin
        result <= (signA < signB)? 32'b1 : 32'b0 ;
    end
SLTU :
    begin
        result <= (A < B)? 32'b1 : 32'b0 ;
    end
SLL :
    begin
        result <= B << A ;
    end
SRL :
    begin
        result <= B >> A ;
    end
SRA :
    begin
        result <= signB >>> A ;
    end
```

```
SLLV :
    begin
        result <= B << A ;
    end
SRLV :
    begin
        result <= B >> A ;
    end
SRAV :
    begin
        result <= signB >>> A ;
    end
JR :
    begin
    end
ADDI :
    begin
        result <= signA + signB ;
    end
ADDIU :
    begin
        result <= signA + signB ;
    end
ANDI :
    begin
        result <= A & B ;
    end
ORI :
    begin
        result <= A | B ;
    end
XORI :
    begin
        result <= A ^ B ;
    end
LUI :
    begin
        result <= { B[15:0] , 16'b0 } ;
    end
LW :
    begin
    end
SW :
    begin
```

```
                end
            BEQ :
                begin
                end
            BNE :
                begin
                end
            SLTI :
                begin
                    result <= (signA < signB)? 32'b1 : 32'b0 ;
                end
            SLTIU :
                begin
                    result <= (A < B)? 32'b1 : 32'b0 ;
                end
            J :
                begin
                end
            JAL :
                begin
                end
            default:
                result <= A + B ;
        endcase
    end

endmodule
```

RegFile.v 文件

```
module RegFile(
    input RF_clk,
    input RF_rst,
    input RF_ena,
    input RF_write,
    input [4:0] RsC,
    input [4:0] RtC,
    input [4:0] RdC,
    input [31:0] RF_Rd_in,
    output [31:0] RF_Rs_out,
```

```
output [31:0] RF_Rt_out
);

// 定义寄存器堆
reg [31:0] RF_Regfiles [31:0] ;

//使能信号为高电平时， 将对应寄存器数据输出
assign RF_Rs_out = (RF_ena)? RF_Regfiles[RsC] : 32'bz ;
assign RF_Rt_out = (RF_ena)? RF_Regfiles[RtC] : 32'bz ;

//初始化寄存器堆
always @(posedge RF_rst)
begin
    RF_Regfiles[0] <= 32'b0 ;
    RF_Regfiles[1] <= 32'b0 ;
    RF_Regfiles[2] <= 32'b0 ;
    RF_Regfiles[3] <= 32'b0 ;
    RF_Regfiles[4] <= 32'b0 ;
    RF_Regfiles[5] <= 32'b0 ;
    RF_Regfiles[6] <= 32'b0 ;
    RF_Regfiles[7] <= 32'b0 ;
    RF_Regfiles[8] <= 32'b0 ;
    RF_Regfiles[9] <= 32'b0 ;
    RF_Regfiles[10] <= 32'b0 ;
    RF_Regfiles[11] <= 32'b0 ;
    RF_Regfiles[12] <= 32'b0 ;
    RF_Regfiles[13] <= 32'b0 ;
    RF_Regfiles[14] <= 32'b0 ;
    RF_Regfiles[15] <= 32'b0 ;
    RF_Regfiles[16] <= 32'b0 ;
    RF_Regfiles[17] <= 32'b0 ;
    RF_Regfiles[18] <= 32'b0 ;
    RF_Regfiles[19] <= 32'b0 ;
    RF_Regfiles[20] <= 32'b0 ;
    RF_Regfiles[21] <= 32'b0 ;
    RF_Regfiles[22] <= 32'b0 ;
    RF_Regfiles[23] <= 32'b0 ;
    RF_Regfiles[24] <= 32'b0 ;
    RF_Regfiles[25] <= 32'b0 ;
    RF_Regfiles[26] <= 32'b0 ;
    RF_Regfiles[27] <= 32'b0 ;
    RF_Regfiles[28] <= 32'b0 ;
    RF_Regfiles[29] <= 32'b0 ;
    RF_Regfiles[30] <= 32'b0 ;
```

