单周期处理器开发

一、 模块定义

1. pc 模块定义

1) 基本描述

pc 的主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，pcAddr 指向 0x00003000，即第一条指令的地址。

2) 模块接口

信号名	方向	描述
npcAddr[31:0]	I	下条指令的地址
clk	I	时钟信号
reset	I	复位信号（1：复位；0：无效）
pcAddr[31:0]	O	32 位指令存储器地址

3) 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，pcAddr 被设置为 0x00003000
2	保存 npcAddr 并输出	在每个时钟信号 clk 的上升沿保存下条指令的地址并输出

2. mux2 模块定义

1) 基本描述

2 选 1 数据选择器。

2) 模块接口

信号名	方向	描述
d0[WIDTH-1:0]	I	输入信号 0
d1[WIDTH-1:0]	I	输入信号 1
s	I	选择信号
y[WIDTH-1:0]	O	输出信号

3) 功能定义

根据输入信号 s 选择相应的输出信号，s 为 0 时输出 d0，为 1 时输出 d1，参数 WIDTH 为信号的宽度，默认为 32 位。

3. alu 模块定义

1) 基本描述

算术逻辑单元，根据 aluCtrlSignal 对输入的两个操作数进行算术运算或逻辑运算。

2) 模块接口

信号名	方向	描述
input1[31:0]	I	操作数 1
input2[31:0]	I	操作数 2
aluCtrlSignal[3:0]	I	ALU 控制信号
allures[31:0]	O	运算结果
zero	O	标记运算结果是否为 0

3) 功能定义

序号	功能名称	功能描述
1	与运算	对两个操作数进行与运算
2	或运算	对两个操作数进行或运算
3	加法运算	对两个操作数进行加法运算
4	减法运算	对两个操作数进行减法运算
5	小于则置 1 运算	若 input1 小于 input2，则置 aluRes 为 1
6	或非运算	对两个操作数进行或非运算

4. dm_2k 模块定义

1) 基本描述

数据存储器为 RAM 类型的存储器，在写入时需要时钟上升沿和控制信号。分配 512 个字存储空间，字长为 32 位，容量为 2K。

2) 模块接口

信号名	方向	描述
addr[10:2]	I	9 位的目标地址，用来定位数据
din[31:0]	I	输入数据，即要写入数据存储器中的数据
we	I	写控制信号，为 1 时可写
clk	I	时钟信号
dout[31:0]	O	输出数据，即从数据存储器中读出的数据

3) 功能定义

用来存储数据，读取数据时不需要控制信号，写入数据时需要时钟上升沿和控制信号。

5. npc 模块定义

1) 基本描述

根据输入信号来计算下一个 PC 值。

2) 模块接口

信号名	方向	描述
inst[25:0]	I	当前指令的低 26 位，用来计算 j 指令跳转地址
immi[31:0]	I	由立即数进行符号扩展后得到的 32 位操作数
branch	I	分支指令控制信号
zero	I	标记 ALU 运算结果是否为 0，用来判断是否分支
jump	I	跳转指令控制信号
pcAddr[31:0]	I	当前 PC 值
npcAddr[31:0]	O	下一个 PC 值

3) 功能定义

根据输入的控制信号以及 zero 的值从 PC+4、分支地址和跳转地址中选择下一个 PC 的值。

6. aluCtrl 模块定义

1) 基本描述

根据 2 位的 aluOp 和 6 位的 funct 功能字段生成 4 位的 ALU 控制信号。

2) 模块接口

信号名	方向	描述
aluOp[1:0]	I	主控制单元生成的 ALUOp
funct[5:0]	I	对于 R 型指令，指令的低 6 位为 funct 字段
aluCtrlSignal[3:0]	O	生成的 4 位 ALU 控制信号

3) 功能定义

序号	指令操作	{aluOp, funct}	aluCtrlSignal	ALU 动作
1	取字/存储字/I 型加	00xxxxxx	0010	加
2	相等分支	01xxxxxx	0110	减
3	R 型加	10xx0000	0010	加
4	R 型减	1xxx0010	0110	减
5	R 型与	10xx0100	0000	与
6	R 型或	10xx0101	0001	或
7	R 型小于则置 1	10xx1010	0111	小于则置 1
8	I 型与	11xxxxxx	0000	与

7. im_4k 模块定义

1) 基本描述

指令存储器为 ROM 类型的存储器，可根据 PC 值读取相应指令并输出。分配 1024 个字存储空间，字长为 32 位，容量为 4K。

2) 模块接口

信号名	方向	描述
addr[11:2]	I	需要读取的指令的地址
dout[31:0]	O	读取到的指令

PS: 由于 PC 的初始地址为 0x00003000，而指令存储器大小为 4K，故利用 PC 的 2~11 位即可忽略 PC 的初始地址并确定要读取的指令。

3) 功能定义

用来存储指令，初始化时从文件"code.txt"中读入指令，之后只允许根据地址从指令存储器中读取指令。

8. signExt 模块定义

1) 基本描述

符号扩展模块，将输入的 16 位数据扩展为 32 位。

2) 模块接口

信号名	方向	描述
before[15:0]	I	输入的 16 位数据
after[31:0]	O	输出的 32 位数据

9. regFile 模块定义

1) 基本描述

寄存器文件，分配 32 个 32 位的寄存器空间。

2) 模块接口

信号名	方向	描述
rsAddr[4:0]	I	rs 寄存器的地址
rtAddr[4:0]	I	rt 寄存器的地址
clk	I	时钟信号

reset	I	复位信号（1：复位；0：无效）
regWriteAddr[4:0]	I	要写入的寄存器的地址
regWriteData[31:0]	I	要写入寄存器的数据
regWriteEn	I	寄存器写控制信号
rsData[31:0]	O	rs 寄存器的数据
rtData[31:0]	O	rt 寄存器的数据

