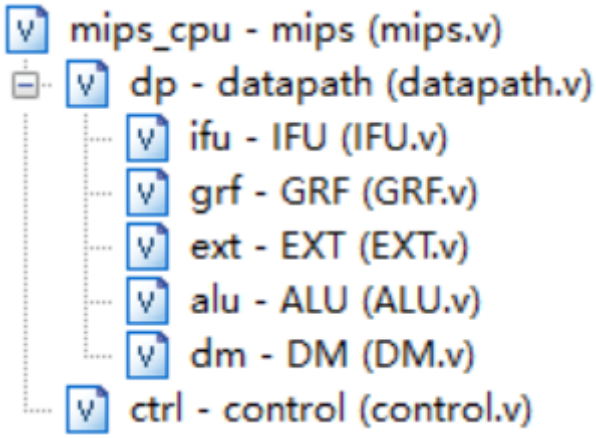
Verilog单周期CPU设计

一、CPU设计方案综述

（一）总体设计概述

本CPU为Verilog实现的32位单周期MIPS-CPU，支持指令集包含`{addu, subu, ori, lw, lh, lb, sw, sh, sb, beq, jal, jr, lui, nop, j, sll, slt}`。为实现相关功能，CPU主要包含IFU(PC+NPC+IM)、GRF、EXT、ALU、DM、CTRL模块。遵循形式化建模综合方法完成设计与实现。

各模块组件解构树状图如下：



（二）数据通路模块定义

1.IFU - 取指令单元

```
module IFU(
    input Clk,
    input Rst,
    input [25:0] IMM,
    input [31:0] RA,
    input [1:0] NPCOp,
    input Zero,
    output [31:0] PC4,
    output [31:0] IM_D,
     output [31:0] PC_sgn
    );
    reg [31:0] PC;
    reg [31:0] IM[0:1023]; //IM[0:4095]
    wire [31:0] NPC;

    wire [31:0] PC_beq;
    integer i;

    assign PC_beq = (Zero == 1'b0) ? PC4 : PC4 +
{{14{IMM[15]}},IMM[15:0],2'b00};
    assign PC4 = PC + 32'd4;
    assign NPC = (NPCOp == 2'b00) ? PC4     :
                 (NPCOp == 2'b01) ? PC_beq :
                 (NPCOp == 2'b10) ? {PC[31:28], IMM, 2'b00} :
                 (NPCOp == 2'b11) ? RA      : -1 ;
```

```
    assign PC_sgn = PC;
    assign IM_D = IM[(PC-32'h00003000)/4];

    always @(posedge Clk) begin
        if(Rst == 1) begin
            PC <= 32'h0000_3000;
        end else begin
            PC <= NPC;
        end
    end

    initial begin
        PC = 32'h0000_3000;
        $readmemh("code.txt",IM);
//      for(i=0;i<4096;i=i+1) begin
//          IM[i] = 32'h0000_0000;
//      end
//      $readmemh("code.txt",IM,3072);
    end

endmodule
```

NPC

信号名	方向	描述
[31:0] PC	Input	32位输入，当前PC值
[15:0] IMM	Input	26位立即数（imm16和imm26的综合考量）
[31:0] RA	Input	jr指令中，所选寄存器32位值输入，目标地址值
[1:0] Op	Input	选择不同NPC输出： `2'b00`：输出顺序地址PC+4 `2'b01`：输出指令beq所得地址 `2'b10`：输出指令jal所得地址 `2'b11`：输出指令jr所得地址
Zero	Input	beq指令中，rs和rt的比较结果： `0`：不相等 `1`：相等
[31:0] PC4	Output	jal指令中，将PC+4存入`$ra`内所需输出
[31:0] NPC	Output	下一条指令目标地址

PC

完全模拟MARS中PC行为。

起始地址：**0x0000_3000**

IM

仍然从0开始时，每次取值时从PC中减去`0x00003000`,即

```
assign IM_D = IM[(PC-32'h00003000)/4];
```

