同濟大學
TONGJI UNIVERSITY

计算机系统结构课程实验
总结报告

实验题目：动态流水线设计与性能定量分析

学号：2152118

姓名：史君宝

指导教师：秦国峰

日期：2023.12.24

一、 实验环境部署与硬件配置说明

操作系统： win10
软件使用：Vivado 和 MARS
开发板使用：FPGA
指令集：MIPS

二、 实验的总体结构

实验要求完成至少 31 条 MIPS 指令的动态流水线 CPU 设计，并支持中断。在 CPU 运行验证程序的过程中，由按键或拨动开关产生一个暂停的中断，再次按键或拨动开关结束中断，继续运行后续的运算，并在数码管上动态显示运算值。

1、动态流水线的总体结构

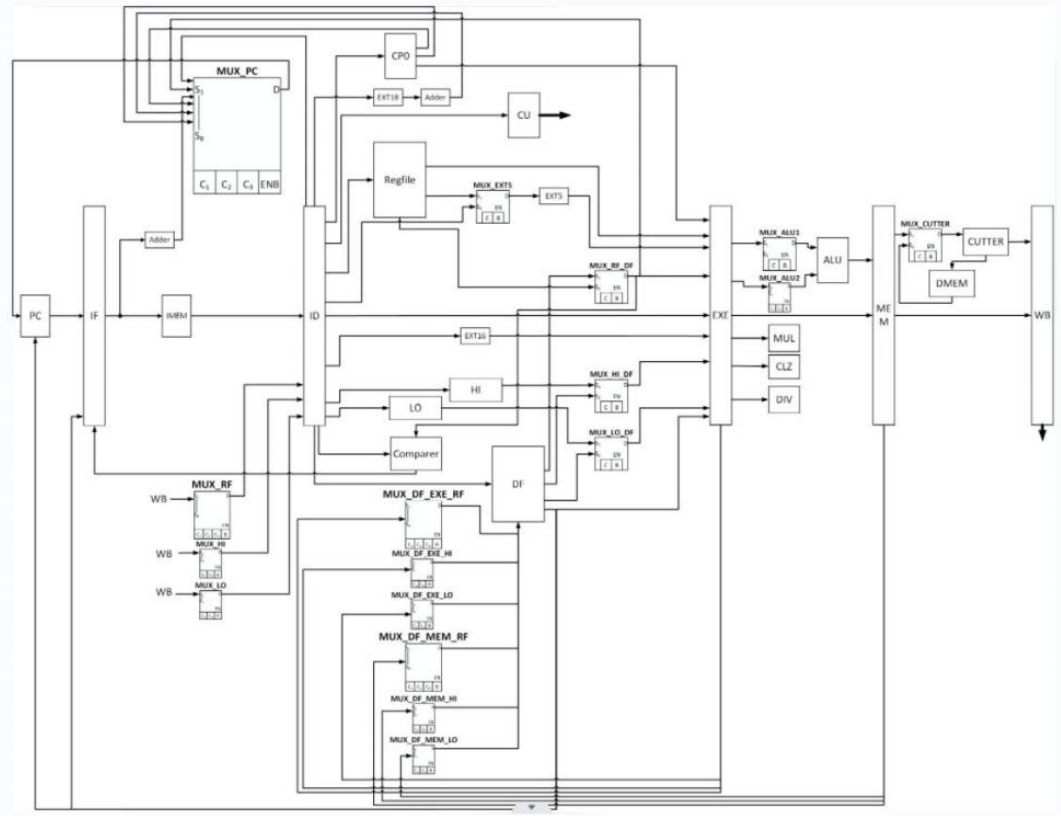
类似于之前设计的静态流水线结构，在我们的动态流水线的设计中，整个结构设计也分为 5 个部分：ID，IF，EXE，MEM，WB。每一级结构都会将上一级结构的结果运用并进一步运算。

首先是指令获取 IF 结构，它能够从指令存储器中获取下一条指令，并将指令传递给下一个阶段。然后是指令译码 ID 结构，它会对指令进行译码，从而确定指令的类型和操作数。并将译码后的指令和操作数传递给下一个阶段。之后是指令执行的 EXE 结构，它执行具体的指令。之后是访存的 MEM 结构，如果指令需要访问内存，在这个阶段可以进行内存读取或写入操作。最后是写回阶段 WB 结构，它会将执行阶段的结果写回寄存器文件。

上述的这些阶段按照特定的顺序连接在一起，就形成一个流水线的 CPU，每个阶段会有专门的功能单元执行相关的操作。这样设计 CPU 可以增强其并行性，因为多个指令可以同时执行，不同指令会占据不同的功能单元，即处在不同的执行阶段。这样提高了 CPU 的并行效率。

2、动态流水线的指令结构

序号	31条	ucosii V2.52	指令	指令说明	指令格式	OP 31-26	RS 25-21	RT 20-16	RD 15-11	SA 10-6	FUNCT 5-0	指令码 16进制
1	√	√	addi	加立即数	addi rt, rs, immediate	001000				00000	100000	20000000
2	√	√	addiu	加立即数（无符号）	addiu rd, rs, immediate	001001						24000000
3	√	√	andi	立即数与	andi rt, rs, immediate	001100						30000000
4	√	√	ori	或立即数	ori rt, rs, immediate	001101						34000000
5	√	√	sltiu	小于立即数置1（无符号）	sltiu rt, rs, immediate	001011						2C000000
6	√	√	lui	立即数加载高位	lui rt, immediate	001111	00000					3C000000
7	√		xori	异或（立即数）	xori rt, rs, immediate	001110			00000	00000	000000	38000000
8	√		slti	小于置1（立即数）	slti rt, rs, immediate	001010			00000	00000	000000	28000000
9	√	√	addu	加（无符号）	addu rd, rs, rt	000000				00000	100001	00000021
10	√	√	and	与	and rd, rs, rt	000000				00000	100100	00000024
11	√	√	beq	相等时分支	beq rs, rt, offset	000100						10000000
12	√	√	bne	不等时分支	bne rs, rt, offset	000101						14000000
13	√	√	j	跳转	j target	000010						08000000
14	√	√	jal	跳转并链接	jal target	000011						0C000000
15	√	√	jr	跳转至寄存器所指地址	jr rs	000000					001000	00000009
16	√	√	lw	取字	lw rt, offset(base)	100011						8C000000
17	√	√	xor	异或	xor rd, rs, rt	000000				00000	100110	00000026
18	√	√	nor	或非	nor rd, rs, rt	000000				00000	100111	00000027
19	√	√	or	或	or rd, rs, rt	000000				00000	100101	00000025
20	√	√	sll	逻辑左移	sll rd, rt, sa	000000	00000				000000	00000000
21	√	√	sllv	逻辑左移（位数可变）	sllv rd, rt, rs	000000				00000	000100	00000004
22	√	√	sltu	小于置1（无符号）	sltu rd, rs, rt	000000				00000	101011	0000002B
23	√	√	sra	算数右移	sra rd, rt, sa	000000	00000				000011	00000003
24	√	√	srl	逻辑右移	srl rd, rt, sa	000000	00000				000010	00000002
25	√	√	subu	减（无符号）	sub rd, rs, rt	000000				00000	100010	00000022
26	√	√	sw	存字	sw rt, offset(base)	101011						AC000000
27	√		add	加	add rd, rs, rt	000000				00000	100000	00000020
28	√		sub	减	sub rd, rs, rt	000000				00000	100010	00000022
29	√		slt	小于置1	slt rd, rs, rt	000000				00000	101010	0000002A
30	√		srlv	逻辑右移（位数可变）	srlv rd, rt, rs	000000				00000	000110	00000006
31	√		srav	算数右移（位数可变）	srav rd, rt, rs	000000				00000	000111	00000007
32		√	clz	前导零计数	clz rd, rs	011100				00000	100000	70000020
33		√	divu	除（无符号）	divu rs, rt	000000			00000	00000	011011	0000001B
34		√	eret	异常返回	eret	010000	10000	00000	00000	00000	011000	42000018
35		√	jalr	跳转至寄存器所指地址，返回地址保存在	jalr rs	000000		00000			001001	00000008
36		√	lb	取字节	lb rt, offset(base)	100000						80000000
37		√	lbu	取字节（无符号）	lbu rt, offset(base)	100100						90000000
38		√	lhu	取半字（无符号）	lhu rt, offset(base)	100101						94000000
39		√	sb	存字节	sb rt, offset(base)	101000						A0000000
40		√	sh	存半字	sh rt, offset(base)	101001						A4000000
41			lh	取半字	lh rt, offset(base)	100001						84000000
42		√	mfc0	读 CP0 寄存器	mfc0 rt, rd	010000	00000			00000	000000	40000000
43		√	mfhi	读 Hi 寄存器	mfhi rd	000000	00000	00000		00000	010000	00000010
44		√	mflo	读 Lo 寄存器	mflo rd	000000	00000	00000		00000	010010	00000012
45		√	mtc0	写 CP0 寄存器	mtc0 rt, rd	010000	00100			00000	000000	40800000
46		√	mthi	写 Hi 寄存器	mthi rd	000000		00000	00000	00000	010001	00000011
47		√	mtlo	写 Lo 寄存器	mtlo rd	000000		00000	00000	00000	010011	00000013
48		√	mul	乘	mul rd, rs, rt	011100				00000	000010	70000002
49		√	multu	乘（无符号）	multu rs, rt	000000			00000	00000	011001	00000019
50		√	syscall	系统调用	syscall	000000					001100	0000000C
51		√	teq	相等异常	teq rs, rt	000000					110100	00000034
52		√	bgez	大于等于0时分支	bgez rs, offset	000001		0001				04010000
53			break	断点	break	000000					001101	0000000D
54			div	除	div rs, rt	000000			00000	00000	011010	0000001A



三、 总体架构部件的解释说明

1、动态流水线总体结构部件的解释说明

```
//##############################################################################
//# IF 部分
//##############################################################################
    wire jump;          // 跳转信号
    wire stall;         // 延迟信号
    wire [31:0] npc_IF; // 始终是npc=pc+4
    wire [31:0] pc_ID;  // ID段返回的pc值
    wire [31:0] pc_IF;  // IF段流出的pc
    wire [31:0] inst_IF;// IF段流出的inst
    assign npc_IF   = pc + 32'd4;
    assign pc_IF    = (stall) ? pc : (jump) ? pc_ID : npc_IF;
    assign inst_IF  = (stall||jump)? 32'b0 : inst;
    PC PC(clk,rst,pc_IF,pc);

//##############################################################################
//# Pipe_IF_ID 流水线寄存器
//##############################################################################
    wire [31:0] inst_ID;
    wire [31:0] npc_ID;

    Pipe_IF_ID Pipe_IF_ID(
        .clk(clk),
        .rst(rst),
        .npc_IF(npc_IF),
        .inst_IF(inst_IF),
        .npc_ID(npc_ID),
        .inst_ID(inst_ID)
    );
```

上面是关于 IF 模块的调用代码，主要作用就是从 COE 的文件中具体的读取相关的指令，送到下一阶段的指令译码。
具体的代码：

```
module Pipe_IF_ID(
    input clk,
    input rst,
    input [31:0] npc_IF,
    input [31:0] inst_IF,
    output reg [31:0] npc_ID = 32'b0,
    output reg [31:0] inst_ID = 32'b0
    );

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            inst_ID <= 32'b0;   npc_ID <= 32'b0;
        end
        else begin
            inst_ID <= inst_IF; npc_ID <= npc_IF;
        end
    end
endmodule
```