```
        RF_Regfiles[31] <= 32'b0 ;
    end

    // 在时钟上升沿来临时， 写入数据
    always @(posedge RF_clk)
    begin
        if(RF_ena && RF_write)
        begin
            if(RdC != 0)
            begin
                RF_Regfiles[RdC] <= RF_Rd_in ;
            end
        end
    end

endmodule
```

PC.v 文件

```
module PC(
    input pc_clk,
    input pc_rst,
    input pc_ena,
    input [31:0] pc_in,    //下一条要执行的指令
    output [31:0] pc_out   //当前要执行的指令
    );

    // 设置 PC 寄存器
    reg [31:0] pc_reg = 32'h00400000 ;

    // 读出要执行的指令
    assign pc_out = (pc_ena)? pc_reg : 32'bz ;

    // 复位信号上升沿时，恢复原值
    always @(posedge pc_rst)
    begin
        if(pc_ena)
        begin
            pc_reg <= 32'h00400000 ;
        end
```

```
    end

    // 时钟信号上升沿时，写入下一条要执行的指令
    always @(posedge pc_clk)
    begin
        if(pc_ena)
            begin
                pc_reg <= pc_in ;
            end
    end

endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

```
module CPU_before_simulation_tb;
    reg clk;
    reg rst;
    wire [31:0] inst;
    wire [31:0] pc;
    reg  [31:0] cnt;

    wire  [31:0] result ;

    integer file_open;

    initial
    begin
        clk = 1'b0;
        rst = 1'b1;
        cnt = 0;
        #10
        rst = 1'b0;
        clk = 1'b0;
        repeat(1000)
```

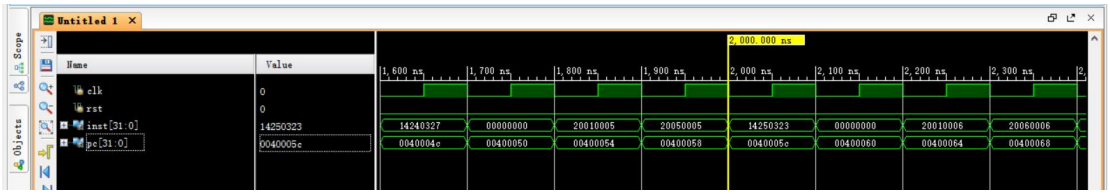
```
        #10 clk = ~clk;
    end


    always @ (posedge clk) begin
        cnt <= cnt + 1'b1;
        file_open   =   $fopen("D:\\31_SingleCPU\\31_real_output.txt",
"a+");
        $fdisplay(file_open, "OP: %d", cnt);
        $fdisplay(file_open, "Instr_addr = %h", sc_inst.inst);
        $fdisplay(file_open,          "$zero          =          %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[0]);
        $fdisplay(file_open,       "$at                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[1]);
        $fdisplay(file_open,       "$v0                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[2]);
        $fdisplay(file_open,       "$v1                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[3]);
        $fdisplay(file_open,       "$a0                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[4]);
        $fdisplay(file_open,       "$a1                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[5]);
        $fdisplay(file_open,       "$a2                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[6]);
        $fdisplay(file_open,       "$a3                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[7]);
        $fdisplay(file_open,       "$t0                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[8]);
        $fdisplay(file_open,       "$t1                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[9]);
        $fdisplay(file_open,       "$t2                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[10]);
        $fdisplay(file_open,       "$t3                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[11]);
        $fdisplay(file_open,       "$t4                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[12]);
        $fdisplay(file_open,       "$t5                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[13]);
        $fdisplay(file_open,       "$t6                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[14]);
        $fdisplay(file_open,       "$t7                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[15]);
        $fdisplay(file_open,       "$s0                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[16]);
```

