Lab 07 流水线 CPU

PB16051448 赵敏帆

实验目的

- 编写 MIPS 指令集流水线 CPU
- 实现指令集中一定数量的指令（16条），在此基础上，增加了 15 条指令。

  > 实现的指令见附录

- 实现了完全转发和分支预测，实现了暂停。
- 实现了下载，可以用开关输入地址，并使用数码管展示。

实验平台

- Vivado 2016.4
- FPGA 开发板 STEP （Artix-7 型）

实验过程及结果

实验具体步骤

1. 分析 CPU 所需要实现的逻辑，设计项目所需的模块
2. 根据需要实现的逻辑编写代码实现基础的流水线 CPU
3. 根据需要实现的指令，对 CPU 进行改进。
4. 编写对应的验证汇编程序
5. 对内存进行例化
6. 编写对应仿真程序，进行仿真测试
7. 下载验证

实验过程——CPU 设计及代码编写

对于代码量较小模块，直接附注代码实现，对于代码量太大的模块，代码在附录中给出。

本部分一共分为 15 模块，其中 PC、ID、regfile、ALU、MEM、开关显示、TOP 模块为核心模块。

define 部分

为了减少后需的代码量，将一些使用较为频繁的常量定义在了 define 文件中，便于之后的使用。

> 这一部分代码在附录中给出

stall 模块设计

stall 信号处理

```
module Stall(
```

```
input rst,
input stall_id,
output reg [5:0] stall
    );
always@(*)
begin
    if(~rst)
        begin
            stall <= 6'b000000;
        end
    else if(stall_id)
        begin
            stall <= 6'b000111;
        end
    else
        begin
            stall <= 6'b000000;
        end
end

endmodule
```

PC 模块设计

该模块负责控制 PC，整个 CPU 的主体设计从这个模块开始。在该模块中增加了 stall 信号和 branch 信号。stall 信号用于解决无法通过转发解决的数据相关，使用它来将流水线推迟等待。而 branch 指令则负责控制跳转指令 pc 的变换。

```
`include "Define.v"
module pc(

    input                           clk,
    input                       rst,
    input  [5:0]                stall,
    input                       branch,
    input  [31:0]               branch_addr,
    output reg [31:0]           pc = 32'd0

);

    always @ (posedge clk) begin
        if (~rst)
            begin
                pc <= 32'h00000000;
            end
        else if (stall[0] == 1'd0)
        begin
            if(branch == 1'b1)
                begin
                    pc <= branch_addr;
                end

            else
```

```
                begin
                    pc <= pc + 4'd4;
                end
        end
    end


endmodule
```

InstRom 模块

该模块用于存储指令，使用 Vivado 中的 IP 核来生成。该模块为 Distributed Simple Port Rom，使用异步读。大小为 32bit*256。

IF_ID 模块

该模块为 IF 和 ID 的中间模块，即流水线的段寄存器部分。在实现 IF_ID 模块时，加入了分支预测的处理信号 last_branch。

```
`include "Define.v"
module if_id(
input clk,
input rst,
input [5:0] stall,
input [31:0] if_pc,
input [31:0] if_inst,
input branch,
output reg [31:0] id_pc,
output reg [31:0] id_inst,
output reg last_branch
    );
always@(posedge clk)
begin
    if(~rst)
        begin
            id_pc <= 32'd0;
            id_inst <= 32'd0;
            last_branch<= 1'd0;
        end
    else if(stall[1] == 1'd1 && stall[2] == 1'd0)
        begin
            id_pc <= 32'd0;
            id_inst <= 32'd0;
            last_branch<= 1'd0;
        end
    else if (stall[1] == 1'd1)
        begin
            id_pc <= id_pc;
            id_inst <= id_inst;
            last_branch<= last_branch;

        end
```

```
    else
        begin
            id_pc <= if_pc;
            id_inst <= if_inst;
            last_branch <= branch;
        end
end
endmodule
```

ID 模块

该部分为译码模块，也是设计中最复杂的模块。在其中实现了 31 条指令的译码。并在该模块中实现转发和暂停信号。

> 该模块由于代码量太大，在附录中给出。

regfile 寄存器文件

寄存器文件模块，实现 CPU 的 32 个寄存器，并在此模块中整合 ALU Source 的选择器进行转发操作。

```
`include "Define.v"
module regfile(
    input                                   clk,
    input                                   rst,
    input                                   we,
    input [31:0]                            waddr,
    input [31:0]                            wdata,
    input                                   re1,
    input [31:0]                            raddr1,
    output reg [31:0]                      rdata1,
    input                                   re2,
    input [31:0]                            raddr2,
    output reg [31:0]                      rdata2

);

    reg[31:0]  regs[0:31];
    always @(posedge clk or negedge rst) begin
        if (!rst) begin
            // reset
            regs[0]<=0;
    regs[1]<=0;
    regs[2]<=0;
    regs[3]<=0;
    regs[4]<=0;
    regs[5]<=0;
    regs[6]<=0;
    regs[7]<=0;
    regs[8]<=0;
    regs[9]<=0;
    regs[10]<=0;
    regs[11]<=0;
    regs[12]<=0;
```

```
    regs[13]<=0;
    regs[14]<=0;
    regs[15]<=0;
    regs[16]<=0;
    regs[17]<=0;
    regs[18]<=0;
    regs[19]<=0;
    regs[20]<=0;
    regs[21]<=0;
    regs[22]<=0;
    regs[23]<=0;
    regs[24]<=0;
    regs[25]<=0;
    regs[26]<=0;
    regs[27]<=0;
    regs[28]<=0;
    regs[29]<=0;
    regs[30]<=0;
    regs[31]<=0;
        end
        else begin
            if((we == 1'd1) && (waddr != 5'h0))
                begin
                    regs[waddr] <= wdata;
                end
        end
    end

    always @ (*) begin
        if(~rst) begin
              rdata1 <= 32'd0;
        end else if(raddr1 == 5'h0) begin
            rdata1 <= 32'd0;
        end else if((raddr1 == waddr) && (we == 1'd1) && (re1 == 1'd1)) begin
          rdata1 <= wdata;
        end else if(re1 == 1'd1) begin
          rdata1 <= regs[raddr1];
        end else begin
          rdata1 <= 32'd0;
        end
    end

    always @ (*) begin
        if(~rst) begin
              rdata2 <= 32'd0;
      end else if(raddr2 == 5'h0) begin
            rdata2 <= 32'd0;
      end else if((raddr2 == waddr) && (we == 1'd1) && (re2 == 1'd1)) begin
          rdata2 <= wdata;
      end else if(re2 == 1'd1) begin
          rdata2 <= regs[raddr2];
      end else begin

          rdata2 <= 32'd0;
```

```
      end
    end

endmodule
```

ID_EX 模块

译码和执行模块之间的流水段寄存器模块。该模块较简单，不作赘述。

```
`include "Define.v"
module id_ex(
    input                   clk,
    input                   rst,
    input [5:0]                 stall,
    input  [7:0]            id_aluop,
    input  [2:0]            id_alusel,
    input  [31:0]           id_reg1,
    input  [31:0]           id_reg2,
    input  [31:0]           id_wd,
    input                   id_wreg,
    input   [31:0]          id_inst,
    output reg[7:0]       ex_aluop,
    output reg[2:0]      ex_alusel,
    output reg[31:0]            ex_reg1,
    output reg[31:0]            ex_reg2,
    output reg[31:0]            ex_wd,
    output reg                  ex_wreg,
    output reg [31:0]           ex_inst
);

    always @ (posedge clk) begin
        if (~rst) begin
            ex_aluop <= `NOP_OP;
            ex_alusel <= `RES_NOP;
            ex_reg1 <= 32'd0;
            ex_reg2 <= 32'd0;
            ex_wd <= 32'd0;
            ex_wreg <= 1'd0;
            ex_inst <= 32'd0;
        end else if(stall[2] == 1'd1 && stall[3] == 1'd0) begin
            ex_aluop <= `NOP_OP;
            ex_alusel <= `RES_NOP;
            ex_reg1 <= 32'd0;
            ex_reg2 <= 32'd0;
            ex_wd <= 32'd0;
            ex_wreg <= 1'd0;
            ex_inst <= 32'd0;
        end else if(stall[2] == 1'd0) begin
            ex_aluop <= id_aluop;
            ex_alusel <= id_alusel;

            ex_reg1 <= id_reg1;
```

```
            ex_reg2 <= id_reg2;
            ex_wd <= id_wd;
            ex_wreg <= id_wreg;
            ex_inst <= id_inst;
        end
    end
endmodule
```