2.GRF

通用寄存器堆

信号名	方向	描述
[4:0] A1	Input	输入rs段要读取的寄存器编号
[4:0] A2	Input	输入rt段要读取的寄存器编号
[4:0] A3	Input	输入rd段要写入的寄存器编号
[31:0] WD	Input	需要写回的值
WE	Input	写入使能端
Clk	Input	时钟信号端
Rst	Input	寄存器复位端
[31:0] RD1	Output	输出A1所选对应寄存器的值
[31:0] RD2	Output	输出A2所选对应寄存器的值

```
module GRF(
     input [31:0] PC, //为了打印信号而传入
    input Clk,
    input Rst,
    input [4:0] A1,
    input [4:0] A2,
    input [4:0] A3,
    input [31:0] WD,
     input RFWr,
    output [31:0] RD1,
    output [31:0] RD2
    );
    integer handle;

    reg [31:0] rf[31:1];
    integer i;

    assign RD1 = (A1 == 0) ? 0 : rf[A1];
    assign RD2 = (A2 == 0) ? 0 : rf[A2];

    always @(posedge Clk) begin
        if(Rst == 1) begin
            for(i=1;i<32;i=i+1)
                rf[i] <= 32'h0000_0000;
        end else if(RFWr == 1 && A3 != 0) begin
            rf[A3] <= WD;
```

```
            $display("@%h: $%d <= %h", PC, A3, WD);
            handle = $fopen("../msg_out_cpu.txt","a");
            $fdisplay(handle,"@%h: $%d <= %h", PC, A3, WD);
            $fclose(handle);
        end
    end

    initial begin
        for(i=1;i<32;i=i+1)
            rf[i] <= 32'h0000_0000;
    end

endmodule
```

3.EXT

实现不同位扩展功能。

信号名	方向	描述
[15:0] I	Input	输入16位立即数
[1:0] EXTOp	Input	选择扩展方式： `2'b00`：无符号扩展 `2'b01`：有符号扩展 `2'b10`：加载到高位
[31:0] O	Output	输出扩展后的32位数

```
module EXT(
    input [15:0] I,
    input [1:0] EXTOp,
    output [31:0] O
    );
    assign O =  (EXTOp == 2'b00) ? {16'h0000,I}     :
                (EXTOp == 2'b01) ? {{16{I[15]}},I}  :
                (EXTOp == 2'b10) ? {I,16'h0000}     : -1 ;

endmodule
```

4.ALU

算数逻辑单元。

信号名	方向	描述
[31:0] A	Input	输入数1
[31:0] B	Input	输入数2
[2:0] ALUOp	Input	运算选择器： `3'b000`：加法运算 `3'b001`：减法运算 `3'b010`：或运算 `3'b011`：左移sll运算 `3'b100`：比较运算(A<B)，用于slt
[31:0] Y	Output	输出运算结果
Zero	Output	A和B比较结果： 0：不相等 1：相等

```
module ALU(
    input [31:0] A,
    input [31:0] B,
     input [4:0] Shamt,
    input [2:0] ALUOp,
    output [31:0] Y,
    output Zero
    );
    assign Y =  (ALUOp == 3'b000) ? A + B :
                (ALUOp == 3'b001) ? A - B :
                (ALUOp == 3'b010) ? A | B :
                (ALUOp == 3'b011) ? B << Shamt :
                (ALUOp == 3'b100) ? A < B : -1;

    assign Zero = (A == B) ? 1 : 0 ;
endmodule
```

5.DM

数据内存。双端模式RAM实现（RAM 的 **Data Interface** 属性设置为 **Separate load and store ports**），容量为$2^{10} \times 32bit$。