然后是 ID 部分：

```
//######################################################################
//# ID 部分
//######################################################################
 //-inst_ID指令解码
   wire [5:0] op,func;
   wire [4:0] rs,rt,rd,sa;
   wire [15:0] imm16;
   wire [25:0] index;
   wire [31:0] sa32_ID;       // sa 拓展
   wire [31:0] uimm32_ID;     // imme(offset) 无符号拓展
   wire [31:0] simm32_ID;     // imme(offset) 有符号拓展
   wire [31:0] offset32;   // offset << 2 拓展
   assign func     = inst_ID[5:0];
   assign sa       = inst_ID[10:6];
   assign imm16    = inst_ID[15:0];
   assign index    = inst_ID[25:0];
   assign op       = inst_ID[31:26];
   assign rs       = inst_ID[25:21];
   assign rt       = inst_ID[20:16];
   assign rd       = inst_ID[15:11];
   assign sa32_ID     = {27'b0, sa};
   assign uimm32_ID   = {16'b0, imm16};
   assign simm32_ID   = {{16{imm16[15]}}, imm16};
   assign offset32 = {{14{imm16[15]}}, imm16, 2'b0};
```

```
//-pc_ID选择
   wire [31:0] rs_data;    // 寄存器rs中的数据
   wire [31:0] rt_data;    // 寄存器rt中的数据
   wire [1:0] mux_pc;      // pc_ID 选择信号
   wire [31:0] pc_add;     // beq bne bgez: pc + offset32
   wire [31:0] pc_jjal;    // j jal: pc[31:28]和index
   wire [31:0] pc_rs;      // jr: reg中rs位置存储的数据
   assign pc_rs  = rs_data;
   assign pc_add  = pc + offset32;
   assign pc_jjal  = {npc_IF[31:28], index, 2'b0};
   assign pc_ID = (mux_pc[1]) ? pc_add : (mux_pc[0]) ? pc_rs : pc_jjal;
```

```
//-控制信号产生
   wire DM_w_ID;           // ID段DMEM写信号
   wire write_ID;          // ID段写信号
   wire [3:0] aluc_ID;     // ID段aluc
   wire [4:0] waddr_ID;    // ID段写地址
   wire mux_alua_ID;           // ID段alua来源选择信号
   wire [1:0] mux_alub_ID;     // ID段alub来源选择信号
   wire [1:0] mux_waddr_ID;    // ID段写地址选择信号
   wire [1:0] mux_wdata_ID;    // ID段写数据选择信号
   assign waddr_ID = (mux_waddr_ID[1]) ? 5'd31 : (mux_waddr_ID[0]) ? rd : rt;
   control control(
       .op(op),
       .func(func),
       .rs_data(rs_data),
       .rt_data(rt_data),
       .jump(jump),
       .DM_w_ID(DM_w_ID),
       .write_ID(write_ID),
       .aluc_ID(aluc_ID),
       .mux_pc(mux_pc),
       .mux_alua_ID(mux_alua_ID),
       .mux_alub_ID(mux_alub_ID),
       .mux_waddr_ID(mux_waddr_ID),
       .mux_wdata_ID(mux_wdata_ID)
   );
```

```
//-冲突处理
   wire write_EXE;         // EXE段写信号
   wire write_MEM;         // MEM段写信号
   wire [4:0] waddr_EXE;   // EXE段写地址
   wire [4:0] waddr_MEM;   // MEM段写地址
   conflict conflict(
       .jump(jump),
       .inst(inst),
       .write_ID(write_ID),
       .write_EXE(write_EXE),
       .write_MEM(write_MEM),
       .waddr_ID(waddr_ID),
       .waddr_EXE(waddr_EXE),
       .waddr_MEM(waddr_MEM),
       .stall(stall)
   );
```

我们需要按照之前看到的指令码的结构对读取到的指令进行译码，主要就是根据指令的种类取相应部分的位来作为信息。之后我们将所获得的信息送到 control 模块，让其产生控制信号。

具体的代码：

```
module control(
    input [5:0] op,
    input [5:0] func,
    input [31:0] rs_data,
    input [31:0] rt_data,

    output jump,
    output DM_w_ID,
    output write_ID,
    output [3:0] aluc_ID,

    output [1:0] mux_pc,
    output mux_alua_ID,
    output [1:0] mux_alub_ID,
    output [1:0] mux_waddr_ID,
    output [1:0] mux_wdata_ID
    );
```

```
    wire r_type = ~(op[5]|op[4]|op[3]|op[2]|op[1]|op[0]);
    wire ADD = r_type&func[5]&~func[4]&~func[3]&~func[2]&~func[1]&~func[0];
    wire ADDU = r_type&func[5]&~func[4]&~func[3]&~func[2]&~func[1]&func[0];
    wire SUB = r_type&func[5]&~func[4]&~func[3]&~func[2]&func[1]&~func[0];
    wire SUBU = r_type&func[5]&~func[4]&~func[3]&~func[2]&func[1]&func[0];
    wire AND = r_type&func[5]&~func[4]&~func[3]&func[2]&~func[1]&~func[0];
    wire OR = r_type&func[5]&~func[4]&~func[3]&func[2]&~func[1]&func[0];
    wire XOR = r_type&func[5]&~func[4]&~func[3]&func[2]&func[1]&~func[0];
    wire NOR = r_type&func[5]&~func[4]&~func[3]&func[2]&func[1]&func[0];
    wire SLT = r_type&func[5]&~func[4]&func[3]&~func[2]&func[1]&~func[0];
    wire SLTU = r_type&func[5]&~func[4]&func[3]&~func[2]&func[1]&func[0];
    wire SLL = r_type&~func[5]&~func[4]&~func[3]&~func[2]&~func[1]&~func[0];
    wire SRL = r_type&~func[5]&~func[4]&~func[3]&~func[2]&func[1]&~func[0];
    wire SRA = r_type&~func[5]&~func[4]&~func[3]&~func[2]&func[1]&func[0];
    wire SLLV = r_type&~func[5]&~func[4]&~func[3]&func[2]&~func[1]&~func[0];
    wire SRLV = r_type&~func[5]&~func[4]&~func[3]&func[2]&func[1]&~func[0];
    wire SRAV = r_type&~func[5]&~func[4]&~func[3]&func[2]&func[1]&func[0];
    wire JR = r_type&~func[5]&~func[4]&func[3]&~func[2]&~func[1]&~func[0];
    wire ADDI = ~op[5]&~op[4]&op[3]&~op[2]&~op[1]&~op[0];
    wire ADDIU = ~op[5]&~op[4]&op[3]&~op[2]&~op[1]&op[0];
    wire ANDI = ~op[5]&~op[4]&op[3]&op[2]&~op[1]&~op[0];
    wire ORI = ~op[5]&~op[4]&op[3]&op[2]&~op[1]&op[0];
    wire XORI = ~op[5]&~op[4]&op[3]&op[2]&op[1]&~op[0];
    wire LUI = ~op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];
    wire LW = op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];
    wire SW = op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];
    wire BEQ = ~op[5]&~op[4]&~op[3]&op[2]&~op[1]&~op[0];
    wire BNE = ~op[5]&~op[4]&~op[3]&op[2]&~op[1]&op[0];
    wire SLTI = ~op[5]&~op[4]&op[3]&~op[2]&op[1]&~op[0];
    wire SLTIU = ~op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];
    wire J = ~op[5]&~op[4]&~op[3]&~op[2]&op[1]&~op[0];
    wire JAL = ~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];
```

```
    assign DM_w_ID = SW;
    assign write_ID = ~(JR|SW|BEQ|BNE|J);
    assign jump = JR|J|JAL|(BEQ&(rs_data == rt_data))|(BNE&(rs_data != rt_data));

    assign aluc_ID[3] = LUI|SLL|SLLV|SLT|SLTI|SLTIU|SLTU|SRA|SRAV|SRL|SRLV;
    assign aluc_ID[2] = AND|ANDI|NOR|OR|ORI|SLL|SLLV|SRA|SRAV|SRL|SRLV|XOR|XORI;
    assign aluc_ID[1] = ADD|ADDI|BEQ|BNE|LW|NOR|SLL|SLLV|SLT|SLTI|SLTIU|SLTU|SUB|SW|XOR|XORI;
    assign aluc_ID[0] = BEQ|BNE|NOR|OR|ORI|SLT|SLTI|SRL|SRLV|SUB|SUBU;

    assign mux_pc = (J|JAL)?2'b00:(JR)?2'b01:(BNE|BEQ)?2'b11:2'bxx;
    assign mux_alua_ID = SLL|SRA|SRL;
    assign mux_alub_ID[1] = ~(ADDI|ADDIU|LUI|LW|SLTI|SW|ANDI|ORI|SLTIU|XORI);
    assign mux_alub_ID[0] = ANDI|ORI|SLTIU|XORI;
    assign mux_wdata_ID[1] = JAL;
    assign mux_wdata_ID[0] = LW;
    assign mux_waddr_ID[1] = JAL;
    assign mux_waddr_ID[0] = ~(ADDI|ADDIU|ANDI|LUI|LW|ORI|SLTI|SLTIU|XORI|JAL);
endmodule
```

完成了上述的指令译码阶段之后，我们需要具体的执行指令，也就是 EXE 模块：

```
//##############################################################################
//# EXE 部分
//##############################################################################
    wire [31:0] alua;
    wire [31:0] alub;
    wire [31:0] alu_EXE;
    assign alua = mux_alua_EXE ? sa32_EXE : rs_data_EXE;
    assign alub = (mux_alub_EXE[1]) ? rt_data_EXE : (mux_alub_EXE[0]) ? uimm32_EXE : simm32_EXE;
    alu alu(alua,alub,aluc_EXE,alu_EXE);
```

```
Pipe_ID_EXE Pipe_ID_EXE(
    .clk(clk),
    .rst(rst),

    .DM_w_ID(DM_w_ID),
    .write_ID(write_ID),
    .mux_alua_ID(mux_alua_ID),
    .mux_alub_ID(mux_alub_ID),
    .mux_wdata_ID(mux_wdata_ID),
    .aluc_ID(aluc_ID),
    .npc_ID(npc_ID),
    .waddr_ID(waddr_ID),
    .sa32_ID(sa32_ID),
    .simm32_ID(simm32_ID),
    .uimm32_ID(uimm32_ID),
    .rs_data_ID(rs_data),
    .rt_data_ID(rt_data),
    .DM_wdata_ID(rt_data),

    .DM_w_EXE(DM_w_EXE),
    .write_EXE(write_EXE),
    .mux_wdata_EXE(mux_wdata_EXE),
    .mux_alua_EXE(mux_alua_EXE),
    .mux_alub_EXE(mux_alub_EXE),
    .aluc_EXE(aluc_EXE),
    .npc_EXE(npc_EXE),
    .waddr_EXE(waddr_EXE),
    .sa32_EXE(sa32_EXE),
    .simm32_EXE(simm32_EXE),
    .uimm32_EXE(uimm32_EXE),
    .rs_data_EXE(rs_data_EXE),
    .rt_data_EXE(rt_data_EXE),
    .DM_wdata_EXE(DM_wdata_EXE)
);
```

```
//##############################################################################
//# Pipe_EXE_MEM 流水线寄存器
//##############################################################################
    wire DM_w_MEM;
    wire [1:0] mux_wdata_MEM;
    wire [31:0] alu_MEM;
    wire [31:0] npc_MEM;
    wire [31:0] DM_wdata_MEM;

    Pipe_EXE_MEM Pipe_EXE_MEM(
        .clk(clk),
        .rst(rst),

        .DM_w_EXE(DM_w_EXE),
        .write_EXE(write_EXE),
        .waddr_EXE(waddr_EXE),
        .mux_wdata_EXE(mux_wdata_EXE),
        .npc_EXE(npc_EXE),
        .alu_EXE(alu_EXE),
        .DM_wdata_EXE(DM_wdata_EXE),

        .DM_w_MEM(DM_w_MEM),
        .write_MEM(write_MEM),
        .waddr_MEM(waddr_MEM),
        .mux_wdata_MEM(mux_wdata_MEM),
        .npc_MEM(npc_MEM),
        .alu_MEM(alu_MEM),
        .DM_wdata_MEM(DM_wdata_MEM)
    );
```

在这一阶段中我们需要根据之前译码所获得的控制信号，执行具体的操作，主要就是利用其中 ALU 模块，执行一个相加，并根据 ALU 的结果结合具体的指令进行执行就可以了。

之后我们需要的就是 MEM 的访存阶段了：

```
//##############################################################################
//# MEM 部分
//##############################################################################
    wire DM_w;
    wire [31:0] DM_addr;
    wire [31:0] DM_rdata;
    wire [31:0] DM_wdata;
    assign DM_addr = alu_MEM;
    assign DM_wdata = DM_wdata_MEM;
    assign DM_w = DM_w_MEM;
    dmem dmem(~clk,rst,DM_w,DM_addr,DM_wdata,DM_rdata);
```

```
//##############################################################################
//# Pipe_MEM_WB 流水线寄存器
//##############################################################################
    wire write_WB;
    wire [4:0] waddr_WB;
    wire [31:0] alu_WB;
    wire [31:0] DM_rdata_WB;
    wire [1:0] mux_wdata_WB;
    wire [31:0] npc_WB;

    Pipe_MEM_WB Pipe_MEM_WB(
        .clk(clk),
        .rst(rst),

        .write_MEM(write_MEM),
        .waddr_MEM(waddr_MEM),
        .mux_wdata_MEM(mux_wdata_MEM),
        .alu_MEM(alu_MEM),
        .npc_MEM(npc_MEM),
        .DM_rdata_MEM(DM_rdata),

        .write_WB(write_WB),
        .waddr_WB(waddr_WB),
        .mux_wdata_WB(mux_wdata_WB),
        .alu_WB(alu_WB),
        .npc_WB(npc_WB),
        .DM_rdata_WB(DM_rdata_WB)
    );
```

我们根据之前的控制信号，进行数据的读取和存入操作。

最后就是写回阶段：

```
//##############################################################################
//# WB 部分
//##############################################################################
    wire [31:0] wdata_WB;
    assign wdata_WB = (mux_wdata_WB[1]) ? npc_WB : (mux_wdata_WB[0]) ? DM_rdata_WB : alu_WB;
    regfile regfile(clk,rst,write_WB,rs,rt,waddr_WB,wdata_WB,rs_data,rt_data);

endmodule
```

2、静态流水线结构部件的运行总结

与上学期所做的多周期 CPU 不同的是，流水线 CPU 的主要设计思路就是将原本的指令执行过程进行一个细分，原来在单周期或多周期的设计中并不怎么注重这个，但是在流水线中我们将一个指令的执行过程分解为若干个阶段，每个阶段由不同的组件负责，完成相应的工作，并将结果传给下一组件。我们让不同的指令同时在不同的阶段执行，充分利用 CPU，提高指令的并行性。而五个阶段的流水线，就是取指（IF）、译码（ID）、执行（EXE）、访存（MEM）和写回（WB）。

