计算机组成原理 P4 CPU 设计实验报告

一、模块规格

1. IM

（1） 端口说明

IM 端口定义表

信号名	方向	描述
address[31:0]	I	地址信号
mipsop[31:0]	o	指令数据

（2） 功能定义

IM 功能描述表

序号	功能定义	功能描述
1	加载地址	将指令的地址加载至 ROM 中
2	寻址	根据 address 找到对应指令

2. GRF

（1） 端口说明

GRF 端口定义表

信号名	方向	描述
RegWrite	I	数据写入信号
clk	I	时钟信号
reset	I	复位信号
WD[31:0]	I	目标寄存器写入数据
WPC[31:0]	I	指令地址
RS[4:0]	I	第一个寄存器源操作数
RT[4:0]	I	第二个寄存器源操作数
RD[4:0]	I	寄存器目标操作数
Readdata1[31:0]	O	第一个源寄存器读出数据
Readdata2[31:0]	O	第二个源寄存器读出数据

（2） 功能定义

GRF 功能描述表

序号	功能定义	功能描述
1	从寄存器堆读数据	读出 rs、rt 地址对应寄存器中的数据到 Readdata1 和 Readdata2
2	向寄存器堆写数据	RegWrite 信号有效时，将 WD 写入 rd 地址对应的寄存器
3	复位	Reset 信号有效时，所有寄存器数据清零。
4	打印输出	`$display("@%h: $%d <= %h", WPC, RD,WD);`

3. ALU

（1） 端口说明

ALU 端口定义表

信号名	方向	描述
A[31:0]	I	参与 ALU 计算的第一个值
B[31:0]	I	参与 ALU 计算的第二个值
ALUop[3:0]	I	ALU 功能选择信号 0000：A&B 0001：A|B 0010：A+B 0110：A-B
result[31:0]	O	ALU 计算的结果
Zero	O	判断 A 于 B 是否相等，相等输出 1

（2） 功能定义

ALU 功能描述表

序号	功能定义	功能描述
1	与运算	reselt = A & B
2	或运算	reselt = A | B
3	加运算	reselt = A + B
4	减运算	reselt = A - B

5	比较运算	If（A==B)；Zero = 1；

4. DM

（1） 端口说明

DM 端口定义表

信号名	方向	描述
Address[31:0]	I	32 位内存地址
Writedata[31:0]	I	内存写入数据
clk	I	时钟信号
reset	I	复位信号
memWrite	I	信号有效时，向内存中写入数据
memRead	I	信号有效时，从内存中读出数据
pc[31:0]	I	指令地址
readdata[31:0]	0	从内存中读出的数据

（2） 功能定义

DM 功能描述表

序号	功能定义	功能描述
1	读数据	根据地址信号从内存中读取数据
2	写数据	根据地址信号向内存中吸入数据
3	复位	Reset 信号有效时，内存清零
4	打印输出	$display("@%h: *%h <= %h",pc, address, writedata);

5. EXT

（1） 端口说明

EXT 端口定义表

信号名	方向	描述
In[15:0]	I	输入 16 位立即数
out [31:0]	0	符号扩展到 32 位

（2） 功能定义

EXT 功能描述表

序号	功能定义	功能描述
1	符号扩展	将 16 位立即数符号扩展至 32 位

6. Controller

（1） 端口说明

Cotroller 端口定义表

信号名	方向	描述
func[5:0]	I	6 位函数码
op[5:0]	I	6 位操作码
RegWrite	O	寄存器写入控制
MemRead	O	内存读取控制
MemWrite	O	内存写入控制
RegDst[1:0]	O	写寄存器的目标寄存器来自 rt/rd/31 操作数
Branch	O	Beq 指令信号
MemtoReg[1:0]	O	写入寄存器选择信号
PCSrc[1:0]	O	nPC 选择信号
Shfit2s	O	需要左移的数(index_26_0/signext)选择信号
ALUop[3:0]	O	ALU 功能选择信号
ALUSrc[1:0]	O	参与 ALU 计算的数据的选择信号