起始地址：**0x0000_0000**

信号名	方向	描述
[31:0] A	Input	输入选择数据存取目标地址
[31:0] WD	Input	输入将要写入的数据内容
DMWr	Input	写入使能端
Clk	Input	时钟信号端
Rst	Input	RAM复位端
[1:0] SSel	Input	store类指令（`sw` `sh` `sb`）选择: `2'b00`：选择`sw`指令需写入的数据 `2'b01`：选择`sh`指令需写入的数据 `2'b10`：选择`sb`指令需写入的数据 `2'b11`：空置
[1:0] LSel	Input	load类指令（`lw` `lh` `lb`）选择: `2'b00`：选择`lw`指令需读出的数据 `2'b01`：选择`lh`指令需读出的数据 `2'b10`：选择`lb`指令需读出的数据 `2'b11`：空置
[31:0] RD	Output	输出读取出的数据

```
module DM(
     input [31:0] PC, //为打印信号而传入
    input Clk,
    input Rst,
    input [31:0] A,
    input [31:0] WD,
    input DMWr,
    input [1:0] SSel,
    input [1:0] LSel,
    output [31:0] RD
    );
    integer handle;

    reg [31:0] DM[0:1023];
    wire [9:0] addr;
    wire [31:0] lwData, swData,shData,sbData;
    wire [15:0] lhData;
    wire [7:0]  lbData;
    wire [31:0] WDReal_addr, WDReal_A; //WDReal_addr是适配addr的内容，WDReal_A是适配A的内容
    integer i;

    assign addr = A[11:2]; //在DM处是大小是控制好的,A是完整地址信号（Byte），addr是字地址信号（4Byte），对应DM索引

    assign lwData = DM[addr];
    assign lhData = (A[1] == 1'b0) ? DM[addr][15:0] : DM[addr][31:16];
    assign lbData = (A[1:0] == 2'b00) ? DM[addr][7:0]   :
                    (A[1:0] == 2'b01) ? DM[addr][15:8] :
                    (A[1:0] == 2'b10) ? DM[addr][23:16]:
```

```
                    (A[1:0] == 2'b11) ? DM[addr][31:24]: -1;
    assign RD = (LSel == 2'b00) ? lwData :
                (LSel == 2'b01) ? {{16{lhData[15]}},lhData} :
                (LSel == 2'b10) ? {{24{lbData[7]}} ,lbData} : -1;

    assign swData = WD;
    assign shData = (A[1] == 1'b0) ? {DM[addr][31:16],WD[15:0]} :
{WD[15:0],DM[addr][15:0]};
    assign sbData = (A[1:0] == 2'b00) ? {DM[addr][31:8] ,WD[7:0]} :
                    (A[1:0] == 2'b01) ? {DM[addr][31:16],WD[7:0], DM[addr][7:0]}
:
                    (A[1:0] == 2'b10) ? {DM[addr][31:24],WD[7:0],DM[addr][15:0]}
:
                    (A[1:0] == 2'b11) ? {WD[7:0],DM[addr][23:0]} : -1;
    assign WDReal_addr = (SSel == 2'b00) ? swData :
                         (SSel == 2'b01) ? shData :
                         (SSel == 2'b10) ? sbData : -1;
    assign WDReal_A = (SSel == 2'b00) ? WD      :
                      (SSel == 2'b01) ? WD[15:0] :
                      (SSel == 2'b10) ? WD[7:0]  : -1;

    always @(posedge Clk) begin
        if(Rst == 1) begin
            for(i=0;i<1024;i=i+1)
                DM[i] <= 32'h0000_0000;
        end else if(DMWr == 1) begin
            DM[addr] <= WDReal_addr;
            $display("@%h: *%h <= %h", PC, A, WDReal_A);
            handle = $fopen("../msg_out_cpu.txt","a");
            $fdisplay(handle,"@%h: *%h <= %h", PC, A, WDReal_A);
            $fclose(handle);
        end
    end

    initial begin
        for(i=0;i<1024;i=i+1)
            DM[i] = 32'h0000_0000;
    end

endmodule
```

（三）数据通路连接总表

未更新，目前都用不上

部件	PC	NPC				IM	RF				EXT	ALU		DM	
输入信号	DI	PC	IMM	RA	Zero	A	A1	A2	A3	WD		A	B	A	WD
addu	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
subu	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
ori	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C		RF.RD1	RF.RD2		
lw	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:00]	RF.RD1	EXT.O	ALU.C	
sw	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]	IM.D[20:16]			IM.D[15:00]	RF.RD1	EXT.O	ALU.C	RF.RD2
beq	NPC.NPC	PC.DO	IM.D[15:00]		ALU.Zero	PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2		
jal	NPC.NPC	PC.DO	IM.D[25:00]			PC.DO			0x1F	NPC.PC4					
jr	NPC.NPC	PC.DO		RF.RD1		PC.DO	IM.D[25:21]								
lui	NPC.NPC	PC.DO				PC.DO			IM.D[20:16]	EXT.O	IM.D[15:00]				
nop	NPC.NPC	PC.DO				PC.DO									
j	NPC.NPC	PC.DO	IM.D[25:00]			PC.DO									
add															
addi															
sll															
sllv															
lh	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:00]	RF.RD1	EXT.O	ALU.C	
lb	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:00]	RF.RD1	EXT.O	ALU.C	
sh	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]	IM.D[20:16]			IM.D[15:00]	RF.RD1	EXT.O	ALU.C	RF.RD2
sb	NPC.NPC	PC.DO				PC.DO	IM.D[25:21]	IM.D[20:16]			IM.D[15:00]	RF.RD1	EXT.O	ALU.C	RF.RD2
COMPLETE	NPC.NPC	PC.DO	IM.D[25:00]	RF.RD1	ALU.Zero	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[20:16] IM.D[15:11] 0x1F	NPC.PC4 DM.RD ALU.C EXT.O	IM.D[15:00]	RF.RD1	RF.RD2 EXT.O	ALU.C	RF.RD2

