

## 小学期数逻实验

### 流水线 MIPS 处理器解决背包问题

姓名：温迪雅

班级：无 97

学号：2019011147

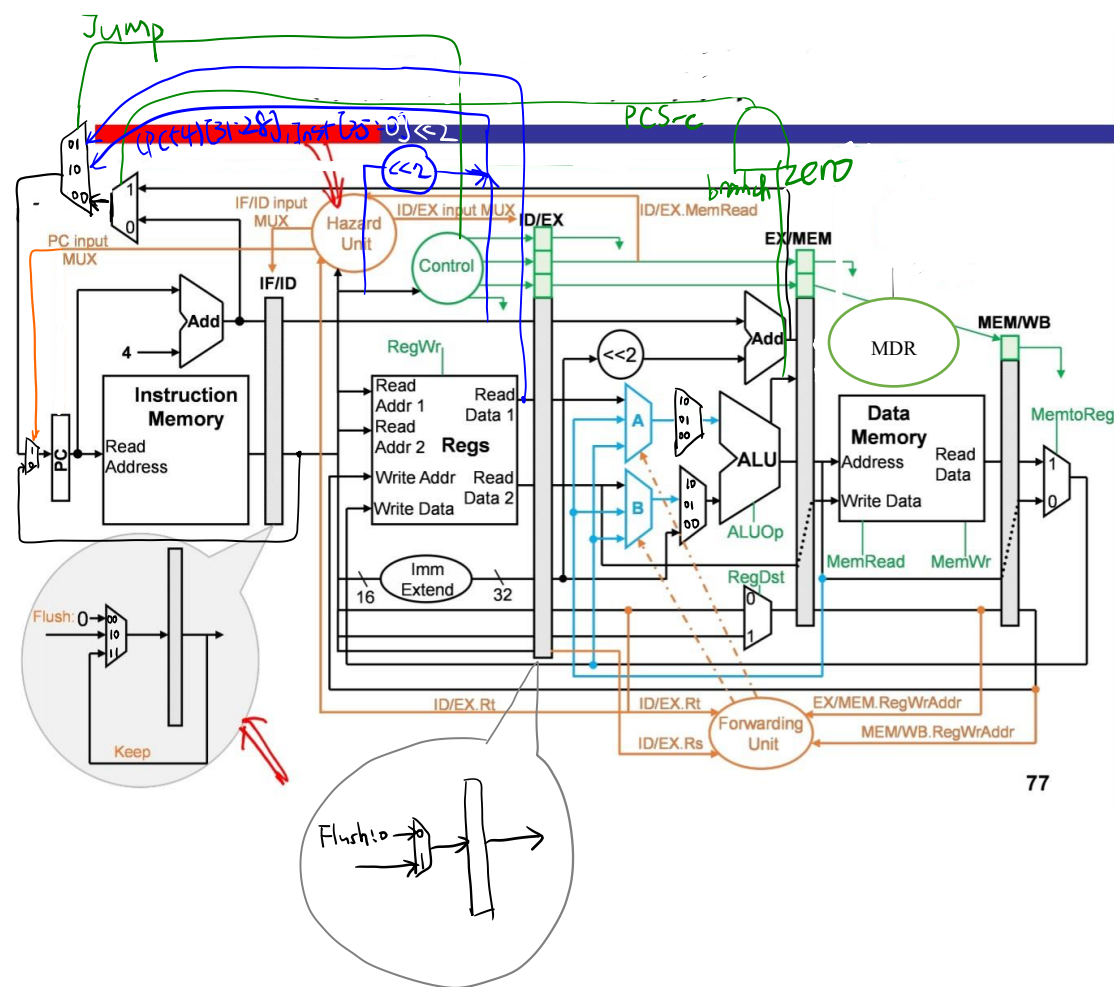
日期：2021.7.13

## 一、实验目的

- (1) 掌握 MIPS 流水线处理器功能模块和数据通路的设计及 RTL 实现；
- (2) 掌握 MIPS 流水线处理器中冒险和数据关联问题及解决方法；
- (3) 掌握 MIPS 流水线处理器的性能分析（最高时钟频率等）；
- (4) 掌握利用 sw 指令进行外设设计的方法。

## 二、设计方案

数据通路：



流水线五个阶段的设计：

步骤\指令类型	R-type	jr, j	jalr, jal
IF	$IR \leftarrow \text{Memory}[PC] ; PC \leftarrow PC + 4$		
ID	$Op \leftarrow IR[31:26] ; A \leftarrow \text{Reg}[IR[25:21]] ; B \leftarrow \text{Reg}[IR[20:16]]$ $\text{If}(j/\text{jal}) PC \leftarrow \{PC[31:28], IR[25:0], 2'b00\}$ $\text{If}(jr/\text{jalr}) PC \leftarrow \text{Reg}[IR[25:21]]$		
EX	$ALUOut \leftarrow A \text{ op } B$ EX/MEM.ALUOut MEM/WB.ALUOut MDR	/	/

MEM	/	/	/
WB	Reg[IR[15:11]]<=ALUOut	/	Reg[31] <= A

步骤\指令类型	Lw	Lui	I-type	Beq
IF	IR<=Memory[PC] ; PC<=PC+4			
ID	Op <= IR[31:26] ; A<=Reg[IR[25:21]] ; B<=Reg[IR[20:16]] If(j/jal) PC<={PC[31:28],IR[25:0],2'b00} If(jr/jalr) PC<=Reg[IR[25:21]]			
EX	ALUout<=A+sign-extend(IR[15:0]) EX/MEM.ALUOut MEM/WB.ALUOut	ALUout<=ImmExtOut+0	ALUout<=A op ImmExtOut	Addr_adder<=PC+(sign-extended(IR[15:0]<<2)) If(A==B) PC<=Addr_adder
MEM	MDR<=Memory[ALUout]	/	/	/
WB	Reg[IR[20:16]]<=MDR	Reg[IR[20:16]]<=ALUout	/	/

步骤\指令类型	Sw	Bne	Blez	Bgtz,bltz
IF	IR<=Memory[PC] ; PC<=PC+4			
ID	Op <= IR[31:26] ; A<=Reg[IR[25:21]] ; B<=Reg[IR[20:16]] If(j/jal) PC<={PC[31:28],IR[25:0],2'b00} If(jr/jalr) PC<=Reg[IR[25:21]]			
EX	ALUout<=A+sign-extend(IR[15:0]) EX/MEM.ALUOut MEM/WB.ALUOut	Addr_adder<=PC+(sign-extended(IR[15:0]<<2)) If(A!=B) PC<=Addr_adder	Addr_adder<=PC+(sign-extended(IR[15:0]<<2)) If(A<=0) PC<=Addr_adder	Bgtz:if(A>0) Bltz:if(A<0)
MEM	Memory[ALUout]<=B	/	/	/
WB	/	/	/	/

冒险及数据关联问题的处理:

(1) 对于同时读写寄存器的冲突, 采用在寄存器内部添加多路选择器, 判断读取的 rs 或 rt 寄存器是否即为将写入的 rd 寄存器, 若是则直接取用即将写入 rd 寄存器的值, 由此完成寄存器支持先写后读的功能。

(2) 对于 lw 竞争采取 1 个 stall+forwarding 的解决方法。添加 hazard unit 通过 OpCode 判断若当前指令为 lw 且 lw 指令的 rt 与当前 IF 阶段的 rs 或 rt 相同则 stall 一个周期, stall 通过将 PC 和 IFIDReg 前的多路选择信号置为 keep 功能, 将 IDEXReg 前的多路选择信号置为 flush 功能 (所有控制信号置无效) 实现, forwarding unit 通过比较 EXMEMReg 和 MEMWBReg 中要写入的寄存器是否与 lw 指令的下一条指令的 rs 或 rt 寄存器相同, 若相同则将 EXMEMReg 或

MEMWBReg 将要写入寄存器的值 Write\_data 转发至 lw 指令下一条指令的 ALU 的输入数据，具体实现通过 forwarding unit 产生 ALU 前多路选择器的控制信号实现。

(3) 分支跳转指令 (beq、bne、blez、bgtz、bltz) 造成的控制冒险采用如下解决方案：分支跳转指令在 EX 阶段通过 OpCode 和 Funct 进行判断，若是分支跳转指令则取消 IF、ID 阶段的指令，亦即使 IFIDReg 和 IDEXReg flush (所有控制信号置无效)。

(4) 跳转指令 (j、jal、jr、jalr) 造成的控制冒险采用如下解决方案：在 ID 阶段通过 OpCode 和 Funct 进行判断，若是跳转指令则取消 IF 阶段的指令，亦即使 IFIDReg flush (所有控制信号置无效)。

外设的设计：

将控制七段数码管的信号 an 和 Cathodes 寄存器放在 DataMemory 中，使用 sw 指令将值写入其地址 (0x40000010) 即完成对七段数码管的控制。