取指（IF）任务：在这个阶段，处理器从存储器中读取当前要执行的指令，取到相应的指令之后，会进行程序计数器（PC）的更新，具体用到 NPC 等等，其中 PC 储存的就是下一条要执行指令的地址。

译码（ID）任务：在这个阶段，处理器会对取得的指令进行译码，确定指令的类型和操作数，并准备相应的操作数。根据指令的信息我们会产生不同的控制信号，用来帮助后续的进程。

执行（EXE）任务：在这个阶段，处理器会根据指令产生的控制信号使用算术逻辑单元 ALU 进行运算，并产生运算结果。

访存（MEM）任务：在这个阶段，某些指令有着特殊的访存要求，我们在设计时，要让这个模块能够实现对存储器的访问，实现相应的功能。

写回（WB）任务：在这个阶段，处理器需要进行寄存器写入操作。

四、 实验仿真过程

1、动态流水线的仿真过程

主要的仿真分为前仿真和后仿真，前者是功能仿真，后者是时序仿真，我们导入具体的 testbench 文件，就可以对其进行仿真：

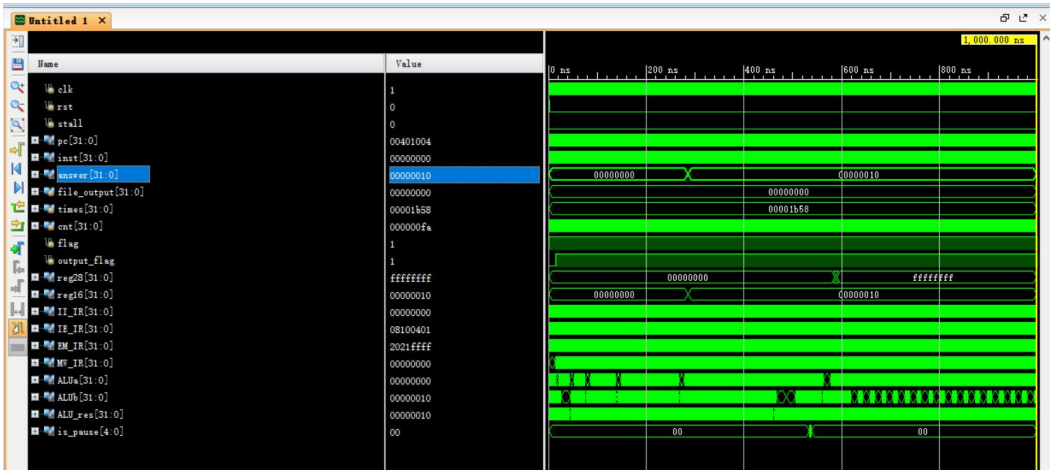
```
module pcpu_top_tb;
    reg clk_in;
    wire clk_real;
    reg reset;
    wire [31:0]inst;
    wire [31:0]pc;
    integer file_output;

    initial  begin
        file_output = $fopen("C:/Users/14065/Desktop/result.txt");
        clk_in = 0;
        reset = 1; #0.5 reset = 0;
    end
    pcpu_top uun(.clk(clk_real),.rst(reset),.inst(inst),.pc(pc));

    always begin
        #0.5 clk_in <= ~clk_in;
    end
    assign clk_real = (inst != 32'b0) ? clk_in : 1'b0;

    always @(posedge clk_real) begin
        $fdisplay(file_output,"pc: %h",pc);
        $fdisplay(file_output,"instr: %h",inst);
        $fdisplay(file_output,"regfiles0: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[0]);
        $fdisplay(file_output,"regfiles1: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[1]);
        $fdisplay(file_output,"regfiles2: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[2]);
        $fdisplay(file_output,"regfiles3: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[3]);
        $fdisplay(file_output,"regfiles4: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[4]);
        $fdisplay(file_output,"regfiles5: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[5]);
        $fdisplay(file_output,"regfiles6: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[6]);
```

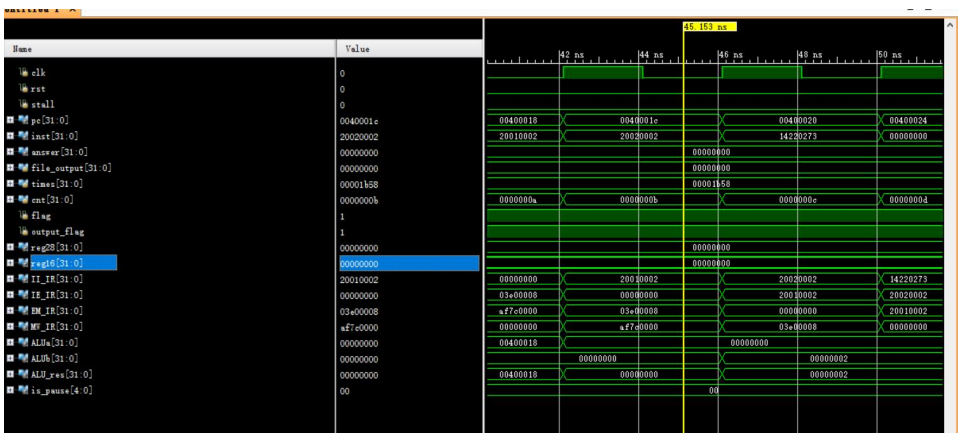
```
        $fdisplay(file_output,"regfiles8: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[8]);
        $fdisplay(file_output,"regfiles9: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[9]);
        $fdisplay(file_output,"regfiles10: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[10]);
        $fdisplay(file_output,"regfiles11: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[11]);
        $fdisplay(file_output,"regfiles12: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[12]);
        $fdisplay(file_output,"regfiles13: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[13]);
        $fdisplay(file_output,"regfiles14: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[14]);
        $fdisplay(file_output,"regfiles15: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[15]);
        $fdisplay(file_output,"regfiles16: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[16]);
        $fdisplay(file_output,"regfiles17: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[17]);
        $fdisplay(file_output,"regfiles18: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[18]);
        $fdisplay(file_output,"regfiles19: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[19]);
        $fdisplay(file_output,"regfiles20: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[20]);
        $fdisplay(file_output,"regfiles21: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[21]);
        $fdisplay(file_output,"regfiles22: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[22]);
        $fdisplay(file_output,"regfiles23: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[23]);
        $fdisplay(file_output,"regfiles24: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[24]);
        $fdisplay(file_output,"regfiles25: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[25]);
        $fdisplay(file_output,"regfiles26: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[26]);
        $fdisplay(file_output,"regfiles27: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[27]);
        $fdisplay(file_output,"regfiles28: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[28]);
        $fdisplay(file_output,"regfiles29: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[29]);
        $fdisplay(file_output,"regfiles30: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[30]);
        $fdisplay(file_output,"regfiles31: %h",pcpu_top_tb.uun.pcpu.regfile.array_reg[31]);
    end
endmodule
```

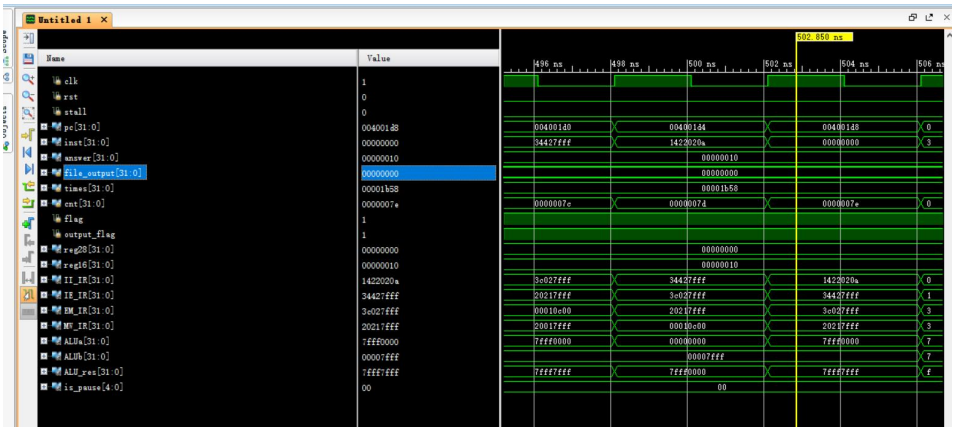
之后我们需要导入具体的 COE 文件，然后分别进行功能仿真和时序仿真就可以了。

五、 实验仿真的波形图某时刻寄存器值的物理意义

1、动态流水线的波形图及某时刻寄存器值的物理意义







相应的 Mars 寄存器：

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	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$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	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		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	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$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	0x00000000
$sp	29	0x00000000
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

具体的寄存器的值如图所示，我们可以看到学过的 32 个寄存器对应的编号和其中存放的具体数值。我们在具体的指令的执行过程中，在使用算术逻辑单元 ALU 进行计算的时候，会使用到上述的寄存器，在访存和写回的阶段也会使用到上述寄存器。

具体的物理意义就是模拟 CPU 的运算过程，实现指令的具体执行。

六、 实验验算数学模型及算法程序

```
int a[m],b[m],c[m],d[m];
a[0]=0;
b[0]=1;
a[i]=a[i-1]+i;
b[i]=b[i-1]+3i;
```

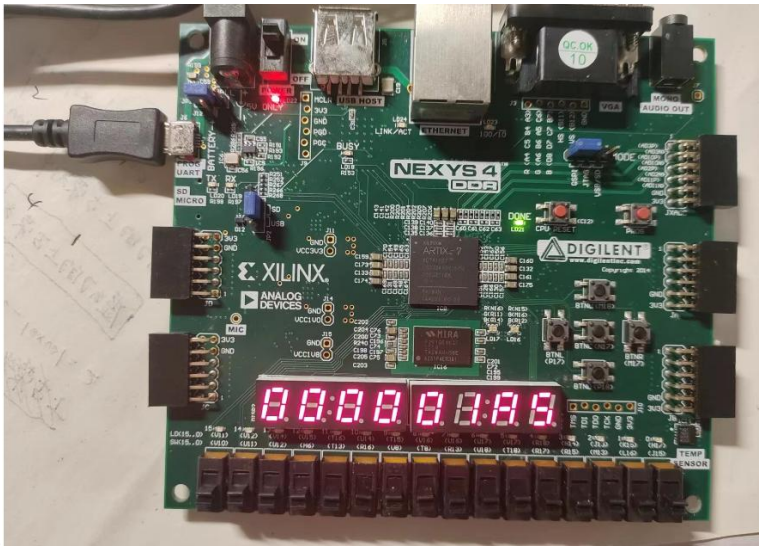
$$c[i]=\begin{cases} a[i], & 0\le i\le 19 \\ a[i]+b[i], & 20\le i\le 39 \\ a[i]*b[i], & 40\le i\le 59 \end{cases}$$

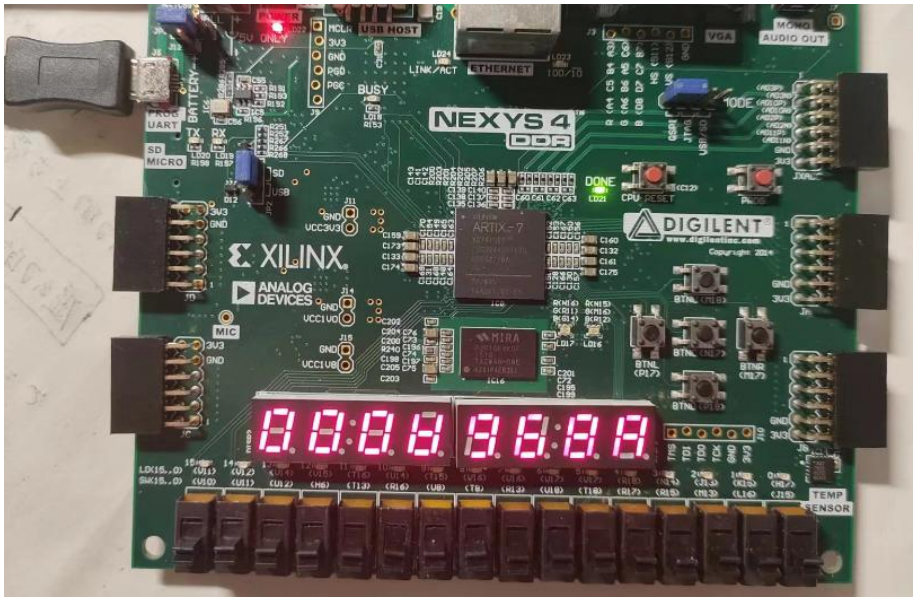
$$d[i]=\begin{cases} b[i], & 0\le i\le 19 \\ a[i]*c[i], & 20\le i\le 39 \\ c[i]*b[i], & 40\le i\le 59 \end{cases}$$