ALU 模块

ALU 模块，亦即执行模块，根据译码得到的信号执行对应的操作。

为了加快计算速度，同时进行逻辑运算、移位运算、数值计算运算，最后通过选择器选择需要的结果。

> 代码在附录中给出。

EX_MEM 模块

执行和访存之间的流水段寄存器模块。

```
`include "Define.v"
module ex_mem(
    input                           clk,
    input                           rst,
    input [5:0]                         stall,
    input [31:0]                    ex_wd,
    input                           ex_wreg,
    input [31:0]                    ex_wdata,
    input [7:0]                         ex_aluop,
    input [31:0]                        ex_mem_addr,
    input [31:0]                        ex_reg2,
    output reg[31:0]                    mem_wd,
    output reg                          mem_wreg,
    output reg[31:0]                    mem_wdata,
    output reg [7:0]                    mem_aluop,
    output reg [31:0]                   mem_mem_addr,
    output reg [31:0]                   mem_reg2
);
    always @ (posedge clk) begin
        if(~rst)
        begin
            mem_wd <= 32'd0;
            mem_wreg <= 31'd0;
            mem_wdata <= 31'd0;
            mem_aluop <= 8'd0;
            mem_mem_addr <= 32'd0;
            mem_reg2 <= 32'd0;
        end
        else if (stall[3] == 1'd1 && stall[4]==1'd0)
            begin
                mem_wd <= 32'd0;
                mem_wreg <= 31'd0;

                mem_wdata <= 31'd0;
```

```
                mem_aluop <= 8'd0;
                mem_mem_addr <= 32'd0;
                mem_reg2 <= 32'd0;
            end
        else
        begin
            mem_wd <= ex_wd;
            mem_wreg <= ex_wreg;
            mem_wdata <= ex_wdata;
            mem_aluop <= ex_aluop;
            mem_mem_addr <= ex_mem_addr;
            mem_reg2 <= ex_reg2;
        end
    end
endmodule
```

MEM 模块

该模块本可以集成在 `TOP` 模块中，但为了方便调试和扩展，将其集成为一个模块来进行处理。主要负责访存前的一些准备工作，根据 `alu_op` 来判断读或写内存，并给出相应的信号给 RAM 进行操作。同时也负责 R-Type 指令的继续后移。

```
`include "Define.v"
module mem(
    input                           rst,
    input  [31:0]                   wd_i,
    input                           wreg_i,
    input  [31:0]                   wdata_i,
    input  [7:0]                    alu_op,
    input  [31:0]                   mem_data_i,
    input  [31:0]                   mem_addr_i,
    input  [31:0]                   mem_reg2,
    output reg [31:0]               wd_o,
    output reg                      wreg_o,
    output reg [31:0]               wdata_o,
    output reg [31:0]               mem_addr_o,
    output reg                      mem_we,
    output reg [31:0]               mem_data_o
);

    always @ (*) begin
        if(~rst) begin
            wd_o <= 32'd0;
            wreg_o <= 1'd0;
            wdata_o <= 32'd0;
            mem_addr_o <= 32'd0;
            mem_we <= 1'd0;
            mem_data_o <= 32'd0;
        end else begin
             wd_o <= wd_i;
            wreg_o <= wreg_i;

            wdata_o <= wdata_i;
```

```
            mem_we <= 1'd0;
            mem_addr_o <= 32'd0;
            case(alu_op)
                `LW_OP:     begin
                    mem_addr_o <= mem_addr_i;
                    mem_we <= 1'd0;
                    wdata_o <= mem_data_i;
                end
                `SW_OP:     begin
                    mem_addr_o <= mem_addr_i;
                    mem_we <= 1'd1;
                    mem_data_o <= mem_reg2;
                end
            default:
            begin
            end
            endcase
        end
    end
endmodule
```

MEM_WB 模块

访存和写回之间的流水段寄存器模块。

```
`include "Define.v"
module mem_wb(
    input                                   clk,
    input                                   rst,
    input [5:0]                             stall,
    input [31:0]                            mem_wd,
    input                                   mem_wreg,
    input [31:0]                            mem_wdata,
    output reg[31:0]                        wb_wd,
    output reg                              wb_wreg,
    output reg[31:0]                        wb_wdata
);

    always @ (posedge clk) begin
        if(~rst) begin
            wb_wd <= 32'd0;
            wb_wreg <= 1'd0;
          wb_wdata <= 32'd0;
        end
        else if (stall[4] ==1'd1 && stall[5] == 1'd0)
            begin
                wb_wd <= 32'd0;
            wb_wreg <= 1'd0;
          wb_wdata <= 32'd0;
            end
        else begin

            wb_wd <= mem_wd;
```

```
            wb_wreg <= mem_wreg;
            wb_wdata <= mem_wdata;
        end
    end
endmodule
```

RAM 模块

采用 Vivado 自带 IP 核生成的 RAM。大小为 32bit*256。

采用 Distributed Dual Port RAM。两个读端口，两个写端口。采用异步读，同步写策略。

之所以使用双端口 RAM，是为了在下载验证时，便于通过开关来控制内存的显示。

开关显示模块

该模块负责对下载验证时的开关状态进行处理，并将对应的内存信息显示到数码管上。

> 代码在附录中给出。

TOP 模块

即 CPU 的顶层模块，例化各个子模块，并处理线路连接。

> 代码在附录中给出

实验过程——汇编程序编写

汇编程序编写及导出

根据实现的指令种类，编写了对应的汇编代码来进行验证。通过 Mars 软件导出对应的十六进制文件并处理成 `.coe` 文件，以便用于对 RAM 模块和 ROM 的例化。

> 代码在附录中给出

汇编程序执行

在 Mars 软件中执行编写的汇编程序，得到如下结果：

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x00000000	0x00000000	0x00000003	0x00000002	0x00000001	0x00000001	0x00000010	0x00000003
0x00002020	0x00000000	0xffffffe4	0x00000001	0x00000000	0x00000008	0x00000002	0x00000000	0x00000001
0x00002040	0x00050000	0x00000004	0x00000004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000001
0x00002060	0x00000001	0x00000002	0x00000003	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00002068
$t1	9	0x00000001
$t2	10	0x00000002
$t3	11	0x00000003
$t4	12	0x00000000
$t5	13	0x00000008
$t6	14	0x00002000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00003ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00000194
hi		0x00000000
lo		0x00000000

实验过程——仿真

仿真代码

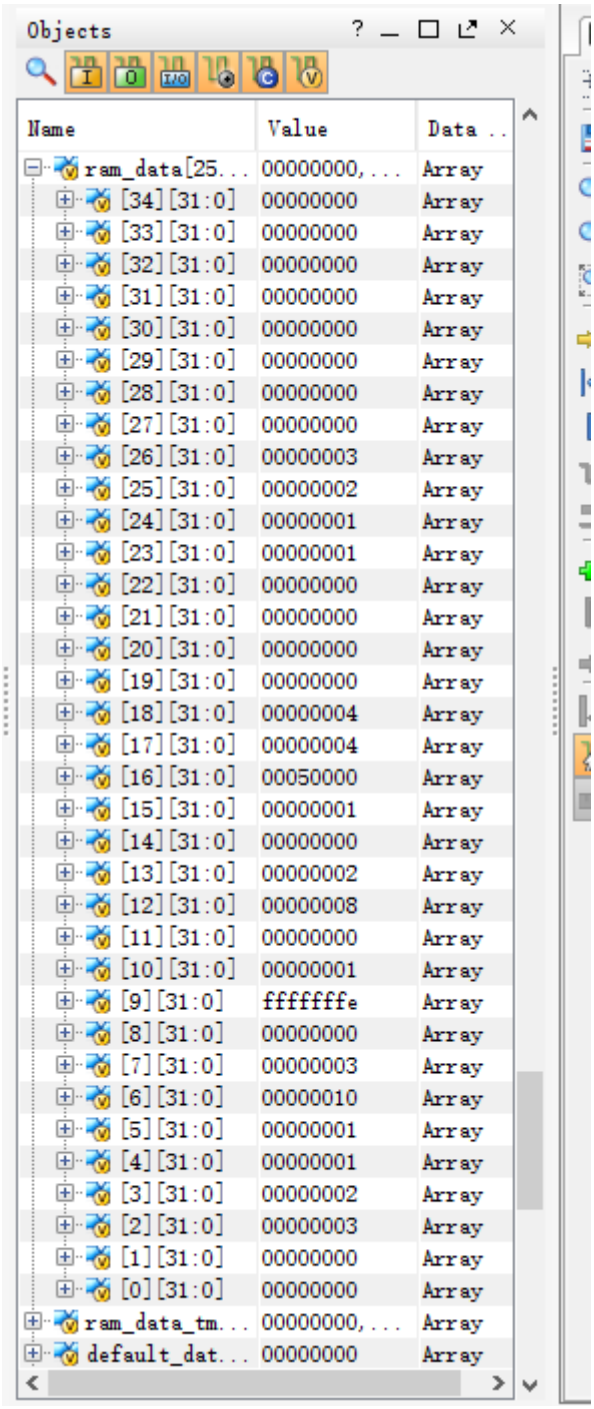
```
module test;

    // Inputs
    reg clk;
    reg rrst;
    reg [7:0] SW;

    // Outputs
    wire [2:0] AN;
    wire [7:0] data;

    // Instantiate the Unit Under Test (UUT)
```