3) 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，将 32 个寄存器全部清零
2	读取寄存器	根据 rs 和 rt 的地址，从寄存器中读取 rs 和 rt 的数据
3	写入寄存器	在时钟上升沿且写控制信号有效时，将给定数据写入目的寄存器中

10.ctrl 模块定义

1) 基本描述

主控制单元，根据操作码生成控制信号。

2) 模块接口

信号名	方向	描述
opcode[5:0]	I	操作码，即指令的 26~31 位
regDst	O	控制写寄存器的目标寄存器来源
aluSrc	O	控制第二个 ALU 操作数的来源
memtoReg	O	控制写入寄存器的数据来源
regWrite	O	寄存器堆写使能
branch	O	标记是否为分支指令
jump	O	标记是否为跳转指令
aluOp[1:0]	O	ALUOp 控制信号

3) 功能定义

序号	功能名称	功能描述
1	R 型指令	产生 R 型指令的控制信号

2	lw	产生 lw 指令的控制信号
3	sw	产生 sw 指令的控制信号
4	beq	产生 beq 指令的控制信号
5	j	产生 j 指令的控制信号
6	addi	产生 addi 指令的控制信号
7	andi	产生 andi 指令的控制信号

11.mips 模块定义

1) 基本描述

实例化以上各个模块，并将各模块用线连接起来。

2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	复位信号（1：复位；0：无效）

12.test_tb 模块定义

1) 基本描述

对 mips 模块进行测试。初始化时产生 25ns 的复位信号，之后产生周期为 40ns 的时钟信号。

二、 程序测试

1. 测试程序

```
1         addi    $t0, $0, 1          # for(i = 1; i <= 10; i++)
2         addi    $t1, $0, 10         # 10 is the upper bound
3  loop1: addi    $t0, $t0, 1         # i++
4         beq     $t0, $t1, exit1     # if i == 10 then exit
5         j       loop1               # jump to loop1
6  exit1: sw      $t0, 0($0)          # save $t0 == 10 in address 0
7         lw      $t3, 0($0)          # load word in address 0, $t3 == 10
8         add     $t0, $t0, $t3       # $t0 = $t0 + $t3, $t0 == 20
9         addi    $t1, $0, 1          # $t1 = 1
10        addi    $t3, $0, 17         # $t3 = 17
11 loop2: sub     $t0, $t0, $t1       # $t0 = $t0 - 1
12        slt     $t2, $t0, $t3       # $t0 < 17 ?
13        beq     $t2, $0, loop2      # for(i = 20; i >= 16; i--)
14        sw      $t0, 4($0)          # save $t0 == 16 in address 4
15        addi    $t1, $0, 15         # $t1 = 15
16        or      $t0, $t0, $t1       # $t0 = $t0 | $t1, $t0 == 31
17        sw      $t0, 8($0)          # save $t0 == 31 in address 8
18        andi    $t4, $t0, 0x13      # test for andi
19        sub     $t1, $t3, $t1       # $t1 = $t3 - $t1, $t1 == 2
20        and     $t0, $t0, $t1       # $t0 = $t0 & $t1, $t0 == 2
21        sw      $t0, 12($0)         # save $t0 == 2 in address 12
```

2. 期望结果

1) 寄存器堆

$t0	8	0x00000002
$t1	9	0x00000002
$t2	10	0x00000001
$t3	11	0x00000011
$t4	12	0x00000013

2) 数据存储器

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00000000	0x0000000a	0x00000010	0x0000001f	0x00000002

3. 测试结果

1) 寄存器堆

/test_tb/mips/regFile/regs[8]	00000002	00000002
/test_tb/mips/regFile/regs[9]	00000002	00000002
/test_tb/mips/regFile/regs[10]	00000001	00000001
/test_tb/mips/regFile/regs[11]	00000011	00000011
/test_tb/mips/regFile/regs[12]	00000013	00000013

2) 数据存储器

/test_tb/mips/dm/dm[3]	00000002	00000002
/test_tb/mips/dm/dm[2]	0000001f	0000001f
/test_tb/mips/dm/dm[1]	00000010	00000010
/test_tb/mips/dm/dm[0]	0000000a	0000000a

三、 问题与建议

1. 在刚写完程序准备进行仿真时，点击总是提示 Error loading design，双击也看不到具体的出错信息。后来经过仔细检查，发现我在实例化模块时将模块的名称错写成了.v 文件的名称。经测试，ModelSim 在编译时不会检查模块名称对应的模块是否存在，因此编程时应注意避免写错。
2. 按照课本 201 页对跳转指令数据通路的描述，当 Jump 控制信号为 0 时会进行跳转，但实际应该是控制信号为 1 时跳转，因此我认为图中右上角的 Mux 的 0 和 1 标反了。查阅英文原版（329 页）后发现中文版确实标反了。