控制信号的设计：

[1:0] ALUA=2'b10 – 将 EX 阶段运算结果转发至 EX 阶段的输入

2'b01 – 将 MEM 阶段的结果转发至 EX 阶段的输入

2'b00 – 不做转发

[1:0] ALUB=2'b10 – 将 EX 阶段运算结果转发至 EX 阶段的输入

2'b01 – 将 MEM 阶段的结果转发至 EX 阶段的输入

2'b00 – 不做转发

[1:0] IFIDInputMux=2'b11 – lw 数据关联冒险引起的 stall

2'b00 – 分支或跳转指令引起的 flush

2'b10 – 正常数据通路

PCInputMux=0 – lw 数据关联冒险引起的 stall

1 – 正常数据通路

IDEXInputMux=0 – lw 数据关联引起的 stall 或分支指令引起的 flush

1 – 正常数据通路

PCSource=IDEXBranchout && Zero //beq 指令

IDEXBranchout&&(!Zero) //bne 指令

IDEXBranchout&&(!ALUOut) //blez

IDEXBranchout&&ALUOut //其他分支指令

[1:0] Jump=2'b00 – 非跳转指令

2'b10 – j 或 jal 指令

2'b01 – jr 或 jalr 指令

RegWrite=0 – 不允许写寄存器堆，1 – 允许写寄存器堆

MemRead=0 – 不允许读数据存储器，1—允许读数据存储器

MemWrite=0 – 不允许写数据存储器，1—允许写数据存储器

[1:0] MemtoReg=2'b00—ALU 计算结果

2'b01—PC

2'b11—从数据存储器中取出的值

[1:0] ALUSrcA=2'b11—Shamt

2'b00—0

Else: A

[1:0] ALUSrcB=2'b11—B

2'b10—ImmExtShift

2'b01—ImmExtOut

2'b00—0

[1:0] RegDst=2'b11—rt

2'b00—rd

Else: ra

[3:0] ALUOp 和 Funct 共同决定 ALU 进行的运算，具体见 ALUControl 模块

其他的有 IDEX-, EXMEM-, MEMWB-前缀的仅是级间寄存器传递信号需要，与其对应的原始控制信号并无二致，在此不再赘述。

### 三、关键代码及文件清单

文件清单：ALU.v ALU 计算单元

ALUControl.v ALU 直接控制信号产生模块

clk\_40k.v 40kHz 时钟分频模块

Controller.v 控制信号产生模块

DataMemory.v 数据和外设 RAM

EXMEMReg.v EX-MEM 阶段间寄存器

IDEXReg.v ID-EX 阶段间寄存器

IFIDReg.v IF-ID 阶段间寄存器

ImmProcess.v 立即数处理模块

InstMemory.v 指令存储器

MEMWBReg.v MEM-WB 阶段间寄存器

myconstraints.xdc 管脚约束文件

PC.v PC 寄存器

PipelineCPU.v 顶层模块

RegisterFile.v 寄存器堆

RegTemp.v 临时寄存器模块

## test\_cpu.v 仿真顶层文件

////////////////////////////////// ALU.v //////////////////////////////////

```
`timescale 1ns / 1ps
```

```
module ALU(ALUConf, Sign, In1, In2, Zero, Result);
```

```
    // Control Signals
```

```
    input [4:0] ALUConf;
```

```
    input Sign;
```

```
    // Input Data Signals
```

```
    input [31:0] In1;
```

```
    input [31:0] In2;
```

```
    // Output
```

```
    output Zero;
```

```
    output reg [31:0] Result;
```

```
    //Zero logic
```

```
    assign Zero = (Result == 0);
```

```
    //ALU logic
```

```
    wire[1:0] ss;
```

```
    assign ss = {In1[31], In2[31]};
```

```
    wire lt_31;
```

```
    assign lt_31 = (In1[30:0] < In2[30:0]);
```

```
    wire lt_signed;
```

```
    assign lt_signed = (In1[31] ^ In2[31])?
```

```
        ((ss == 2'b01)? 0: 1): lt_31;
```

```
    wire gt_31;
```

```
    assign gt_31 = (In1[30:0] > In2[30:0]);
```

```
    wire gt_signed;
```

```
    assign gt_signed = (In1[31] ^ In2[31])?
```

```
        ((ss == 2'b10)? 0: 1): gt_31;
```

```

always @(*) begin
    case (ALUConf)
        5'b000000: Result <= In1 + In2;
        5'b000001: Result <= In1 | In2;
        5'b000010: Result <= In1 & In2;
        5'b000110: Result <= In1 - In2;
        5'b001111: Result <= {31'h00000000, Sign? lt_signed: (In1 < In2)};
        5'b010000: Result <= {31'h00000000, Sign? gt_signed: (In1 > In2)};
        5'b011000: Result <= ~(In1 | In2);
        5'b011010: Result <= In1 ^ In2;
        5'b100000: Result <= (In2 >> In1[4:0]);
        5'b110000: Result <= ({32{In2[31]}}, In2) >> In1[4:0];
        5'b110010: Result <= (In2 << In1[4:0]);
        5'b110100: Result <= In1 & (~In2); // A·(~B)
        default: Result <= 32'h00000000;
    endcase
end

endmodule

//////////////////////////////// ALUControl.v //////////////////////////////////
`timescale 1ns / 1ps
module ALUControl(ALUOp, Funct, ALUConf, Sign);
    //Control Signals
    input [3:0] ALUOp;
    //Inst. Signals
    input [5:0] Funct;
    //Output Control Signals
    output reg [4:0] ALUConf;
    output Sign;

    parameter aluADD = 5'b000000;
    parameter aluOR  = 5'b000001;
    parameter aluAND = 5'b000010;
    parameter aluSUB = 5'b000110;

```

```
parameter aluSLT = 5'b00111;
parameter aluSGT = 5'b01000;
parameter aluNOR = 5'b01100;
parameter aluXOR = 5'b01101;
parameter aluSRL = 5'b10000;
parameter aluSRA = 5'b11000;
parameter aluSLL = 5'b11001;
parameter aluCHAJI = 5'b11010;

assign Sign = (ALUOp[2:0] == 3'b010)? ~Funct[0]: ~ALUOp[3];

reg [4:0] aluFunct;
always @(*)
    case (Funct)
        6'b00_0000: aluFunct <= aluSLL;
        6'b00_0010: aluFunct <= aluSRL;
        6'b00_0011: aluFunct <= aluSRA;
        6'b10_0000: aluFunct <= aluADD;
        6'b10_0001: aluFunct <= aluADD;
        6'b10_0010: aluFunct <= aluSUB;
        6'b10_0011: aluFunct <= aluSUB;
        6'b10_0100: aluFunct <= aluAND;
        6'b10_0101: aluFunct <= aluOR;
        6'b10_0110: aluFunct <= aluXOR;
        6'b10_0111: aluFunct <= aluNOR;
        6'b10_1010: aluFunct <= aluSLT;
        6'b10_1011: aluFunct <= aluSLT;

        //0x29=0010 1001 = 101001
        6'b10_1001: aluFunct <= aluCHAJI;
        default: aluFunct <= aluADD;
    endcase

always @(*)
    case (ALUOp[2:0])
```



```

        3'b000: ALUConf <= aluADD;
        3'b001: ALUConf <= aluSUB;
        3'b100: ALUConf <= aluAND;
        3'b101: ALUConf <= aluSLT;
        3'b011: ALUConf <= aluSGT;
        3'b010: ALUConf <= aluFunct;
        default: ALUConf <= aluADD;
    endcase

endmodule

////////////////////////////////// clk_40k.v ////////////////////////////////////////////
module clk_40k(
    clk,
    reset,
    clk_40K
);

input          clk;
input          reset;
output         clk_40K;

reg            clk_40K;

parameter     CNT = 16'd1250;

reg            [15:0] count;

always @(posedge clk or posedge reset)
begin
    if(reset) begin
        clk_40K <= 1'b0;
        count <= 16'd0;
    end
    else begin

```

```

        count <= (count==CNT-16'd1) ? 16'd0 : count + 16'd1;
        clk_40K <= (count==16'd0) ? ~clk_40K : clk_40K;
    end
end

endmodule

///////////////////////////////// Controller.v ///////////////////////////////////
`timescale 1ns / 1ps
module Controller(OpCode, Funct,ExtOp,LuiOp,
                  Jump,Branch,RegWrite,ALUSrcA,ALUSrcB,ALUOp,
                  RegDst,MemRead,MemWrite,MemtoReg);
    //Input Signals
    input  [5:0] OpCode;
    input  [5:0] Funct;
    //Output Control Signals
    output reg ExtOp;
    output reg LuiOp;
    output reg [1:0] Jump;
    output reg Branch;
    output reg RegWrite;
    output reg [1:0] ALUSrcA;
    output reg [1:0] ALUSrcB;
    output reg [3:0] ALUOp;
    output reg [1:0] RegDst;
    output reg MemRead;
    output reg MemWrite;
    output reg [1:0] MemtoReg;

    reg [3:0] inst_type;

    always @(*)
    begin
        case(OpCode)
            6'h00: inst_type <= (Funct == 6'h08)? 4'b0001 : //jr

```

```

        (Funct == 6'h09)? 4'b0010 : //jalr
        4'b0000 ; //R-type
6'h02: inst_type <= 4'b0001 ; //j
6'h03: inst_type <= 4'b0010 ; //jal
6'h23: inst_type <= 4'b0011 ; //lw
6'h2b: inst_type <= 4'b0111 ; //sw
6'h0f: inst_type <= 4'b0100 ; //lui
6'h08: inst_type <= 4'b0101 ; //I-type
6'h09: inst_type <= 4'b0101 ; //I-type
6'h0c: inst_type <= 4'b0101 ; //I-type
6'h0a: inst_type <= 4'b0101 ; //I-type
6'h0b: inst_type <= 4'b0101 ; //I-type
6'h04: inst_type <= 4'b0110 ; //beq
6'h05: inst_type <= 4'b0110 ; //bne
6'h06: inst_type <= 4'b1001 ; //blez
6'h07: inst_type <= 4'b1001 ; //bgtz
6'h01: inst_type <= 4'b1011 ; //bltz
default: inst_type <= 4'b1000; // no command/ cancel command
endcase
case(inst_type)
    4'b1000: // no command or cancel command
    begin
        Jump <= 2'b00;
        Branch <= 0;
        RegWrite <= 0;
        MemRead <= 0;
        MemWrite <= 0;
    end
    4'b0000: // R-type
    begin
        Jump <= 2'b00;
        Branch <= 0;
        RegWrite <= 1;
        ALUSrcA <= (Funct==6'h00 || Funct==6'h02 ||