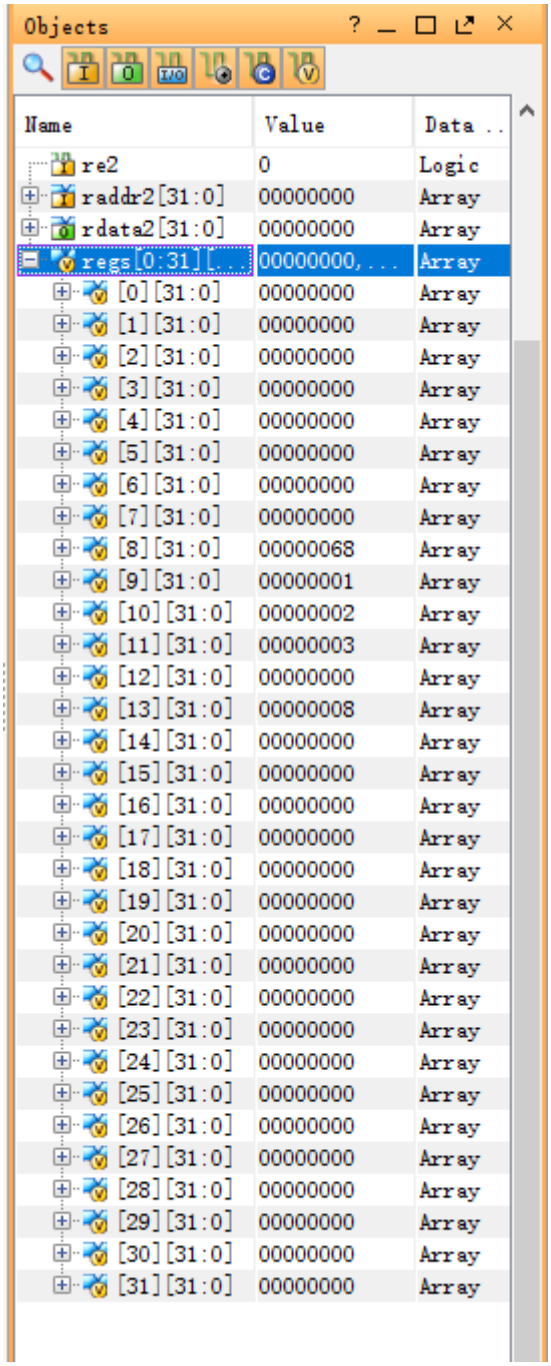
```
    cpu_top uut (
        .CLK50MHZ(clk),
        .BTNL(rrst),
        .SW(SW),
        .AN(AN),
        .SEG(data)
    );
    always
    begin
        #1;
        clk=~clk;
        rrst=1;
        SW<=SW+1;
    end
    initial begin
        // Initialize Inputs
        rrst=0;
        clk=1;
        SW=0;
        #100;
    end
endmodule
```

仿真执行结果

Vivado 仿真结果与 Mars 执行结果一致。

其中 regfile 部分的差异由栈指针以及存储起始位置不同（相差 0x2000）而造成，属于正常现象。


Objects
Name
Value
Data ..
ram_data[25...
00000000,...
Array
[34][31:0]
00000000
Array
[33][31:0]
00000000
Array
[32][31:0]
00000000
Array
[31][31:0]
00000000
Array
[30][31:0]
00000000
Array
[29][31:0]
00000000
Array
[28][31:0]
00000000
Array
[27][31:0]
00000000
Array
[26][31:0]
00000003
Array
[25][31:0]
00000002
Array
[24][31:0]
00000001
Array
[23][31:0]
00000001
Array
[22][31:0]
00000000
Array
[21][31:0]
00000000
Array
[20][31:0]
00000000
Array
[19][31:0]
00000000
Array
[18][31:0]
00000004
Array
[17][31:0]
00000004
Array
[16][31:0]
00050000
Array
[15][31:0]
00000001
Array
[14][31:0]
00000000
Array
[13][31:0]
00000002
Array
[12][31:0]
00000008
Array
[11][31:0]
00000000
Array
[10][31:0]
00000001
Array
[9][31:0]
fffffffe
Array
[8][31:0]
00000000
Array
[7][31:0]
00000003
Array
[6][31:0]
00000010
Array
[5][31:0]
00000001
Array
[4][31:0]
00000001
Array
[3][31:0]
00000002
Array
[2][31:0]
00000003
Array
[1][31:0]
00000000
Array
[0][31:0]
00000000
Array
ram_data_tm...
00000000,...
Array
default_dat...
00000000
Array



实验过程——下载验证

配置好管脚文件后，经过综合、布线的步骤，生成 bitstream 文件，并下载到 FPGA 开发板上，成功实现通过开关控制内存数据的读取，并在数码管上进行显示。在实验检查现场给助教检查无误。由于 FPGA 开发板已收回，此处不再给出结果演示。

核心代码附录

实现的指令

```
ADD
ADDU

ADDI
```

```
XOR
XORI
LW
SW
SUB
SUBU
AND
ANDI
OR
NOR
ORI
LUI
SLL
SLLV
SRL
SRLV
SRA
SRAV
J
JR
BGTZ
BNE
BEQ
BGEZ
BLEZ
BLTZ
NOP
```

define 部分代码

```
`define AND  6'b100100
`define OR   6'b100101
`define XOR 6'b100110
`define NOR 6'b100111
`define ANDI 6'b001100
`define ORI  6'b001101
`define XORI 6'b001110
`define LUI 6'b001111
`define SLL  6'b000000
`define SLLV  6'b000100
`define SRL  6'b000010
`define SRLV  6'b000110
`define SRA  6'b000011
`define SRAV  6'b000111
`define SLT  6'b101010
`define SLTU  6'b101011
`define ADD  6'b100000
`define ADDU  6'b100001
`define SUB  6'b100010

`define SUBU  6'b100011
```

```
`define ADDI  6'b001000
`define ADDIU  6'b001001
`define J  6'b000010
`define JR  6'b001000
`define BEQ  6'b000100
`define BGEZ  5'b00001
`define BGTZ  6'b000111
`define BLEZ  6'b000110
`define BLTZ  5'b00000
`define BNE  6'b000101
`define LW  6'b100011
`define SW  6'b101011
`define NOP 6'b000000


`define SPECIAL_INST 6'b000000
`define REGIMM_INST 6'b000001
`define SPECIAL2_INST 6'b011100


`define AND_OP   8'b00100100
`define OR_OP    8'b00100101
`define XOR_OP  8'b00100110
`define NOR_OP  8'b00100111
`define ANDI_OP  8'b01011001
`define ORI_OP  8'b01011010
`define XORI_OP  8'b01011011
`define LUI_OP  8'b01011100

`define SLL_OP  8'b01111100
`define SLLV_OP  8'b00000100
`define SRL_OP  8'b00000010
`define SRLV_OP  8'b00000110
`define SRA_OP  8'b00000011
`define SRAV_OP  8'b00000111


`define SLT_OP  8'b00101010
`define SLTU_OP  8'b00101011
`define ADD_OP  8'b00100000
`define ADDU_OP  8'b00100001
`define SUB_OP  8'b00100010
`define SUBU_OP  8'b00100011
`define ADDI_OP  8'b01010101
`define ADDIU_OP  8'b01010110



`define J_OP  8'b01001111
`define JR_OP  8'b00001000
`define BEQ_OP  8'b01010001
`define BGEZ_OP  8'b01000001

`define BGTZ_OP  8'b01010100
```

```
`define BLEZ_OP  8'b01010011
`define BLTZ_OP  8'b01000000
`define BNE_OP  8'b01010010

`define LW_OP  8'b11100011
`define SW_OP  8'b11101011


`define NOP_OP    8'b00000000


`define RES_LOGIC 3'b001
`define RES_SHIFT 3'b010
`define RES_MOVE 3'b011
`define RES_ARITHMETIC 3'b100
`define RES_JUMP_BRANCH 3'b110
`define RES_LOAD_STORE 3'b111

`define RES_NOP 3'b000
```