（四）控制器CTRL模块定义

1.控制信号定义

控制信号	描述
[1:0] NPCOp	执行跳转指令时控制选择NPC输出值： （即NPC的Op信号） `2'b00`：输出顺序地址PC+4 `2'b01`：输出指令beq所得地址 `2'b10`：输出指令jal所得地址 `2'b11`：输出指令jr所得地址
[1:0] M1Sel	选择GRF模块A3（写入寄存器）的输入信号： `2'b00`：接入IM.D[20:16]信号（也是A2处信号） `2'b01`：接入IM.D[15:11]信号 `2'b10`：接入常量0x1F，即`$ra` `2'b11`：空置
[1:0] M2Sel	选择GRF模块WD（写入内容共）的输入信号： `2'b00`：接入NPC.PC4，jal指令存入PC+4的地址 `2'b01`：接入DM.RD，内存数据的回写 `2'b10`：接入ALU.C，计算数据回写 `2'b11`：接入EXT.O，位扩展数据回写（lui）
RFWr	GRF的写入使能端
[1:0] EXTOp	位扩展方式： `2'b00`：零扩展 `2'b01`：符号扩展 `2'b10`：加载至高16位 `2'b11`：空置
M3Sel	选择ALU模块B的输入信号： `0`：接入RF.RD2，接收寄存器取出内容 `1`：接入EXT.O，接收位扩展后的16位立即数内容
[2:0] ALUOp	ALU的运算模式： `3'b000`：加法运算 `3'b001`：减法运算 `3'b010`：或运算 `3'b011`：左移sll运算 `3'b100`：比较运算(A<B)，用于slt
DMWr	DM的写入使能端

```
module control(
    input [5:0] op,
    input [5:0] funct,
    output [1:0] NPCOp,
    output [1:0] EXTOp,
    output [2:0] ALUOp,
    output [1:0] SSel,
    output [1:0] LSel,
    output [1:0] M1Sel,
    output [1:0] M2Sel,
    output M3Sel,
    output RFWr,
```

```
    output DMWr
    );
```

2.控制器逻辑真值表

未更新，目前都用不上

func	10_0001	10_0011						00_1000						
op	00_0000	00_0000	00_1101	10_0011	10_1011	00_0100	00_0011	00_0000	00_1111	00_0010	10_0001	10_0000	10_1001	10_1000
	addu	subu	ori	lw	sw	beq	jal	jr	lui	j	lh	lb	sh	sb
NPCOp[1]	0	0	0	0	0	0	1	1	0	1	0	0	0	0
NPCOp[0]	0	0	0	0	0	1	0	1	0	0	0	0	0	0
M1Sel[1]	0	0	0	0	x	x	1	x	0	x	0	0	x	x
M1Sel[0]	1	1	0	0	x	x	0	x	0	x	0	0	x	x
M2Sel[1]	1	1	1	0	x	x	0	x	1	x	0	0	x	x
M2Sel[0]	0	0	0	1	x	x	0	x	1	x	1	1	x	x
RFWr	1	1	1	1	0	0	1	0	1	0	1	1	0	0
EXTOp[1]	x	x	0	0	0	x	x	x	1	x	0	0	0	0
EXTOp[0]	x	x	0	1	1	x	x	x	0	x	1	1	1	1
M3Sel	0	0	1	1	1	0	x	x	x	x	1	1	1	1
ALUOp[1]	0	0	1	0	0	x	x	x	x	x	0	0	0	0
ALUOp[0]	0	1	0	0	0	x	x	x	x	x	0	0	0	0
DMWr	0	0	0	0	1	0	0	0	0	0	0	0	1	1
SSel[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	1
SSel[0]	0	0	0	0	0	0	0	0	0	0	0	0	1	0
LSel[1]	0	0	0	0	0	0	0	0	0	0	0	1	0	0
LSel[0]	0	0	0	0	0	0	0	0	0	0	1	0	0	0