```

```
        Funct==6'h03)? 2'b11 : 2'b10; // 2'b11-Shamt,2'b00-
0,else:A
        ALUSrcB <= 2'b11; // 2'b11-B,2'b01-ImmExtOut,2'b10-
ImmExtShift,2'b00-0
        ALUOp[3] <= OpCode[0];
        ALUOp[2:0] <= 3'b010;
        RegDst <= 2'b00; // 2'b11-rt,2'b00-rd,else:ra
        MemRead <= 0;
        MemWrite <= 0;
        MemtoReg <= 2'b00;
    end
4'b0001: // jr,j
    begin
        Jump <= (OpCode == 6'h02)? 2'b10 : 2'b01;
        Branch <= 0;
        RegWrite <= 0;
        MemRead <= 0;
        MemWrite <= 0;
    end
4'b0010: // jalr,jal
    begin
        Jump <= (OpCode == 6'h03)? 2'b10 : 2'b01;
        Branch <= 0;
        RegWrite <= 1;
        RegDst <= 2'b10; // 2'b11-rt,2'b00-rd,else:ra
        MemRead <= 0;
        MemWrite <= 0;
        MemtoReg <= 2'b01;
    end
4'b0011: // lw
    begin
        Jump <= 2'b00;
        Branch <= 0;
        RegWrite <= 1;
        ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
```

```
        ExtOp <= 1;
        LuiOp <= 0;
        ALUSrcB <= 2'b01; // 2'b11-B,2'b01-ImmExtOut,2'b10-
ImmExtShift,2'b00-0
        ALUOp[3] <= OpCode[0];
        ALUOp[2:0] <= 3'b000;
        RegDst <= 2'b11; // 2'b11-rt,2'b00-rd,else:ra
        MemRead <= 1;
        MemWrite <= 0;
        MemtoReg <= 2'b11;
    end
4'b0100: // lui
    begin
        Jump <= 2'b00;
        Branch <= 0;
        RegWrite <= 1;
        ALUSrcA <= 2'b00; // 2'b11-Shamt,2'b00-0,else:A
        ExtOp <= 0;
        LuiOp <= 1;
        ALUSrcB <= 2'b01; // 2'b11-B,2'b01-ImmExtOut,2'b10-
ImmExtShift,2'b00-0
        ALUOp[3] <= OpCode[0];
        ALUOp[2:0] <= 3'b000;
        RegDst <= 2'b11; // 2'b11-rt,2'b00-rd,else:ra
        MemRead <= 0;
        MemWrite <= 0;
        MemtoReg <= 2'b00;
    end
4'b0101: // I-type
    begin
        Jump <= 2'b00;
        Branch <= 0;
        RegWrite <= 1;
        ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
        LuiOp <= 0;
```

```
    ExtOp <= 1;
    ALUSrcB <= 2'b01; // 2'b11-B,2'b01-ImmExtOut,2'b10-
ImmExtShift,2'b00-0
    ALUOp[3] <= OpCode[0];
    ALUOp[2:0] <= (OpCode == 6'h0c)? 3'b100 :
        (OpCode == 6'h0a || OpCode == 6'h0b)? 3'b101 : 3'b000;
    RegDst <= 2'b11; // 2'b11-rt,2'b00-rd,else:ra
    MemRead <= 0;
    MemWrite <= 0;
    MemtoReg <= 2'b00;
end
4'b0110: // beq,bne
begin
    Jump <= 2'b00;
    Branch <= 1;
    RegWrite <= 0;
    ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
    LuiOp <= 0;
    ExtOp <= 1;
    ALUSrcB <= 2'b11; // 2'b11-B,2'b01-ImmExtOut,2'b10-
ImmExtShift,2'b00-0
    ALUOp[3] <= 0;
    ALUOp[2:0] <= 3'b001;
    MemRead <= 0;
    MemWrite <= 0;
end
4'b1001: //blez,bgtz
begin
    Jump <= 2'b00;
    Branch <= 1;
    RegWrite <= 0;
    ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
    LuiOp <= 0;
    ExtOp <= 1;
```

```
        ALUSrcB <= 2'b11; // 2'b11-B,2'b01-ImmExtOut,2'b10-ImmExtShift,2'b00-0
    ALUOp[3] <= 0;
    ALUOp[2:0] <= 3'b011;
    MemRead <= 0;
    MemWrite <= 0;
end
4'b1011: //bltz
begin
    Jump <= 2'b00;
    Branch <= 1;
    RegWrite <= 0;
    ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
    LuiOp <= 0;
    ExtOp <= 1;
    ALUSrcB <= 2'b11; // 2'b11-B,2'b01-ImmExtOut,2'b10-ImmExtShift,2'b00-0
    ALUOp[3] <= 0;
    ALUOp[2:0] <= 3'b101;
    MemRead <= 0;
    MemWrite <= 0;
end
4'b0111: // sw
begin
    Jump <= 2'b00;
    Branch <= 0;
    RegWrite <= 0;
    ALUSrcA <= 2'b10; // 2'b11-Shamt,2'b00-0,else:A
    ExtOp <= 1;
    LuiOp <= 0;
    ALUSrcB <= 2'b01; // 2'b11-B,2'b01-ImmExtOut,2'b10-ImmExtShift,2'b00-0
    ALUOp[3] <= OpCode[0];
    ALUOp[2:0] <= 3'b000;
    MemRead <= 0;
```

```
        MemWrite <= 1;
    end
    default:begin end
endcase
end

endmodule

//////////////////////////////// DataMemory.v //////////////////////////////////
`timescale 1ns / 1ps
module DataMemory(reset, clk, Address, Write_data,
                  MemRead, MemWrite, Mem_data);
    //Input Clock Signals
    input reset;
    input clk;
    //Input Data Signals
    input [31:0] Address;
    input [31:0] Write_data;
    //Input Control Signals
    input MemRead;
    input MemWrite;
    //Output Data
    output [31:0] Mem_data;

    reg [3:0] AN;
    reg [7:0] CATHODES;

    parameter RAM_SIZE = 512;
    parameter RAM_SIZE_BIT = 8;

    reg [31:0] RAM_data[RAM_SIZE - 1: 0];
    reg [1:0] cnt;
    reg [3:0] figure3,figure2,figure1,figure0;

    //read data
```



```
assign Mem_data = MemRead? RAM_data[Address[RAM_SIZE_BIT +  
1:2]]: 32'h00000000;
```

```
//write data
```

```
integer i;
```

```
always @(posedge reset or posedge clk) begin
```

```
    if (reset) begin
```

```
        RAM_data[0] <= 32'h0000000c;
```

```
        RAM_data[1] <= 32'h0000000d;
```

```
        RAM_data[2] <= 32'h00000002;
```

```
        RAM_data[3] <= 32'h00000001;
```

```
        RAM_data[4] <= 32'h00000003;
```

```
        RAM_data[5] <= 32'h00000003;
```

```
        RAM_data[6] <= 32'h00000004;
```

```
        RAM_data[7] <= 32'h00000005;
```

```
        RAM_data[8] <= 32'h00000007;
```

```
        RAM_data[9] <= 32'h00000009;
```

```
        RAM_data[10] <= 32'h00000002;
```

```
        RAM_data[11] <= 32'h0000000c;
```

```
        RAM_data[12] <= 32'h00000001;
```

```
        RAM_data[13] <= 32'h0000000a;
```

```
        RAM_data[14] <= 32'h00000003;
```

```
        RAM_data[15] <= 32'h00000014;
```

```
        RAM_data[16] <= 32'h00000002;
```

```
        RAM_data[17] <= 32'h0000000f;
```

```
        RAM_data[18] <= 32'h00000001;
```

```
        RAM_data[19] <= 32'h00000008;
```

```
        RAM_data[20] <= 32'h00000004;
```

```
        RAM_data[21] <= 32'h0000001a;
```

```
        RAM_data[22] <= 32'h00000002;
```

```
        RAM_data[23] <= 32'h0000001a;
```

```
        RAM_data[24] <= 32'h00000003;
```

```
        RAM_data[25] <= 32'h0000000f;
```

```
        for (i = 26; i < RAM_SIZE; i = i + 1)
```

```
            RAM_data[i] <= 32'h00000000;
```

```
AN <= 4'b1111;
CATHODES <= 8'b11111111;
cnt <= 0;
end else if (MemWrite) begin
    if(Address == 32'h40000010) begin
        figure3 <= Write_data[15:12];
        figure2 <= Write_data[11:8];
        figure1 <= Write_data[7:4];
        figure0 <= Write_data[3:0];
        case(cnt)
            2'b00:
                begin
                    CATHODES<=(figure3==0)?8'b1100_0000:
                        (figure3==1)?8'b1111_1001:
                        (figure3==2)?8'b1010_0100:
                        (figure3==3)?8'b1011_0000:
                        (figure3==4)?8'b1001_1001:
                        (figure3==5)?8'b1001_0010:
                        (figure3==6)?8'b1000_0010:
                        (figure3==7)?8'b1111_1000:
                        (figure3==8)?8'b1000_0000:
                        (figure3==9)?8'b1001_0000:
                        (figure3==10)?8'b1000_1000://A
                        (figure3==11)?8'b1000_0011://b
                        (figure3==12)?8'b1100_0110://C
                        (figure3==13)?8'b1010_0001://d
                        (figure3==14)?8'b1000_0110://E
                        (figure3==15)?8'b1000_1110://F
                        8'b1111_1111;
                    AN<=4'b0111;
                    cnt <= cnt+1;
                end
            2'b01:
                begin
```

```
CATHODES<=(figure2==0)?8'b1100_0000:
    (figure2==1)?8'b1111_1001:
    (figure2==2)?8'b1010_0100:
    (figure2==3)?8'b1011_0000:
    (figure2==4)?8'b1001_1001:
    (figure2==5)?8'b1001_0010:
    (figure2==6)?8'b1000_0010:
    (figure2==7)?8'b1111_1000:
    (figure2==8)?8'b1000_0000:
    (figure2==9)?8'b1001_0000:
    (figure2==10)?8'b1000_1000://A
    (figure2==11)?8'b1000_0011://b
    (figure2==12)?8'b1100_0110://C
    (figure2==13)?8'b1010_0001://d
    (figure2==14)?8'b1000_0110://E
    (figure2==15)?8'b1000_1110://F
    8'b1111_1111;
AN<=4'b1011;
cnt <= cnt+1;
end
2'b10:
begin
    CATHODES<=(figure1==0)?8'b1100_0000:
        (figure1==1)?8'b1111_1001:
        (figure1==2)?8'b1010_0100:
        (figure1==3)?8'b1011_0000:
        (figure1==4)?8'b1001_1001:
        (figure1==5)?8'b1001_0010:
        (figure1==6)?8'b1000_0010:
        (figure1==7)?8'b1111_1000:
        (figure1==8)?8'b1000_0000:
        (figure1==9)?8'b1001_0000:
        (figure1==10)?8'b1000_1000://A
        (figure1==11)?8'b1000_0011://b
        (figure1==12)?8'b1100_0110://C
```

```
(figure1==13)?8'b1010_0001://d
(figure1==14)?8'b1000_0110://E
(figure1==15)?8'b1000_1110://F
8'b1111_1111;
AN<=4'b1101;
cnt <= cnt+1;
end
2'b11:
begin
CATHODES<=(figure0==0)?8'b1100_0000:
(figure0==1)?8'b1111_1001:
(figure0==2)?8'b1010_0100:
(figure0==3)?8'b1011_0000:
(figure0==4)?8'b1001_1001:
(figure0==5)?8'b1001_0010:
(figure0==6)?8'b1000_0010:
(figure0==7)?8'b1111_1000:
(figure0==8)?8'b1000_0000:
(figure0==9)?8'b1001_0000:
(figure0==10)?8'b1000_1000://A
(figure0==11)?8'b1000_0011://b
(figure0==12)?8'b1100_0110://C
(figure0==13)?8'b1010_0001://d
(figure0==14)?8'b1000_0110://E
(figure0==15)?8'b1000_1110://F
8'b1111_1111;
AN<=4'b1110;
cnt<=0;
end
endcase
end
else begin
RAM_data[Address[RAM_SIZE_BIT + 1:2]] <= Write_data;
end
end
```

end

endmodule

////////////////////////////////// EXMEMReg.v //////////////////////////////////////

`timescale 1ns / 1ps

```
module EXMEMReg(reset, clk, PC_i, ALUOutin, Write_Data_in,
                Write_Register_in, MemReadin, MemWritein,
                MemtoRegin, RegWritein, PC_o, ALUOutout,
                Write_Data_out, Write_Register_out, MemReadout,
                MemWriteout, MemtoRegout, RegWriteout);
```

//Input Clock Signals

input reset;

input clk;

//Input Data

input [31:0] PC\_i;

input [31:0] ALUOutin;

input [31:0] Write\_Data\_in;

input [4:0] Write\_Register\_in;

//Input Control Signals

input MemReadin;

input MemWritein;

input [1:0] MemtoRegin;

input RegWritein;

//Output Data

output reg [31:0] PC\_o;

output reg [31:0] ALUOutout;

output reg [31:0] Write\_Data\_out;

output reg [4:0] Write\_Register\_out;

//Output Control Signals

output reg MemReadout;

output reg MemWriteout;

output reg [1:0] MemtoRegout;

output reg RegWriteout;

```

always@(posedge reset or posedge clk) begin
    if (reset) begin
        RegWriteout <= 0;
        MemReadout <= 0;
        MemWriteout <= 0;
        PC_o <= 32'h00000000;
    end else begin
        PC_o <= PC_i;
        ALUOutout <= ALUOutin;
        Write_Data_out <= Write_Data_in;
        Write_Register_out <= Write_Register_in;
        MemReadout <= MemReadin;
        MemWriteout <= MemWritein;
        MemtoRegout <= MemtoRegin;
        RegWriteout <= RegWritein;
    end
end

endmodule

////////////////////////////////// IDEXReg.v ////////////////////////////////////
`timescale 1ns / 1ps
module IDEXReg(reset, clk, OpCode_i,
PC_i, ReadData1in, ReadData2in, Shamtin, ImmExtOutin, ImmExtShiftin, rsin, rdin, rtin,

Branchin, RegWritein, ALUSrcAin, ALUSrcBin, ALUOpin, Functin,

RegDstin, MemReadin, MemWritein, MemtoRegin, ReadData1out,

ReadData2out, Shamtout, ImmExtOutout, ImmExtShiftout, rsout, rdout, rtout, Branch
out, RegWriteout,

ALUSrcAout, ALUSrcBout, ALUOpout, Functout, RegDstout, MemReadout, Mem
Writeout,

MemtoRegout, PC_o, OpCode_o);

```

```
//Input Clock Signals
input reset;
input clk;
//Input Data
input [5:0] OpCode_i;
input [31:0] PC_i;
input [31:0] ReadData1in;
input [31:0] ReadData2in;
input [4:0] Shamtin;
input [31:0] ImmExtOutin;
input [31:0] ImmExtShiftin;
input [4:0] rsin;
input [4:0] rdin;
input [4:0] rtin;
input [5:0] Functin;
//Input Control Signals
input Branchin;
input RegWritein;
input [1:0] ALUSrcAin;
input [1:0] ALUSrcBin;
input [3:0] ALUOpin;
input [1:0] RegDstin;
input MemReadin;
input MemWritein;
input [1:0] MemtoRegin;
//Output Data
output reg [5:0] OpCode_o;
output reg [31:0] ReadData1out;
output reg [31:0] ReadData2out;
output reg [4:0] Shamtout;
output reg [31:0] ImmExtOutout;
output reg [31:0] ImmExtShiftout;
output reg [4:0] rsout;
output reg [4:0] rdout;
output reg [4:0] rtout;
```

```
output reg [5:0] Functout;
//Output Control Signals
output reg Branchout;
output reg RegWriteout;
output reg [1:0] ALUSrcAout;
output reg [1:0] ALUSrcBout;
output reg [3:0] ALUOpout;
output reg [1:0] RegDstout;
output reg MemReadout;
output reg MemWriteout;
output reg [1:0] MemtoRegout;
output reg [31:0] PC_o;

always@(posedge reset or posedge clk) begin
    if (reset) begin
        Branchout <= 0;
        RegWriteout <= 0;
        MemReadout <= 0;
        MemWriteout <= 0;
        PC_o <= 32'h00000000;
    end else begin
        OpCode_o <= OpCode_i;
        ReadData1out <= ReadData1in;
        ReadData2out <= ReadData2in;
        Shamtout <= Shamtin;
        ImmExtOutout <= ImmExtOutin;
        ImmExtShiftout <= ImmExtShiftin;
        rsout <= rsin;
        rdout <= rdin;
        rtout <= rtin;
        Branchout <= Branchin;
        RegWriteout <= RegWritein;
        ALUSrcAout <= ALUSrcAin;
        ALUSrcBout <= ALUSrcBin;
        ALUOpout <= ALUOpin;
```



```

        RegDstout <= RegDstin;
        MemReadout <= MemReadin;
        MemWriteout <= MemWritein;
        MemtoRegout <= MemtoRegin;
        Functout <= Functin;
        PC_o <= PC_i;
    end
end
endmodule

```

```

////////////////////////////////// IFIDReg.v ////////////////////////////////////

```

```

`timescale 1ns / 1ps

```

```

module IFIDReg(reset, clk, PC_IFIDi, Instruction_IFIDi,
               PC_IFIDo, Instruction_IFIDo);