ID 模块代码

```
`include "Define.v"


module id(

    input                           rst,
    input [31:0]                    pc_i,
    input [31:0]                    inst,

    input                           ex_wreg_i,
    input [31:0]                    ex_wdata_i,
    input [31:0]                    ex_wd_i,

    input                           mem_wreg_i,
    input [31:0]                    mem_wdata_i,
    input [31:0]                    mem_wd_i,

    input [31:0]                    reg1_data_i,
    input [31:0]                    reg2_data_i,

    input                           last_branch,
    input [7:0]                     ex_aluop_i,

    output reg                      reg1_read_o,
    output reg                      reg2_read_o,
    output reg[31:0]                reg1_addr_o,
    output reg[31:0]                reg2_addr_o,


    output reg[7:0]                 aluop_o,
```

```
    output reg[2:0]                     alusel_o,
    output reg[31:0]                    reg1_o,
    output reg[31:0]                    reg2_o,
    output reg[31:0]                    wd_o,
    output reg                          wreg_o,
    output reg                          branch,
    output reg [31:0]                   branch_addr,
    output [31:0]                       this_inst,
    output                              stall_id
);

    reg stall_reg1;
    reg stall_reg2;
    wire [31:0] inst_i = (last_branch)? 32'd0: inst;
    wire [5:0] op = inst_i[31:26];
    wire [4:0] op2 = inst_i[10:6];
    wire [5:0] op3 = inst_i[5:0];
    wire [4:0] op4 = inst_i[20:16];
    reg[31:0]   imm;
    reg instvalid;
    wire [31:0] immsll = {{14{inst_i[15]}},inst_i[15:0],2'b00};
    wire [31:0] pc_4 = pc_i + 32'd4;
    assign this_inst = inst;

    always @ (*) begin
        if (~rst) begin
            aluop_o <= 8'd0;
            alusel_o <= 3'd0;
            wd_o <= 32'd0;
            wreg_o <= 1'd0;
            instvalid <= 1'd0;
            reg1_read_o <= 1'b0;
            reg2_read_o <= 1'b0;
            reg1_addr_o <= 32'd0;
            reg2_addr_o <= 32'd0;
            imm <= 32'h0;
            branch_addr <= 32'd0;
            branch <= 1'd0;
      end else begin
            aluop_o <= 8'd0;
            alusel_o <= 3'd0;
            wd_o <= inst_i[15:11];
            wreg_o <= 1'd0;
            instvalid <= 1'b1;
            reg1_read_o <= 1'b0;
            reg2_read_o <= 1'b0;
            reg1_addr_o <= inst_i[25:21];
            reg2_addr_o <= inst_i[20:16];
            imm <= 32'd0;
            branch_addr <= 32'd0;
            branch <= 1'd0;
          case (op)

            `SPECIAL_INST:      begin
```

```
                case (op2)
                    5'b00000:           begin
                        case (op3)
                            `OR:    begin
                                wreg_o <= 1'd1;     aluop_o <= `OR_OP;
                                alusel_o <= `RES_LOGIC;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `AND:   begin
                                wreg_o <= 1'd1;     aluop_o <= `AND_OP;
                                alusel_o <= `RES_LOGIC;   reg1_read_o <= 1'b1;  reg2_read_o <=
1'b1;
                                instvalid <= 1'd0;
                                end
                            `XOR:   begin
                                wreg_o <= 1'd1;     aluop_o <= `XOR_OP;
                                alusel_o <= `RES_LOGIC;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `NOR:   begin
                                wreg_o <= 1'd1;     aluop_o <= `NOR_OP;
                                alusel_o <= `RES_LOGIC;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SLLV: begin
                                wreg_o <= 1'd1;     aluop_o <= `SLL_OP;
                                alusel_o <= `RES_SHIFT;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SRLV: begin
                                wreg_o <= 1'd1;     aluop_o <= `SRL_OP;
                                alusel_o <= `RES_SHIFT;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SRAV: begin
                                wreg_o <= 1'd1;     aluop_o <= `SRA_OP;
                                alusel_o <= `RES_SHIFT;     reg1_read_o <= 1'b1;    reg2_read_o
<= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SLT: begin
                                wreg_o <= 1'd1;     aluop_o <= `SLT_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SLTU: begin

                                wreg_o <= 1'd1;     aluop_o <= `SLTU_OP;
```

```
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `ADD:
                                begin
                                    wreg_o <= 1'd1;
                                    aluop_o <= `ADD_OP;
                                    alusel_o <= `RES_ARITHMETIC;
                                    reg1_read_o <= 1'd1;
                                    reg2_read_o <= 1'd1;
                                    instvalid <= 1'd0;
                                end
                            `ADDU: begin
                                wreg_o <= 1'd1;     aluop_o <= `ADDU_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SLT: begin
                                wreg_o <= 1'd1;     aluop_o <= `SLT_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SLTU: begin
                                wreg_o <= 1'd1;     aluop_o <= `SLTU_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SUB: begin
                                wreg_o <= 1'd1;     aluop_o <= `SUB_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `SUBU: begin
                                wreg_o <= 1'd1;     aluop_o <= `SUBU_OP;
                                alusel_o <= `RES_ARITHMETIC;        reg1_read_o <= 1'b1;
reg2_read_o <= 1'b1;
                                instvalid <= 1'd0;
                                end
                            `JR: begin
                                wreg_o <= 1'd0;
                                aluop_o <= `JR_OP;
                                alusel_o <= `RES_JUMP_BRANCH;
                                reg1_read_o <= 1'b1;
                                reg2_read_o <= 1'b0;
                                branch_addr <= reg1_o;
                            branch <= 1'd1;
                                instvalid <= 1'd0;

                                end
```

```
                default:    begin
                end
              endcase
             end
            default: begin
            end
        endcase
        end
`ORI:           begin
    wreg_o <= 1'd1;     aluop_o <= `OR_OP;
    alusel_o <= `RES_LOGIC; reg1_read_o <= 1'b1;    reg2_read_o <= 1'b0;
    imm <= {16'h0, inst_i[15:0]};       wd_o <= inst_i[20:16];
    instvalid <= 1'd0;
end
`ANDI:          begin
    wreg_o <= 1'd1;     aluop_o <= `AND_OP;
    alusel_o <= `RES_LOGIC; reg1_read_o <= 1'b1;    reg2_read_o <= 1'b0;
    imm <= {16'h0, inst_i[15:0]};       wd_o <= inst_i[20:16];
    instvalid <= 1'd0;
    end
`XORI:          begin
    wreg_o <= 1'd1;     aluop_o <= `XOR_OP;
    alusel_o <= `RES_LOGIC; reg1_read_o <= 1'b1;    reg2_read_o <= 1'b0;
    imm <= {16'h0, inst_i[15:0]};       wd_o <= inst_i[20:16];
    instvalid <= 1'd0;
    end
`LUI:           begin
    wreg_o <= 1'd1;     aluop_o <= `OR_OP;
    alusel_o <= `RES_LOGIC; reg1_read_o <= 1'b1;    reg2_read_o <= 1'b0;
    imm <= {inst_i[15:0], 16'h0};       wd_o <= inst_i[20:16];
    instvalid <= 1'd0;
    instvalid <= 1'd0;
    end
`ADDI:          begin
    wreg_o <= 1'd1;     aluop_o <= `ADDI_OP;
    alusel_o <= `RES_ARITHMETIC; reg1_read_o <= 1'b1;   reg2_read_o <= 1'b0;
        imm <= {{16{inst_i[15]}}, inst_i[15:0]};        wd_o <= inst_i[20:16];
        instvalid <= 1'd0;
    end
`ADDIU:         begin
    wreg_o <= 1'd1;     aluop_o <= `ADDIU_OP;
    alusel_o <= `RES_ARITHMETIC; reg1_read_o <= 1'b1;   reg2_read_o <= 1'b0;
        imm <= {{16{inst_i[15]}}, inst_i[15:0]};        wd_o <= inst_i[20:16];
        instvalid <= 1'd0;
    end
`J:         begin
    wreg_o <= 1'd0;
    aluop_o <= `J_OP;
    alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b0;  reg2_read_o <= 1'b0;
    branch_addr <= {pc_4[31:28], inst_i[25:0], 2'b00};
    branch <= 1'd1;
    instvalid <= 1'd0;

     end
```

```
            `BEQ:          begin
               wreg_o <= 1'd0;     aluop_o <= `BEQ_OP;
               alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b1;
               instvalid <= 1'd0;
               if(reg1_o == reg2_o) begin
                   branch_addr <= pc_4 + immsll;
                   branch <= 1'd1;
               end
               end
            `BGTZ:          begin
               wreg_o <= 1'd0;     aluop_o <= `BGTZ_OP;
               alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b0;
               instvalid <= 1'd0;
               if((reg1_o[31] == 1'b0) && (reg1_o != 32'd0)) begin
                   branch_addr<= pc_4 + immsll;
                   branch <= 1'd1;
               end
               end
            `BLEZ:          begin
               wreg_o <= 1'd0;     aluop_o <= `BLEZ_OP;
               alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b0;
               instvalid <= 1'd0;
               if((reg1_o[31] == 1'b1) || (reg1_o == 32'd0)) begin
                   branch_addr <= pc_4 + immsll;
                   branch <= 1'd1;
               end
               end
            `BNE:           begin
               wreg_o <= 1'd0;     aluop_o <= `BLEZ_OP;
               alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b1;
               instvalid <= 1'd0;
               if(reg1_o != reg2_o) begin
                   branch_addr <= pc_4 + immsll;
                   branch <= 1'd1;
               end
               end
            `LW:            begin
               wreg_o <= 1'd1;     aluop_o <= `LW_OP;
               alusel_o <= `RES_LOAD_STORE; reg1_read_o <= 1'b1;   reg2_read_o <= 1'b0;
                   wd_o <= inst_i[20:16]; instvalid <= 1'd0;
               end
            `SW:            begin
               wreg_o <= 1'd0;     aluop_o <= `SW_OP;
               reg1_read_o <= 1'b1;    reg2_read_o <= 1'b1; instvalid <= 1'd0;
               alusel_o <= `RES_LOAD_STORE;
               end
            `REGIMM_INST:       begin
                   case (op4)
                       `BGEZ:  begin
                           wreg_o <= 1'd0;     aluop_o <= `BGEZ_OP;
                       alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b0;
                       instvalid <= 1'd0;

                       if(reg1_o[31] == 1'b0) begin
```

```
                        branch_addr <= pc_4 + immsll;
                        branch <= 1'd1;
                    end
                    end
                    `BLTZ:      begin
                      wreg_o <= 1'd0;       aluop_o <= `BLTZ_OP;
                    alusel_o <= `RES_JUMP_BRANCH; reg1_read_o <= 1'b1;  reg2_read_o <= 1'b0;
                    instvalid <= 1'd0;
                    if(reg1_o[31] == 1'b1) begin
                        branch_addr <= pc_4 + immsll;
                        branch <= 1'd1;
                    end
                    end
                    default:    begin
                    end
                endcase
            end
          default:            begin
          end
        endcase         //case op

        if (inst_i[31:21] == 11'b00000000000) begin
          if (op3 == `SLL) begin
                wreg_o <= 1'd1;     aluop_o <= `SLL_OP;
                alusel_o <= `RES_SHIFT; reg1_read_o <= 1'b0;    reg2_read_o <= 1'b1;
                imm[4:0] <= inst_i[10:6];       wd_o <= inst_i[15:11];
                instvalid <= 1'd0;
            end else if ( op3 == `SRL ) begin
                wreg_o <= 1'd1;     aluop_o <= `SRL_OP;
                alusel_o <= `RES_SHIFT; reg1_read_o <= 1'b0;    reg2_read_o <= 1'b1;
                imm[4:0] <= inst_i[10:6];       wd_o <= inst_i[15:11];
                instvalid <= 1'd0;
            end else if ( op3 == `SRA ) begin
                wreg_o <= 1'd1;     aluop_o <= `SRA_OP;
                alusel_o <= `RES_SHIFT; reg1_read_o <= 1'b0;    reg2_read_o <= 1'b1;
                imm[4:0] <= inst_i[10:6];       wd_o <= inst_i[15:11];
                instvalid <= 1'd0;
            end
          end

    end
end


always @ (*) begin
stall_reg1 <= 1'd0;
    if(~rst) begin
        reg1_o <= 32'd0;
    end
    else if (ex_aluop_i == `LW_OP && ex_wd_i == reg1_addr_o && reg1_read_o == 1'b1)
        begin
            stall_reg1 <= 1'd1;

        end
```

```
        else if((reg1_read_o == 1'b1) && (ex_wreg_i == 1'b1)
                                && (ex_wd_i == reg1_addr_o)) begin
            reg1_o <= ex_wdata_i;
        end else if((reg1_read_o == 1'b1) && (mem_wreg_i == 1'b1)
                                && (mem_wd_i == reg1_addr_o)) begin
            reg1_o <= mem_wdata_i;
      end else if(reg1_read_o == 1'b1) begin
        reg1_o <= reg1_data_i;
      end else if(reg1_read_o == 1'b0) begin
        reg1_o <= imm;
      end else begin
        reg1_o <= 32'd0;
      end
    end

    always @ (*) begin
    stall_reg2 <= 1'd0;
        if(~rst) begin
            reg2_o <= 32'd0;
        end
        else if (ex_aluop_i == `LW_OP && ex_wd_i == reg2_addr_o && reg2_read_o == 1'b1)
            begin
                stall_reg2 <= 1'd1;
            end
        else if((reg2_read_o == 1'b1) && (ex_wreg_i == 1'b1)
                                && (ex_wd_i == reg2_addr_o)) begin
            reg2_o <= ex_wdata_i;
        end else if((reg2_read_o == 1'b1) && (mem_wreg_i == 1'b1)
                                && (mem_wd_i == reg2_addr_o)) begin
            reg2_o <= mem_wdata_i;
      end else if(reg2_read_o == 1'b1) begin
        reg2_o <= reg2_data_i;
      end else if(reg2_read_o == 1'b0) begin
        reg2_o <= imm;
      end else begin
        reg2_o <= 32'd0;
      end
    end
    assign stall_id = stall_reg1 | stall_reg2;

endmodule
```