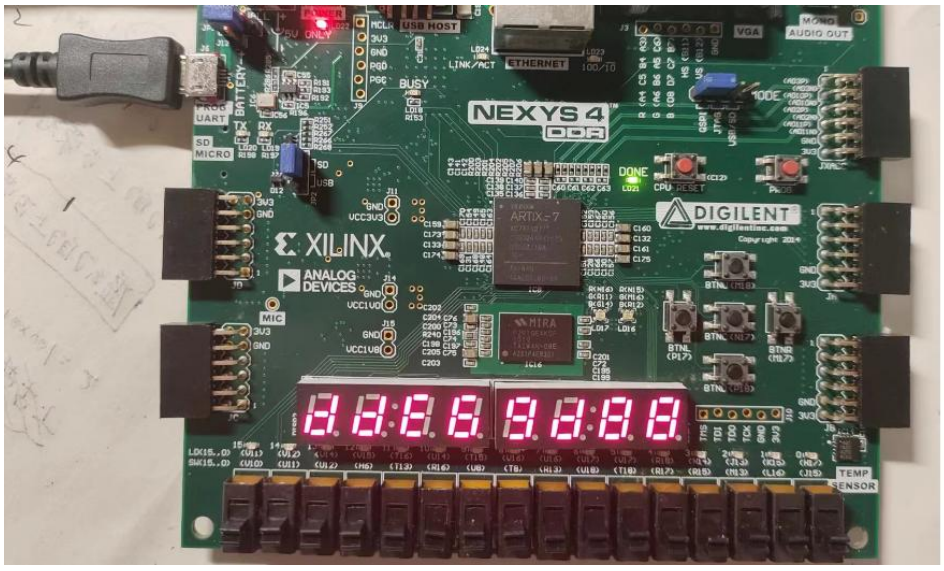
七、 实验验算程序下板测试过程与实现

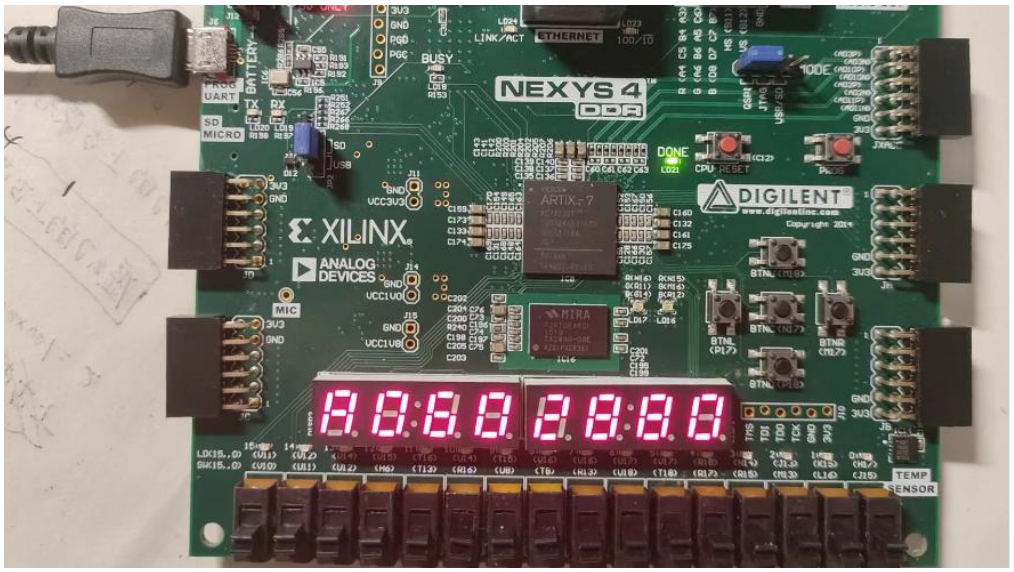
我们利用 MARS 中导出为 COE 文件，进行下板测试，修改相应的 XDC 约束文件，配置相应的管脚。为了使实验结果清晰可见，我们使用七段数码管来观察具体的实验现象。

七段数码管不停的变化，程序正在进行运算：

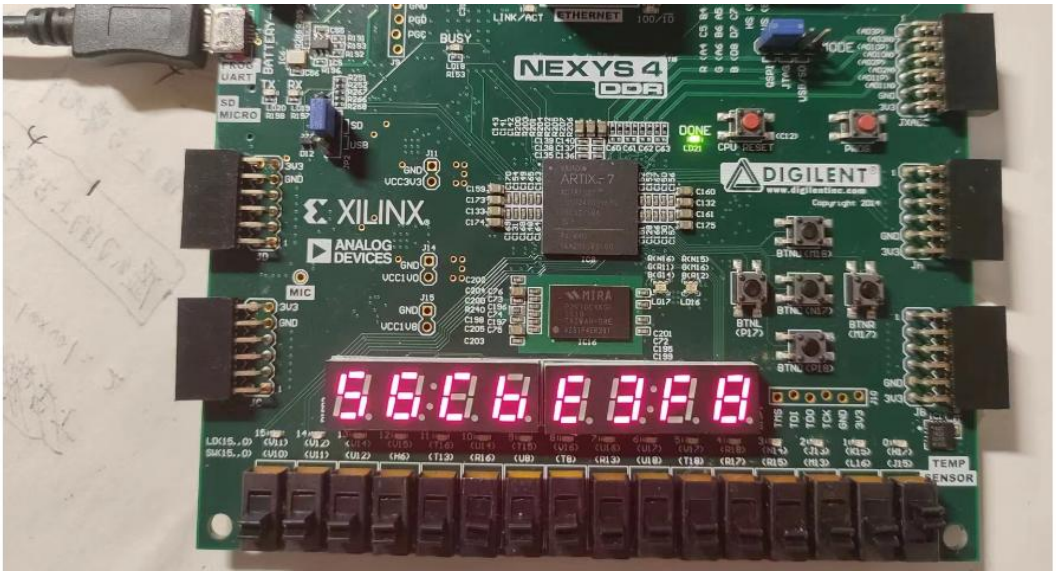


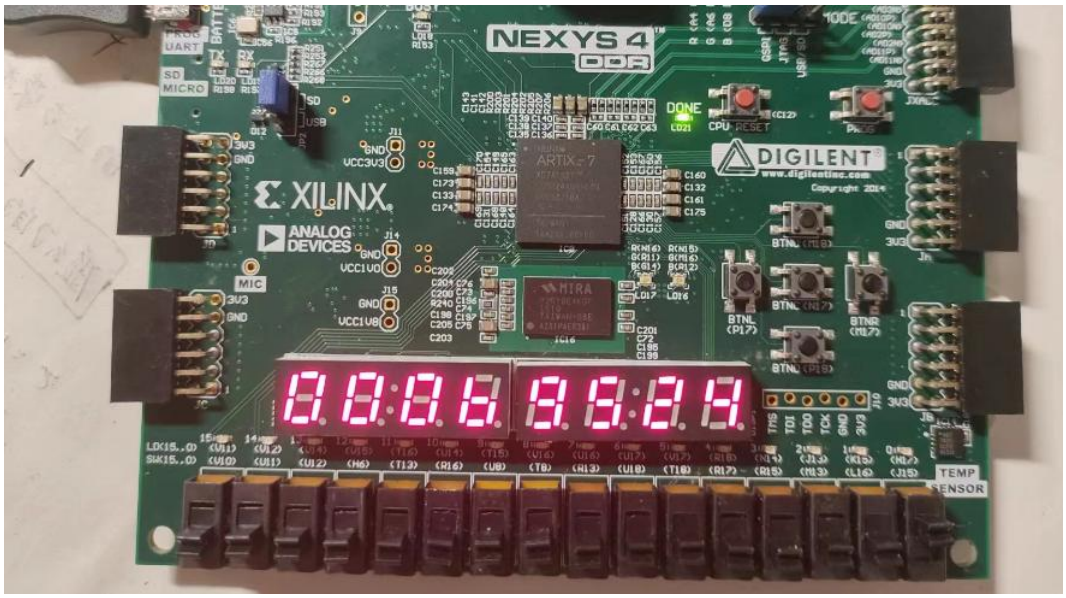

NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
TEMP SENSOR

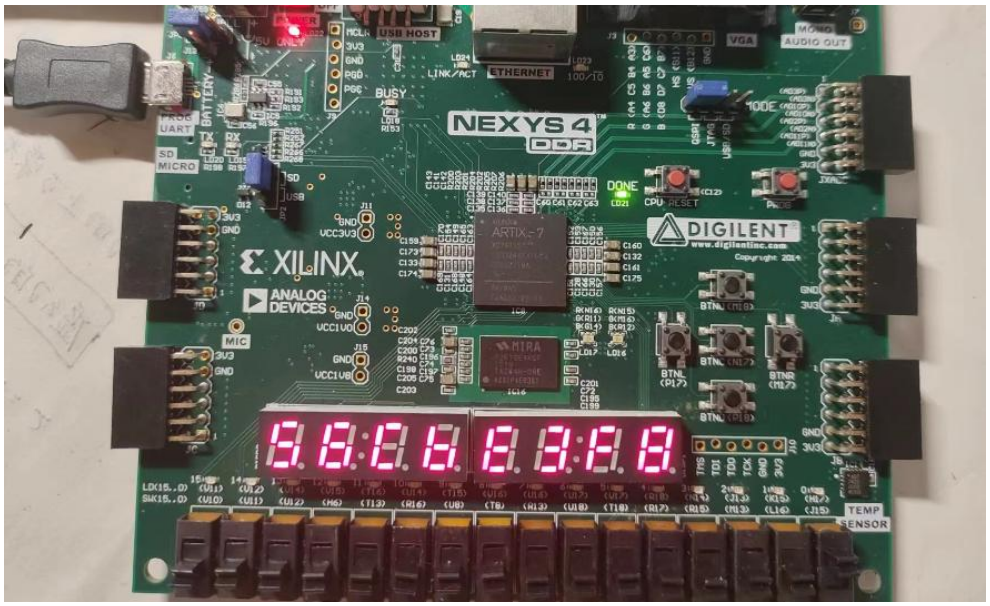

NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
TEMP SENSOR


NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES
ETHERNET
TEMP SENSOR

按下中断后，会停止运算：


NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES


NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES


NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES

八、 流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析）

1、 动态流水线的性能指标定性分析

动态流水线的性能指标定性分析，主要可以从以下几个方面来进行分析。分别是吞吐量，延迟，流水线效率和加速比。

吞吐量是指单位时间内完成的指令数量。由于静态流水线能够在同一时间执行多个指令，因此可以提高处理器的吞吐量。我们注意到如果我们减少每个阶段的执行时间，就能够相应的提高吞吐量，但是这也造成了一定的流水线的风险程度，因此选择适宜的阶段周期或者适当的在原有的 5 个阶段中增加阶段都是可以增加流水线吞吐量的方法。

延迟是指从指令进入流水线到完成执行所需的时间。由于指令在流水线中依次经过不同的阶段，每个阶段都需要一定的时间来完成操作。静态流水线的延迟取决于流水线的深度和各个阶段的执行时间。

同时不当的延迟可能会增加并行运行的风险，我们在具体的时序仿真可以看到，信号的传递的门逻辑本身有一定的延迟，会加大相关流水线的风险。

流水线效率是指流水线在单位时间内实际执行指令的比例。流水线效率受到流水线冒险（如结构冒险、数据冒险和控制冒险）的影响。如果流水线冒险发生频率较高，会导致流水线效率下降，因为流水线需要暂停等待冒险解决。较好的流水线设计可以减少冒险的发生，从而提高流水线效率。

加速比是指流水线处理器在相同工作量下的性能提升比例。由于流水线可以并行执行多个指令，因此可以提高处理器的性能。加速比的计算公式为：加速比 = 非流水线处理器的执行时间 / 流水线处理器的执行时间。

九、 总结与体会

本次我们使用之前写的多周期 CPU 进行修改，将其修改成简单的流水线 CPU，感受颇多。

与上学期所做的多周期 CPU 不同的是，流水线 CPU 的主要设计思路就是将原本的指令执行过程进行一个细分，原来在单周期或多周期的设计中并不怎么注重这个，但是在流水线中我们将一个指令的执行过程分解为若干个阶段，每个阶段由不同的组件负责，完成相应的工作，并将结果传给下一组件。

我们让不同的指令同时在不同的阶段执行，充分利用 CPU，提高指令的并行性。而五个阶段的流水线，就是取指（IF）、译码（ID）、执行（EXE）、访存（MEM）和写回（WB）。