（2） 功能定义

Cotroller 功能描述表

序号	功能定义	功能描述
1	写寄存器的目标选择	0：写寄存器的目标寄存器号来自 rt 字段 1: 写寄存器的目标寄存器号来自 rd 字段

		2: 目标寄存器 31
2	寄存器写入控制	寄存器堆写使能有效
3	内存读取控制	数据存储器读使能有效
4	内存写入控制	数据存储器写使能有效
5	Beq 指令信号	Branch 有效时指令为 beq
6	写入寄存器选择信号	00：向寄存器写入 ALU 计算结果 01：向寄存器写入从 DM 读取的值 10：向寄存器写入 adder（pc+4）的结果
7	参与 ALU 计算的数据的选择信号	00：Readdata2 01：sign-ext 10：0-ext 11：加载立即数到高位
8	ALU 功能选择信号	0000：A&B 0001：A|B 0010：A+B 0110：A-B
9	选择需要左移 2 位的数	1：index_26_0 0：signext
10	下一周期 pc 选择信号	00：pc+4 01：pc+4+shift2 or pc+4 10：PC31…28||shift2 11：Rdata1

7. adder

（1） 端口说明

adder 端口定义表

信号名	方向	描述
PC[31:0]	I	当前 PC 的值
PC_4[31:0]	O	Pc+4

（2） 功能定义

adder 功能描述表

序号	功能定义	功能描述
1	计算 PC+4	计算 PC+4

8. nPC

（1）端口说明

nPC 端口定义表

信号名	方向	描述
PC4[31:0]	I	PC+4 的值
Shift2[31:0]	I	跳转指令需要跳转的位数
Rdata1[31:0]	I	Jr 指令跳转的位数
PCsrc[1：0]	I	跳转指令控制信号
Equal	I	Beq 判断相等标志
Clk	I	时钟信号
reset	I	复位信号
Nextpc [31:0]	O	下一周期 PC 的值

（1） 功能定义

nPC 功能描述表

序号	功能定义	功能描述

1	计算下一周期 pc	Pcsrc==0
		Nextpc <= pc4;
		Pcsrc==1
		if(equal)
		Nextpc <= pc4 + shift2;
		else
		Nextpc <= pc4;
		pcsrc === 2
		Nextpc<= {pc4[31:28],shift2[27:0]};
		pcsrc === 3
		Nextpc <= Rdata1;

2	复位	复位信号有效时，Nextpc <= 32'h3000;

9. Shift2

（1）端口说明

Shift2 端口定义表

信号名	方向	描述
a[31:0]	I	输入值
Shift2[31:0]	O	左移两位后的值

（2）功能定义

Shift2 功能描述表

序号	功能定义	功能描述
1	左移两位	将输入值左移两位

二、控制器设计

1. 控制信号意义

控制信号	功能定义	功能描述
RegWrite	寄存器写入控制	寄存器写入使能有效
MemRead	内存读取控制	内存读取使能有效
MemWrite	内存写入控制	内存写入使能有效
RegDst[1:0]	写寄存器的目标寄存器 来自 rt/rd/31 操作数	0： rt 1: rd 2: 目标寄存器 31
Branch	Beq 指令信号	指令时 beq
MemtoReg[1:0]	写入寄存器选择信号	00：向寄存器写入 ALU 计算结果 01：向寄存器写入从 DM 读取的值 10：向寄存器写入 adder（pc+4）的结果
PCSrc[1:0]	nPC 选择信号	00：pc+4 01：pc+4+shift2 or pc+4 10：PC31…28||shift2 11：Rdata1

Shfit2s	选择需要左移 2 位的数	1：index_26_0 0：signext
ALUop[3:0]	ALU 功能选择信号	0000：A&B 0001：A|B 0010：A+B 0110：A-B
ALUSrc[1:0]	参与 ALU 计算的数据的选择信号	00：Readdata2 01：sign-ext 10：0-ext 11：加载立即数到高位

2. 控制信号真值表

	op	func	RegWrite	MemRead	MemWrite	RegDst	branch
addu	000000	100001	1	0	0	01	0
subu	000000	100011	1	0	0	01	0
ori	001101		1	0	0	00	0
lw	100011		1	1	0	00	0
sw	101011		0	0	1	00	0
lui	001111		1	0	0	00	0
beq	000100		0	0	0	00	1
nop	000000	000000	0	0	0	00	0
jal	000011		1	0	0	10	0
jr	000000	001000	0	0	0	00	0

	op	func	MemtoReg	PCSrc	ALUop	shift2s	AlUSrc
addu	000000	100001	00	00	0010	0	00
subu	000000	100011	00	00	0110	0	00
ori	001101		00	00	0001	0	10
lw	100011		01	00	0010	0	01
sw	101011		00	00	0010	0	01

lui	001111		00	00	0010	0	11
beq	000100		00	01	0110	0	00
nop	000000	000000	00	00	0010	0	00
jal	000011		10	10	0010	1	00
jr	000000	001000	00	11	0010	0	00