ALU 模块代码

```
`include "Define.v"

module ex(

    input                   rst,


    input [7:0]             aluop_i,
```

```
    input [2:0]             alusel_i,
    input [31:0]            reg1_i,
    input [31:0]            reg2_i,
    input [31:0]            wd_i,
    input                   wreg_i,
    input  [31:0]           ex_inst,


    output reg[31:0]        wd_o,
    output reg              wreg_o,
    output reg[31:0]        wdata_o,
    output  [7:0]           aluop_o,
    output  [31:0]          mem_addr_o,
    output  [31:0]          reg2_o
);
    assign reg2_o = reg2_i;
    reg [31:0] logic;
    reg [31:0] shift;
    reg [31:0] math;
    wire [31:0] reg2_i_mux = ((aluop_i == `SUB_OP) || (aluop_i == `SUBU_OP) ||(aluop_i ==
`SLT_OP) ) ? (~reg2_i)+1 : reg2_i;
    wire [31:0] result_sum = reg1_i + reg2_i_mux;
    wire [31:0] ov_sum = ((!reg1_i[31] && !reg2_i_mux[31]) && result_sum[31]) || ((reg1_i[31] &&
reg2_i_mux[31]) && (!result_sum[31]));
    assign  aluop_o = aluop_i;
    assign mem_addr_o = reg1_i + {{16{ex_inst[15]}},ex_inst[15:0]};

    always @ (*) begin
        if(~rst) begin
            logic <= 32'd0;
        end else begin
            case (aluop_i)
                `OR_OP:         begin
                    logic <= reg1_i | reg2_i;
                end
                `AND_OP:        begin
                    logic <= reg1_i & reg2_i;
                end
                `NOR_OP:        begin
                    logic <= ~(reg1_i |reg2_i);
                end
                `XOR_OP:        begin
                    logic <= reg1_i ^ reg2_i;
                end
                default:                begin
                    logic <= 32'd0;
                end
            endcase
        end
    end

    always @ (*) begin

        if(~rst) begin
```

```
            shift <= 32'd0;
        end else begin
            case (aluop_i)
                `SLL_OP:            begin
                    shift <= reg2_i << reg1_i[4:0] ;
                end
                `SRL_OP:        begin
                    shift <= reg2_i >> reg1_i[4:0];
                end
                `SRA_OP:        begin
                    shift <= ({32{reg2_i[31]}} << (6'd32-{1'b0, reg1_i[4:0]})) | reg2_i >>
reg1_i[4:0];
                end
                default:                begin
                    shift <= 32'd0;
                end
            endcase
        end
    end
    always @ (*) begin
        if(~rst) begin
            math <= 32'd0;
        end else begin
            case (aluop_i)
                `ADD_OP, `ADDU_OP, `ADDI_OP, `ADDIU_OP:     begin
                    math <= result_sum;
                end
                `SUB_OP, `SUBU_OP:      begin
                    math <= result_sum;
                end
                default:                begin
                    math <= 32'd0;
                end
            endcase
        end
    end

 always @ (*) begin
     wd_o <= wd_i;
     if(((aluop_i == `ADD_OP) || (aluop_i == `ADDI_OP) ||
          (aluop_i == `SUB_OP)) && (ov_sum == 1'b1)) begin
        wreg_o <= 1'd0;
     end else begin
      wreg_o <= wreg_i;
     end
     case ( alusel_i )
        `RES_LOGIC:     begin
            wdata_o <= logic;
        end
        `RES_SHIFT:     begin
            wdata_o <= shift;
        end

        `RES_ARITHMETIC:    begin
```

```
            wdata_o <= math;
        end
        default:                    begin
            wdata_o <= 32'd0;
        end
     endcase
 end

endmodule
```

