2021年北航计组P4课下设计报告

一 CPU设计方案简述

（一）总体设计概述

本CPU为Verilog实现的单周期MIPS - CPU，支持的指令集包含{addu、subu、lw、sw、beq、lui、 ori 、jal、jr、nop}。为了实现这些功能，CPU主要包含了IM、GRF、ALU、DM、IFU、NPC、Controller、MUX这些模块。

（二）关键模块设计

1.GRF

GRF端口定义

端口名	方向	描述
clk	I	时钟信号
reset	I	复位信号， 1 复位，0无效
WE	I	写使能信号，1写入，0不能写入
A1	I	5位地址输入信号，将对应的寄存器数据读出到RD1
A2	I	5位地址输入信号，将对应的寄存器数据读出到RD2
A3	I	5位地址输入信号，将数WD存储到所对应的寄存器
WD3	I	32位输入数据
RD1	O	输出A1地址所对应的寄存器数据
RD2	O	输出A2地址所对应的寄存器数据
pc	I	用于输出$display中的地址指令信息

功能定义

序号	功能名称	描述
1	复位	reset信号有效时，所有寄存器存储的数据清零
2	读数据	读A1,A2所对应的寄存器数据到RD1，RD2
3	写数据	当WE有效且时钟上升沿到来时，将WD3存入到A3所对应的寄存器，并且输出对应的地址指令以及相应的寄存器信息

2. DM

DM端口定义

端口名	方向	描述
A [31：0]	I	读入五位地址信号
WD [31:0]	I	32位信号，存入A[11:2]所对应的地址
clk	I	时钟信号
reset	I	清零信号
WE	I	读入使能信号，当1时，存入数据
RD [31:0]	O	32位信号，输出A[11:2]所对应的寄存器的数据
pc	I	对应的指令地址信息

DM功能描述

序号	功能名称	描述
1	清零	当reset信号位1时，里的数据清零
2	读入数据	当WE信号为1，将WD存储到addr对应的寄存器

备注： DM的实质是1024*32bit的寄存器，addr为对应的寄存器地址，addr为10位，取A的11-2位。

3.ALU

ALU 端口简述

端口名	方向	描述
DataA [31:0]	I	数据A
DataB [31:0]	I	数据B
ALUctrl [2:0]	I	运算控制选择信号
result [31:0]	O	输出结果
ALUzero	O	当result结果为0，ALUzero输出为1

ALU 功能介绍

ALUctrl	运算描述
000	A&B
001	A|B
010	A + B
011	A - B
100	A⊕B

4.IFU

端口名	方向	描述
clk	I	时钟信号
reset	I	清零信号
[31:0] npc	I	下一地址
[31:0] pc	O	当前地址
[5:0] op	O	根据当前指令地址取指后的opcode
[4:0] rs	O	根据当前指令地址取指后的rs寄存器地址
[4:0] rt	O	根据当前指令地址取指后的rt寄存器地址
[4:0] rd	O	根据当前指令地址取指后的rd寄存器地址
[5:0] func	O	根据当前指令地址取指后的function
[15:0] imm	O	根据当前指令地址取指后的立即数
[25:0]index	O	根据当前指令地址取指后的转跳地址

具体实现

```
module IFU(
    input clk,
    input reset,
    input [31:0] npc,
    output  reg [31:0] pc,
    output [31:0] instr,
    output [5:0] op,
    output [4:0] rs,
    output [4:0] rt,
    output [4:0] rd,
    output [5:0] func,
    output [15:0] imm,
    output [25:0] index
    );
     reg [31:0] im [1023:0];
     wire [11:2] addr ;

     assign addr = pc [11:2];
     assign instr = im [addr];
     assign op = instr [31:26];
     assign rs = instr [25:21];
     assign rt = instr [20:16];
     assign rd = instr [15:11];
     assign func = instr [5:0];
     assign imm = instr [15:0];
     assign index = instr [25:0];

     initial begin
             pc <= 32'h0000_3000;
             $readmemh("code.txt",im);
```

```
    end
    always@(posedge clk) begin
         if ( reset == 1'b1) begin
                pc <= 32'h0000_3000;
           end
           else begin
                pc <= npc;
           end

    end

endmodule
```

功能介绍

1. 根据复位信号，将地址信号同步复位至0x0000_3000
2. 根据当前npc的输入，在时钟上升沿，更新pc的输出
3. 取指，根据当前pc所指向的地址，取出指令，并将指令的opcode、func、寄存器、立即数、跳转地址等信息分别输出。

