P4.Verilog 开发单周期 cpu

一、实验目的

使用 verilog 设计一个单周期 CPU。

二、设计要求

1.处理器使用 Verilog 设计。

2.处理器能支持 addu、subu、ori、lw、sw、beq、lui、nop、jal、jr 指令。

3 处理器为单周期设计

三、实验设计

一、模块设计

1.PC 模块

（1）基本描述

PC 主要功能是完成地址转移工作，当复位信号有效时，将地址变为首地址，否则维持原有地址不变，将地址传输给指令存储器（IM）。

（2）模块接口

信号名	方向	描述
inp	I	输入地址。
Clk	I	时钟信号。
Reset	I	复位信号。
outp[31:0]	O	当前的 32 位地址。

（3）功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，PC 被设置为 0x00003000。
2	转移地址	当复位信号为 0 时，PC 将输入的地址转移给 IM。

2.IM 模块

（1）基本描述

IM 的主要功能是根据当前的地址，将地址相对应的 32 位数据导出并进行后续操作。

（2）模块接口

信号名	方向	描述
add	I	输入当前的地址。
cod	O	输出地址所对应的 32 位数据。

（3）功能定义

序号	功能名称	功能描述
1	取指令	根据当前输入的地址，从存储器里取出对应的数据进行操作。

3.GRF 模块

（1）基本描述

GPR 主要功能是利用寄存器以实现对数据的取出和存入操作。通过一个 32 位 MIPS 指令对指令中的指定寄存器的值进行读或写操作。

（2）模块接口

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号。
Rw	I	寄存器写信号，当信号有效时，可以向寄存器里写入数据。
Rr1[25:21]	I	读入数据的寄存器地址 1。
Rr2[20:16]	I	读入数据的寄存器地址 2。
Wr[4:0]	I	写寄存器的地址。
Wd[31:0]	I	写寄存器的数据。
Rd1[31:0]	O	rr1 中数据的输出。
Rd2[31:0]	O	rr2 中数据的输出。

（3）功能定义

序号	功能名称	功能描述
1	取数	取出 RS 接口和 RT 接口对应的寄存器中存储的值。
2	写数	将指定的 32 位数据写入指定的寄存器中。

4.ALU 模块

（1）基本描述

ALU 的功能是对指定的两个 32 位数进行加、减、或运算以及对两个数进行大小比较。

（2）模块接口

信号名	方向	描述
I1	I	第一个 32 位数据的输入。
I2	I	第二个 32 位数据的输入。
Aluop[1:0]	I	alu 输出数据的控制信号。 00：输出加法结果。 01：输出减法结果。 10：输出或运算结果。
R1	O	运算输出结果。
Z	O	两个数判断大小的结果。

（3）功能定义

序号	功能名称	功能描述
1	加法运算	取出 RS 接口和 RT 接口对应的寄存器中存储的值。
2	减法运算	将指定的 32 位数据写入指定的寄存器中。
3	或运算	对输入的两个数进行或运算。
4	比较大小	将输入的两个数进行比较大小。

5.DM 模块

（1）基本描述

DM 的功能是对数据存储器里指定的地址里的数据进行读或写操作。

（2）模块接口

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号。
Addr[11:2]	I	写入或读出数据的地址。
Memw	I	写数据操作信号。
Inp[31:0]	I	写入的数据。
Outp[31:0]	O	读出的数据。

（3）功能定义

序号	功能名称	功能描述
1	写数据	将 inp 写入向 addr 指定的地址中。
2	读数据	将 addr 地址中的数据输出到 outp。

6. EXT 模块

（1）基本描述

EXT 的功能是对一个十六位二进制数进行扩展，并根据扩展信号判断进行符号扩展还是零扩展。

（2）模块接口

信号名	方向	描述
Extd[15:0]	I	需要扩展的 16 位数据。
Ex2	I	符号扩展信号。
Ex1	I	加零信号。
Exo[31:0]	O	扩展完毕的 32 位数据。

（3）功能定义

序号	功能名称	功能描述
1	符号扩展	ext2 为 1 时进行符号扩展。
2	高位扩展	ext2 为 0 时，且 ext1 为 1 时，进行高位零扩展。
3	低位扩展	ext2 为 0 时，且 ext1 为 0 时，进行低位零扩展。