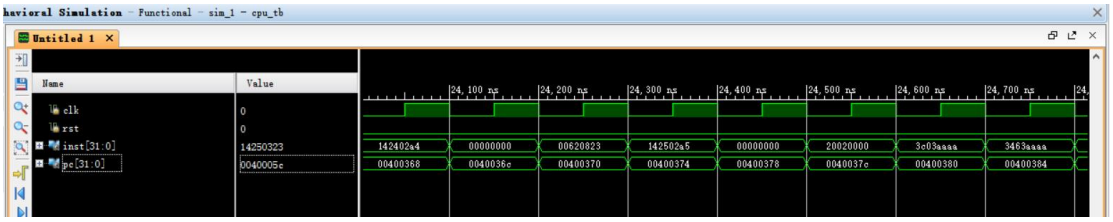
```
        $fdisplay(file_open,       "$s1                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[17]);
        $fdisplay(file_open,       "$s2                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[18]);
        $fdisplay(file_open,       "$s3                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[19]);
        $fdisplay(file_open,       "$s4                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[20]);
        $fdisplay(file_open,       "$s5                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[21]);
        $fdisplay(file_open,       "$s6                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[22]);
        $fdisplay(file_open,       "$s7                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[23]);
        $fdisplay(file_open,       "$t8                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[24]);
        $fdisplay(file_open,       "$t9                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[25]);
        $fdisplay(file_open,       "$k0                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[26]);
        $fdisplay(file_open,       "$k1                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[27]);
        $fdisplay(file_open,       "$gp                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[28]);
        $fdisplay(file_open,       "$sp                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[29]);
        $fdisplay(file_open,       "$fp                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[30]);
        $fdisplay(file_open,       "$ra                    =       %h",
sc_inst.single_cpu.Regfile.RF_Regfiles[31]);
        $fdisplay(file_open, "$pc   = %h\n", sc_inst.pc);
        $fclose(file_open);
    end

    sccomp_dataflow sc_inst(
        .clk_in(clk),
        .reset(rst),
        .inst(inst),
        .pc(pc),

        .result(result)
    );

endmodule
```

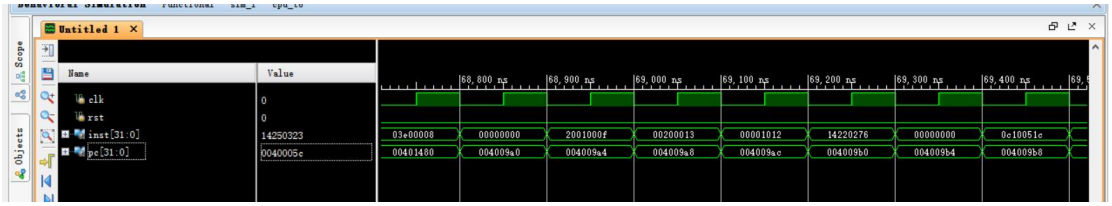
五、实验结果

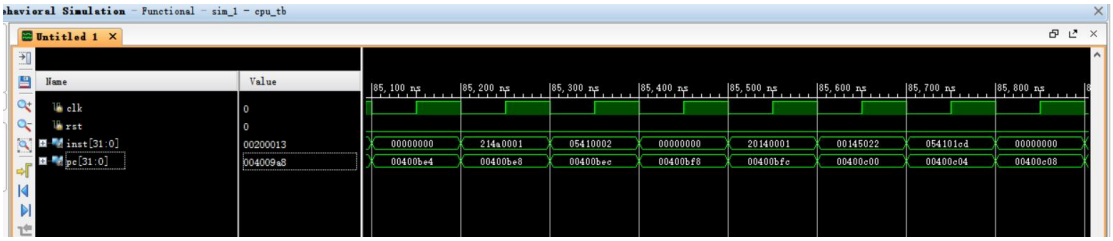
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