十、 附件（所有程序）

1、 动态流水线的设计程序

```
module DPCPU(
        input           clk,
        input           rst,
        input   [31:0]  instr_in,
        input   [31:0]  DMEM_out,

        input           stall,

        output  [31:0]  pc,
        output  [31:0]  DMEM_addr,
        output  [31:0]  DMEM_data,
        output          DMEM_wena,
        output          DMEM_rena,

        output  [31:0]  answer
    );
    //

    //连线
    //CU input
    wire            ID_is_branch;
    wire    [31:0]  II_IR_out,
                    IE_IR_out,
                    EM_IR_out,
                    MW_IR_out;
    //CU output
    wire            RF_wena,
                    RF_rena,
                    ID_equal,
                    ext16_sext,
                    MUX2_jpc_s,
                    MUX2_jump_s,
                    MUX2_II_IR_s,
                    MUX2_ext5_s,
                    MUX2_pusha_s,
                    MUX2_pushb_s,
                    MUX2_EM_res_s,
                    MUX2_MW_res_s;
    wire            MUX2_cmp_pusha_s,
                    MUX2_cmp_pushb_s;
    wire    [31:0]  MUX2_cmp_pusha_out,
                    MUX2_cmp_pushb_out;
    wire    [1:0]   MUX4_PC_s,
```

```
                MUX4_IE_ALUa_s,
                MUX4_IE_ALUb_s;
wire    [3:0]   aluc;
wire    [4:0]   is_pause;
wire    [4:0]   WB_waddr,
                ID_rsc,
                ID_rtc,
                ID_sa;
wire    [15:0]  ID_immed;
wire    [25:0]  ID_jaddr;
```

```
//中间连线
wire            zero,
                carry,
                negative,
                overflow;

wire    [4:0]   MUX2_ext5_out;
```

```
wire    [31:0]  MUX2_jpc_out;
wire    [31:0]  MUX4_PC_out,
                PC_out;
wire    [31:0]  NPC_out;
wire    [31:0]  MUX2_jump_out;
wire    [31:0]  MUX2_II_IR_out;
wire    [31:0]  II_NPC_out;
wire    [31:0]  MW_res_out,
                RF_data1,
                RF_data2;
wire    [31:0]  ext5_out;
wire    [31:0]  ext16_out;
wire    [31:0]  j_pc;
wire    [31:0]  b_pc;
wire    [31:0]  ALU_r,
                MUX2_pusha_out,
                MUX2_pushb_out;
wire    [31:0]  MUX4_IE_ALUa_out;
wire    [31:0]  IE_ALUa_out;
wire    [31:0]  MUX4_IE_ALUb_out;
wire    [31:0]  IE_ALUb_out;
wire    [31:0]  IE_wdata_out;
wire    [31:0]  MUX2_EM_res_out;
wire    [31:0]  EM_res_out;
```

```
wire    [31:0]  EM_wdata_out;
wire    [31:0]  MUX2_MW_res_out;
wire    [31:0]  hi,
                lo;
assign pc=PC_out;
assign  DMEM_addr=EM_res_out;
assign  DMEM_data=EM_wdata_out;
//CU
Controller      CU(
    .II_IR_out      (II_IR_out),
    .IE_IR_out      (IE_IR_out),
    .EM_IR_out      (EM_IR_out),
    .MW_IR_out      (MW_IR_out),
    .ID_is_branch   (ID_is_branch),

    .stall          (stall),

    .RF_wena        (RF_wena),
    .RF_rena        (RF_rena),
    .DMEM_rena      (DMEM_rena),
    .DMEM_wena      (DMEM_wena),
    .ID_equal       (ID_equal),
    .ext16_sext     (ext16_sext),
    .MUX2_jpc_s     (MUX2_jpc_s),
    .MUX2_jump_s    (MUX2_jump_s),
    .MUX2_II_IR_s   (MUX2_II_IR_s),
    .MUX2_ext5_s    (MUX2_ext5_s),
    .MUX2_cmp_pusha_s   (MUX2_cmp_pusha_s),
    .MUX2_cmp_pushb_s   (MUX2_cmp_pushb_s),
    .MUX2_pusha_s   (MUX2_pusha_s),
    .MUX2_pushb_s   (MUX2_pushb_s),
    .MUX2_EM_res_s  (MUX2_EM_res_s),
    .MUX2_MW_res_s  (MUX2_MW_res_s),
    .MUX4_PC_s      (MUX4_PC_s),
    .MUX4_IE_ALUa_s (MUX4_IE_ALUa_s),
    .MUX4_IE_ALUb_s (MUX4_IE_ALUb_s),
    .aluc           (aluc),
    .is_pause       (is_pause),
    .WB_waddr       (WB_waddr),
    .ID_rsc         (ID_rsc),
    .ID_rtc         (ID_rtc),
    .ID_sa          (ID_sa),
    .ID_immed       (ID_immed),
    .ID_jaddr       (ID_jaddr)
```

```
);
//IF
PCReg           cpu_PC(
    .clk        (clk),
    .rst        (rst),
    .pc_in      (MUX4_PC_out),
    .pc_out     (PC_out)
);
/*IMEM*/
NPC             cpu_NPC(
    .data_in    (PC_out),
    .data_out   (NPC_out)
);

MUX2            MUX2_jump(
    .data0      (MUX2_jpc_out),
    .data1      (b_pc),
    .s          (MUX2_jump_s),
    .out        (MUX2_jump_out)
);

MUX4            MUX4_PC(
    .data0      (NPC_out),
    .data1      (PC_out),
    .data2      (MUX2_jump_out),
    .data3      (`INT_ENTRY),
    .s          (MUX4_PC_s),
    .out        (MUX4_PC_out)
);
//IF/ID
```

```
MUX2            MUX2_II_IR(
    .data0      (instr_in),
    .data1      (32'b0),
    .s          (MUX2_II_IR_s),
    .out        (MUX2_II_IR_out)
);
```

```
SegmentReg      II_IR(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`IF_STATE]),
    .next_pause (is_pause[`ID_STATE]),
    .data_in    (MUX2_II_IR_out),
```

```
    .data_out   (II_IR_out)
);
```

```
SegmentReg      II_NPC(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`IF_STATE]),
    .next_pause (is_pause[`ID_STATE]),
    .data_in    (NPC_out),
    .data_out   (II_NPC_out)
);
//ID
```

```
Regfiles        cpu_ref(
    .clk        (clk),
    .rst        (rst),
    .wena       (RF_wena),
    .rena       (RF_rena),
    .wdata      (MW_res_out),
    .waddr      (WB_waddr),
    .raddr1     (ID_rsc),
    .raddr2     (ID_rtc),
    .rdata1     (RF_data1),
    .rdata2     (RF_data2),
    .answer     (answer)
);
```

```
MUX2            MUX2_cmp_pusha(
    .data0      (RF_data1),
    .data1      (MUX2_pusha_out),
    .s          (MUX2_cmp_pusha_s),
    .out        (MUX2_cmp_pusha_out)
);
MUX2            MUX2_cmp_pushb(
    .data0      (RF_data2),
    .data1      (MUX2_pushb_out),
    .s          (MUX2_cmp_pushb_s),
    .out        (MUX2_cmp_pushb_out)
);
cmp             cpu_cmp(
    .num1       (MUX2_cmp_pusha_out),
    .num2       (MUX2_cmp_pushb_out),
    .cmp_equal  (ID_equal),
    .is_branch  (ID_is_branch)
```

```
);
```

```
MUX25           MUX2_ext5(
    .data0      (ID_sa),
    .data1      (RF_data1[4:0]),
    .s          (MUX2_ext5_s),
    .out        (MUX2_ext5_out)
);
```

```
ext5            CPU_ext5(
    .a          (MUX2_ext5_out),
    .sext       (1'b0),
    .b          (ext5_out)
);
```

```
ext16           CPU_ext16(
    .a          (ID_immed),
    .sext       (ext16_sext),
    .b          (ext16_out)
);
```

```
II              CPU_II(
    .a          (II_NPC_out[31:28]),
    .b          (ID_jaddr),
    .r          (j_pc)
);


MUX2             MUX2_jpc(
    .data0      (j_pc),
    .data1      (RF_data1),
    .s          (MUX2_jpc_s),
    .out        (MUX2_jpc_out)
);

add             branch_add(
    .a          (II_NPC_out),
    .b          ({{(14){ID_immed[15]}},ID_immed,2'b0}),
    .r          (b_pc)
);
//ID/EXE
```

```
MUX2            MUX2_pusha(
    .data0      (MUX2_EM_res_out),
    .data1      (MUX2_MW_res_out),
```

```
    .s          (MUX2_pusha_s),
    .out        (MUX2_pusha_out)
);
MUX2            MUX2_pushb(
    .data0      (MUX2_EM_res_out),
    .data1      (MUX2_MW_res_out),
    .s          (MUX2_pushb_s),
    .out        (MUX2_pushb_out)
);
MUX4            MUX4_IE_ALUa(
    .data0      (RF_data1),
    .data1      (ext5_out),
    .data2      (MUX2_pusha_out),
    .data3      (II_NPC_out),
    .s          (MUX4_IE_ALUa_s),
    .out        (MUX4_IE_ALUa_out)
);
```

```
SegmentReg      IE_ALUa(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`ID_STATE]),
    .next_pause (is_pause[`EXE_STATE]),
    .data_in    (MUX4_IE_ALUa_out),
    .data_out   (IE_ALUa_out)
);
```

```
MUX4            MUX4_IE_ALUb(
    .data0      (RF_data2),
    .data1      (ext16_out),
    .data2      (MUX2_pushb_out),
    .data3      (32'd4),
    .s          (MUX4_IE_ALUb_s),
    .out        (MUX4_IE_ALUb_out)
);
```

```
SegmentReg      IE_ALUb(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`ID_STATE]),
    .next_pause (is_pause[`EXE_STATE]),
    .data_in    (MUX4_IE_ALUb_out),
    .data_out   (IE_ALUb_out)
);
```

```
SegmentReg      IE_IR(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`ID_STATE]),
    .next_pause (is_pause[`EXE_STATE]),
    .data_in    (II_IR_out),
    .data_out   (IE_IR_out)
);
```

```
SegmentReg      IE_wdata(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`ID_STATE]),
    .next_pause (is_pause[`EXE_STATE]),
    .data_in    (RF_data2),
    .data_out   (IE_wdata_out)
);
//EXE
```

```
ALU             cpu_ALU(
    .a          (IE_ALUa_out),
    .b          (IE_ALUb_out),
    .aluc       (aluc),
    .r          (ALU_r),
    .zero       (zero),
    .carry      (carry),
    .negative   (negative),
    .overflow   (overflow)
);
```

```
MULT            cpu_MULT(
    .a          (IE_ALUa_out),
    .b          (IE_ALUb_out),
    .hi         (hi),
    .lo         (lo)
);
//EXE/MEM
```

```
SegmentReg      EM_IR(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`EXE_STATE]),
    .next_pause (is_pause[`MEM_STATE]),
```

```
    .data_in    (IE_IR_out),
    .data_out   (EM_IR_out)
);
```

```
MUX2            MUX2_EM_res(
    .data0      (ALU_r),
    .data1      (lo),
    .s          (MUX2_EM_res_s),
    .out        (MUX2_EM_res_out)
);
```

```
SegmentReg      EM_res(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`EXE_STATE]),
    .next_pause (is_pause[`MEM_STATE]),
    .data_in    (MUX2_EM_res_out),
    .data_out   (EM_res_out)
);
```

```
SegmentReg      EM_wdata(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`EXE_STATE]),
    .next_pause (is_pause[`MEM_STATE]),
    .data_in    (IE_wdata_out),
    .data_out   (EM_wdata_out)
);
//MEM
//DMEM
```

```
//wena,rena in CU
//MEM/WB
```

```
MUX2            MUX2_MW_res(
    .data0      (EM_res_out),
    .data1      (DMEM_out),
    .s          (MUX2_MW_res_s),
    .out        (MUX2_MW_res_out)
);
SegmentReg      MW_res(
    .clk        (clk),
    .rst        (rst),
    .prev_pause (is_pause[`MEM_STATE]),
```

```
        .next_pause (is_pause[`WB_STATE]),
        .data_in    (MUX2_MW_res_out),
        .data_out   (MW_res_out)
    );
    SegmentReg      MW_IR(
        .clk        (clk),
        .rst        (rst),
        .prev_pause (is_pause[`MEM_STATE]),
        .next_pause (is_pause[`WB_STATE]),
        .data_in    (EM_IR_out),
        .data_out   (MW_IR_out)
    );
    //WB
    //Regfiles
endmodule
```

```
module sccomp_dataflow(
    input           clk,