5. NPC

端口名	方向	描述
[31:0] pc	I	当前指令地址
[15:0] imm	I	指令立即数
[25:0] index	I	指令跳转数
[31: 0] RD1	I	当指令为jr时，需转跳的rs寄存器所存储的地址
branch	I	beq指令信号
ALUzero	I	beq指令判断信号
jump	I	jump指令信号
jr	I	jr指令判断信号
npc	O	下一指令

功能介绍

1.jr信号为1时，npc为当前rs寄存器中的地址

2.jump信号为1时，npc的值为 pc31-28 || index || 00

3 .branch 和 ALUzero信号同时为1时，npc为 PC +4+ sign_ext(imm)||00

4.上述情况都不满足时，npc为pc+4

代码实现

```
module NPC(
    input [31:0] pc,
    input [15:0] imm,
    input [25:0] index,
    input [31:0] RD1,
    input branch,
    input ALUzero,
    input jump, // jal and j instruction
    input jr,
    output reg [31:0] npc
    );

     always@(*) begin
           if ( jr == 1'b1 ) begin
                   npc = RD1;
             end
             else if ( jump == 1'b1 ) begin
                   npc = {pc[31:28], index , {2{1'b0}}};
             end
             else if ( (branch == 1'b1 && ALUzero == 1'b1)) begin
                   npc = pc + 4 + {{14{imm[15]}} , imm , 2'b00};
             end
             else begin
                   npc = pc +4 ;
             end
     end


endmodule
```

6. controller

端口	方向	描述
[5:0] opcode	I	指令的opcode码
[5:0] func	I	指令的func码
MemtoReg	O	GRF输入端的MUX信号选择
MemWrite	O	DM写入信号
[2:0] ALUctrl	O	ALU控制信号
ALUSrc	O	ALU输入端的MUX控制信号
RegDst	O	GRF的A3端的控制器输入信号
RegWrite	O	GRF写入信号
jump	O	转跳指令信号
branch	O	beq指令信号
[1:0] EXT	O	ALUdataB输入端的MUX控制信号
jal	O	jal转跳指令的控制信号
jr	O	jr转跳指令的控制信号

控制器真值表

控制信号	R	lw	sw	beq	j	ori	Lui
	000000	100011	101011	000100	000010	001101	001111
MemWrite	0	0	1	0	0	0	0
RegWrite	1	1	0	0	0	1	1
RegDst	1	0	X	X	X	0	0
ALUSrc	0	1	1	0	X	X	X
MemtoReg	0	1	X	X	X	0	0
ALUop	010	000	000	001	XXX	011	100
branch	0	0	0	1	X	0	0
jump	0	0	0	0	1	0	0
EXT	00	00	00	00	XX	01	10
jal	0	0	0	0	0	0	0
jr	0	0	0	0	0	0	0

控制信号	jal	jr
op	000011	000000
MemWrite	0	0
RegWrite	1	0
RegDst	X	X
ALUSrc	X	X
MemtoReg	X	X
ALUop	XXX	XXX
branch	X	X
jump	1	X
EXT	XX	XX
jal	1	0
jr	0	1