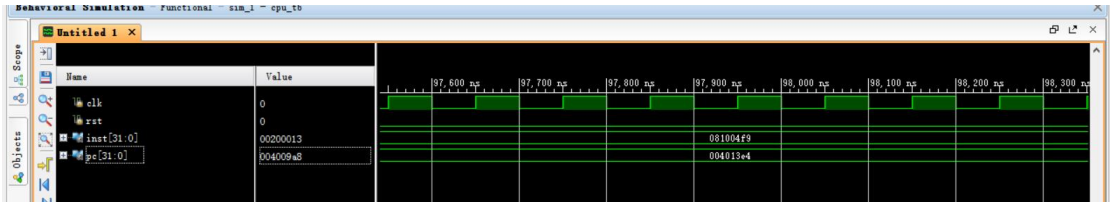
前仿真：





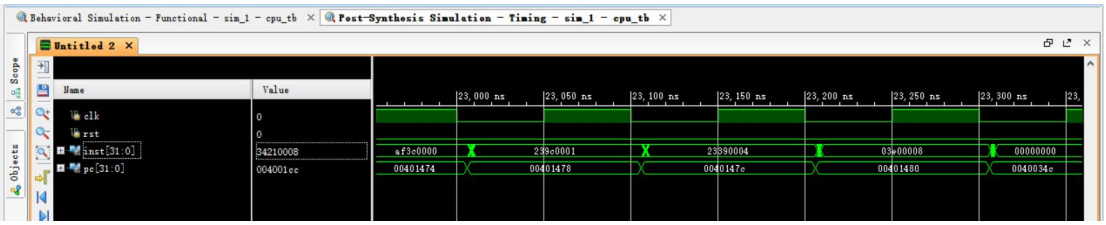


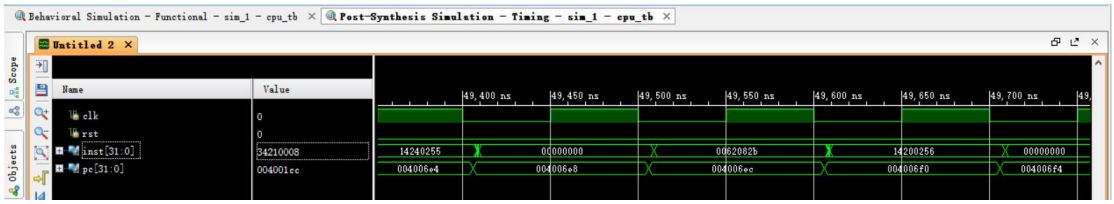


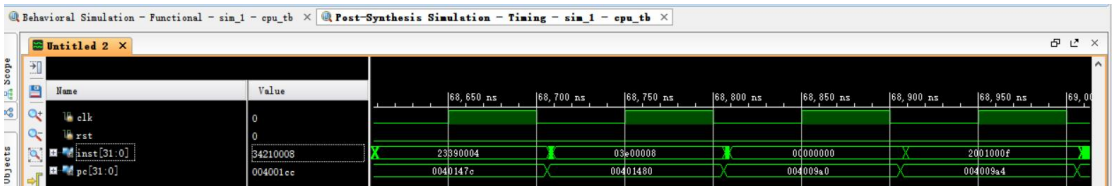


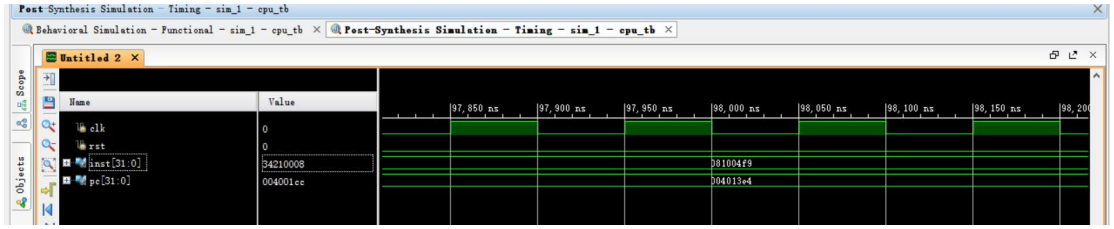
后仿真：


Untitled 2
Name
Value
clk
rst
inst[31:0]
pc[31:0]
0
0
34210008
004001cc
11,728.395 ns
11,700 ns
11,750 ns
11,800 ns
11,850 ns
11,900 ns
11,950 ns
12,000 ns
3c011001
34210008
3c02ffff
3442ffff
ac220000
004001c8
004001cc
004001d0
00400144
00400148