    input           rst,
    input           stall,
    output  [31:0]  pc,
    output  [31:0]  inst,
    output  [31:0]  answer
    );
    /*数据通路*/
    wire    [31:0]  DMEM_out,DMEM_addr,DMEM_data;
    wire    [31:0]  IMEM_addr_in=(pc-`PC_INIT)>>2,
                    DMEM_addr_in=(DMEM_addr-`DMEM_BASE)>>2;
    wire            DMEM_wena,DMEM_rena;
    DPCPU           sccpu(
        .clk        (clk),
        .rst        (rst),
        .instr_in   (inst),
        .DMEM_out   (DMEM_out),
        .stall      (stall),
        .pc         (pc),
        .DMEM_addr  (DMEM_addr),
        .DMEM_data  (DMEM_data),
        .DMEM_wena  (DMEM_wena),
        .DMEM_rena  (DMEM_rena),
```

```
        .answer     (answer)
    );
    DMEM            cpu_DMEM(
        .clk        (clk),
        .rst        (rst),
        .wena       (DMEM_wena),
        .rena       (DMEM_rena),
        .addr       (DMEM_addr_in[`ADDR_BYTES-1:0]),
        .data_in    (DMEM_data),
        .data_out   (DMEM_out)
    );
    IMEM            cpu_IMEM(
        .addr       (IMEM_addr_in[10:0]),
        .instr      (inst)
    );
endmodule
```

```
module add(
    input   [31:0]  a,
    input   [31:0]  b,

    output  [31:0]  r
    );
    assign r=a+b;
endmodule
```

```

module ALU(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output [31:0] r,
    output zero,
    output carry,
    output negative,
    output overflow
    );
parameter
    addu=4'd0,
    add =4'd2,
    subu=4'd1,
    sub =4'd3,
```

```
    _and=4'd4,
    _or =4'd5,
    _xor=4'd6,
    _nor=4'd7,
    lui1=4'd8,
    lui2=4'd9,
    sltu=4'd10,
    slt =4'd11,
    sra =4'd12,
    srl =4'd13,
    sll =4'd14,
    slr =4'd15;
reg signed  [31:0]res;
reg         [32:0]sres;
wire signed [31:0]sa=a,sb=b;
always @ (*)begin
    sres=33'b0;
    case(aluc)
        add:begin
            res<=a+b;sres<={sa[31],sa}+{sb[31],sb};
        end
        addu:begin
            res<=sa+sb;
        end
        sub:begin
            res<=a-b;sres<={sa[31],sa}-{sb[31],sb};
        end
        subu:begin
            res<=sa-sb;
        end
        _and:begin
            res<=a&b;
        end
        _or:begin
            res<=a|b;
        end
        _xor:begin
            res<=a^b;
        end
        _nor:begin
            res<=~(a|b);
        end
        lui1:begin
            res<={b[15:0],16'b0};
```

```
        end
        lui2:begin
            res<={b[15:0],16'b0};
        end
        slt:begin
            res<=(sa<sb)?32'b1:32'b0;
        end
        sltu:begin
            res<=(a<b)?32'b1:32'b0;
        end
        sra:begin
            res<=sb>>>a[4:0];
        end
        sll,slr:begin
            res<=b<<a[4:0];
        end
        srl:begin
            res<=b>>a[4:0];
        end
        default:begin
            res<=32'b0;
        end
    endcase
end
assign r=res[31:0];
assign zero=(aluc==slt||aluc==sltu)?((a==b)?1'b1:1'b0):((r==32'b0)?1'b1:1'b0);
assign
carry=(aluc==addu||aluc==subu||aluc==sltu||aluc==sra||aluc==sll||aluc==srl)?res[31]:1'b0;
assign negative=(aluc==slt)?((res==32'b1)?1'b1:1'b0):(res[31]);
assign overflow=(aluc==add||aluc==sub)?(sres[32]^sres[31]):1'b0;
endmodule
```

```
module cmp(
    input   [31:0]  num1,
    input   [31:0]  num2,
    input           cmp_equal,

    output          is_branch
    );
    reg     res;
    always @ * begin
```

```
        //beq use it
        if(cmp_equal) begin
            res=(num1==num2);
        end
        //bne use it
        else begin
            res=(num1!=num2);
        end
    end
    assign is_branch=res;
endmodule
```

```
module Controller(
    input   [31:0]  II_IR_out,
                    IE_IR_out,
                    EM_IR_out,
                    MW_IR_out,
    input           ID_is_branch,
    input           stall,

    output          RF_wena,
                    RF_rena,
                    DMEM_rena,
                    DMEM_wena,
                    ID_equal,
                    ext16_sext,
                    MUX2_jpc_s,
                    MUX2_jump_s,
                    MUX2_II_IR_s,
                    MUX2_cmp_pusha_s,
                    MUX2_cmp_pushb_s,
                    MUX2_ext5_s,
                    MUX2_pusha_s,
                    MUX2_pushb_s,
                    MUX2_EM_res_s,
                    MUX2_MW_res_s,

    output  [1:0]   MUX4_PC_s,
                    MUX4_IE_ALUa_s,
                    MUX4_IE_ALUb_s,
    output  [3:0]   aluc,
    output  [4:0]   is_pause,
```

```
                WB_waddr,
                ID_rsc,
                ID_rtc,
                ID_sa,
output  [15:0]  ID_immed,
output  [25:0]  ID_jaddr
);
wire    [`CODE_NUM:0]   Code_type   [1:4];
wire    [4:0]           rsc         [1:4];
wire    [4:0]           rtc         [1:4];
wire    [4:0]           rdc         [1:4];
wire    [4:0]           sa          [1:4];
wire    [15:0]          immed       [1:4];
wire    [25:0]          j_addr      [1:4];
/* 译码器 */
Decoder     ID_Decoder(
    .instr      (II_IR_out),
    .rsc        (rsc[`ID_STATE]),
    .rtc        (rtc[`ID_STATE]),
    .rdc        (rdc[`ID_STATE]),
    .sa         (sa[`ID_STATE]),
    .immed      (immed[`ID_STATE]),
    .j_addr     (j_addr[`ID_STATE]),
    .code       (`IDC)
);
Decoder     EXE_Decoder(
    .instr      (IE_IR_out),
    .rsc        (rsc[`EXE_STATE]),
    .rtc        (rtc[`EXE_STATE]),
    .rdc        (rdc[`EXE_STATE]),
    .sa         (sa[`EXE_STATE]),
    .immed      (immed[`EXE_STATE]),
    .j_addr     (j_addr[`EXE_STATE]),
    .code       (`EXEC)
);
Decoder     MEM_Decoder(
    .instr      (EM_IR_out),
    .rsc        (rsc[`MEM_STATE]),
    .rtc        (rtc[`MEM_STATE]),
    .rdc        (rdc[`MEM_STATE]),
    .sa         (sa[`MEM_STATE]),
    .immed      (immed[`MEM_STATE]),
    .j_addr     (j_addr[`MEM_STATE]),
    .code       (`MEMC)
```

```
    );
    Decoder     WB_Decoder(
        .instr      (MW_IR_out),
        .rsc        (rsc[`WB_STATE]),
        .rtc        (rtc[`WB_STATE]),
        .rdc        (rdc[`WB_STATE]),
        .sa         (sa[`WB_STATE]),
        .immed      (immed[`WB_STATE]),
        .j_addr     (j_addr[`WB_STATE]),
        .code       (`WBC)
    );
    //conflict judge
    wire    IDCrRsc=~(`IDC[`_SLL]|`IDC[`_SRL]|`IDC[`_SRA]|
                `IDC[`_LUI]|`IDC[`_J]|`IDC[`_JAL]|`IDC[`_BRK]);
    wire    IDCrRtc=~(`IDC[`_ADDI]|`IDC[`_ADDIU]|`IDC[`_ANDI]|`IDC[`_ORI]|`IDC[`_XORI]|
                `IDC[`_SLTI]|`IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_LW]|`IDC[`_SW]|
                `IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|`IDC[`_BRK]);
    wire    EXECwRF=~(`EXEC[`_SW]|`EXEC[`_BEQ]|`EXEC[`_BNE]|`EXEC[`_J]|`EXEC[`_JR]|
                `EXEC[`_BRK]);
    wire    MEMCwRF=~(`MEMC[`_SW]|`MEMC[`_BEQ]|`MEMC[`_BNE]|`MEMC[`_J]|`MEMC[`_JR]|
                `MEMC[`_BRK]);
    wire    [4:0]
            EXEC_waddr= (`EXEC[`_JAL])? 5'd31:
                        (`EXEC[`_ADDI]|`EXEC[`_ADDIU]|`EXEC[`_ANDI]|`EXEC[`_ORI]|`EXEC[`_XORI]|
                        `EXEC[`_SLTI]|`EXEC[`_SLTIU]|`EXEC[`_LUI]|`EXEC[`_LW])?rtc[`EXE_STATE]:
                        rdc[`EXE_STATE],
           MEMC_waddr= (`MEMC[`_JAL])? 5'd31:
                        (`MEMC[`_ADDI]|`MEMC[`_ADDIU]|`MEMC[`_ANDI]|`MEMC[`_ORI]|`MEMC[`_XORI]|
                        `MEMC[`_SLTI]|`MEMC[`_SLTIU]|`MEMC[`_LUI]|`MEMC[`_LW])?rtc[`MEM_STATE]:
                        rdc[`MEM_STATE];
    //前推不考虑 nop
    wire    EXEC_pusha= (II_IR_out!=32'b0) && (IE_IR_out!=32'b0) &&
                        IDCrRsc &&  EXECwRF &&
                        (rsc[`ID_STATE]==EXEC_waddr)&& (EXEC_waddr!=5'b0) &&
(rsc[`ID_STATE]!=5'b0);
    wire    EXEC_pushb= (II_IR_out!=32'b0) && (IE_IR_out!=32'b0) &&
                        IDCrRtc &&  EXECwRF &&
                        (rtc[`ID_STATE]==EXEC_waddr)&& (EXEC_waddr!=5'b0) &&
(rtc[`ID_STATE]!=5'b0);
    wire    MEMC_pusha= (II_IR_out!=32'b0) && (EM_IR_out!=32'b0) &&
                        IDCrRsc &&  MEMCwRF &&
                        (rsc[`ID_STATE]==MEMC_waddr)&& (MEMC_waddr!=5'b0) &&
(rsc[`ID_STATE]!=5'b0);
    wire    MEMC_pushb= (II_IR_out!=32'b0) && (EM_IR_out!=32'b0) &&
```

```
                    IDCrRtc &&  MEMCwRF &&
                    (rtc[`ID_STATE]==MEMC_waddr)&& (MEMC_waddr!=5'b0) &&
(rtc[`ID_STATE]!=5'b0);
    wire    ID_MustStop=(`EXEC[`_LW]) && (II_IR_out!=32'b0) &&(
                        (IDCrRsc &&(rtc[`EXE_STATE]==rsc[`ID_STATE]))||
                        (IDCrRtc &&(rtc[`EXE_STATE]==rtc[`ID_STATE]))
                    );
    reg     [4:0]   pause;
    always @(*) begin
        if(stall)begin
            pause=`HALT;
        end
        else if(ID_MustStop) begin
            pause=`ID_PAUSE;
        end
        else begin
            pause=`NO_PAUSE;
        end
    end
    assign  is_pause=pause;
    wire    II_IR_drop=`IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|((`IDC[`_BEQ]|`IDC[`_BNE])&
ID_is_branch);
    assign  RF_wena=~(`WBC[`_SW]|`WBC[`_BEQ]|`WBC[`_BNE]|`WBC[`_J]|`WBC[`_JR]|`WBC[`_BRK]);
    assign  RF_rena=1'b1;
    assign  DMEM_rena=`MEMC[`_LW];
    assign  DMEM_wena=`MEMC[`_SW];
    assign  ID_equal=`IDC[`_BEQ];
    assign  ext16_sext=~(`IDC[`_ADDIU]|`IDC[`_ANDI]|`IDC[`_ORI]|`IDC[`_XORI]|
                        `IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_LW]|`IDC[`_SW]);
    assign  WB_waddr=(`WBC[`_JAL])   ?   5'd31   :
                    (`WBC[`_ADDI]|`WBC[`_ADDIU]|`WBC[`_ANDI]|`WBC[`_ORI]|`WBC[`_XORI]|`WBC[`_S
LTI]|`WBC[`_SLTIU]|`WBC[`_LUI]|`WBC[`_LW])?rtc[`WB_STATE]:
                    rdc[`WB_STATE];

    assign  aluc[0]=`EXEC[`_SUB]|`EXEC[`_SUBU]|`EXEC[`_OR]|`EXEC[`_NOR]|`EXEC[`_SLT]|`EXEC[`_S
RL]|
                    `EXEC[`_SRLV]|`EXEC[`_ORI]|`EXEC[`_SLTI];
    assign  aluc[1]=`EXEC[`_ADD]|`EXEC[`_SUB]|`EXEC[`_XOR]|`EXEC[`_NOR]|`EXEC[`_SLT]|`EXEC[`_S
LTU]|
                    `EXEC[`_SLL]|`EXEC[`_SLLV]|`EXEC[`_ADDI]|`EXEC[`_XORI]|`EXEC[`_LW]|`EXEC[`_
SW]|
                    `EXEC[`_SLTI]|`EXEC[`_SLTIU];
    assign  aluc[2]=`EXEC[`_AND]|`EXEC[`_OR]|`EXEC[`_XOR]|`EXEC[`_NOR]|`EXEC[`_SLL]|`EXEC[`_SR
L]|`EXEC[`_SRA]|
```

```