```

```

    //Input Clock Signals

```

```

    input reset;

```

```

    input clk;

```

```

    //Input Data

```

```

    input [31:0] PC_IFIDi;

```

```

    input [31:0] Instruction_IFIDi;

```

```

    //Output Data

```

```

    output reg [31:0] PC_IFIDo;

```

```

    output reg [31:0] Instruction_IFIDo;

```

```

    always@(posedge reset or posedge clk) begin

```

```

        if (reset) begin

```

```

            PC_IFIDo <= 32'h00000000;

```

```

            Instruction_IFIDo <= {6'h11,24'h000000,2'b00}; // no command

```

```

0x11

```

```

        end else begin

```

```

            PC_IFIDo <= PC_IFIDi;

```

```

            Instruction_IFIDo <= Instruction_IFIDi;

```

```

        end

```

```

    end

```

```

endmodule

```

```
//////////////////////////////// ImmProcess.v //////////////////////////////////
`timescale 1ns / 1ps
module ImmProcess(ExtOp, LuiOp, Immediate, ImmExtOut, ImmExtShift);
    //Input Control Signals
    input ExtOp; //0'-zero extension, '1'-signed extension
    input LuiOp; //for lui instruction
    //Input
    input [15:0] Immediate;
    //Output
    output [31:0] ImmExtOut;
    output [31:0] ImmExtShift;

    wire [31:0] ImmExt;

    assign ImmExt = {ExtOp? {16{Immediate[15]}}: 16'h0000, Immediate};
    assign ImmExtShift = ImmExt << 2;
    assign ImmExtOut = LuiOp? {Immediate, 16'h0000}: ImmExt;

endmodule

//////////////////////////////// InstMemory.v //////////////////////////////////
`timescale 1ns / 1ps
module InstMemory(reset, clk, Address, Mem_data);
    //Input Clock Signals
    input reset;
    input clk;
    //Input Data Signals
    input [31:0] Address;
    //Output Data
    output [31:0] Mem_data;

    parameter RAM_SIZE_BIT = 8;
    parameter RAM_INST_SIZE = 60;
```

```
reg [31:0] RAM_data[RAM_INST_SIZE - 1: 0];

//read data
assign Mem_data = RAM_data[Address[RAM_SIZE_BIT + 1:2]];

//write data
integer i;
always @(posedge reset or posedge clk) begin
    if (reset) begin
        // init instruction memory
        //main:
        //add $a1 $0 $0 0x00002820
        RAM_data[8'd0] <= {6'h00,5'd0,5'd0,5'd5,5'd0,6'h20};
        //lw $s0 4($a1) 0x8cb00004
        RAM_data[8'd1] <= {6'h23,5'd5,5'd16,16'h0004};
        //lw $s1 0($a1) 0x8cb10000
        RAM_data[8'd2] <= {6'h23,5'd5,5'd17,16'h0000};
        //addi $a1 $a1 8 0x20a50008
        RAM_data[8'd3] <= {6'h08,5'd5,5'd5,16'h0008};
        //addi $a0 $s0 0 0x22040000
        RAM_data[8'd4] <= {6'h08,5'd16,5'd4,16'h0000};
        //addi $a2 $s1 0 0x22260000
        RAM_data[8'd5] <= {6'h08,5'd17,5'd6,16'h0000};
        //jal knapsack_dp_loop 0x0c00000b
        RAM_data[8'd6] <= {6'h03,26'd11};

        //beforeloop:
        //拆分 addi $t0 $0 32'h40000010
        //lui $t0 16'h4000
        RAM_data[8'd7] <= {6'h0f,5'd0,5'd8,16'h4000};
        //addi $t0 $t0 16'h0010
        RAM_data[8'd8] <= {6'h08,5'd8,5'd8,16'h0010};
        //loop:
        //sw $v0 0($t0)
```

```
RAM_data[8'd9] <= {6'h2b,5'd8,5'd2,16'h0000};
//beq $0 $0 loop 11->9 -2=16'hfff0 0x1000fff0
RAM_data[8'd10] <= {6'h04,5'd0,5'd0,16'hfff0};

//knapsack_dp_loop:
//addi $sp $sp -12 0x23bdfbf4
RAM_data[8'd11] <= {6'h08,5'd29,5'd29,16'hfff4};
//sw $ra 8($sp) 0xafbf0008
RAM_data[8'd12] <= {6'h2b,5'd29,5'd31,16'h0008};
//sw $s0 4($sp) 0xafb00004
RAM_data[8'd13] <= {6'h2b,5'd29,5'd16,16'h0004};
//sw $s1 0($sp) 0xafb10000
RAM_data[8'd14] <= {6'h2b,5'd29,5'd17,16'h0000};
//addi $s2 $0 26 # $s2 = MAX_CAPACITY + 1 0x20120040
RAM_data[8'd15] <= {6'h08,5'd0,5'd18,16'h0040};
//add $t2 $0 $0 0x00005020
RAM_data[8'd16] <= {6'h00,5'd0,5'd0,5'd10,5'd0,6'h20};
//sll $s2 $s2 2 0x00129080
RAM_data[8'd17] <= {6'h00,5'd0,5'd18,5'd18,5'd2,6'h00};
//sub $sp $sp $s2 0x03b2e822
RAM_data[8'd18] <= {6'h00, 5'd29,5'd18,5'd29,5'd0,6'h22};

//for:
//add $t3 $t2 $sp 0x015d5820
RAM_data[8'd19] <= {6'h00,5'd10,5'd29,5'd11,5'd0,6'h20};
//sw $0 0($t3) 0xad600000
RAM_data[8'd20] <= {6'h2b,5'd11,5'd0,16'h0000};
//addi $t2 $t2 4 0x214a0004
RAM_data[8'd21] <= {6'h08,5'd10,5'd10,16'h0004};
//bne $t2 $s2 for 23-->19 offset=-4=0xffff 0x1552fffc
RAM_data[8'd22] <= {6'h05,5'd10,5'd18,16'hfff0};
//add $t0 $0 $0 0x00004020
RAM_data[8'd23] <= {6'h00,5'd0,5'd0,5'd8,5'd0,6'h20};

//for2:
```

```
//sll $t4 $t0 3    0x000850c0
RAM_data[8'd24] <= {6'h00,5'd0,5'd8,5'd12,5'd3,6'h00};
//add $t1 $t4 $a1    0x01854820
RAM_data[8'd25] <= {6'h00,5'd12,5'd5,5'd9,5'd0,6'h20};
//lw $t2 0($t1)    0x8d2a0000
RAM_data[8'd26] <= {6'h23,5'd9,5'd10,16'h0000};
//addi $t1 $t1 4    0x21290004
RAM_data[8'd27] <= {6'h08,5'd9,5'd9,16'h0004};
//lw $t3 0($t1)    0x8d2b0000
RAM_data[8'd28] <= {6'h23, 5'd9,5'd11,16'h0000};
//addi $t5 $a2 0    0x20cd0000
RAM_data[8'd29] <= {6'h08,5'd6,5'd13,16'h0000};

//for3:
//拆分 blt $t5 $t2 next3    if($t5<$t2) $t5-$t2<0
//sub $s4 $t5 $t2    $t8=$t5-$t2    0x01aa8022
RAM_data[8'd30] <=
{6'h00,5'd13,5'd10,5'd20,5'd0,6'h22};
//bltz $s4 next3 32->43    11    0x0600000b
RAM_data[8'd31] <=
{6'h01,5'd20,5'd0,16'h000b};
//sll $t6 $t5 2    0x000d7080
RAM_data[8'd32] <=
{6'h00,5'd0,5'd13,5'd14,5'd2,6'h00};
//add $t6 $t6 $sp    0x01dd7020
RAM_data[8'd33] <=
{6'h00,5'd14,5'd29,5'd14,5'd0,6'h20};
//lw $t9 0($t6)    0x8dd10000
RAM_data[8'd34] <=
{6'h23,5'd14,5'd25,16'h0000};
//sub $t7 $t5 $t2    0x0116a7822
RAM_data[8'd35] <=
{6'h00,5'd13,5'd10,5'd15,5'd0,6'h22};
//sll $t7 $t7 2    0x000f7880
```

```
RAM_data[8'd36] <=
{6'h00,5'd0,5'd15,5'd15,5'd2,6'h00};
//add $t7 $t7 $sp 0x01fd7820
RAM_data[8'd37] <=
{6'h00,5'd15,5'd29,5'd15,5'd0,6'h20};
//lw $t8 0($t7) 0x8df00000
RAM_data[8'd38] <=
{6'h23,5'd15,5'd24,16'h0000};
//add $t8 $t8 $t3 0x020b8020
RAM_data[8'd39] <=
{6'h00,5'd24,5'd11,5'd24,5'd0,6'h20};
//拆分 bgt $t9 $t8 next3 if($t9>$t8) $t9-$t8>0
//sub $s3 $t9 $t8 0x02309822
RAM_data[8'd40] <=
{6'h00,5'd25,5'd24,5'd19,5'd0,6'h22};
//bgtz $s3 offset 42->43 = 1 = 16'h0001
0x1e600001
RAM_data[8'd41] <=
{6'h07,5'd19,5'd0,16'h0001};
//sw $t8 0($t6) 0xadd00000
RAM_data[8'd42] <=
{6'h2b,5'd14,5'd24,16'h0000};
//next3:
//addi $t5 $t5 -1 0x21adffff
RAM_data[8'd43] <=
{6'h08,5'd13,5'd13,16'hffff};
//拆分 bgez $t5 for3
//sub $s3 $0 $t5 $s3 = -$t5
0x000b9822
RAM_data[8'd44] <=
{6'h00,5'd0,5'd13,5'd19,5'd0,6'h22};
//blez $s3 offset 46->30 -16=0xffff0
0x1a60fff0
RAM_data[8'd45] <=
{6'h06,5'd19,5'd0,16'hfff0};
```

```

        //addi $t0 $t0 1    0x21080001
        RAM_data[8'd46] <= {6'h08,5'd8,5'd8,16'h0001};
        //bne $t0 $a0 for2 48->24 -24=0xffe8    0x1504ffe8
        RAM_data[8'd47] <= {6'h05,5'd8,5'd4,16'hffe8};

//continue:(for)
        //sll $t0 $a2 2    0x00064080
        RAM_data[8'd48] <= {6'h00,5'd0,5'd6,5'd8,5'd2,6'h00};
        //add $t0 $t0 $sp    0x011d4020
        RAM_data[8'd49] <= {6'h00,5'd8,5'd29,5'd8,5'd0,6'h20};
        //lw $v0 0($t0)    0x8d020000
        RAM_data[8'd50] <= {6'h23,5'd8,5'd2,16'h0000};
        //add $sp $sp $s2    0x03b2e820
        RAM_data[8'd51] <= {6'h00,5'd29,5'd18,5'd29,5'd0,6'h20};
        //lw $s1 0($sp)    0x8fd10000
        RAM_data[8'd52] <= {6'h23,5'd29,5'd17,16'h0000};
        //lw $s0 4($sp)    0x8fb00004
        RAM_data[8'd53] <= {6'h23,5'd29,5'd16,16'h0004};
        //lw $ra 8($sp)    0x8fbf0008
        RAM_data[8'd54] <= {6'h23,5'd29,5'd31,16'h0008};
        //addi $sp $sp 12    0x23bd000c
        RAM_data[8'd55] <= {6'h08,5'd29,5'd29,16'h000c};
        //jr $ra    0x03e00008
        RAM_data[8'd56] <= {6'h00,5'd31,15'd0,6'h08};

    end

end

endmodule

//////////////////////////////// MEMWBReg.v //////////////////////////////////
`timescale 1ns / 1ps
module MEMWBReg(reset, clk, PC_i, ALUOutin, Write_Data_in,
                Write_Register_in, MemtoRegin, RegWritein,
                PC_o, ALUOutout, Write_Data_out, Write_Register_out,