实现代码

```
module Controller(
    input [5:0] opcode,
    input [5:0] func,
    output reg  MemtoReg,
    output reg MemWrite,
    output reg [2:0] ALUCtrl,
    output reg ALUSrc,
    output reg RegDst,
    output reg RegWrite,
    output reg jump,
    output reg branch,
    output reg [1:0] EXT,
     output reg jal,
     output reg jr
    );
     always@(*) begin
         case(opcode)
                 6'b000000: begin
                                 case(func)
                                           6'b100001:begin //addu
                                                   RegWrite = 1;
                                                    RegDst = 1 ;
                                                    ALUSrc = 0;
                                                    branch = 0;
                                                   MemWrite = 0;
                                                    MemtoReg = 0 ;
                                                    ALUCtrl = 3'b010;
                                                    jump = 0;
                                                    EXT = 2'b00;
                                                    jal = 0;
                                                    jr = 0;
```

```
                                        end
                                        6'b100011:begin //subu
                                             RegWrite = 1;
                                              RegDst = 1 ;
                                              ALUSrc = 0;
                                              branch = 0;
                                             MemWrite = 0;
                                              MemtoReg = 0 ;
                                              ALUCtrl = 3'b011;
                                              jump = 0;
                                              EXT = 2'b00;
                                              jal = 0;
                                              jr = 0;

                                        end
                                        6'b001000: begin //jr
                                             RegWrite = 0;
                                              RegDst = 0 ;
                                              ALUSrc = 0;
                                              branch = 0;
                                             MemWrite = 0;
                                              MemtoReg = 0 ;
                                              ALUCtrl = 3'b000;
                                              jump = 0;
                                              EXT = 2'b00;
                                              jal = 0;
                                              jr = 1;
                                        end
                                        default : begin
                                             RegWrite = 0;
                                              RegDst = 0 ;
                                              ALUSrc = 0;
                                              branch = 0;
                                             MemWrite = 0;
                                              MemtoReg = 0 ;
                                              ALUCtrl = 3'b000;
                                              jump = 0;
                                              EXT = 2'b00;
                                              jal = 0 ;
                                              jr = 0 ;

                                        end
                                endcase

                end
                6'b100011:begin //lw
                             RegWrite = 1;
                                 RegDst = 0 ;
                                 ALUSrc = 1;
                                 branch = 0;
                                 MemWrite = 0;
                                 MemtoReg = 1 ;
                                 ALUCtrl = 3'b010;
                                 jump = 0;
                                 EXT = 2'b00;
                                 jal = 0;
                                 jr = 0;
                end
```

```
                    6'b101011:begin//sw
                                RegWrite = 0;
                                    RegDst = 0 ;
                                    ALUSrc = 1;
                                    branch = 0;
                                    MemWrite = 1;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b010;
                                    jump = 0;
                                    EXT = 2'b00;
                                    jal = 0;
                                    jr = 0;
                    end
                    6'b000100:begin//beq
                                RegWrite = 0;
                                    RegDst = 0 ;
                                    ALUSrc = 0;
                                    branch = 1;
                                    MemWrite = 0;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b011;
                                    jump = 0;
                                    EXT = 2'b00;
                                    jal = 0;
                                    jr = 0;
                    end
                    6'b001101:begin // ori
                                RegWrite = 1;
                                    RegDst = 0 ;
                                    ALUSrc = 0;
                                    branch = 0;
                                    MemWrite = 0;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b001;
                                    jump = 0;
                                    EXT = 2'b01;
                                    jal = 0;
                                    jr = 0;
                    end
                    6'b001111:begin //lui
                                RegWrite = 1;
                                    RegDst = 0 ;
                                    ALUSrc = 0 ;
                                    branch = 0;
                                    MemWrite = 0;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b100;
                                    jump = 0;
                                    EXT = 2'b10;
                                    jal = 0;
                                    jr = 0;
                    end
                    6'b000011:begin//jal
                                RegWrite = 1;
                                    RegDst = 0 ;
                                    ALUSrc = 0;
                                    branch = 0;
                                    MemWrite = 0;
```

```
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b000;
                                    jump = 1 ;
                                    EXT = 2'b00;
                                    jal = 1 ;
                                    jr = 0 ;
                    end
                    6'b000010:begin //j
                                RegWrite = 0;
                                    RegDst = 0 ;
                                    ALUSrc = 0;
                                    branch = 0;
                                    MemWrite = 0;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b000;
                                    jump = 1 ;
                                    EXT = 2'b00;
                                    jal = 0;
                                    jr = 0;

                    end
                    default : begin //
                                RegWrite = 0;
                                    RegDst = 0 ;
                                    ALUSrc = 0;
                                    branch = 0;
                                    MemWrite = 0;
                                    MemtoReg = 0 ;
                                    ALUCtrl = 3'b000;
                                    jump = 0 ;
                                    EXT = 2'b00;
                                    jal = 0;
                                    jr = 0;
                    end

          endcase
     end
endmodule
```