三、测试代码

```
.text
  ori $t1,$0,5
  ori $t2,$0,3
  ori $t3,$t1,4

  addu $t0,$t1,$t2
  subu $t0,$t1,$t2
  lui $s0,1234
  beq $t1,$t3,label
  j label
  lui $s0,1
label:
  sw $t0,0($t7)
  lw $s2,0($t7)
  lui $s1,1
  jal label_2
label_2:
  beq $s1,$0,end
  sw $t0,0($t7)
  lw $s2,0($t7)
  lui $s1,0
  jr $ra
end:
  lui $s1,1
```

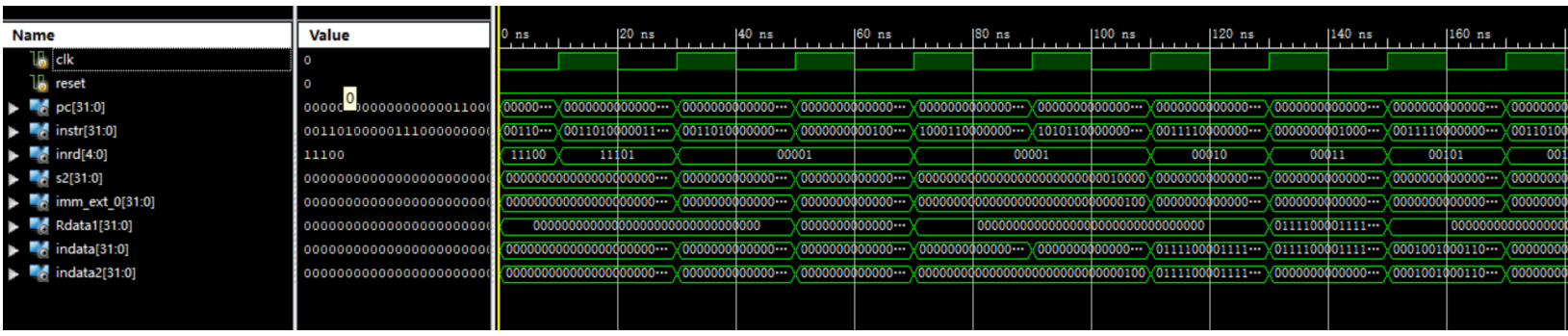
```
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
@00003000: $ 9 <= 00000005
@00003004: $10 <= 00000003
@00003008: $11 <= 00000005
@0000300c: $ 8 <= 00000008
@00003010: $ 8 <= 00000002
@00003014: $16 <= 04d20000
@00003024: *00000000 <= 00000002
@00003028: $18 <= 00000002
@0000302c: $17 <= 00010000
@00003030: $31 <= 00003034
@00003038: *00000000 <= 00000002
@0000303c: $18 <= 00000002
@00003040: $17 <= 00000000
@00003048: $17 <= 00010000
ISim> |
```

```
      nop
    34090005
    340a0003
    352b0004
    012a4021
    012a4023
    3c1004d2
    112b0002
    08000c09
    3c100001
    ade80000
    8df20000
    3c110001
    0c000c0d
    12200004
    ade80000
    8df20000
    3c110000
    03e00008
    3c110001
    00000000

341c0000
341d0000
34013456
00210821
8c010004
ac010004
3c027878
00411823
3c051234
34040005
```

```
00000000
ac85ffff
8c83ffff
10650003
00000000
10000011
00000000
34670404
10e3000e
00000000
3c087777
3508ffff
00080023
34001100
00e65021
34080000
34090001
340a0001
010a4021
1109fffe
0c000c22
00000000
014a5021
1000ffff
014a5021
03e00008
00000000
```

```
@00003000: $28 <= 00000000
@00003004: $29 <= 00000000
@00003008: $ 1 <= 00003456
@0000300c: $ 1 <= 000068ac
@00003010: $ 1 <= 00000000
@00003014: *00000004 <= 00000000
@00003018: $ 2 <= 78780000
@0000301c: $ 3 <= 78780000
@00003020: $ 5 <= 12340000
@00003024: $ 4 <= 00000005
@0000302c: *00000004 <= 12340000
@00003030: $ 3 <= 12340000
@00003044: $ 7 <= 12340404
@00003050: $ 8 <= 77770000
@00003054: $ 8 <= 7777ffff
@00003060: $10 <= 12340404
@00003064: $ 8 <= 00000000
@00003068: $ 9 <= 00000001
@0000306c: $10 <= 00000001
@00003070: $ 8 <= 00000001
@00003070: $ 8 <= 00000002
@00003078: $31 <= 0000307c
@00003088: $10 <= 00000002
@00003080: $10 <= 00000004
ISim> |
```