```

```
        MemtoRegout,RegWriteout);

//Input Clock Signals
input reset;
input clk;

//Input Data
input [31:0] PC_i;
    input [31:0] ALUOutin;
input [31:0] Write_Data_in;
input [4:0] Write_Register_in;
//Input Control Signals
input [1:0] MemtoRegin;
input RegWritein;

//Output Data
output reg [31:0] PC_o;
output reg [31:0] ALUOutout;
output reg [31:0] Write_Data_out;
output reg [4:0] Write_Register_out;
//Output Control Signals
output reg [1:0] MemtoRegout;
output reg RegWriteout;


always@(posedge reset or posedge clk) begin
    if (reset) begin
        RegWriteout <= 0;
        PC_o <= 32'h00000000;
    end else begin
        PC_o <= PC_i;
        ALUOutout <= ALUOutin;
        Write_Data_out <= Write_Data_in;
        Write_Register_out <= Write_Register_in;
        MemtoRegout <= MemtoRegin;
        RegWriteout <= RegWritein;
    end
end
```



```
endmodule
```

```
//////////////////////////////// myconstraints.xdc //////////////////////////////////
```

```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN T18 [get_ports reset]
set_property PACKAGE_PIN W7 [get_ports {Cathodes[0]}]
set_property PACKAGE_PIN W6 [get_ports {Cathodes[1]}]
set_property PACKAGE_PIN U8 [get_ports {Cathodes[2]}]
set_property PACKAGE_PIN V8 [get_ports {Cathodes[3]}]
set_property PACKAGE_PIN U5 [get_ports {Cathodes[4]}]
set_property PACKAGE_PIN V5 [get_ports {Cathodes[5]}]
set_property PACKAGE_PIN U7 [get_ports {Cathodes[6]}]
set_property PACKAGE_PIN V7 [get_ports {Cathodes[7]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathodes[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

```
//////////////////////////////// PC.v //////////////////////////////////
```

```
`timescale 1ns / 1ps
```

```

module PC(reset, clk, PC_i, PC_o);
    //Input Clock Signals
    input reset;
    input clk;
    //Input PC
    input [31:0] PC_i;
    //Output PC
    output reg [31:0] PC_o;

    always@(posedge reset or posedge clk)
    begin
        if(reset) begin
            PC_o <= 0;
        end else begin
            PC_o <= PC_i;
        end
    end
end

endmodule

//////////////////////////////// PipelineCPU.v //////////////////////////////////
`timescale 1ns / 1ps
//create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports
clk]
module PipelineCPU (reset,clk,an,Cathodes);
    //Input Clock Signals
    input reset;
    input clk;

    //output
    output reg [3:0] an;
    output reg [7:0] Cathodes;

    //forwarding and hazard unit
    wire [1:0] ALUA;
    wire [1:0] ALUB;

```

```
//IF stage
//PC
wire [31:0] IFIDPCAddout;
wire [31:0] PC_o;
wire [31:0] PC_i;
wire [31:0] PC_new;
wire [31:0] PC_Jump_Address;
wire PCSource;
wire Branch;
wire Zero;
wire PCInputMux;
wire [1:0] Jump;
//InstMemory
wire [31:0] Instruction;
```

```
//IFIDReg
wire [31:0] PC_IFIDi;
wire [31:0] PC_IFIDo;
wire [31:0] Instruction_IFIDi;
wire [31:0] Instruction_IFIDo;
wire [1:0] IFIDInputMux;
```

```
//ID stage
//Decode
wire [5:0] OpCode;
wire [4:0] rs;
wire [4:0] rt;
wire [4:0] rd;
wire [4:0] Shamt;
wire [5:0] Funct;
//Controller
```

```
wire ExtOp;
wire LuiOp;
wire RegWrite;
wire MemRead;
wire MemWrite;
wire [1:0] MemtoReg;
wire [1:0] ALUSrcA;
wire [1:0] ALUSrcB;
wire [1:0] RegDst;
wire [3:0] ALUOp;
//ImmProcess
wire [15:0] Immediate;
wire [31:0] ImmExtOut;
wire [31:0] ImmExtShift;
//RegisterFile
wire [4:0] Write_Register;
wire [31:0] Reg_Write_data;
wire [31:0] Read_data1;
wire [31:0] Read_data2;

//IDEXReg
wire IDEXInputMux;
wire IDEXBranchin;
wire IDEXBranchout;
wire IDEXRegWritein;
wire IDEXRegWriteout;
wire IDEXMemReadin;
wire IDEXMemReadout;
wire IDEXMemWritein;
wire IDEXMemWriteout;
wire [1:0] IDEXMemtoRegout;
wire [1:0] IDEXALUSrcAout;
wire [1:0] IDEXALUSrcBout;
wire [1:0] IDEXRegDstout;
```

```
wire [3:0] IDEXALUOpout;
wire [4:0] IDEXShamtout;
wire [4:0] IDEXrsout;
wire [4:0] IDEXrdout;
wire [4:0] IDEXrtout;
wire [5:0] IDEXFunctout;
wire [5:0] IDEXOpCode_o;
wire [31:0] IDEXReadData1out;
wire [31:0] IDEXReadData2out;
wire [31:0] IDEXImmExtOutout;
wire [31:0] IDEXImmExtShiftout;
wire [31:0] PC_IDEXo;

//EX stage
//PCAdder
wire [31:0] IDEXPCAddout;
//ALUControl
wire Sign;
wire [4:0] ALUConf;
//ALU
wire [31:0] DataA;
wire [31:0] DataB;
wire [31:0] In1;
wire [31:0] In2;
wire [31:0] ALUOut;

//EXMEMReg
wire EXMEMMemReadout;
wire EXMEMMemWriteout;
wire [1:0] EXMEMMemtoRegout;
wire EXMEMRegWriteout;
wire [4:0] EXMEMWriteRegisterin;
wire [4:0] EXMEMWriteRegisterout;
```

```
wire [31:0] EXMEMALUOutout;
wire [31:0] PC_EXMEMo;
wire [31:0] EXMEMWrite_Data_out;

//MEM stage
//DataMemory
wire [31:0] DataMemory_data;
//MDR
wire [31:0] MDR_o;

//MEMWBReg
wire MEMWBRegWriteout;
wire [1:0] MEMWBMemtoRegout;
wire [4:0] MEMWBWriteRegisterout;
wire [31:0] PC_MEMWBBo;
wire [31:0] MEMWBWrite_Data_out;
wire [31:0] MEMWBALUOut;

//WB stage
wire [31:0] MEMWBWrite_Data_decided;

wire clk_40k_gen;
reg [3:0] AN;
reg [7:0] CATHODES;

parameter ra = 5'b11111;

clk_40k myclk_40k(clk,reset,clk_40k_gen);

////////// forwarding & hazard unit //////////
//forwarding
assign ALUA = (EXMEMRegWriteout && (EXMEMWriteRegisterout != 0)
&& (EXMEMWriteRegisterout == IDEXrsout) )? 2'b10 :
```

```

        ( MEMWBRegWriteout && (MEMWBWriteRegisterout !=
0) &&( MEMWBWriteRegisterout == IDEXrsout) &&
( EXMEMWriteRegisterout != IDEXrsout || ~EXMEMRegWriteout) )? 2'b01 :
        2'b00;

    assign ALUB = (EXMEMRegWriteout && (EXMEMWriteRegisterout != 0)
&&(EXMEMWriteRegisterout == IDEXrtout) )? 2'b10 :
        ( MEMWBRegWriteout && (MEMWBWriteRegisterout !=
0) &&( MEMWBWriteRegisterout == IDEXrtout) &&
( EXMEMWriteRegisterout != IDEXrtout || ~EXMEMRegWriteout) )? 2'b01 :
        2'b00;

    //hazard
    assign IFIDInputMux = (IDEXMemReadout && ( (IDEXrtout == rs) ||
(IDEXrtout == rt) ) )?2'b11 :
        PCSource? 2'b00 :
        (OpCode == 6'h02 || OpCode == 6'h03 ||
((OpCode == 6'h00) && (Funct == 6'h08 || Funct == 6'h09)))?2'b00 :
        2'b10;

    assign PCInputMux = (IDEXMemReadout && ( (IDEXrtout == rs) ||
(IDEXrtout == rt) ) )? 0 : 1;

    assign IDEXInputMux = (IDEXMemReadout && ( (IDEXrtout == rs) ||
(IDEXrtout == rt) ) )? 0 :
        PCSource? 0 : 1;

    ////////////////////////////////// IF stage //////////////////////////////////
    //PC register
    //PC(reset, clk, PC_i, PC_o);
    assign IFIDPCAddout = PC_o + 4;
    //beq 6'h04, bne 6'h05, blez 6'h06, bgtz 6'h07, bltz 6'h01
    assign PCSource = (IDEXOpCode_o == 6'h04)? IDEXBranchout && Zero :
        (IDEXOpCode_o == 6'h05)? IDEXBranchout &&
(!Zero) :
        (IDEXOpCode_o == 6'h06)? IDEXBranchout &&
(!ALUOut):
        IDEXBranchout && ALUOut;

    // assign PC_i = reset? 0 : (Jump==2'b10)? PC_Jump_Address :

```

```

//          (Jump==2'b01)? Read_data1 :
//          PCSource? IDEXPCAddout : IFIDPCAddout;
assign PC_i = reset? 0 : PCSource? IDEXPCAddout :
          (Jump == 2'b10)? PC_Jump_Address :
          (Jump == 2'b01)? Read_data1 :
          IFIDPCAddout ;

PC myPC(reset,clk_40k_gen,PC_new,PC_o);
assign PC_new = reset? 0 : PCInputMux? PC_i : PC_IFIDo;

//Instruction Memory
//InstMemory(reset, clk, Address, Mem_data);
InstMemory myInstMemory(reset,clk_40k_gen,PC_o,Instruction);
////////////////////////////////////

//IFIDReg(reset, clk, PC_IFIDi, Instruction_IFIDi,
//          PC_IFIDo, Instruction_IFIDo);
assign Instruction_IFIDi = (IFIDInputMux == 2'b00)?
{6'h11,24'h000000,2'b00} : // cancel command in IF if Opcode i 0x11(I defined this)
          (IFIDInputMux == 2'b11)?

Instruction_IFIDo :

          Instruction;

assign PC_IFIDi = (IFIDInputMux == 2'b11)? PC_IFIDo : IFIDPCAddout;
IFIDReg myIFIDReg(reset,clk_40k_gen,PC_IFIDi,
          Instruction_IFIDi,PC_IFIDo,Instruction_IFIDo);

////////////////////////////////////

//////////////////////////////////// ID stage //////////////////////////////////////

//Decode
assign OpCode = Instruction_IFIDo[31:26];
assign rs = Instruction_IFIDo[25:21];
assign rt = Instruction_IFIDo[20:16];
assign rd = Instruction_IFIDo[15:11];
assign Shamt = Instruction_IFIDo[10:6];
assign Funct = Instruction_IFIDo[5:0];

```



```

//Controller
//Controller(OpCode, Funct,ExtOp,LuiOp,
//           Jump,Branch,RegWrite,ALUSrcA,ALUSrcB,ALUOp,
//           RegDst,MemRead,MemWrite,MemtoReg);
Controller myController(Instruction_IFIDo[31:26],
                        Instruction_IFIDo[5:0],ExtOp,LuiOp,Jump,
Branch,RegWrite,ALUSrcA,ALUSrcB,ALUOp,
                        RegDst,MemRead,MemWrite,MemtoReg);

//ImmExt, ImmExtShift
//ImmProcess(ExtOp, LuiOp, Immediate, ImmExtOut, ImmExtShift);
assign Immediate = Instruction_IFIDo[15:0];
ImmProcess
myImmProcess(ExtOp,LuiOp,Immediate,ImmExtOut,ImmExtShift);

//RegisterFile
//RegisterFile(reset, clk, RegWrite, Read_register1, Read_register2,
//           Write_Register, Write_data, Read_data1, Read_data2);
assign Write_Register = MEMWBWriteRegisterout;
RegisterFile
myRegisterFile(reset,clk_40k_gen,MEMWBRegWriteout,rs,rt,MEMWBWriteRegisterout,
MEMWBWrite_Data_decided,Read_data1,Read_data2);
//Jump address
assign PC_Jump_Address =
{PC_IFIDo[31:28],Instruction_IFIDo[25:0],2'b00};

/////////////////////////////////////////////////////////////////

//IDEXReg
//IDEXReg(reset, clk,
OpCode_i,PC_i,ReadData1in,ReadData2in,Shamtin,ImmExtOutin,

```