3.控制器实现结构

与逻辑产生指令信号

```
parameter special   = 6'b00_0000;
    parameter ADDU      = 6'b10_0001;   //funct
    parameter SUBU      = 6'b10_0011;   //funct
    parameter ORI       = 6'b00_1101;
    parameter SLL       = 6'b00_0000;   //funct

    parameter LW        = 6'b10_0011;
    parameter LH        = 6'b10_0001;
    parameter LB        = 6'b10_0000;
    parameter SW        = 6'b10_1011;
    parameter SH        = 6'b10_1001;
    parameter SB        = 6'b10_1000;

    parameter BEQ       = 6'b00_0100;
    parameter JAL       = 6'b00_0011;
    parameter JR        = 6'b00_1000;   //funct
    parameter J         = 6'b00_0010;
    parameter LUI       = 6'b00_1111;
    parameter SLT       = 6'b10_1010;   //funct

    wire addu,subu,ori,lw,lh,lb,sw,sh,sb,beq,jal,jr,j,lui,sll,slt;

    assign addu = (op == special) & (funct == ADDU);
    assign subu = (op == special) & (funct == SUBU);
    assign jr   = (op == special) & (funct == JR  );
    assign sll  = (op == special) & (funct == SLL );
    assign slt  = (op == special) & (funct == SLT );
    assign ori  = (op == ORI);
    assign lw   = (op == LW );
    assign lh   = (op == LH );
    assign lb   = (op == LB );
    assign sw   = (op == SW );
```

```
    assign sh   = (op == SH );
    assign sb   = (op == SB );
    assign beq  = (op == BEQ);
    assign jal  = (op == JAL);
    assign j    = (op == J  );
    assign lui  = (op == LUI);
```

或逻辑产生控制信号

```
    assign NPCOp[1] = jal | jr | j ;
    assign NPCOp[0] = beq | jr ;
    assign EXTOp[1] = lui ;
    assign EXTOp[0] = lw | lh | lb | sw | sh | sb ;
    assign ALUOp[2] = slt ;
    assign ALUOp[1] = ori | sll ;
    assign ALUOp[0] = subu | sll ;
    assign SSel[1]  = sb ;
    assign SSel[0]  = sh ;
    assign LSel[1]  = lb ;
    assign LSel[0]  = lh ;
    assign M1Sel[1] = jal ;
    assign M1Sel[0] = addu | subu | sll | slt ;
    assign M2Sel[1] = addu | subu | ori | lui | sll | slt ;
    assign M2Sel[0] = lw | lui | lh | lb ;
    assign M3Sel    = ori | lw | lh | lb | sw | sh | sb ;
    assign RFWr     = addu | subu | ori | lw | lh | lb | jal | lui | sll | slt ;
    assign DMWr     = sw | sh | sb ;
```

二、测试方案

尝试着通过缝合的方式实现自动化（核心步骤来自于其他同学）

流程思路

- [x] 1.**C++/Python**：生成大量包含指定指令的mips程序（`test_stg_mips.asm`）
- [x] 2.**命令行**：导出该mips程序指令的机器码（`test_stg_code.txt`）
- [x] 3.**魔改MARS**：产生测试所需的IM、DM存入信息
- [x] 4.**命令行**：导出该mips程序在魔改MARS运行产生的测试所需信息（`msg_out_mars.txt`）
- [x] 5.**Verilog**：微修CPU文件使得testbench运行后可将输出信息导出相应文件（`msg_out_cpu.txt`）
- [ ] 6.**命令行**：操纵ISE运行仿真相应CPU文件从而得到输出信息
- [x] 7.**C++/Python**：对拍`msg_out_mars.txt`和`msg_out_cpu.txt`，得出测试结果