                  `EXEC[`_SLLV]|`EXEC[`_SRLV]|`EXEC[`_SRAV]|`EXEC[`_ANDI]|`EXEC[`_ORI]|`EXEC[
`_XORI];
    assign  aluc[3]=`EXEC[`_SLT]|`EXEC[`_SLTU]|`EXEC[`_SLL]|`EXEC[`_SRL]|`EXEC[`_SRA]|`EXEC[`_
SLLV]|`EXEC[`_SRLV]|
                  `EXEC[`_SLTI]|`EXEC[`_SLTIU]|`EXEC[`_LUI];

    assign  ID_rsc=rsc[`ID_STATE];
    assign  ID_rtc=rtc[`ID_STATE];
    assign  ID_sa=sa[`ID_STATE];
    assign  ID_immed=immed[`ID_STATE];
    assign  ID_jaddr=j_addr[`ID_STATE];
    // MUX2_jpc_s: 0 j_pc from II,1 rdata1(jr)
    assign  MUX2_jpc_s=`IDC[`_JR];
    // MUX2_jump: 0 MUX2_jpc_out,1 branch_addr
    assign  MUX2_jump_s=(`IDC[`_BEQ]|`IDC[`_BNE])&ID_is_branch;
    //MUX2_II_IR: 0 instr_in,1 32'b0(drop)
    assign MUX2_II_IR_s=II_IR_drop;
    //MUX2_ext5_s 0:ID_sa 1: RF_data1
    assign MUX2_ext5_s=`IDC[`_SLLV]|`IDC[`_SRLV]|`IDC[`_SRAV];
    assign MUX2_cmp_pusha_s=EXEC_pusha || MEMC_pusha;
    assign MUX2_cmp_pushb_s=EXEC_pushb || MEMC_pushb;
    //MUX2_pusha: 0 ALU.r 1:MUX2_MW_res_out(MEM.res)
    //MUX2_pushb similar
    //both push,choose nearest EXE data
    assign MUX2_pusha_s=~EXEC_pusha && MEMC_pusha;
    assign MUX2_pushb_s=~EXEC_pushb && MEMC_pushb;
    //MUX2_EM_res: 0 ALU.r,1 MULT.lo
    assign MUX2_EM_res_s=`EXEC[`_MUL];
    //MUX2_MW_res: 0 EM_res_out,1 DMEM_data_out
    assign MUX2_MW_res_s=`MEMC[`_LW];
    //MUX4_PC_s[0] 0:NPC_out|MUX2_jump_out  1:PC_out|int_entry
    //MUX4_PC_s[1] 0:NPC_out|PC_out         1:MUX2_jump_out|int_entry
    assign MUX4_PC_s[0]=is_pause[`IF_STATE] |
                        `IDC[`_BRK];
    assign MUX4_PC_s[1]=`IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|((`IDC[`_BEQ]|`IDC[`_BNE])&
ID_is_branch)|
                        `IDC[`_BRK];
    //MUX4_IE_ALUa[0]   0:RF.data1|MUX2_pusha_out     1:ext5_out|NPC_out
    //MUX4_IE_ALUa[1]   0:RF.data1|ext5_out           1:MUX2_pusha_out|NPC_out
    assign MUX4_IE_ALUa_s[0]=   (`IDC[`_SLL]|`IDC[`_SRL]|`IDC[`_SRA]|
                               `IDC[`_SLLV]|`IDC[`_SRLV]|`IDC[`_SRAV]|
                               `IDC[`_JAL]) && !EXEC_pusha && !MEMC_pusha;
    assign MUX4_IE_ALUa_s[1]=   EXEC_pusha|MEMC_pusha|
                               `IDC[`_JAL];
```

```
    //MUX4_IE_ALUb[0]:  0:RF.data2|MUX2_pushb_out   1:ext16_out|32'd4
    //MUX4_IE_ALUb[1]:  0:RF.data2|ext16_out       1:MUX2_pushb_out|32'd4
    assign  MUX4_IE_ALUb_s[0]=  (`IDC[`_ADDI]|`IDC[`_ADDIU]|`IDC[`_ANDI]|`IDC[`_ORI]|`IDC[`_XO
RI]|
                               `IDC[`_SLTI]|`IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_LW]|`IDC[`_SW]|
                               `IDC[`_JAL])&& !EXEC_pushb && !MEMC_pushb;
    assign  MUX4_IE_ALUb_s[1]=  EXEC_pushb|MEMC_pushb|
                               `IDC[`_JAL];
endmodule
```

```
module Decoder(
    input       [31:0]          instr,
    output      [4:0]           rsc,
    output      [4:0]           rtc,
    output      [4:0]           rdc,
    output      [4:0]           sa,
    output      [15:0]          immed,
    output      [25:0]          j_addr,
    output  reg [`CODE_NUM:0]   code
    );
    wire [5:0]opcode = instr[31:26];
    wire [5:0]func = instr[5:0];
    assign rsc=instr[25:21];
    assign rtc=instr[20:16];
    assign rdc=instr[15:11];
    assign sa =instr[10: 6];
    assign immed=instr[15:0];
    assign j_addr =instr[25:0];
    always @ * begin
        if(opcode==6'b0)begin
            case(func)
              12'h020:code=`ADD;            //add
              12'h021:code=`ADDU;            //addu
              12'h022:code=`SUB;            //sub
              12'h023:code=`SUBU;            //subu
              12'h024:code=`AND;            //and
              12'h025:code=`OR;            //or
              12'h026:code=`XOR;            //xor
              12'h027:code=`NOR;            //nor
              12'h02a:code=`SLT;            //slt
              12'h02b:code=`SLTU;            //sltu
```

```
            12'h000:code=`SLL;           //sll
            12'h002:code=`SRL;           //srl
            12'h003:code=`SRA;           //sra
            12'h004:code=`SLLV;           //sllv
            12'h006:code=`SRLV;           //srlv
            12'h007:code=`SRAV;           //srav
            12'h008:code=`JR;           //jr
            //6'b001101:code=56'h0000_0000_4000_00;//break
            12'h00d:code=`BRK;           //break
            default:code=32'h0;
          endcase
      end
      else begin
          case(opcode)
             6'b001000:code=`ADDI;  //addi
             6'b001001:code=`ADDIU;  //addiu
             6'b001100:code=`ANDI;  //andi
             6'b001101:code=`ORI;  //ori
             6'b001110:code=`XORI;  //xori
             6'b100011:code=`LW;  //lw
             6'b101011:code=`SW;  //sw
             6'b000100:code=`BEQ;  //beq
             6'b000101:code=`BNE;  //bne
             6'b001010:code=`SLTI;  //slti
             6'b001011:code=`SLTIU;  //sltiu
             6'b001111:code=`LUI;  //lui
             6'b000010:code=`J;  //j
             6'b000011:code=`JAL;  //jal
             6'b011100:code=`MUL;    //mul
             default:  code=32'h0;
          endcase
      end
   end
endmodule
```

```
module Divider(
    input clk_in,
    input reset,
    output clk_out
    );
    reg clk=1'b0;
    reg [23:0]cnt;
```

```
    always @(posedge clk_in or posedge reset)begin
        if(reset)begin
            clk<=1'b0;
            cnt<=24'b0;
        end
        else begin
            if(cnt==24'h00ffff) begin
                clk<=~clk;
                cnt<=2'b0;
            end
            else begin
                cnt<=cnt+1;
            end
        end
    end
    assign clk_out=clk;
endmodule
```

```
module DMEM(
    input                   clk,
    input                   rst,
    input                   wena,
    input                   rena,
    input   [`ADDR_BYTES-1:0] addr,
    input   [31:0]          data_in,
    output  [31:0]          data_out
    );
    reg [31:0] ram_array[0:`MAX_MEMORY];
    integer i;
    always@(posedge clk or posedge rst) begin
        if (rst)begin
            for(i=0;i<=`MAX_MEMORY;i=i+1) begin
                ram_array[i]<=32'b0;
            end
        end
        else begin
            ram_array[addr]=(wena)?data_in:ram_array[addr];
        end
    end
    assign data_out=(rena)?ram_array[addr]:32'bz;
endmodule
```

```
module ext5#(parameter WIDTH = 5)(
    input [WIDTH - 1:0] a,
    input sext,     //sext 有效为符号扩展，否则 0 扩展
    output [31:0] b //32 位输出数据，
);
assign b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{{(32-WIDTH){1'b0}},a};
endmodule
```

```
module ext16#(parameter WIDTH = 16)(
    input [WIDTH - 1:0] a,
    input sext,     //sext 有效为符号扩展，否则 0 扩展
    output [31:0] b //32 位输出数据，
);
assign b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{{(32-WIDTH){1'b0}},a};
endmodule
```

```
module II(
    input   [3:0]   a,
    input   [25:0]  b,
    output  [31:0]  r
    );
    assign r={a,b,2'b0};
endmodule
```

```
module IMEM(
    input   [10:0]  addr,
    output  [31:0]  instr
    );
    dist_mem_gen_0      imem(
        addr,
        instr
    );
endmodule
```

```
module MULT(
    input [31:0] a,
    input [31:0] b,
```

```
    output [31:0] hi,
    output [31:0] lo
    );
    wire [63:0] moved_a[31:0];
    wire [63:0] z;
    assign moved_a[0]=(b[0])?(-{{32{a[31]}},a}):64'b0;
    assign moved_a[1]=(b[1]^b[0])?(b[1]?(-{{31{a[31]}},a,1'b0}):({{31{a[31]}},a,1'b0})):64'b0;
    assign moved_a[2]=(b[2]^b[1])?(b[2]?(-{{30{a[31]}},a,2'b0}):({{30{a[31]}},a,2'b0})):64'b0;
    assign moved_a[3]=(b[3]^b[2])?(b[3]?(-{{29{a[31]}},a,3'b0}):({{29{a[31]}},a,3'b0})):64'b0;
    assign moved_a[4]=(b[4]^b[3])?(b[4]?(-{{28{a[31]}},a,4'b0}):({{28{a[31]}},a,4'b0})):64'b0;
    assign moved_a[5]=(b[5]^b[4])?(b[5]?(-{{27{a[31]}},a,5'b0}):({{27{a[31]}},a,5'b0})):64'b0;
    assign moved_a[6]=(b[6]^b[5])?(b[6]?(-{{26{a[31]}},a,6'b0}):({{26{a[31]}},a,6'b0})):64'b0;
    assign moved_a[7]=(b[7]^b[6])?(b[7]?(-{{25{a[31]}},a,7'b0}):({{25{a[31]}},a,7'b0})):64'b0;
    assign moved_a[8]=(b[8]^b[7])?(b[8]?(-{{24{a[31]}},a,8'b0}):({{24{a[31]}},a,8'b0})):64'b0;
    assign moved_a[9]=(b[9]^b[8])?(b[9]?(-{{23{a[31]}},a,9'b0}):({{23{a[31]}},a,9'b0})):64'b0;
    assign
moved_a[10]=(b[10]^b[9])?(b[10]?(-{{22{a[31]}},a,10'b0}):({{22{a[31]}},a,10'b0})):64'b0;
    assign
moved_a[11]=(b[11]^b[10])?(b[11]?(-{{21{a[31]}},a,11'b0}):({{21{a[31]}},a,11'b0})):64'b0;
    assign
moved_a[12]=(b[12]^b[11])?(b[12]?(-{{20{a[31]}},a,12'b0}):({{20{a[31]}},a,12'b0})):64'b0;
    assign
moved_a[13]=(b[13]^b[12])?(b[13]?(-{{19{a[31]}},a,13'b0}):({{19{a[31]}},a,13'b0})):64'b0;
    assign
moved_a[14]=(b[14]^b[13])?(b[14]?(-{{18{a[31]}},a,14'b0}):({{18{a[31]}},a,14'b0})):64'b0;
    assign
moved_a[15]=(b[15]^b[14])?(b[15]?(-{{17{a[31]}},a,15'b0}):({{17{a[31]}},a,15'b0})):64'b0;
    assign
moved_a[16]=(b[16]^b[15])?(b[16]?(-{{16{a[31]}},a,16'b0}):({{16{a[31]}},a,16'b0})):64'b0;
    assign
moved_a[17]=(b[17]^b[16])?(b[17]?(-{{15{a[31]}},a,17'b0}):({{15{a[31]}},a,17'b0})):64'b0;
    assign
moved_a[18]=(b[18]^b[17])?(b[18]?(-{{14{a[31]}},a,18'b0}):({{14{a[31]}},a,18'b0})):64'b0;
    assign
moved_a[19]=(b[19]^b[18])?(b[19]?(-{{13{a[31]}},a,19'b0}):({{13{a[31]}},a,19'b0})):64'b0;
    assign
moved_a[20]=(b[20]^b[19])?(b[20]?(-{{12{a[31]}},a,20'b0}):({{12{a[31]}},a,20'b0})):64'b0;
    assign
moved_a[21]=(b[21]^b[20])?(b[21]?(-{{11{a[31]}},a,21'b0}):({{11{a[31]}},a,21'b0})):64'b0;
    assign
moved_a[22]=(b[22]^b[21])?(b[22]?(-{{10{a[31]}},a,22'b0}):({{10{a[31]}},a,22'b0})):64'b0;
    assign
moved_a[23]=(b[23]^b[22])?(b[23]?(-{{9{a[31]}},a,23'b0}):({{9{a[31]}},a,23'b0})):64'b0;
```

```
    assign
moved_a[24]=(b[24]^b[23])?(b[24]?(-{{8{a[31]}},a,24'b0}):({{8{a[31]}},a,24'b0})):64'b0;
    assign
moved_a[25]=(b[25]^b[24])?(b[25]?(-{{7{a[31]}},a,25'b0}):({{7{a[31]}},a,25'b0})):64'b0;
    assign
moved_a[26]=(b[26]^b[25])?(b[26]?(-{{6{a[31]}},a,26'b0}):({{6{a[31]}},a,26'b0})):64'b0;
    assign
moved_a[27]=(b[27]^b[26])?(b[27]?(-{{5{a[31]}},a,27'b0}):({{5{a[31]}},a,27'b0})):64'b0;
    assign
moved_a[28]=(b[28]^b[27])?(b[28]?(-{{4{a[31]}},a,28'b0}):({{4{a[31]}},a,28'b0})):64'b0;