四、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？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

因为指令都是 32 位，即 4 字节，所以下一条指令地址位于 PC+4，故舍弃最后两位，选择 [11:2]

addr 来源于 ALU、npc、controller 的计算结果

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 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

针对 IFU 中的 PC 寄存器，GRF 中的寄存器和 DM 中的数据。

因为如果不清零的话会使寄存器和 RAM 中的数据在下一个时钟沿上升时传到下一个部件中造成数据错误。

3. 列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

1. 方法一(case):

```
case(opcode)
        6'b000000:begin
            if(func==100001) addu=1;
            if(func==100011) subu=1;
            if(func==001000) jr=1;
        end
        6'b100011:lw=1;
        6'b101011:sw=1;
        6'b000100:beq=1;
        6'b001101:ori=1;
        6'b001111:lui=1;
        6'b000011:jal=1;
```

方法二（assign 语句）:

```
        assign addu = (opcode==6'b000000 & func==100001);
        assign subu = (opcode==6'b000000 & func==100011);
        assign lw = (opcode==6'b100011);
        assign sw = (opcode==6'b101011);
        assign beq = (opcode==6'b000100);
        assign ori = (opcode==6'b001101);
        assign lui = (opcode==6'b001111);
        assign jal = (opcode==6'b000011);
        assign jr = (opcode==6'b000000 & func==001000);
```

方法三（宏定义）:

```
        `define addu   (opcode==6'b000000 & func==100001);
        `define subu   (opcode==6'b000000 & func==100011);
         `define lw   (opcode==6'b100011);
         `define sw   (opcode==6'b101011);
         `define beq   (opcode==6'b000100);
         `define ori   (opcode==6'b001101);
         `define lui   (opcode==6'b001111);
```

4.根据你所列举的编码方式，说明他们的优缺点。

1：优点：将 R 型指令放入同一个 case，可读性强，

缺点：case 语句必须用于 always 模块中。

2：优点：代码量相对较少，

缺点：可维护性差。

2：优点：方便程序的修改，

缺点：嵌套定义影响程序的可读性，容易出错。

1. 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。

addi 为加立即数，操作为 ：

```
temp = (GPR[rs]31 || GPR[rs]) + sign_ extend(immediate)
if temp32 ≠ temp31
Signal Exception(Integer Overflow)
else GPR[rt] ← temp31..0
endif
```

addiu 为无符号加立即数，操作为 ：

```
GPR[rt] GPR[rs] + sign_ extend(immediate)
```

在忽略溢出的情况下，对 rt 进行的操作是相同的。

add 为符号加，操作为

```
temp = (GPR[rs]31 || GPR[rs]) + (GPR[rt]31 || GPR[rt])
if temp32 ≠ temp31
Signal Exception(Integer Overflow)
else GPR[rd] ← temp31..0
end if
```

addu 为无符号加，操作为

```
GPR[rd] = GPR[rs] + GPR[rt]
```

在忽略溢出的情况下，addu 与 add 等价

2. 根据自己的设计说明单周期处理器的优缺点。

优点：简化了处理器，结构简单，便于理解。

缺点：单周期 CPU 的 CPI 为 1，但时钟周期为最长的 ，但时钟周期为最长

的 load 指令执行 时间，效率较多周期较低。

3. 简要说明 jal、jr 和堆栈的关系。

jal 和 jr 通常成对使用，用于函数的调用，调用时一些寄存器变量的值需要用栈来维护防止被更改。

五、设计参考