最后每一步的运行都包装到一个C++程序中

内容

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>
#include<algorithm>
#include<cstdlib>
#include<ctime>
using namespace std;
ofstream outfile("test_stg_mips.asm");

struct op{
    int type,r1,r2,r3;
    op(){}
    op(int a1,int a2,int a3,int a4){
        type=a1,r1=a2,r2=a3,r3=a4;
    }
};

int size_im, size_dm, size_op, small_reg, ra;
op a[10010]; //instruction
vector<int> branch[10010];
int is_small[40],small[40],val[40],cnt;

int r(int a,int b,int except1){//range: [a...b]-{except1} 由于1号寄存器使用时可能出现
错误，所以生成指令不包含1号寄存器
    int t0=rand()*2+rand()%2;
    int res=t0%(b-a+1)+a;
    return ((res==1)&&except1==1)?0:res;
}

int t[10010];
void get_small(){//随机选取若干个寄存器用于lw和sw，他们的值小于dm大小且不变
    int i;
    for(i=1;i<=26;i++) t[i]=i;
    for(i=1;i<=26;i++) swap(t[i],t[rand()%30+1]); // t[i] range from 1...30,
thus t[i]+1 range from 2...31
    for(i=1;i<=small_reg;i++) small[i]=t[i]+1,is_small[t[i]+1]=1;
}
int nosmall(){//随机一个不用于lw和sw的寄存器
    int u=r(0,27,1);
    while(is_small[u]==1){u=r(0,27,1);}
    return u;
}
void print(int x){
    if(a[x].type==0) outfile<<"nop"<<endl;
    else if(a[x].type==1) outfile<<"addu $"<<a[x].r1<<",$"<<a[x].r2<<",$"
<<a[x].r3<<endl;
    else if(a[x].type==2) outfile<<"subu $"<<a[x].r1<<",$"<<a[x].r2<<",$"
<<a[x].r3<<endl;
    else if(a[x].type==3) outfile<<"lui $"<<a[x].r1<<","<<a[x].r3<<endl;
    else if(a[x].type==4) outfile<<"ori $"<<a[x].r1<<",$"<<a[x].r2<<","
<<a[x].r3<<endl;
    else if(a[x].type==5) outfile<<"lw $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
```

```
    else if(a[x].type==6) outfile<<"sw $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
    else if(a[x].type==7) outfile<<"beq $"<<a[x].r1<<",$"<<a[x].r2<<",branch"
<<a[x].r3<<endl;
    else if(a[x].type==8) outfile<<"sll $"<<a[x].r1<<",$"<<a[x].r2<<","
<<a[x].r3<<endl;
    else if(a[x].type==9) outfile<<"slt $"<<a[x].r1<<",$"<<a[x].r2<<",$"
<<a[x].r3<<endl;
    //else if(a[x].type==10) outfile<<"addiu $"<<a[x].r1<<",$"<<a[x].r2<<","
<<a[x].r3<<endl;
    else if(a[x].type==11) outfile<<"lb $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
    else if(a[x].type==12) outfile<<"sb $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
    else if(a[x].type==13) outfile<<"lh $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
    else if(a[x].type==14) outfile<<"sh $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;

    //else if(a[x].type==15) outfile<<"bne $"<<a[x].r1<<",$"<<a[x].r2<<",branch"
<<a[x].r3<<endl;
    else if(a[x].type==16) outfile<<"j "<<"branch"<<a[x].r3<<endl;
    else if(a[x].type==17) outfile<<"jal "<<"branch"<<a[x].r3<<endl;
    else if(a[x].type==18) outfile<<"jr $ra"<<endl;
    //else if(a[x].type==19) outfile<<"lhu $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
    //else if(a[x].type==20) outfile<<"lbu $"<<a[x].r1<<","<<a[x].r3<<"($"
<<a[x].r2<<")"<<endl;
}

void generateMips(){
    // 0nop 1addu 2subu 3lui 4ori 5lw 6sw 7beq 8sll 9slt 10addiu 11lb 12sb 13lh
14sh 15bne
    // 16j 17jal 18jr 19lhu 20lbu

    int i,j;
    srand(time(0));
    size_im=200;//size_of_im : number of the instructions generated
    size_dm=128;//size_of_dm
    size_op=18;//size of instruction set