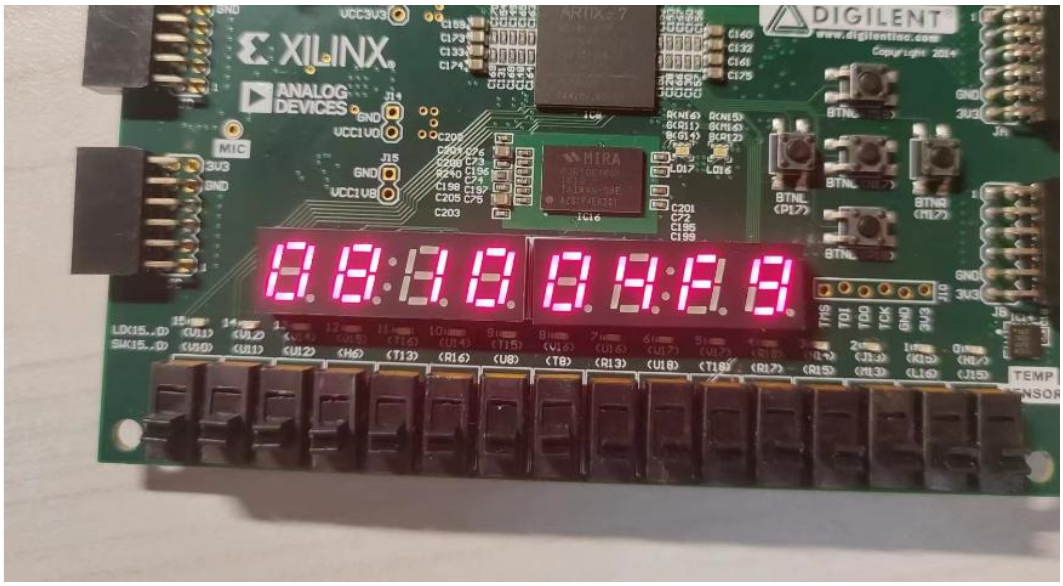

Behavioral Simulation - Functional - sim_1 - cpu_tb
Post-Synthesis Simulation - Timing - sim_1 - cpu_tb
Untitled 2
Scope
Objects
Name
Value
clk
rst
inst[31:0]
pc[31:0]
0
0
34210008
004001cc
23,000 ns
23,050 ns
23,100 ns
23,150 ns
23,200 ns
23,250 ns
23,300 ns
af3c0000
239c0001
23390004
03e00008
00000000
00401474
00401478
0040147c
00401480
0040034c