7.npc 模块

（1）基本描述

npc 的功能是对下一次指令的地址进行计算。

（2）模块接口

信号名	方向	描述
Dpc[31:0]	I	当前的地址。
Imm[25:0]	I	当前地址对应的数据的后 26 位数
Wb	I	判断是否为 beq。
Wja	I	判断是否为 jal。
Wjr	I	判断是否为 jr
Zero	I	判断两个操作数是否相等
Dra[31:0]	I	31 号寄存器的值
Pca4[31:0]	O	当前地址偏移四后的结果
Npc[31:0]	O	下一条指令地址

（3）功能定义

序号	功能名称	功能描述
1	计算地址	根据当前指令和当前指令类型判断下一指令地址。
2	存储 31 号寄存器	计算当前指令的下一条指令地址，并传给 GRF。

8.control 模块

（1）基本描述

control 模块的功能是对各部分的运算、存储、读取等进行控制。

（2）模块接口

信号名	方向	描述
Op[5:0]	I	六位 op 信号。
fun[5:0]	I	六位 func 信号。
regdst	O	写地址控制。
alusrc	O	ALU 操作数控制。
regw	O	GRF 写控制。
memw	O	DM 写控制。
whbeq	O	判断指令是否为 beq 指令。
memtor	O	GRF 写入数据控制。

Whext	O	判断位扩展的方式。
extw	O	判断是高位扩展还是低位扩展。
whjal	O	判断指令是否为 jal 指令。
whjr	O	判断指令是否为 jr 指令。
Aluop[1:0]	O	判断 alu 执行的操作。

（3）功能定义

序号	功能名称	功能描述
1	总体控制	根据输入信号，判断需要控制的数据，从而得到我们想要的结果。

（4）控制器真值表

指令	addu	subu	ori	lw	sw	beq	lui	jal	jr
func	100001	100011	n/a						001000
op	000000	000000	001101	100011	101011	000100	001111	000011	000000
regdst	1	1	0	0	x	x	0	x	x
alusrc	0	0	1	1	1	0	1	x	x
regw	1	1	1	1	0	0	1	1	0
memw	0	0	0	0	1	0	0	0	0
whbeq	0	0	0	0	0	1	0	0	0
memtor	0	0	0	1	x	x	0	x	x
Whext	x	x	0	1	1	x	0	x	x
extw	x	x	1	x	x	x	0	x	X
whjal	0	0	0	0	0	0	0	1	0
whjr	0	0	0	0	0	0	0	0	1
Aluop[1:0]	00	01	10	00	00	01	00	x	x

二、模块间连接

```
module mips(clk,reset);
    input clk;  //时钟信号
    input reset;  ///复位信号
    wire [31:0] p1;
    wire [31:0] p2;
    wire [31:0] p3;
    wire [31:0] p4;
    wire [31:0] r1;
    wire [31:0] r2;
    wire [31:0] r3;
    wire [31:0] d1;
    wire [31:0] d2;
    wire [31:0] re1;
    wire [31:0] re2;
    wire [31:0] re3;
    wire [4:0] wri;
    wire [31:0] wad;
    wire regd;
    wire alus;
    wire regwr;
    wire memwr;
    wire beq;
    wire mtr;
    wire ex;
    wire exw;
    wire ja;
    wire jr;
    wire [1:0] aop;
    wire z1;
    PC qpc(.clk(clk),.reset(reset),.inp(p3),.outp(p1));
    IM qim(.add(p1[11:2]),.cod(p2));
pcc
qpcc(.dpc(p1),.wb(beq),.wja(ja),.wjr(jr),.imm(p2[25:0]),.pca4(p4),.np
c(p3),.dra(r1),.zero(z1));
    GRF
qgrf(.clk(clk),.reset(reset),.rw(regwr),.rr1(p2[25:21]),.rr2(p2[20:16
]),.wr(wri),.wd(re3),.rd1(r1),.rd2(r2),.whjal(ja),.jalad(p4));
```

```
    EXT qext(.extd(p2[15:0]),.ex1(exw),.ex2(ex),.exo(d1));
    ALU qalu(.i1(r1),.i2(d2),.aluop(aop),.r1(re1),.z(z1));
    DM
qdm(.clk(clk),.reset(reset),.addr(re1),.memw(memwr),.inp(r2),.outp(re
2));
    control
qcontrol(.op(p2[31:26]),.fun(p2[5:0]),.regdst(regd),.alusrc(alus),.re
gw(regwr),.memw(memwr),.whbeq(beq),.memtor(mtr),.whext(ex),.extw(exw)
,.whjal(ja),.whjr(jr),.aluop(aop));
    MUX alumux(.A(r2),.B(d1),.sel(alus),.result(d2)); //多路选择器
    MUX2 regmux(.A(p2[20:16]),.B(p2[15:11]),.sel(regd),.result(wri));
    MUX datamux(.A(re1),.B(re2),.sel(mtr),.result(re3));
endmodule
```