```

//          ImmExtShiftin,rsin,rdin,rtin,Branchin,RegWritein,
//
ALUSrcAin,ALUSrcBin,ALUOpin,Funcin,RegDstin,MemReadin,
//
MemWritein,MemtoRegin,ReadData1out,ReadData2out,Shamtout,
//          ImmExtOutout,ImmExtShiftout,rsout,rdout,rtout,Branchout,
//
RegWriteout,ALUSrcAout,ALUSrcBout,ALUOpout,Funcout,RegDstout,
//          MemReadout,MemWriteout,MemtoRegout,PC_o,OpCode_o);
assign IDEXBranchin = IDEXInputMux? Branch : 0;
assign IDEXRegWritein = IDEXInputMux? RegWrite : 0;
assign IDEXMemReadin = IDEXInputMux? MemRead : 0;
assign IDEXMemWritein = IDEXInputMux? MemWrite : 0;
IDEXReg
myIDEXReg(reset,clk_40k_gen,OpCode,PC_IFIDo,Read_data1,Read_data2,Shamt,I
mmExtOut,
          ImmExtShift,rs,rd,rt,IDEXBranchin,IDEXRegWritein,
          ALUSrcA,ALUSrcB,ALUOp,Func,RegDst,IDEXMemReadin,
IDEXMemWritein,MemtoReg,IDEXReadData1out,IDEXReadData2out,
IDEXShamtout,IDEXImmExtOutout,IDEXImmExtShiftout,IDEXrsout,IDEXrdout,
          IDEXrtout,IDEXBranchout,IDEXRegWriteout,
IDEXALUSrcAout,IDEXALUSrcBout,IDEXALUOpout,IDEXFuncout,
IDEXRegDstout,IDEXMemReadout,IDEXMemWriteout,IDEXMemtoRegout,PC_ID
EXo,IDEXOpCode_o);

////////// EX stage//////////

//ALUControl
//ALUControl(ALUOp, Func, ALUConf, Sign);
ALUControl
myALUControl(IDEXALUOpout,IDEXFuncout,ALUConf,Sign);

```

```

//ALU
//ALU(ALUConf, Sign, In1, In2, Zero, Result);
assign DataA = (ALUA == 2'b00)? IDEXReadData1out :
                (ALUA == 2'b10)? EXMEMALUOutout :
                (MEMWBMemtoRegout == 2'b11)?
MEMWBWrite_Data_out :
                (MEMWBMemtoRegout == 2'b00)? MEMWBALUOut :
0;

assign DataB = (ALUB == 2'b00)? IDEXReadData2out :
                (ALUB == 2'b10)? EXMEMALUOutout :
                (MEMWBMemtoRegout == 2'b11)?
MEMWBWrite_Data_out :
                (MEMWBMemtoRegout == 2'b00)?
MEMWBALUOut : 0;

assign In1 = (IDEXALUSrcAout == 2'b00)? 0 :
              (IDEXALUSrcAout == 2'b11)? IDEXShamtout : DataA;
assign In2 = (IDEXALUSrcBout == 2'b00)? 0 :
              (IDEXALUSrcBout == 2'b11)? DataB :
              (IDEXALUSrcBout == 2'b10)? IDEXImmExtShiftout :
IDEXImmExtOutout;

ALU myALU(ALUConf,Sign,In1,In2,Zero,ALUOut);

//choose reg destination,2'b00-rs,2'b01-rt,2'b10-ra
assign EXMEMWriteRegisterin = (IDEXRegDstout == 2'b00)? IDEXrdout :
                               (IDEXRegDstout == 2'b11)?
IDEXrtout :ra;

//PCAdder
assign IDEXPCAddout = PC_IDEXo + IDEXImmExtShiftout;

////////////////////////////////////
//EXMEMReg
//EXMEMReg(reset, clk,PC_i,ALUOutin,Write_Data_in,
//          Write_Register_in,MemReadin,MemWritein,

```

```

//      MemtoRegin,RegWritein,PC_o,ALUOutout,
//      Write_Data_out,Write_Register_out,MemReadout,
//      MemWriteout,MemtoRegout,RegWriteout);
    EXMEMReg
myEXMEMReg(reset,clk_40k_gen,PC_IDEXo,ALUOut,IDEXReadData2out,EXME
MWriteRegisterin,

IDEXMemReadout,IDEXMemWriteout,IDEXMemtoRegout,IDEXRegWriteout,

    PC_EXMEMo,EXMEMALUOutout,EXMEMWrite_Data_out,EXMEMWriteRe
gisterout,

    EXMEMMemReadout,EXMEMMemWriteout,EXMEMMemtoRegout,EXMEM
RegWriteout);

////////// MEM stage //////////

//DataMemory
//DataMemory(reset, clk, Address, Write_data, MemRead, MemWrite,
Mem_data);
    DataMemory
myDataMemory(reset,clk_40k_gen,EXMEMALUOutout,EXMEMWrite_Data_out,

EXMEMMemReadout,EXMEMMemWriteout,DataMemory_data);

//Memory data register
//RegTemp(reset, clk, Data_i, Data_o);
RegTemp MDR(reset,clk_40k_gen,DataMemory_data,MDR_o);

//////////

//MEMWBReg
//MEMWBReg(reset, clk,PC_i,ALUOutin,Write_Data_in,
//      Write_Register_in,MemtoRegin,RegWritein,
//      PC_o,ALUOutout,Write_Data_out,Write_Register_out,

```

```

//          MemtoRegout,RegWriteout);
MEMWBReg
myMEMWBReg(reset,clk_40k_gen,PC_EXMEMo,EXMEMALUOutout,DataMemor
y_data,

EXMEMWriteRegisterout,EXMEMMemtoRegout,EXMEMRegWriteout,

PC_MEMWBo,MEMWBALUOut,MEMWBWrite_Data_out,

MEMWBWriteRegisterout,MEMWBMemtoRegout,MEMWBRegWriteout);

////////// WB stage //////////
assign MEMWBWrite_Data_decided = (MEMWBMemtoRegout == 2'b11)?
MEMWBWrite_Data_out :

(MEMWBMemtoRegout ==

2'b00)? MEMWBALUOut :

PC_MEMWBo;
always @(posedge reset or posedge clk_40k_gen)
begin
  if(reset)begin
    an <= 4'b1111;
    Cathodes <= 8'b11111111;
  end
  else
  begin
    an <= myDataMemory.AN;
    Cathodes <= myDataMemory.CATHODES;
  end
end
endmodule

////////// RegisterFile.v //////////
`timescale 1ns / 1ps
module RegisterFile(reset, clk, RegWrite, Read_register1, Read_register2,
Write_Register, Write_data, Read_data1, Read_data2);

```

```
//Input Clock Signals
input reset;
input clk;
//Input Control Signals
input RegWrite;
//Input Data Signals
input [4:0] Read_register1;
input [4:0] Read_register2;
input [4:0] Write_Register;
input [31:0] Write_data;
//Output Data Signals
output [31:0] Read_data1;
output [31:0] Read_data2;

reg [31:0] RF_data[31:1];

//read data
assign Read_data1 = (RegWrite && (Read_register1 == Write_Register))?
Write_data :
                                (Read_register1 == 5'b000000)? 32'h00000000:
RF_data[Read_register1];
assign Read_data2 = (RegWrite && (Read_register2 == Write_Register))?
Write_data :
                                (Read_register2 == 5'b000000)? 32'h00000000:
RF_data[Read_register2];

integer i;
always @(posedge reset or posedge clk) begin
    if (reset) begin
        for (i = 1; i < 32; i = i + 1) begin
            RF_data[i] <= 32'h00000000;
        end
    end else if (RegWrite && (Write_Register != 5'b000000)) begin
        RF_data[Write_Register] <= Write_data;
    end
end
```

```
end
```

```
endmodule
```

```
//////////////////////////////// RegTemp.v //////////////////////////////////
```

```
`timescale 1ns / 1ps
```

```
module RegTemp(reset, clk, Data_i, Data_o);
```

```
    //Input Clock Signals
```

```
    input reset;
```

```
    input clk;
```

```
    //Input Data
```

```
    input [31:0] Data_i;
```

```
    //Output Data
```

```
    output reg [31:0] Data_o;
```

```
    always@(posedge reset or posedge clk) begin
```

```
        if (reset) begin
```

```
            Data_o <= 32'h00000000;
```

```
        end else begin
```

```
            Data_o <= Data_i;
```

```
        end
```

```
    end
```

```
endmodule
```

```
//////////////////////////////// test_cpu.v //////////////////////////////////
```