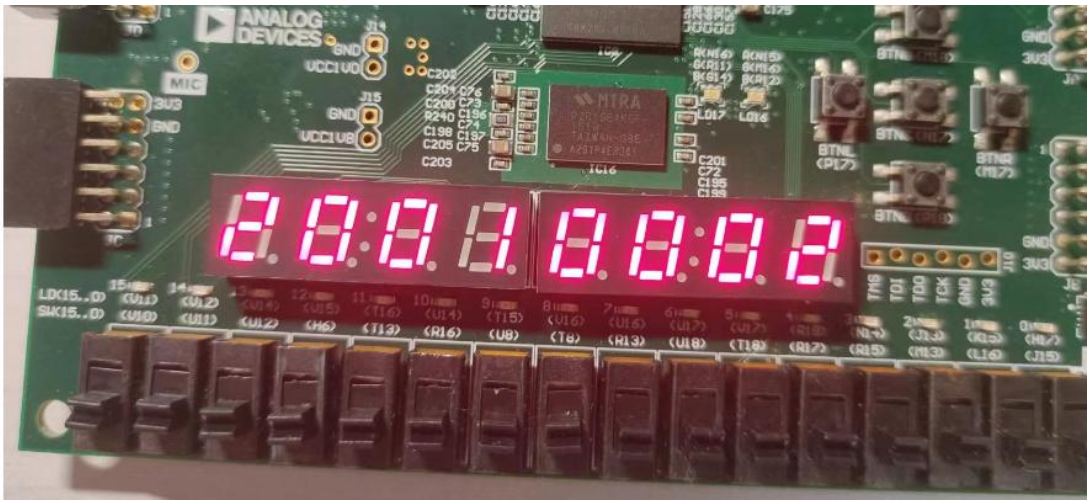

Behavioral Simulation - Functional - sim_1 - cpu_tb
Post-Synthesis Simulation - Timing - sim_1 - cpu_tb
Untitled 2
Scope
Objects
Name
Value
clk
rst
inst[31:0]
pc[31:0]
0
0
34210008
004001cc
49,400 ns
49,450 ns
49,500 ns
49,550 ns
49,600 ns
49,650 ns
49,700 ns
14240255
00000000
0062082b
14200256
00000000
004006e4
004006e8
004006ec
004006f0
004006f4

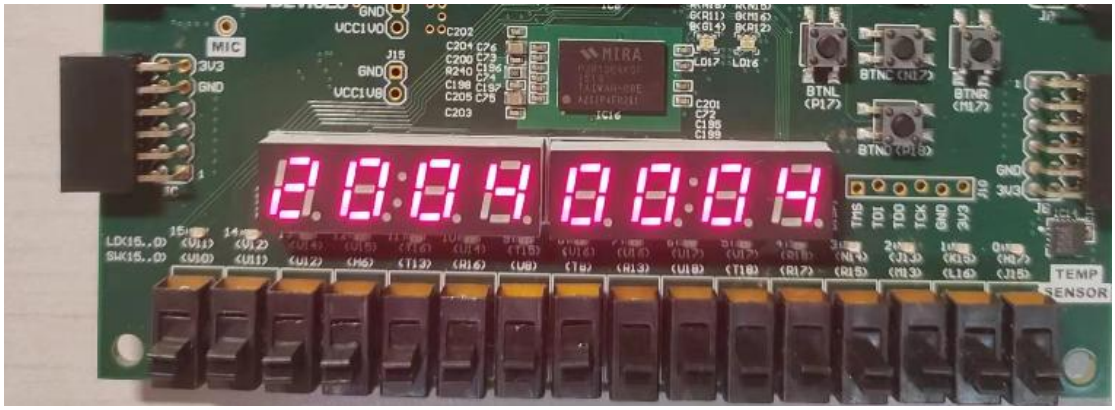

Behavioral Simulation - Functional - sim_1 - cpu_tb
Post-Synthesis Simulation - Timing - sim_1 - cpu_tb
Untitled 2
Name
Value
clk
rst
inst[31:0]
pc[31:0]
0
0
34210008
004001cc
68,650 ns
68,700 ns
68,750 ns
68,800 ns
68,850 ns
68,900 ns
68,950 ns
23390004
03e00008
00000000
2001000f
0040147c
00401480
004009a0
004009a4