    small_reg=10;//最前面small_reg个指令都是ori，先对寄存器进行赋值
    get_small();
    for(i=1;i<=small_reg;i++)
        a[i]=(op(4,small[i],0,val[small[i]]=r(0,size_dm-1,0)));
    for(i=small_reg+1;i<=size_im;i++){
        int op0=r(0,size_op-1,0),r1,r2,r3;
        if(op0==0) a[i]=(op(0,0,0,0));
        else if(op0==1||op0==2||op0==9){//addu subu
            a[i]=op(op0,nosmall(),r(0,27,1),r(0,27,1));
        }
        else if(op0==3||op0==4||op0==10){//lui ori
            a[i]=(op(op0,nosmall(),r(0,27,1),r(0,0x0000ffff,0)));
        }
        else if(op0==8){
            a[i]=(op(op0,nosmall(), r(0, 27, 1), r(0, 31, 0)));
        }
        else if(op0==5){//lw
```

```
            r1=r(0,size_dm-1,0)*4;
            r2=r(1,small_reg,0);
            a[i]=(op(op0,nosmall(),small[r2],r1-val[small[r2]]));
        }
        else if(op0==13 || op0 == 19){// lh lhu
            r1=r(0, size_dm-1, 0)*2;
            r2=r(1, small_reg, 0);
            a[i]=(op(op0,nosmall(),small[r2],r1-val[small[r2]]));
        }
        else if(op0==6){//sw
            r1=r(0,size_dm-1,0)*4;
            r2=r(1,small_reg,0);
            a[i]=(op(op0,r(0,27,1),small[r2],r1-val[small[r2]]));
        }
        else if(op0==14){//sh
            r1=r(0,size_dm-1,0)*2;
            r2=r(1,small_reg,0);
            a[i]=(op(op0,r(0,27,1),small[r2],r1-val[small[r2]]));
        }
        else if(op0==11 || op0 == 20){//lb lbu
            r1=r(0, size_dm-1, 0);
            r2=r(1, small_reg, 0);
            a[i]=(op(op0, nosmall(), small[r2], r1-val[small[r2]]));
        }
        else if(op0==12){//sb
            r1=r(0, size_dm-1, 0);
            r2=r(1, small_reg, 0);
            a[i]=(op(op0, r(0, 27, 1), small[r2], r1-val[small[r2]]));
        }
        else if(op0==7||op0==15|| op0 == 16 || op0 == 17){//beq j jal由于往上跳转可
能出现死循环，所以只生成往下跳转
            r3=r(i,size_im,0);
            a[i]=(op(op0,r(0,27,1),r(0,27,1),++cnt));
            branch[r3].push_back(cnt);
            if(op0 == 17) ra = r3;

        }
        else if(op0 == 18 && ra){//jr
            a[i]=op(op0,nosmall(),r(0,27,1),r(0,27,1));
            ra = 0;
        }
    }

    for(i=1;i<=size_im;i++){
        print(i);
        for(j=0;j<branch[i].size();j++)
            outfile<<"branch"<<branch[i][j]<<":"<<endl;
    }

    outfile.close();
    return;
}

void fileCmp(){
    // FILE *fp_cpu,*fp_mars;
    // char tmp_cpu[100],tmp_mars[100];
    int cnt=0, flag = 1;
    string tmp_cpu,tmp_mars;
```

```
    ifstream cpufile("msg_out_cpu.txt");
    ifstream marfile("msg_out_mars.txt");
    ofstream cmpfile("cmp_result.txt");

    // fp_cpu   = fopen("msg_out_cpu.txt","r");
    // fp_mars = fopen("msg_out_mars.txt","r");
    // while( fgets(tmp_cpu,100,fp_cpu)!=NULL &&
fgets(tmp_mars,100,fp_mars)!=NULL ){
    //  cnt++;

    // }
    //while(cpufile >> tmp_cpu && marfile >> tmp_mars){
    while(getline(cpufile,tmp_cpu) && getline(marfile,tmp_mars)){
        cnt++;
        if(tmp_cpu != tmp_mars){
            flag = 0;
            cmpfile << "mismatch - line" << cnt << "\t: ";
            cmpfile << "[cpu] " << tmp_cpu << "\t[mars] " << tmp_mars << endl;
        }
    }
    if(flag) cmpfile << "no error";
}

int main(){
    generateMips();
    system("java -jar MARS.jar nc mc CompactDataAtZero dump .text HexText
test_stg_code.txt test_stg_mips.asm > msg_out_mars.txt");