开关显示模块代码

```
module seg(
input clk,
input rst,
input [31:0]data,
output reg [7:0]SEG,
output reg [2:0]AN
    );
    reg [15:0]count;
    always @(posedge clk or negedge rst) begin
    if(!rst)
    begin
    count<=0;
    AN<=0;
    end
    else if (count==16'd49999)
    begin
    count<=0;
    if( AN==3'b111)
    AN<=0;
    else
    AN<=AN+1'b1;
    end
    else
    count<=count+1'b1;
    end
    always @(posedge clk) begin
            case(AN)
    3'b000:
           case(data[3:0])
           4'b0000: SEG[7:0]=8'b11000000;
           4'b0001: SEG[7:0]=8'b11111001;
           4'b0010: SEG[7:0]=8'b10100100;
           4'b0011: SEG[7:0]=8'b10110000;
           4'b0100: SEG[7:0]=8'b10011001;
           4'b0101: SEG[7:0]=8'b10010010;
           4'b0110: SEG[7:0]=8'b10000010;
           4'b0111: SEG[7:0]=8'b11011000;
           4'b1000: SEG[7:0]=8'b10000000;
           4'b1001: SEG[7:0]=8'b10010000;
```

```
        4'b1010: SEG[7:0]=8'b10001000;
        4'b1011: SEG[7:0]=8'b10000011;
        4'b1100: SEG[7:0]=8'b11000110;
        4'b1101: SEG[7:0]=8'b10100001;
        4'b1110: SEG[7:0]=8'b10000110;
        4'b1111: SEG[7:0]=8'b10001110;
        default SEG[7:0]=8'b01111111;
        endcase
     3'b001:
            case(data[7:4])
                4'b0000: SEG[7:0]=8'b11000000;
                4'b0001: SEG[7:0]=8'b11111001;
                4'b0010: SEG[7:0]=8'b10100100;
                4'b0011: SEG[7:0]=8'b10110000;
                4'b0100: SEG[7:0]=8'b10011001;
                4'b0101: SEG[7:0]=8'b10010010;
                4'b0110: SEG[7:0]=8'b10000010;
                4'b0111: SEG[7:0]=8'b11011000;
                4'b1000: SEG[7:0]=8'b10000000;
                4'b1001: SEG[7:0]=8'b10010000;
                4'b1010: SEG[7:0]=8'b10001000;
                4'b1011: SEG[7:0]=8'b10000011;
                4'b1100: SEG[7:0]=8'b11000110;
                4'b1101: SEG[7:0]=8'b10100001;
                4'b1110: SEG[7:0]=8'b10000110;
                4'b1111: SEG[7:0]=8'b10001110;
                default SEG[7:0]=8'b01111111;
                endcase
    3'b010:
    case(data[11:8])
               4'b0000: SEG[7:0]=8'b11000000;
               4'b0001: SEG[7:0]=8'b11111001;
               4'b0010: SEG[7:0]=8'b10100100;
               4'b0011: SEG[7:0]=8'b10110000;
               4'b0100: SEG[7:0]=8'b10011001;
               4'b0101: SEG[7:0]=8'b10010010;
               4'b0110: SEG[7:0]=8'b10000010;
               4'b0111: SEG[7:0]=8'b11011000;
               4'b1000: SEG[7:0]=8'b10000000;
               4'b1001: SEG[7:0]=8'b10010000;
               4'b1010: SEG[7:0]=8'b10001000;
               4'b1011: SEG[7:0]=8'b10000011;
               4'b1100: SEG[7:0]=8'b11000110;
               4'b1101: SEG[7:0]=8'b10100001;
               4'b1110: SEG[7:0]=8'b10000110;
               4'b1111: SEG[7:0]=8'b10001110;
               default SEG[7:0]=8'b01111111;
               endcase
    3'b011:
    case(data[15:12])
               4'b0000: SEG[7:0]=8'b11000000;
               4'b0001: SEG[7:0]=8'b11111001;

               4'b0010: SEG[7:0]=8'b10100100;
```

```
            4'b0011: SEG[7:0]=8'b10110000;
            4'b0100: SEG[7:0]=8'b10011001;
            4'b0101: SEG[7:0]=8'b10010010;
            4'b0110: SEG[7:0]=8'b10000010;
            4'b0111: SEG[7:0]=8'b11011000;
            4'b1000: SEG[7:0]=8'b10000000;
            4'b1001: SEG[7:0]=8'b10010000;
            4'b1010: SEG[7:0]=8'b10001000;
            4'b1011: SEG[7:0]=8'b10000011;
            4'b1100: SEG[7:0]=8'b11000110;
            4'b1101: SEG[7:0]=8'b10100001;
            4'b1110: SEG[7:0]=8'b10000110;
            4'b1111: SEG[7:0]=8'b10001110;
            default SEG[7:0]=8'b01111111;
            endcase
            3'b100:
    case(data[19:16])
            4'b0000: SEG[7:0]=8'b11000000;
            4'b0001: SEG[7:0]=8'b11111001;
            4'b0010: SEG[7:0]=8'b10100100;
            4'b0011: SEG[7:0]=8'b10110000;
            4'b0100: SEG[7:0]=8'b10011001;
            4'b0101: SEG[7:0]=8'b10010010;
            4'b0110: SEG[7:0]=8'b10000010;
            4'b0111: SEG[7:0]=8'b11011000;
            4'b1000: SEG[7:0]=8'b10000000;
            4'b1001: SEG[7:0]=8'b10010000;
            4'b1010: SEG[7:0]=8'b10001000;
            4'b1011: SEG[7:0]=8'b10000011;
            4'b1100: SEG[7:0]=8'b11000110;
            4'b1101: SEG[7:0]=8'b10100001;
            4'b1110: SEG[7:0]=8'b10000110;
            4'b1111: SEG[7:0]=8'b10001110;
            default SEG[7:0]=8'b01111111;
            endcase
            3'b101:
    case(data[23:20])
            4'b0000: SEG[7:0]=8'b11000000;
            4'b0001: SEG[7:0]=8'b11111001;
            4'b0010: SEG[7:0]=8'b10100100;
            4'b0011: SEG[7:0]=8'b10110000;
            4'b0100: SEG[7:0]=8'b10011001;
            4'b0101: SEG[7:0]=8'b10010010;
            4'b0110: SEG[7:0]=8'b10000010;
            4'b0111: SEG[7:0]=8'b11011000;
            4'b1000: SEG[7:0]=8'b10000000;
            4'b1001: SEG[7:0]=8'b10010000;
            4'b1010: SEG[7:0]=8'b10001000;
            4'b1011: SEG[7:0]=8'b10000011;
            4'b1100: SEG[7:0]=8'b11000110;
            4'b1101: SEG[7:0]=8'b10100001;
            4'b1110: SEG[7:0]=8'b10000110;

            4'b1111: SEG[7:0]=8'b10001110;
```

```
                default SEG[7:0]=8'b01111111;
                endcase
                3'b110:
    case(data[27:24])
                4'b0000: SEG[7:0]=8'b11000000;
                4'b0001: SEG[7:0]=8'b11111001;
                4'b0010: SEG[7:0]=8'b10100100;
                4'b0011: SEG[7:0]=8'b10110000;
                4'b0100: SEG[7:0]=8'b10011001;
                4'b0101: SEG[7:0]=8'b10010010;
                4'b0110: SEG[7:0]=8'b10000010;
                4'b0111: SEG[7:0]=8'b11011000;
                4'b1000: SEG[7:0]=8'b10000000;
                4'b1001: SEG[7:0]=8'b10010000;
                4'b1010: SEG[7:0]=8'b10001000;
                4'b1011: SEG[7:0]=8'b10000011;
                4'b1100: SEG[7:0]=8'b11000110;
                4'b1101: SEG[7:0]=8'b10100001;
                4'b1110: SEG[7:0]=8'b10000110;
                4'b1111: SEG[7:0]=8'b10001110;
                default SEG[7:0]=8'b01111111;
                endcase
                3'b111:
    case(data[31:28])
                4'b0000: SEG[7:0]=8'b11000000;
                4'b0001: SEG[7:0]=8'b11111001;
                4'b0010: SEG[7:0]=8'b10100100;
                4'b0011: SEG[7:0]=8'b10110000;
                4'b0100: SEG[7:0]=8'b10011001;
                4'b0101: SEG[7:0]=8'b10010010;
                4'b0110: SEG[7:0]=8'b10000010;
                4'b0111: SEG[7:0]=8'b11011000;
                4'b1000: SEG[7:0]=8'b10000000;
                4'b1001: SEG[7:0]=8'b10010000;
                4'b1010: SEG[7:0]=8'b10001000;
                4'b1011: SEG[7:0]=8'b10000011;
                4'b1100: SEG[7:0]=8'b11000110;
                4'b1101: SEG[7:0]=8'b10100001;
                4'b1110: SEG[7:0]=8'b10000110;
                4'b1111: SEG[7:0]=8'b10001110;
                default SEG[7:0]=8'b01111111;
                endcase
     endcase
    end

endmodule
```