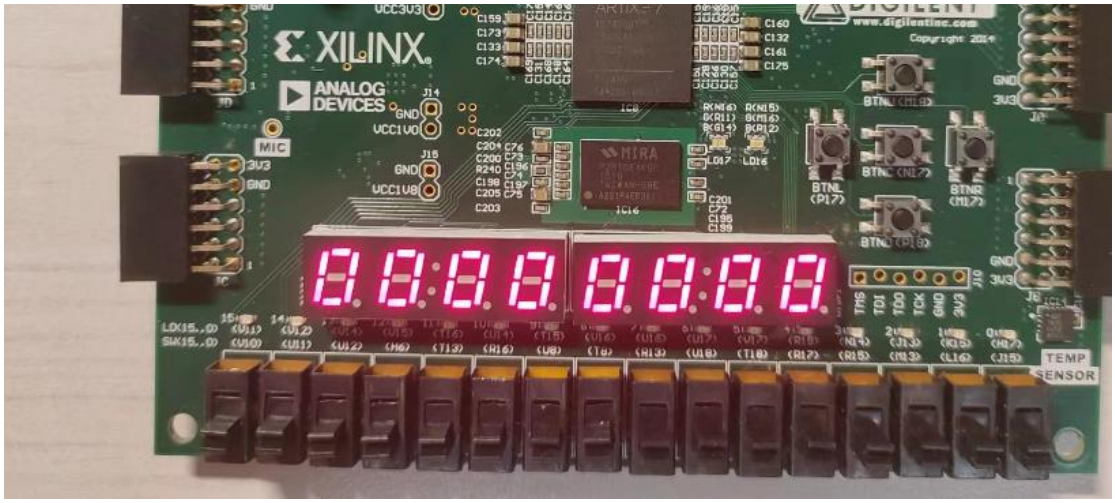

Post-Synthesis Simulation - Timing - sim_1 - cpu_tb
Behavioral Simulation - Functional - sim_1 - cpu_tb
Post-Synthesis Simulation - Timing - sim_1 - cpu_tb
Untitled 2
Scope
Objects
Name
Value
clk
rst
inst[31:0]
pc[31:0]
0
0
34210008
004001cc
97,850 ns
97,900 ns
97,950 ns
98,000 ns
98,050 ns
98,100 ns
98,150 ns
081004f9
004013e4

下板：

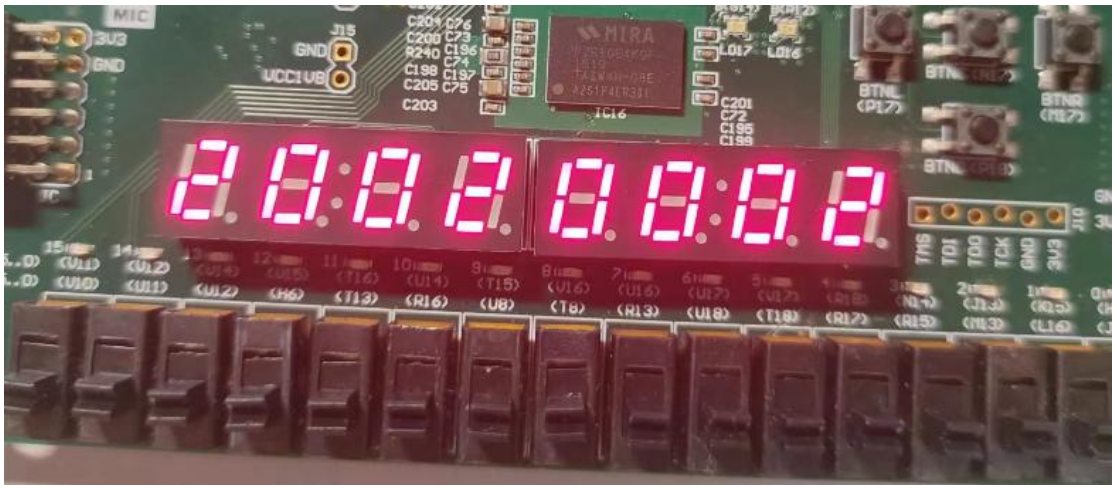

XILINX
ANALOG DEVICES
DIGILENT
MIRA
MIC
BTNL (P17)
BTNC (N17)
BTNR (M17)
TEMP SENSOR


ANALOG DEVICES
MIRA
MIC
BTNL (P17)
BTNR (M17)


MIRA
MIC
BTNL (P17)
BTNC (N17)
BTNR (M17)
TEMP SENSOR


XILINX
ANALOG DEVICES
MIC
TEMP SENSOR


XILINX
ANALOG DEVICES
MIC
TEMP SENSOR


MIC
MIRA