    //fileCmp();
    return 0;
}
```

三、思考题

1.思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

1.与逻辑识别+或逻辑生成

（1）assign

```
parameter special   = 6'b00_0000;
    parameter ADDU      = 6'b10_0001;   //funct
    parameter SUBU      = 6'b10_0011;   //funct
    parameter ORI       = 6'b00_1101;
    parameter SLL       = 6'b00_0000;   //funct

    parameter LW        = 6'b10_0011;
    parameter LH        = 6'b10_0001;
    parameter LB        = 6'b10_0000;
    parameter SW        = 6'b10_1011;
    parameter SH        = 6'b10_1001;
    parameter SB        = 6'b10_1000;
```

```
    parameter BEQ       = 6'b00_0100;
    parameter JAL       = 6'b00_0011;
    parameter JR        = 6'b00_1000;   //funct
    parameter J         = 6'b00_0010;
    parameter LUI       = 6'b00_1111;
    parameter SLT       = 6'b10_1010;   //funct

    wire addu,subu,ori,lw,lh,lb,sw,sh,sb,beq,jal,jr,j,lui,sll,slt;

    assign addu = (op == special) & (funct == ADDU);
    assign subu = (op == special) & (funct == SUBU);
    assign jr   = (op == special) & (funct == JR  );
    assign sll  = (op == special) & (funct == SLL );
    assign slt  = (op == special) & (funct == SLT );
    assign ori  = (op == ORI);
    assign lw   = (op == LW );
    assign lh   = (op == LH );
    assign lb   = (op == LB );
    assign sw   = (op == SW );
    assign sh   = (op == SH );
    assign sb   = (op == SB );
    assign beq  = (op == BEQ);
    assign jal  = (op == JAL);
    assign j    = (op == J  );
    assign lui  = (op == LUI);

    /* ================================================== */

    assign NPCOp[1] = jal | jr | j ;
    assign NPCOp[0] = beq | jr ;
    assign EXTOp[1] = lui ;
    assign EXTOp[0] = lw | lh | lb | sw | sh | sb ;
    assign ALUOp[2] = slt ;
    assign ALUOp[1] = ori | sll ;
    assign ALUOp[0] = subu | sll ;
    assign SSel[1]  = sb ;
    assign SSel[0]  = sh ;
    assign LSel[1]  = lb ;
    assign LSel[0]  = lh ;
    assign M1Sel[1] = jal ;
    assign M1Sel[0] = addu | subu | sll | slt ;
    assign M2Sel[1] = addu | subu | ori | lui | sll | slt ;
    assign M2Sel[0] = lw | lui | lh | lb ;
    assign M3Sel    = ori | lw | lh | lb | sw | sh | sb ;
    assign RFWr     = addu | subu | ori | lw | lh | lb | jal | lui | sll | slt ;
    assign DMWr     = sw | sh | sb ;
```

（2）always(*)

就是将assign的一些关系翻译成顺序块的写法

2.真值表

根据输入输出每位的真值表直接暴力书写主析取范式。

方式	优点	缺点
与或逻辑（assign）	直观，易调试；接近物理电路模型	代码堆积不美观?
与或逻辑（always）	更接近高级语言的写法	总感觉离电路模型远了，容易出现纰漏
真值表	代码量小一点吧，精力转移到画真值表	可迭代性差，调试难

2.在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset所驱动的部件具有什么共同特点?

都是具有存储功能的模块（寄存器）

3.C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。

因为addi相对addiu仅多了对相加后temp是否溢出的判断（`temp[32]==temp[31]`），若溢出则进入异常“IntegerOverflow”。若忽略了溢出， 两者指令操作一样；同理，add相对addu也是一样的。

4.根据自己的设计说明单周期处理器的优缺点。

优点	缺点
一个周期全数据通路中执行一条指令，简单规整，不易出错，调试容易	一个周期全数据通路只能执行一条指令，运行效率低，CPU频率被限制