```
`timescale 1ns / 1ps
```

```
module test_cpu();
```

```
    reg reset;
```

```
    reg clk;
```

```
    wire [3:0] an;
```

```
    wire [7:0] Cathodes;
```

```
    PipelineCPU PipelineCPU_1(reset, clk,an,Cathodes);
```

```

initial begin
    reset = 1;
    clk = 1;
    #100 reset = 0;
end

always #50 clk = ~clk;

```

```
endmodule
```

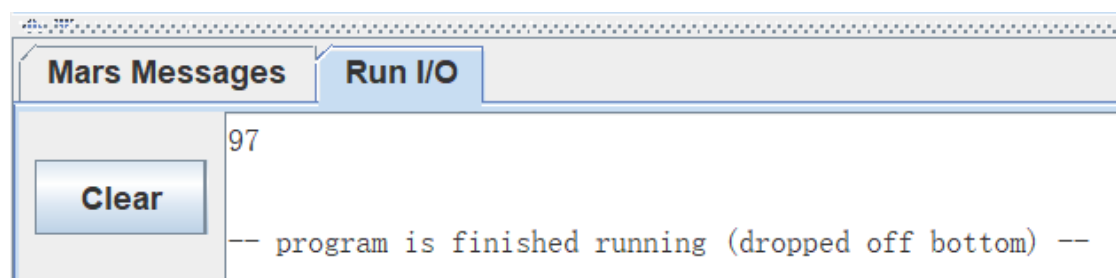
#### 四、仿真结果及分析

测试文件：

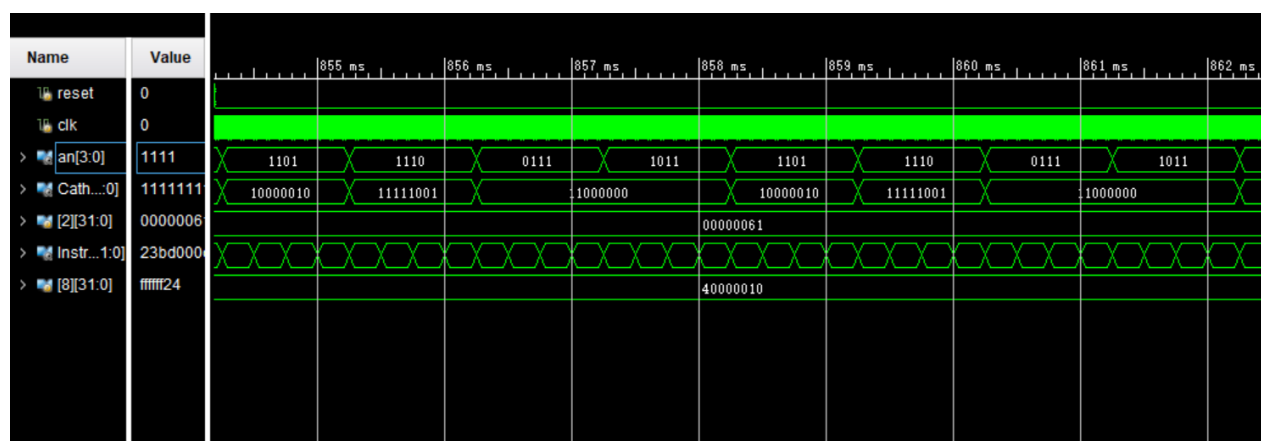
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000	0c	00	0d	00	02	00	01	00	03	00	03	00	04	00	05	00
00000010	07	00	09	00	02	00	0c	00	01	00	0a	00	03	00	14	00
00000020	02	00	0f	00	01	00	08	00	04	00	1a	00	02	00	1a	00
00000030	03	00	0f	00												

0x0c 代表背包承重容量为 12, 0x0d 代表一共有 13 个物品，之后每两个非零值代表一个物品的属性，比如以 0x02 和 0x01 为例，前者代表该物品重量为 2，后者代表该物品价值为 1。

利用 MARS 运行汇编代码得到结果：



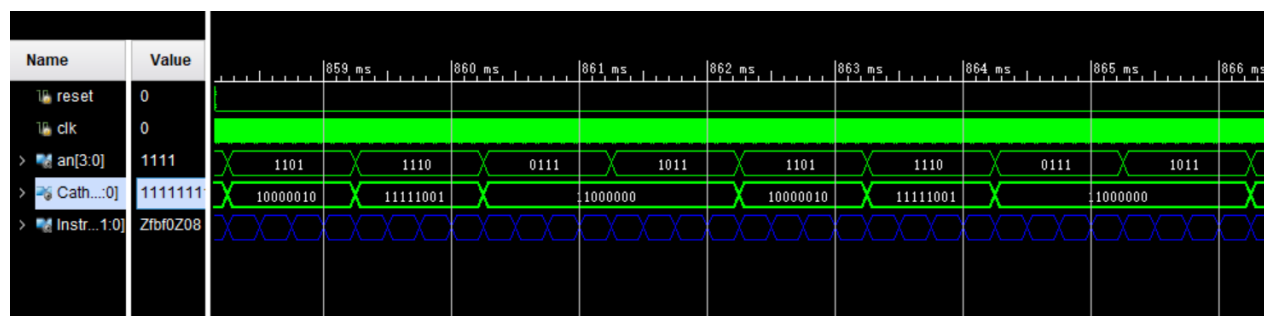
前仿：





2 号寄存器最终值为 0x00000061，换算为十进制： $6*16+1=97$ ，结果正确。七段数码管控制信号 an 每 1ms 变化一次，在 0111→1011→1101→1110→0111 之间循环，0111 时的 Cathodes 信号为 11000000 即最高位的 0，1011 时 Cathodes 信号为 11000000 即第二位 0，1101 时 Cathodes 信号为 10000010 即第三位 6，1110 时 Cathodes 信号为 11111001 即最低位 1。因而在数码管上应显示 0061 即该背包问题解的十六进制值的最低 4 位。正确！

后仿：



后仿 Instruction 信号一直为高阻状态，分析可能是 implementation 中存在信号合并，该信号与其他信号功能相同被合并优化。观察 an 和 Cathodes 的变化规律，与前仿时相同，故而功能正确！

## 五、综合情况

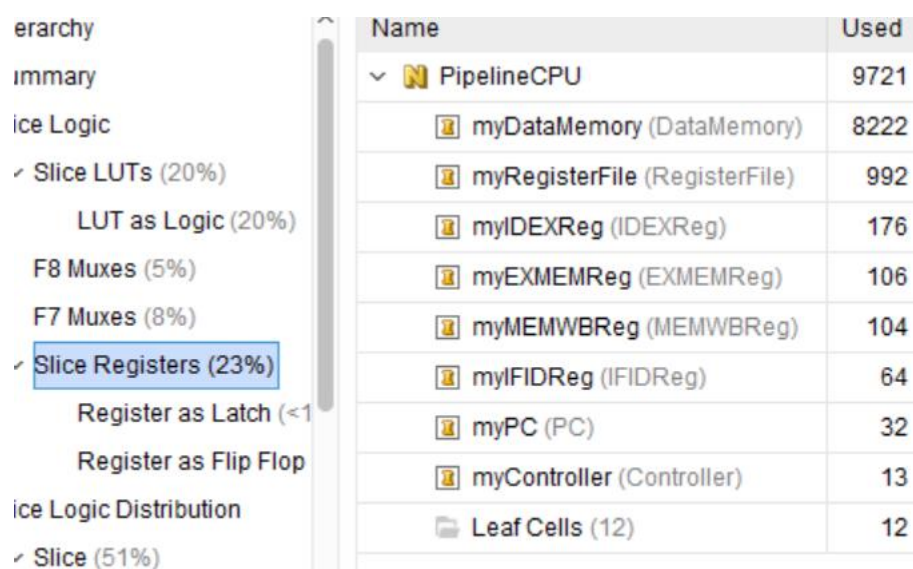
### 1. 面积情况

erarchy	Name	Used
immary	▼ PipelineCPU	4197
ice Logic	myDataMemory (DataMemory)	2677
✓ Slice LUTs (20%)	myRegisterFile (RegisterFile)	611
LUT as Logic (20%)	myALU (ALU)	502
F8 Muxes (5%)	Leaf Cells (312)	312
F7 Muxes (8%)	myController (Controller)	36
✓ Slice Registers (23%)	myInstMemory (InstMemory)	29
Register as Latch (<1	myImmProcess (ImmProcess)	17
Register as Flip Flop	myALUControl (ALUControl)	10
ice Logic Distribution	myEXMEMReg (EXMEMReg)	1
✓ Slice (51%)	myIDEXReg (IDEXReg)	1
SLICEM	myMEMWBReg (MEMWBReg)	1
SLICEL		

组合逻辑资源占用情况：总共使用了 4197 个查找表，其中数据存储器 myDataMemory 占比最高，使用了 2677 个查找表，第二是寄存器堆 myRegisterFile，使用了 611 个查找表，第三是计算单元模块 myALU，使用了

502 个查找表，其次是一些 leaf cells，使用了 312 个查找表，其他模块使用的查找表与前面相比很少，在此不赘述。

上述的这些组合逻辑资源使用情况与我的预期相一致，因为根据我的汇编代码（见 InstMemory.v 中的指令），cpu 将进行自底向上的动态规划，不断地开辟栈空间，存储、改写、读取数据，故而数据存储器使用的组合逻辑资源最多，达到总组合逻辑资源使用的一半以上也是意料之中的，其次寄存器堆占比第二是因为数据的运算都是通过将数据搬移到寄存器中，对寄存器中的值进行运算之后写回数据存储器的，由之前所说，本设计在解决背包问题时涉及大量的数据改写，故而寄存器堆使用组合逻辑资源第二也是意料之中的。ALU 计算单元模块占比第三是因为本 cpu 支持的指令类型较多，运算种类较多，而且 ALU 的输入数据存在转发，故而存在输入数据的多路选择，因而占用组合逻辑资源第三。



The screenshot shows the logic hierarchy on the left and a table of resource usage on the right. The hierarchy includes: erarchy, immary, ice Logic, Slice LUTs (20%), LUT as Logic (20%), F8 Muxes (5%), F7 Muxes (8%), Slice Registers (23%), Register as Latch (<1), Register as Flip Flop, ice Logic Distribution, and Slice (51%). The table lists the following resources and their usage counts:

Name	Used
▼ PipelineCPU	9721
myDataMemory (DataMemory)	8222
myRegisterFile (RegisterFile)	992
myIDEXReg (IDEXReg)	176
myEXMEMReg (EXMEMReg)	106
myMEMWBReg (MEMWBReg)	104
myIFIDReg (IFIDReg)	64
myPC (PC)	32
myController (Controller)	13
Leaf Cells (12)	12

时序逻辑寄存器资源占用情况：总共使用了 9721 个寄存器，其中数据存储器 myDataMemory 使用了 8222 个，占比最高，也是寄存器资源占用的主要原因，占比第二的是寄存器堆，但只使用了 992 个，远小于数据存储器所使用的，之后 4 个级间寄存器，IDEXReg, EXMEMReg, MEMWBReg, IFIDReg 使用的寄存器递减，其他模块使用寄存器较少，在此不赘述。

本设计中，存储主要依靠寄存器实现，由于本设计在解决背包问题时进行了大量数据的存储、读取、改写，故而数据存储器使用的寄存器占比最高也是意料之中的。除此之外存储数据量最大的就是寄存器堆了，这是由于对数据的改写操作都是通过寄存器进行运算后完成的，故而进行大量数据的改写也意味着使用了较多的寄存器，故而寄存器堆占比第二。之后四个级间寄存器，由于控制信号在 ID 阶段产生，故而 IFIDReg 中仅存储少量数据，不存储控制信号，故而存储量最小，使用的寄存器最少，至于 IDEXReg, EXMEMReg,

MEMWBReg，由于随着流水线阶段的不断推进，部分控制信号使用完毕后不再需要向后传递，故而这三个级间寄存器的存储量逐渐递减，故而使用的寄存器数量也逐渐递减。

## 2.时序性能

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -2.574 ns	Worst Hold Slack (WHS): 0.024 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): -294.578 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 268	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 18909	Total Number of Endpoints: 18909	Total Number of Endpoints: 9709

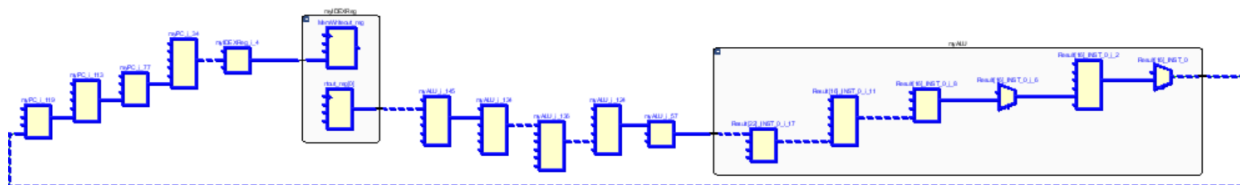
Timing constraints are not met.

添加的时钟约束为 `create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports clk]` 亦即 100MHz 的时钟，由 timing 结果可看到 hold time 和 pulse width time 满足了要求，而 setup time slack 为负值，说明本设计达不到 100MHz 时钟情况下的 setup time 的要求，计算最高工作频率：

$$T_{min} = 10n - 0.024n + 2.574n = 12.55ns$$

$$f_{max} = 1/T_{min} = 79.68MHz。$$

关键路径：

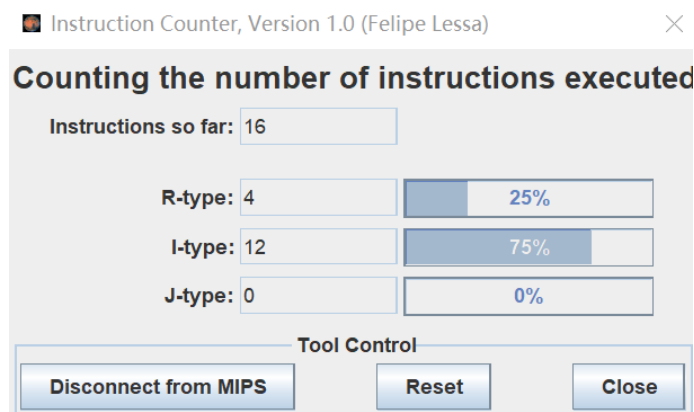


Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay
Path 1	-2.574	16	14	68	myIDEXReg/rtout_reg[0]/C	myIDEXReg/MemWriteout_reg/D	12.558	3.497

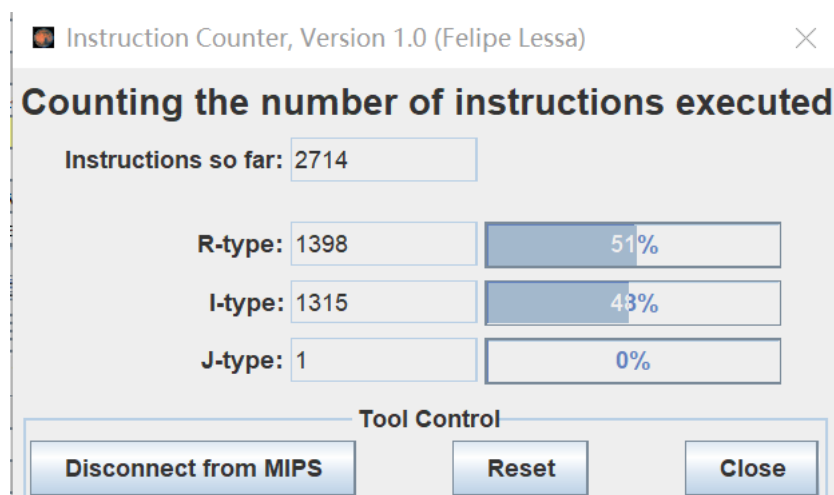
IDEXReg 中存储的数据经时钟上升沿进入 EX 阶段，最终反作用于 IDEXReg，故而推测是分支跳转指令（beq 等）先在 EX 阶段进行判断，若判断跳转则 flush IDEXReg，该路径为关键路径。

使用 MARS 确定完成算法的指令总数：

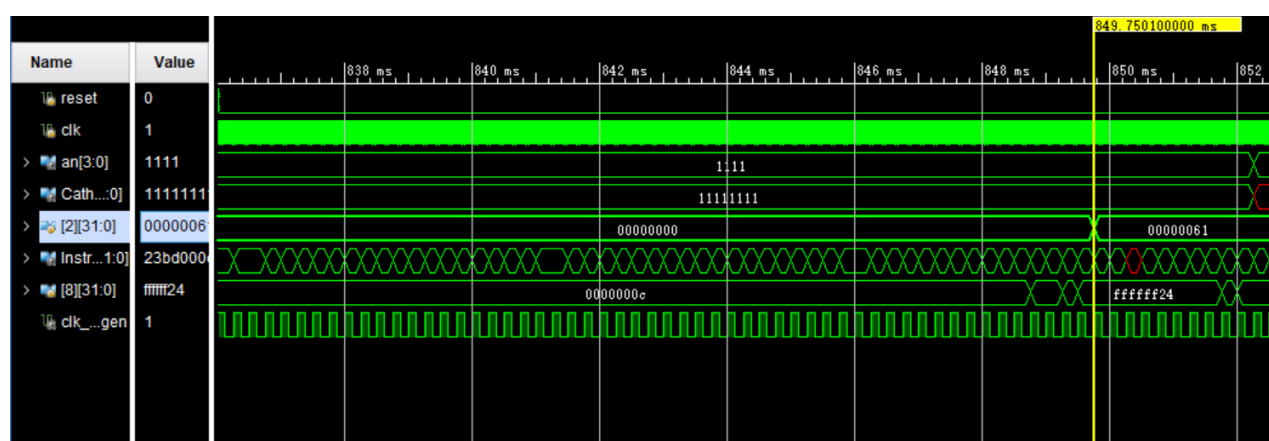
syscall 等原代码中读文件部分使用的指令数：



总指令数:



故而本代码的指令总数为  $2714 - 16 = 2698$



本代码在 849.75ms 时完成对背包问题的求解，由图中可以看出，每 2ms 包含了 8 个 clk\_gen 周期，故而总共时钟周期为 3399 个。

$$CPI = 3399 / 2698 = 1.26$$

上面计算出最高时钟频率为 79.68MHz，我们按 75MHz 进行计算，则时钟周期为 13.33ns，每秒时钟周期数目为  $1 \div (13.33 \times 10^{-9}) = 75 \times 10^6$ ，CPI=1.26 则每秒执行指令数目为  $75 \times 10^6 / 1.26 = 59.52 \times 10^6$

这个数值并不非常理想，经过分析，我发现可以通过调换部分指令的顺序（在不影响正确执行的情况下）来减少 lw 数据关联冒险产生的 stall。

优化前的代码：