7. MUX

MUX主要分成三个类型的选择器

1.GRF A3端的地址选择输入 （由控制信号 RegDst 和jr信号控制）

2.GRF WD3端的写入数据选择输入（由信号MemtoReg 和 Jal控制）

3.ALU输入端dataB的选择输入 （有ALUSrc和EXT输入信号控制）

```
module RegA_MUX(
     input [4:0] rt,
      input [4:0] rd,
      input RegDst,
      input jal,
      output reg [4:0] A3

    );
```

```
     always@(*) begin
                if(jal == 1'b1) begin
                     A3 = 5'b11111;
                end
                else if ( RegDst == 1'b0) begin
                     A3 = rt;
                end
                else if ( RegDst == 1'b1) begin
                     A3 = rd ;
                end
       end

endmodule
module ALUdataB_MUX(
     input [31:0] RD2,
      input [15:0] imm,
      input ALUSrc,
      input [1:0] EXT,
      output reg [31:0]  ALUdataB
         );

     always@(*)begin
          if ( EXT == 2'b00 ) begin
                 if ( ALUSrc == 1'b0 ) begin
                       ALUdataB =  RD2;
                  end
                  else  begin
                       ALUdataB = {{16{imm[15]}},imm};
                  end
            end
          else if ( EXT == 2'b01 ) begin
                    ALUdataB  = {{16{1'b0}},imm};
            end
            else if ( EXT == 2'b10 ) begin
                    ALUdataB  = {imm,{16{1'b0}}};
            end
            else begin
                    ALUdataB  = ALUdataB;
            end
    end

endmodule
module RegD_MUX(
       input [31:0] ALUresult,
         input [31:0] RD,
         input [31:0] pc,
         input  MemtoReg,
         input  jal,
       output reg [31:0] WriteData
      );
        always@(*) begin
             if ( jal == 1'b1 ) begin
                  WriteData = pc + 4;
              end
              else begin
                  if ( MemtoReg == 1'b0 ) begin
                          WriteData = ALUresult;
                    end
```

```
                    else begin
                            WriteData = RD;
                    end
                end

        end
endmodule
```

二 测试方法

代码自动生成，使用了讨论区jrt同学的自动生成器

然后将魔改的mars输出数据和ise的输出数据导入到两个txt文档

个人写了一个文档自动比较器来比较输出

```
#include<stdio.h>
#include<string.h>
#define max 1024
char string1[max], string2[max];
int len1,len2;
int num;
int i;
int main()
{
    FILE * str1, *str2;
    str1 = fopen("ise_output.txt","r") ;
    str2 = fopen("mars_output.txt","r");
    while(fgets(string1,max,str1)!=NULL)
    {
        num++;
        fgets(string2,max,str2);
        len1=strlen(string1);
        len2=strlen(string2);
        if ( len1 != len2)
        {
            printf(" Attention! Line %d has a mistake",num);
        }
        else
        {
            for(i=0;i<len1;i++)
            {
                if(string1[i]!=string2[i])
                {
                    printf(" Attention! Line %d has a mistake.We expect output
%s, but we get the output %s",num,string1,string2);
                    return 0;
                }
            }
        }

    }

    printf("AK");
    return 0;

}
```

三 思考题

1.根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的?

lw，sw在计算存储地址的时候都是以字节为单位，而我们设计的DM的存储是32bit，即以一字为存储单位，所以在计算DM存储的实际地址时，需要除以4，即将输入地址的后两位忽略，取11-2位，addr即输入地址的11-2位。

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.方法一使用 always@（*） + case（）语句，如上给出的controller代码

2. 模拟logisim的与或门阵列

```
assign ori = !op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0];

assign ALUSrc = sw|lw;
```

3.使用宏定义+case()语句

```
`define R 6'b000000
`define lw 6'b100011
`define sw  6'b101011
always@(*) begin
    case(opcode)
        R:begin
        end
        lw:begin
        end
        sw: begin
        end
end
```

第一、三种方式结构清晰，再修改错误时可以对照真值表查看case中的语句，找错误非常方便，但是代码冗长，第三种比较第一种使用宏定义，在写case语句时更能方便找到对应指令对应的控制信号。

第二种方式模仿logisim的与或门阵列，逻辑比较简单清晰，但是无法清晰找到指令对应的控制信号。

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?

部件内部都存在着时序逻辑，存在着复位的需求，不复位可能会影响后续的操作。

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。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。

add的指令执行过程中，自动生成一个高位temp32，temp32与temp31比较，如果不相等，则代表溢出，会反馈"IntegerOverflow"，不输出有效数据，如果temp32与temp31，则输出相加后的数据。addu指令则是执行过程中忽略溢出，直接输出相加后的效果，所以如果忽略溢出，则addu和add等价。

4. 根据自己的设计说明单周期处理器的优缺点。

优点：设计结构清晰，逻辑简单，分模块实现，容易构建代码。

缺点：时间效率较低，单周期执行，而且实现指令较少，如要添加更多的指令，要增加更多的控制信号，较为不便。