    assign
moved_a[29]=(b[29]^b[28])?(b[29]?(-{{3{a[31]}},a,29'b0}):({{3{a[31]}},a,29'b0})):64'b0;
    assign
moved_a[30]=(b[30]^b[29])?(b[30]?(-{{2{a[31]}},a,30'b0}):({{2{a[31]}},a,30'b0})):64'b0;
    assign
moved_a[31]=(b[31]^b[30])?(b[31]?(-{{1{a[31]}},a,31'b0}):({{1{a[31]}},a,31'b0})):64'b0;
    assign z=moved_a[0]+moved_a[1]+moved_a[2]+moved_a[3]+moved_a[4]
            +moved_a[5]+moved_a[6]+moved_a[7]+moved_a[8]+moved_a[9]
            +moved_a[10]+moved_a[11]+moved_a[12]+moved_a[13]+moved_a[14]
            +moved_a[15]+moved_a[16]+moved_a[17]+moved_a[18]+moved_a[19]
            +moved_a[20]+moved_a[21]+moved_a[22]+moved_a[23]+moved_a[24]
            +moved_a[25]+moved_a[26]+moved_a[27]+moved_a[28]+moved_a[29]
            +moved_a[30]+moved_a[31];
    assign hi=z[63:32];
    assign lo=z[31:0];
endmodule
```

```
module MUX2(
    input       [31:0]  data0,
    input       [31:0]  data1,
    input               s,
    output      [31:0]  out
    );
    assign out=(s)?data1:data0;
endmodule
```

```

module MUX4(
    input       [31:0]  data0,
    input       [31:0]  data1,
    input       [31:0]  data2,
    input       [31:0]  data3,
```

```
    input       [1:0]   s,
    output      [31:0]  out
    );
    assign out= (s==2'b00)?data0:
                (s==2'b01)?data1:
                (s==2'b10)?data2:
                data3;
endmodule
```

```
module MUX25(
    input       [4:0]  data0,
    input       [4:0]  data1,
    input               s,
    output      [4:0]  out
    );
    assign out=(s)?data1:data0;
endmodule
```

```
module NPC(
    input   [31:0]  data_in,
    output  [31:0]  data_out
    );
    assign data_out=data_in+32'h4;
endmodule
```

```
module PCReg(
    input           clk,
    input           rst,
    input   [31:0]  pc_in,
    output  [31:0]  pc_out
    );
    reg [31:0]  pc;
    always @ (posedge clk or posedge rst)begin
        if(rst)
            pc<=`PC_INIT;
         else
            pc<=pc_in;
    end
    assign pc_out=pc;
endmodule
```

```
module Regfiles(
    input           clk,
    input           rst,
    input           wena,
    input           rena,
    input   [31:0]  wdata,
    input   [4:0]   waddr,
    input   [4:0]   raddr1,
    input   [4:0]   raddr2,

    output  [31:0]  rdata1,
    output  [31:0]  rdata2,
    output  [31:0]  answer
    );
    reg [31:0]  array_reg   [31:0];
    integer i;
    always @(posedge clk or posedge rst)begin
        if(rst) begin
            for(i=0;i<=31;i=i+1)
                array_reg[i]<=32'b0;
        end
        else begin
            array_reg[waddr]<=(wena && waddr!=5'b0)?wdata:array_reg[waddr];
        end
    end
    assign rdata1=(rena)?((wena && waddr==raddr1 && waddr!=5'b0)?wdata:array_reg[raddr1]):32'bz;
    assign rdata2=(rena)?((wena && waddr==raddr2 && waddr!=5'b0)?wdata:array_reg[raddr2]):32'bz;
    assign answer=array_reg[16];
endmodule
```

```
module seg7x16(
     input clk,
     input reset,
     input cs,
     input [31:0] i_data,
     output [7:0] o_seg,
     output [7:0] o_sel
    );

    reg [14:0] cnt;
     always @ (posedge clk, posedge reset)
      if (reset)
```

```
      cnt <= 0;
    else
      cnt <= cnt + 1'b1;

  wire seg7_clk = cnt[14];

   reg [2:0] seg7_addr;

   always @ (posedge seg7_clk, posedge reset)
     if(reset)
        seg7_addr <= 0;
      else
        seg7_addr <= seg7_addr + 1'b1;

   reg [7:0] o_sel_r;

   always @ (*)
     case(seg7_addr)
        7 : o_sel_r = 8'b01111111;
        6 : o_sel_r = 8'b10111111;
        5 : o_sel_r = 8'b11011111;
        4 : o_sel_r = 8'b11101111;
        3 : o_sel_r = 8'b11110111;
        2 : o_sel_r = 8'b11111011;
        1 : o_sel_r = 8'b11111101;
        0 : o_sel_r = 8'b11111110;
      endcase

   reg [31:0] i_data_store;
   always @ (posedge clk, posedge reset)
     if(reset)
        i_data_store <= 0;
      else if(cs)
        i_data_store <= i_data;

   reg [7:0] seg_data_r;
   always @ (*)
     case(seg7_addr)
        0 : seg_data_r = i_data_store[3:0];
        1 : seg_data_r = i_data_store[7:4];
        2 : seg_data_r = i_data_store[11:8];
        3 : seg_data_r = i_data_store[15:12];
        4 : seg_data_r = i_data_store[19:16];
        5 : seg_data_r = i_data_store[23:20];
```

```
        6 : seg_data_r = i_data_store[27:24];
        7 : seg_data_r = i_data_store[31:28];
       endcase

    reg [7:0] o_seg_r;
    always @ (posedge clk, posedge reset)
      if(reset)
        o_seg_r <= 8'hff;
       else
        case(seg_data_r)
          4'h0 : o_seg_r <= 8'hC0;
        4'h1 : o_seg_r <= 8'hF9;
        4'h2 : o_seg_r <= 8'hA4;
        4'h3 : o_seg_r <= 8'hB0;
        4'h4 : o_seg_r <= 8'h99;
        4'h5 : o_seg_r <= 8'h92;
        4'h6 : o_seg_r <= 8'h82;
        4'h7 : o_seg_r <= 8'hF8;
        4'h8 : o_seg_r <= 8'h80;
        4'h9 : o_seg_r <= 8'h90;
        4'hA : o_seg_r <= 8'h88;
        4'hB : o_seg_r <= 8'h83;
        4'hC : o_seg_r <= 8'hC6;
        4'hD : o_seg_r <= 8'hA1;
        4'hE : o_seg_r <= 8'h86;
        4'hF : o_seg_r <= 8'h8E;
        endcase

    assign o_sel = o_sel_r;
    assign o_seg = o_seg_r;

endmodule
```

```
module SegmentReg(
    input           clk,
    input           rst,
    input           prev_pause,
    input           next_pause,
    input   [31:0]  data_in,

    output  [31:0]  data_out
    );
    reg [31:0]  data;
```

```
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            data<=32'h0;
        end
        else if(prev_pause&&~next_pause)
            data<=32'h0;
        else if(prev_pause)
            data<=data;
        else begin
            data<=data_in;
        end
    end
    assign data_out=data;
endmodule
```

```
module testDPCPU;
reg             clk,
                rst,
                stall;
wire    [31:0]  pc,
                inst,
                answer;

integer file_output;
integer times=7000;
integer cnt=0;
```

```
reg             flag=1'b1;
reg             output_flag=1'b0;
```

```
sccomp_dataflow uut(
    .clk(clk),
    .rst(rst),
    .stall(stall),
    .inst(inst),
    .pc(pc),
    .answer(answer)
);
initial begin
    file_output = $fopen("D:\\CPU_AutoCheck\\yanshou1_result.txt");
    clk= 0;
```

```
    rst = 1;
    stall=1'b0;
    #0.1;
    rst = 0;
    repeat(times) #2 clk=~clk;
    flag=1'b0;
    #2;clk=~clk;
end
```

```
wire    [31:0]  reg28=uut.sccpu.cpu_ref.array_reg[28],
                reg16=uut.sccpu.cpu_ref.array_reg[16];
wire    [31:0]
                II_IR=uut.sccpu.II_IR_out,
                IE_IR=uut.sccpu.IE_IR_out,
                EM_IR=uut.sccpu.EM_IR_out,
                MW_IR=uut.sccpu.MW_IR_out;
wire    [31:0]  ALUa=uut.sccpu.cpu_ALU.a,
                ALUb=uut.sccpu.cpu_ALU.b;
wire    [31:0]  ALU_res=uut.sccpu.MUX2_EM_res_out;
wire    [4:0]   is_pause=uut.sccpu.is_pause;
//wire    [31:0]  RF_data1=uut.sccpu.RF_data1,
//                RF_data2=uut.sccpu.RF_data2;
always @(posedge clk) begin
    if(!output_flag && EM_IR!=32'b0) begin
        output_flag<=1'b1;
    end
    else begin
        output_flag<=output_flag;
    end
    if(uut.sccpu.II_IR_out!=32'h0000000d) begin
        cnt=cnt+1;
    end
    else begin
        $display("instr num: %d",cnt);
    end
    if(flag && output_flag)begin
        if(uut.sccpu.MW_IR_out!=32'h0)begin
        $fdisplay(file_output, "pc: %h", pc);
        $fdisplay(file_output, "instr: %h", uut.sccpu.MW_IR_out);
        $fdisplay(file_output, "regfile0: %h", uut.sccpu.cpu_ref.array_reg[0]);
        $fdisplay(file_output, "regfile1: %h", uut.sccpu.cpu_ref.array_reg[1]);
        $fdisplay(file_output, "regfile2: %h", uut.sccpu.cpu_ref.array_reg[2]);
        $fdisplay(file_output, "regfile3: %h", uut.sccpu.cpu_ref.array_reg[3]);
        $fdisplay(file_output, "regfile4: %h", uut.sccpu.cpu_ref.array_reg[4]);
```

```
        $fdisplay(file_output, "regfile5: %h", uut.sccpu.cpu_ref.array_reg[5]);
        $fdisplay(file_output, "regfile6: %h", uut.sccpu.cpu_ref.array_reg[6]);
        $fdisplay(file_output, "regfile7: %h", uut.sccpu.cpu_ref.array_reg[7]);
        $fdisplay(file_output, "regfile8: %h", uut.sccpu.cpu_ref.array_reg[8]);
        $fdisplay(file_output, "regfile9: %h", uut.sccpu.cpu_ref.array_reg[9]);
        $fdisplay(file_output,"regfile10: %h",uut.sccpu.cpu_ref.array_reg[10]);
        $fdisplay(file_output,"regfile11: %h",uut.sccpu.cpu_ref.array_reg[11]);
        $fdisplay(file_output,"regfile12: %h",uut.sccpu.cpu_ref.array_reg[12]);
        $fdisplay(file_output,"regfile13: %h",uut.sccpu.cpu_ref.array_reg[13]);
        $fdisplay(file_output,"regfile14: %h",uut.sccpu.cpu_ref.array_reg[14]);
        $fdisplay(file_output,"regfile15: %h",uut.sccpu.cpu_ref.array_reg[15]);
        $fdisplay(file_output,"regfile16: %h",uut.sccpu.cpu_ref.array_reg[16]);
        $fdisplay(file_output,"regfile17: %h",uut.sccpu.cpu_ref.array_reg[17]);
        $fdisplay(file_output,"regfile18: %h",uut.sccpu.cpu_ref.array_reg[18]);
        $fdisplay(file_output,"regfile19: %h", uut.sccpu.cpu_ref.array_reg[19]);
        $fdisplay(file_output,"regfile20: %h", uut.sccpu.cpu_ref.array_reg[20]);
        $fdisplay(file_output,"regfile21: %h", uut.sccpu.cpu_ref.array_reg[21]);
        $fdisplay(file_output,"regfile22: %h", uut.sccpu.cpu_ref.array_reg[22]);
        $fdisplay(file_output,"regfile23: %h", uut.sccpu.cpu_ref.array_reg[23]);
        $fdisplay(file_output,"regfile24: %h", uut.sccpu.cpu_ref.array_reg[24]);
        $fdisplay(file_output,"regfile25: %h", uut.sccpu.cpu_ref.array_reg[25]);
        $fdisplay(file_output,"regfile26: %h", uut.sccpu.cpu_ref.array_reg[26]);
        $fdisplay(file_output,"regfile27: %h", uut.sccpu.cpu_ref.array_reg[27]);
        $fdisplay(file_output,"regfile28: %h", uut.sccpu.cpu_ref.array_reg[28]);
        $fdisplay(file_output,"regfile29: %h", uut.sccpu.cpu_ref.array_reg[29]);
        $fdisplay(file_output,"regfile30: %h", uut.sccpu.cpu_ref.array_reg[30]);
        $fdisplay(file_output,"regfile31: %h", uut.sccpu.cpu_ref.array_reg[31]);
        end
    end
    else if(!flag)begin
        $fclose(file_output);
        $finish;
    end
end
endmodule
```