四、Mips 程序测试

```
lui $t0,0x36a0  #立即数 0x36a0 加载至 t0 寄存器高位。
lui $t1,0x002a  #立即数 0x002a 加载至 t1 寄存器高位。
ori $s0,$zero,0x0003  #zero 寄存器中的数与立即数 0x00000003 进行或运算，结果储
存在 t3 寄存器中。
ori $s1,$zero,0x0001  #zero 寄存器中的数与立即数 0x00000001 进行或运算，结果储
存在 s1 寄存器中。
ori $s2,$zero,0x0002  #zero 寄存器中的数与立即数 0x00000002 进行或运算，结果储
存在 s2 寄存器中。
ori $t4,$zero,0x0004  #zero 寄存器中的数与立即数 0x00000004 进行或运算，结果储
存在 t4 寄存器中。
loop:sw $t0,0($t3)  #把 t0 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 0 所
指向的 RAM 中。
subu $t0,$t0,$t1  #t0 寄存器中的值减去 t1 寄存器中的值后存到 t0 寄存器中。
addu $t3,$t3,$t4  #t4 寄存器中的值加上 t3 寄存器中的值后存到 t3 寄存器中。
addu $s2,$s2,$s1  #s1 寄存器中的值加上 s2 寄存器中的值后存到 s2 寄存器中。
beq $s2,$s0,loop2  #判断 s0 的值和 s2 的值是否相等，相等转 loop。
loop3:addu $s3,$s2,$s1  #s1 寄存器中的值加上 s2 寄存器中的值后存到 s3 寄存器中。
ori $s3,$s3,0x34a2  #s3 寄存器中的数与立即数 0x000034a2 进行或运算，结果储存在
s3 寄存器中。
subu $s3,$s3,$s0 #s3 寄存器中的值减去 s0 寄存器中的值后存到 s3 寄存器中。
addu $t5,$s3,$s1 #s1 寄存器中的值加上 s3 寄存器中的值后存到 t5 寄存器中。
addu $s5,$s3,$t5 #s3 寄存器中的值加上 t5 寄存器中的值后存到 s5 寄存器中。
```

subu $t8,$s5,$s2 #s5 寄存器中的值减去 s2 寄存器中的值后存到 t8 寄存器中。

jr $ra #跳转回 ra 中存储的指令

loop2:subu $t3,$t3,$t4 #t3 寄存器中的值减去 t4 寄存器中的值后存到 t3 寄存器中。

lw $t5,0($t3) #把 t3 寄存器的值加偏移量 4 当作地址读取存储器中的值存入 t5。

ori $t6,$t5,0xc020 #zero 寄存器中的数与立即数 0x00000003 进行或运算，结果储存在 t3 寄存器中。

sw $t6,4($t3) #把 t6 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 4 所指向的 RAM 中。

subu $s2,$s2,$s1 #s2 寄存器中的值减去 s1 寄存器中的值后存到 s2 寄存器中。

beq $s2,$s1,loop2 #判断 s2 的值和 s1 的值是否相等，相等转 loop2。

jal loop3 #跳转到 loop3，并将当前地址加 4 存到 ra 中

addu $t3,$t3,$t4 #t4 寄存器中的值加上 t3 寄存器中的值后存到 t3 寄存器中。

lui $s7,0x33a4 #立即数 0x33a4 加载至 s7 寄存器高位。

sw $s7,4($t3) #把 s7 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 4 所指向的 RAM 中。