TOP 模块代码

```
`include "Define.v"
```

```
module cpu_top(
input CLK50MHZ,
input BTNL,
input [7:0] SW,
output [2:0] AN,
output [7:0] SEG
    );
    wire clk=CLK50MHZ;
    wire rst = BTNL;
    wire [31:0] pc;
    wire [31:0] id_pc_i;
    wire [31:0] id_inst_i;

    wire [7:0] id_aluop_o;
    wire [2:0] id_alusel_o;
    wire [31:0] id_reg1_o;
    wire [31:0] id_reg2_o;
    wire  id_wreg_o;
    wire [31:0] id_wd_o;
    wire [7:0] ex_aluop_i;
    wire [2:0] ex_alusel_i;
    wire [31:0] ex_reg1_i;
    wire [31:0] ex_reg2_i;
    wire  ex_wreg_i;
    wire [31:0] ex_wd_i;

    wire  ex_wreg_o;
    wire [31:0] ex_wd_o;
    wire [31:0] ex_wdata_o;

    wire  mem_wreg_i;
    wire [31:0] mem_wd_i;
    wire [31:0] mem_wdata_i;

    wire  mem_wreg_o;
    wire [31:0] mem_wd_o;
    wire [31:0] mem_wdata_o;

    wire  wb_wreg_i;
    wire [31:0] wb_wd_i;
    wire [31:0] wb_wdata_i;

    wire  reg1_read;
    wire  reg2_read;
    wire [31:0] reg1_data;
    wire [31:0] reg2_data;
    wire [31:0] reg1_addr;
    wire [31:0] reg2_addr;
    wire [31:0] if_inst;

    wire  branch;
    wire  [31:0] branch_addr;

    wire  last_branch;
```

```
wire  [31:0] this_inst;
wire  [31:0] ex_inst;
wire  [31:0] ex_mem_addr_o;
wire  [7:0] ex_aluop_o;
wire  [7:0] ex_aluop;
wire  [31:0] mem_mem_addr;
wire  [7:0] mem_aluop;
wire  [31:0] ex_reg2;
wire  [31:0] mem_reg2;
wire  stall_id;
wire  [5:0] stall;
wire  [31:0] num;
wire  page;


seg u_seg(
    .clk (clk),
    .rst (rst),
    .data (num),
    .SEG (SEG),
    .AN (AN)
    );

Stall u_Stall(
    .rst(rst),
    .stall_id(stall_id),
    .stall(stall)
);

pc u_pc(
    .clk(clk),
    .rst(rst),
    .stall(stall),
    .branch(branch),
    .branch_addr(branch_addr),
    .pc(pc)

);
dist_mem_gen_0 u_InstRom(
    .a(pc[9:2]), // input [7 : 0] a
    .spo(if_inst) // output [31 : 0] spo
);

if_id u_if_id(
    .clk        (clk),
    .rst        (rst),
    .stall  (stall),
    .if_pc  (pc),
    .if_inst    (if_inst),
    .branch (branch),
    .id_pc  (id_pc_i),
    .id_inst    (id_inst_i),

    .last_branch(last_branch)
```

```
);



id u_id(
    .rst(rst),
    .pc_i(id_pc_i),
    .inst(id_inst_i),


    .ex_wreg_i(ex_wreg_o),
    .ex_wdata_i(ex_wdata_o),
    .ex_wd_i(ex_wd_o),

    .mem_wreg_i(mem_wreg_o),
    .mem_wdata_i(mem_wdata_o),
    .mem_wd_i(mem_wd_o),
    .reg1_data_i(reg1_data),
    .reg2_data_i(reg2_data),
    .last_branch(last_branch),
    .ex_aluop_i(ex_aluop_i),

    .reg1_read_o(reg1_read),
    .reg2_read_o(reg2_read),

    .reg1_addr_o(reg1_addr),
    .reg2_addr_o(reg2_addr),

    .aluop_o(id_aluop_o),
    .alusel_o(id_alusel_o),
    .reg1_o(id_reg1_o),
    .reg2_o(id_reg2_o),
    .wd_o(id_wd_o),
    .wreg_o(id_wreg_o),
    .branch(branch),
    .branch_addr(branch_addr),
    .this_inst(this_inst),
    .stall_id(stall_id)
);

regfile u_regfile(
    .clk (clk),
    .rst (rst),
    .we (wb_wreg_i),
    .waddr (wb_wd_i),
    .wdata (wb_wdata_i),
    .re1 (reg1_read),
    .raddr1 (reg1_addr),
    .rdata1 (reg1_data),
    .re2 (reg2_read),
    .raddr2 (reg2_addr),
    .rdata2 (reg2_data)

);
```

```
  id_ex u_id_ex(
      .clk(clk),
      .rst(rst),
      .stall(stall),

      .id_aluop(id_aluop_o),
      .id_alusel(id_alusel_o),
      .id_reg1(id_reg1_o),
      .id_reg2(id_reg2_o),
      .id_wd(id_wd_o),
      .id_wreg(id_wreg_o),
      .id_inst(this_inst),

      .ex_aluop(ex_aluop_i),
      .ex_alusel(ex_alusel_i),
      .ex_reg1(ex_reg1_i),
      .ex_reg2(ex_reg2_i),
      .ex_wd(ex_wd_i),
      .ex_wreg(ex_wreg_i),
      .ex_inst(ex_inst)
  );

  ex u_ex(
      .rst(rst),

      .aluop_i(ex_aluop_i),
      .alusel_i(ex_alusel_i),
      .reg1_i(ex_reg1_i),
      .reg2_i(ex_reg2_i),
      .wd_i(ex_wd_i),
      .wreg_i(ex_wreg_i),
      .ex_inst(ex_inst),

      .wd_o(ex_wd_o),
      .wreg_o(ex_wreg_o),
      .wdata_o(ex_wdata_o),
      .aluop_o(ex_aluop_o),
      .mem_addr_o(ex_mem_addr_o),
      .reg2_o(ex_reg2)
  );

ex_mem u_ex_mem(
      .clk(clk),
      .rst(rst),
      .stall(stall),

      .ex_wd(ex_wd_o),
      .ex_wreg(ex_wreg_o),
      .ex_wdata(ex_wdata_o),
      .ex_aluop(ex_aluop_o),
      .ex_mem_addr(ex_mem_addr_o),

      .ex_reg2(ex_reg2),
```

```
        .mem_wd(mem_wd_i),
        .mem_wreg(mem_wreg_i),
        .mem_wdata(mem_wdata_i),
        .mem_aluop(mem_aluop),
        .mem_mem_addr(mem_mem_addr),
        .mem_reg2(mem_reg2)

    );

    wire  [31:0] mem_read;
    wire  [31:0] mem_addr_o;
    wire  mem_we;
    wire  [31:0] mem_write;
    mem u_mem(
        .rst(rst),
        .wd_i(mem_wd_i),
        .wreg_i(mem_wreg_i),
        .wdata_i(mem_wdata_i),
        .alu_op(mem_aluop),
        .mem_data_i(mem_read),
        .mem_addr_i(mem_mem_addr),
        .mem_reg2(mem_reg2),
        .wd_o(mem_wd_o),
        .wreg_o(mem_wreg_o),
        .wdata_o(mem_wdata_o),
        .mem_addr_o(mem_addr_o),
        .mem_we(mem_we),
        .mem_data_o(mem_write)
    );

    mem_wb u_mem_wb(
        .clk(clk),
        .rst(rst),
        .stall(stall),

        .mem_wd(mem_wd_o),
        .mem_wreg(mem_wreg_o),
        .mem_wdata(mem_wdata_o),

        .wb_wd(wb_wd_i),
        .wb_wreg(wb_wreg_i),
        .wb_wdata(wb_wdata_i)

    );


dist_mem_gen_1 u_RAM (
  .a(mem_addr_o[9:2]), // input [7 : 0] a
  .d(mem_write), // input [31 : 0] d
  .dpra(SW), // input [7 : 0] dpra

  .clk(clk), // input clk
```

```
  .we(mem_we), // input we
  .spo(mem_read), // output [31 : 0] spo
  .dpo(num) // output [31 : 0] dpo
);

endmodule
```