```

//add $t6 $t6 $sp 0x01dd7020
RAM_data[8'd33] <= {6'h00,5'd14,5'd29,5'd14,5'd0,6'h20};
//lw $t9 0($t6) 0x8dd10000
RAM_data[8'd34] <= {6'h23,5'd14,5'd25,16'h0000};
//sub $t7 $t5 $t2 0x0116a7822
RAM_data[8'd35] <= {6'h00,5'd13,5'd10,5'd15,5'd0,6'h22};
//sll $t7 $t7 2 0x000f7880
RAM_data[8'd36] <= {6'h00,5'd0,5'd15,5'd15,5'd2,6'h00};
//add $t7 $t7 $sp 0x01fd7820
RAM_data[8'd37] <= {6'h00,5'd15,5'd29,5'd15,5'd0,6'h20};
//lw $t8 0($t7) 0x8df00000
RAM_data[8'd38] <= {6'h23,5'd15,5'd24,16'h0000};
//add $t8 $t8 $t3 0x020b8020
RAM_data[8'd39] <= {6'h00,5'd24,5'd11,5'd24,5'd0,6'h20};
//拆分bgt $t9 $t8 next3 if($t9>$t8) $t9-$t8>0
//sub $s3 $t9 $t8 0x02309822
RAM_data[8'd40] <= {6'h00,5'd25,5'd24,5'd19,5'd0,6'h22};
//bgtz $s3 offset 42->43 = 1 = 16'h0001 0x1e600001
RAM_data[8'd41] <= {6'h07,5'd19,5'd0,16'h0001};

```

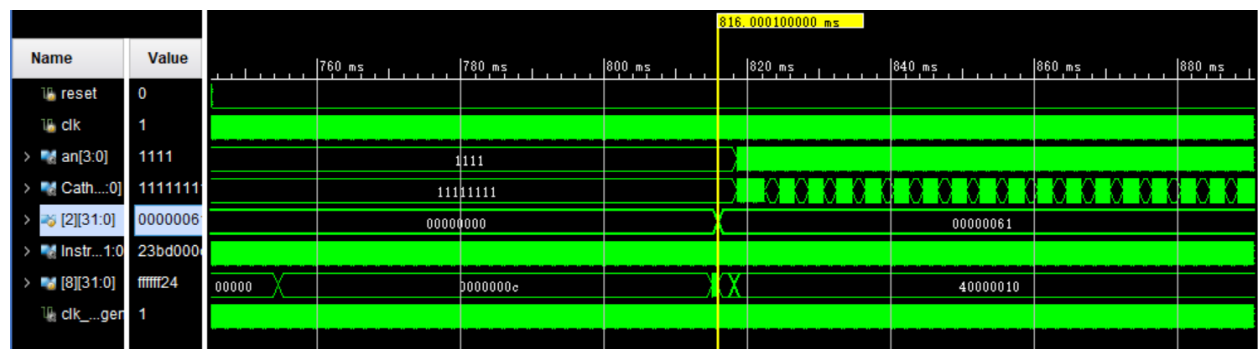
优化后的代码:

```

//add $t6 $t6 $sp 0x01dd7020
RAM_data[8'd33] <= {6'h00,5'd14,5'd29,5'd14,5'd0,6'h20};
//sub $t7 $t5 $t2 0x0116a7822
RAM_data[8'd34] <= {6'h00,5'd13,5'd10,5'd15,5'd0,6'h22};
//sll $t7 $t7 2 0x000f7880
RAM_data[8'd35] <= {6'h00,5'd0,5'd15,5'd15,5'd2,6'h00};
//add $t7 $t7 $sp 0x01fd7820
RAM_data[8'd36] <= {6'h00,5'd15,5'd29,5'd15,5'd0,6'h20};
//lw $t8 0($t7) 0x8df00000
RAM_data[8'd37] <= {6'h23,5'd15,5'd24,16'h0000};
//lw $t9 0($t6) 0x8dd10000
RAM_data[8'd38] <= {6'h23,5'd14,5'd25,16'h0000};
//add $t8 $t8 $t3 0x020b8020
RAM_data[8'd39] <= {6'h00,5'd24,5'd11,5'd24,5'd0,6'h20};
//拆分bgt $t9 $t8 next3 if($t9>$t8) $t9-$t8>0
//sub $s3 $t9 $t8 0x02309822
RAM_data[8'd40] <= {6'h00,5'd25,5'd24,5'd19,5'd0,6'h22};
//bgtz $s3 offset 42->43 = 1 = 16'h0001 0x1e600001
RAM_data[8'd41] <= {6'h07,5'd19,5'd0,16'h0001};

```

优化后总指令执行数不变，仍为 2698 条，但时钟周期有所减少:



同优化前 2ms 包含 8 个时钟周期，则一共使用了  $816 \times 4 = 3264$  个时钟周期，优于优化前的 3399 个时钟周期。

$$CPI = 3264 / 2698 = 1.21$$

上面计算出最高时钟频率为 79.68MHz，我们按 75MHz 进行计算，则时钟周期为 13.33ns，每秒时钟周期数目为  $1 \div (13.33 \times 10^{-9}) = 75 \times 10^6$ ，CPI=1.21 则每秒执行指令数目为  $75 \times 10^6 / 1.21 = 61.98 \times 10^6$

## 六、硬件调试情况

本设计于 7 月 9 日下午验收。

由于在烧录到板子上之前做了充分的仿真工作，故而烧录到板子上一次即成功，并没有出现问题，故而本部分主要分析在仿真过程中遇到的问题。

(1) 设置存储器时需要事先预估需要的存储空间来设置存储器大小，否则可能出现存储大小不够的情况。我在编写时沿用了春季学期多周期处理器的代码，然而当时设置的存储器中存储指令的部分只有 32 个字节，对于当时的指令来说绰绰有余，但本次设计的背包问题解决方案有 50 余条指令，该存储空间不够。一开始未意识到这个问题，发现运行到一半多之后取出的指令全是不定态 X，仔细分析发现是存储器不够导致后续指令并没有存入指令存储器造成的。

(2) 寄存器“先写后读”的编写。由于数逻理论课上一开始就讲到了通过硬件设计使寄存器支持先写后读，故而在第一次编写时我直接默认了这个设定，导致程序出错，后来通过在寄存器内部添加多路选择器，判断当读取的 rs 或 rt 寄存器与将写入的 rd 寄存器相同时直接取用即将写入 rd 的值。

(3) 细节问题，诸如 beq 的 offset 计算错误（少加了 1），lw 偏移量 4 写为 0 等，在此不赘述。

总体而言，本次实验代码编写较为顺利。

## 七、思想体会

首先，通过本次实验，我更加深刻地理解了理论课上讲解的流水线的数据通路的设计，包括其五个阶段和四个级间寄存器，其中在设计四个级间寄存器时更深入地理解了信号和数据一样随着流水线流动的意思。

其次，本次实验也使我更熟练地掌握了流水线中存在的三类冒险，即结构冒险、数据关联冒险和控制冒险的处理方法，比如寄存器同时读写存在的结构冒险可以通过在寄存器内部添加多路选择器使其支持先写后读的功能，相较于多周期处理器添加了 PC 的 Adder 以解决在 EX 阶段同时需要对寄存器和 PC 进行计算的结构冒险；数据关联冒险方面主要是 lw 指令引起的数据关联冒险，采取的解决方法是利用 hazard unit 通过 OpCode 判断是否为 lw 指令，若是则比较下一条指令的 rs 或 rt 是否为 lw 指令的 rt，若是则 stall 一个周期（通过 flush，



令 IFIDReg 所有控制信号无效), 若产生数据关联冒险的指令并非 lw 紧接着的下一条指令则可以通过完全的 forwarding unit 将 lw 指令计算出即将写入的数据旁路转发至存在冒险的指令的 ALU 的输入数据处; 控制冒险主要是分支指令 (beq 等) 和跳转指令 (j 等), 主要通过提前判断 (分支指令在 EX 阶段判断, 跳转指令在 ID 阶段判断) 来解决, 若判断发生跳转则取消流水线上该指令后的指令 (分支指令跳转则取消 IF、ID 阶段的指令, 跳转指令跳转则取消 IF 阶段的指令)。

第三, 在优化 CPI 的过程中, 我更加深刻地理解了在保证代码执行结果正确的情况下, 通过调整 lw 指令的顺序可以减少 stall 的周期从而达到降低 CPI 的效果, 也意识到掌握流水线 cpu 执行时可能产生的冒险的情况下, 编写指令时应多加注意这点, 尽可能让流水线 cpu 高效。

第四, 我充分认识到了仿真工作的重要性。春季学期的前几次实验因为比较简单, 所以即使仿真并不充分, 将代码烧录到板子上也较容易调试, 然而像多周期处理器、流水线处理器, 由于其结构复杂, 可能出问题的地方比较多, 而且通常是内部某些信号出了问题, 如果只看板子的最终结果不对是很难找到问题所在。而仿真可以查看任何信号的波形, 可以清楚地看到信号波形的变化, 非常有利于 debug, 虽然仿真结果正确后上板子未必百分百正确, 但会很大程度上地提高上板子的正确率。

第五, 我深刻地意识到细节决定成败的意义。从我写的硬件调试情况部分可以看到, 本次实验我并没有遇到相比较起来更有技术含量的算法错误或者逻辑错误等, 遇到的 bug 全是比较低级的错误, 如果我在写完代码之后不急于做仿真, 而是再检查一遍, 就很有可能可以避免这些低级错误。虽然是低级错误, 然而在逆向找 bug 的时候还是花费了不少时间, 所以我深刻地意识到了注意细节的重要性, 也会在今后的生活中注重这一点。

总体而言, 本次实验使我获益匪浅。感谢老师和助教的辛苦付出!