addu $s1,$s1,$s1 #s1 寄存器中的值加上 s1 寄存器中的值后存到 s1 寄存器中。

lw $s6,0($t3) #把 t3 寄存器的值加偏移量 0 当作地址读取存储器中的值存入 s6。

subu $a0,$s6,$s1 #s6 寄存器中的值减去 s1 寄存器中的值后存到 a0 寄存器中。

机器码：3c0836a0 3c09002a 34100003 34110001 34120002 340c0004 ad680000 01094023 016c5821 02519021 12500007 02519821 367334a2 02709823 02716821 026da821 02b2c023 03e00008 016c5823 8d6d0000 35aec020 ad6e0004 02519023 1251fffa 0c000c0b 016c5821 3c1733a4 ad770004 02318821 8d760000 02d12023

预期结果：

$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x36a0c01e
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x36760000
$t1	9	0x002a0000
$t2	10	0x00000000
$t3	11	0x00000004
$t4	12	0x00000004
$t5	13	0x000034a1
$t6	14	0x36a0c020
$t7	15	0x00000000
$s0	16	0x00000003
$s1	17	0x00000002
$s2	18	0x00000002
$s3	19	0x000034a0
$s4	20	0x00000000
$s5	21	0x00006941
$s6	22	0x36a0c020
$s7	23	0x33a40000
$t8	24	0x0000693f
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003064
pc		0x0000307c
hi		0x00000000
lo		0x00000000

Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x36a00000	0x36a0c020	0x33a40000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000

实际结果：

00003064	00000000
00000000	00000000
00000000	00000000
00000000	0000693F
33A40000	36A0C020
00006941	00000000
000034A0	00000002
00000002	00000003
00000000	36A0C020
000034A1	00000004
00000004	00000000
002A0000	36760000
00000000	00000000
00000000	36A0C01E
00000000	00000000
00000000	00000000

00000000	00000000
00000000	00000000
00000000	00000000
00000000	00000000
00000000	33A40000
36A0C020	36A00000

五、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 addr 信号从 ALU 经过加法运算传输过来，原地址和偏移量是四的倍数，计算结果也是 4 的倍数，而存储器的地址是一串连续的数，所以需要将结果的后两位 0 移除，并且位数要为 10，所以是[11:2]。

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？

IM（指令存储器）、DM（数据存储器）、GRF（寄存器堆）

IM 复位回到程序首地址；

DM、GRF 需要将数据全部清零，如果不清零，会对下一次测试的数据有影响而导致错误发生。

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。

二进制编码：ori = 001101；

独热编码：

ori=00000000000000000000000000000000000000000000000000010000000000

00；

格雷码：001011；

4.根据你所列举的编码方式，说明他们的优缺点。

二进制编码优点：直观，容易实现，运算规则简单。缺点：位数较多，太繁琐。

独热编码优点：直观，1 的位置就是表示的数字。缺点：位数太多，过于复杂。

格雷码：优点：可靠性编码，错误最小化的编码方式。缺点：无法直观表示一个十进制数。

5.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 。

如果不考虑溢出，就不会执行 SignalException(IntegerOverflow)而去执行 GPR[rt] ← temp31..0，而 temp31..0 与 GPR[rs] + sign_extend(immediate)的结果相

同。

6.根据自己的设计说明单周期处理器的优缺点。

优点：在一个周期内计算出结果，快捷，方便。

缺点：不同类型的指令需要的原件不同，时间消耗将有高有低。。

例如：addu 指令需要 IM、GFR、ALU、GFR，jr 指令只需要 IM、GFR、npc 即可，时间消耗不同。

7.简要说明 jal、jr 和堆栈的关系。

堆栈与递归等价，在递归时需要用到函数的调用，函数的调用需要 jal 跳转和 jr 的地址返回，将跳转的地址压入堆栈，就可以实现递归。