汇编代码

此代码为在 Mars 中运行的代码，实际导出的代码经过了对内存地址的对应修改

> Mars 默认的内存起始地址从 0x00002000 开始，在设计的 CPU 中内存起始地址从 0x0 开始。

```
  .text
      addi $8, $0, 8192
      xor $9, $9, $9
      xor $10, $10, $10
      xor $11, $11, $11
      xor $12, $12, $12
      addi $9, $9, 1
      addi $10, $10, 2
      addi $11, $11, -1
      lw $12, 0($8)
      addi $8, $8, 4
      add $13, $9, $11
      sw $13, 0($8)
      addi $8, $8, 4
      add $13, $9, $10
      sw $13, 0($8)
      addi $8, $8, 4
      sub $13, $9, $11
      sw $13, 0($8)
      addi $8, $8, 4
      subu $13, $10, $9
      sw $13, 0($8)
      addi $8, $8, 4
      and $13, $9, $11
      sw $13, 0($8)
      addi $8, $8, 4
      andi $13, $11, 16
      sw $13, 0($8)
      addi $8, $8, 4
      or $13, $9, $10
      sw $13, 0($8)
      addi $8, $8, 4
      nor $13, $11, $9
      sw $13, 0($8)
      addi $8, $8, 4
      xor $13, $11, $9
      sw $13, 0($8)

      addi $8, $8, 4
```

```
b:    addi $13, $13, 1
      bgtz $13, a
      j b
a:    sw $13, 0($8)
      addi $8, $8, 4
      bne $13, $9, c
      xor $13, $13, $13
c:    sw $13, 0($8)
      addi $8, $8, 4
      addi $14, $0, 200
      xor $13, $13, $13
      jr $14
      addi $13, $13, 16
      addi $13, $13, 8
      sw $13, 0($8)
      addi $8, $8, 4
      addi $14, $0,8192
      lw $9, 0($14)
      add $9, $10, $11
      add $11, $9, $9
      sw $11, 0($8)
       xori $11,$9,1
      addi $8,$8,4
      sw $11,0($8)
      ori $11,$9,1
      addi $8,$8,4
      sw $11,0($8)
      lui $11,5
      addi $8,$8,4
      sw $11,0($8)
      sll $11,$9,2
      addi $8,$8,4
      sw $11,0($8)
      sllv $11,$9,$10
      addi $8,$8,4
      sw $11,0($8)
      srl $11,$9,1
      addi $8,$8,4
      sw $11,0($8)
      srlv $11,$9,$10
      addi $8,$8,4
      sw $11,0($8)
      sra $11,$9,2
      addi $8,$8,4
      sw $11,0($8)
      srav $11,$9,$10
      addi $8,$8,4
      sw $11,0($8)
      beq $11,$9,d
      addi $11,$11,1
d:    addi $8,$8,4
      sw $11,0($8)

      bgez $11,e
```

```
      addi $11,$11,1
e:    addi $8,$8,4
      sw $11,0($8)
      blez $11,f
      addi $11,$11,1
f:     addi $8,$8,4
      sw $11,0($8)
      bltz $11,g
      addi $11,$11,1
g:    addi $8,$8,4
      sw $11,0($8)
out:  j out
```

.coe文件代码

InstRom 例化文件

```
memory_initialization_radix = 16;
memory_initialization_vector =
20080000,
01294826,
014a5026,
016b5826,
018c6026,
21290001,
214a0002,
216bffff,
8d0c0000,
21080004,
012b6820,
ad0d0000,
21080004,
012a6820,
ad0d0000,
21080004,
012b6822,
ad0d0000,
21080004,
01496823,
ad0d0000,
21080004,
012b6824,
ad0d0000,
21080004,
316d0010,
ad0d0000,
21080004,
012a6825,
ad0d0000,
21080004,

01696827,
```

```
ad0d0000,
21080004,
01696826,
ad0d0000,
21080004,
21ad0001,
1da00001,
08000025,
ad0d0000,
21080004,
15a90001,
01ad6826,
ad0d0000,
21080004,
200e00c8,
01ad6826,
01c00008,
21ad0010,
21ad0008,
ad0d0000,
21080004,
200e0000,
8dc90000,
014b4820,
01295820,
ad0b0000,
392b0001,
21080004,
ad0b0000,
352b0001,
21080004,
ad0b0000,
3c0b0005,
21080004,
ad0b0000,
00095880,
21080004,
ad0b0000,
01495804,
21080004,
ad0b0000,
00095842,
21080004,
ad0b0000,
01495806,
21080004,
ad0b0000,
00095883,
21080004,
ad0b0000,
01495807,
21080004,

ad0b0000,
```

```
11690001,
216b0001,
21080004,
ad0b0000,
05610001,
216b0001,
21080004,
ad0b0000,
19600001,
216b0001,
21080004,
ad0b0000,
05600001,
216b0001,
21080004,
ad0b0000,
08000065;
```

RAM 例化文件

```
memory_initialization_radix = 16;
memory_initialization_vector =
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0;
```