CPU设计文档

一.模块规格

1.IFU

a.模块接口

表 1

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| nPC\_sel | I | beq指令判断信号  0:非beq指令  1:是beq指令 |
| zero | I | R[rs] 和R[rt]大小比较信号  0: R[rs] != R[rt]  1: R[rs] = R[rt] |
| imm[15:0] | I | 立即数 |
| Reset | I | 异步复位信号  1：复位  0：无效 |
| Clk | I | 时钟信号 |
| Instr[31:0] | O | 32位指令 |

b.功能定义

表 2

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算NPC | 若nPC\_sel和zero均为1：  PC←PC+4+signed\_ext(imm||02)  其他：  PC←PC+4 |
| 2 | 取出指令Instr | 根据PC寄存器的地址从ROM取出对应指令 |
| 3 | 复位 | 若复位信号若有效则清零PC |

2.GRF

a.模块接口

表 3

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[4:0] | I | 读入的第一个寄存器的地址 |
| A2[4:0] | I | 读入的第二个寄存器的地址 |
| A3[4:0] | I | 写入的寄存器序号 |
| WD[31:0] | I | 写入寄存器堆的32位信号 |
| WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| Clk | I | 时钟信号 |
| Reset | I | 异步复位信号  1：复位  0：无效 |
| RD1[31:0] | O | 输出的A1对应的寄存器的32位数据 |
| RD2[31:0] | O | 输出的A2对应的寄存器的32位数据 |

b.功能定义

表 4

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 读出A1、A2地址对应寄存器中存储的数据到RD1,RD2 |
| 2 | 写数据 | 时钟上升沿到来时若WE有效，将WD写入A3对应的寄存器中 |
| 3 | 复位 | Reset信号有效时，所有寄存器的数值被清零 |

3.ALU

a.模块接口

表 5

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 参与计算的第一个32位数据 |
| B[31:0] | I | 参与计算的第二个32位数据 |
| ALUOp[1:0] | I | 计算类型  00：加法  01：减法  10：或运算 |
| C[31:0] | O | 计算得到的32位数据 |

b.功能定义

表 6

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法运算 | C=A+B |
| 2 | 减法运算 | C=A-B |
| 3 | 或运算 | C=A | B |
| 4 | 比较大小 | 当计算类型为减法时，  C>0: A>B  C=0: A=B  C<0: A<B |

4.DM

a.模块接口

表 7

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[4:0] | I | 读或写DM的地址 |
| DataIn[31:0] | I | 写入的32位数据 |
| DMwrite | I | 写使能信号：  0：不进行写入操作  1：进行写入操作 |
| Clk | I | 时钟信号 |
| Reset | I | 异步复位信号  1：复位  0：无效 |
| DataOut[31:0] | O | 读出对应地址的 32位数据 |

b.功能定义

表 8

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读存储器 | 将Addr对应的RAM的地址中的数据取出并输出 |
| 2 | 写存储器 | 时钟上升沿到来时，若DMwrite有效，将32位数据写入Addr对应的RAM的地址区域 |
| 3 | 复位 | 复位信号有效时，将RAM的数据清零 |

5.EXT

a.模块接口

表 9

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16[15:0] | I | 被扩展的16位立即数 |
| EXTOp[1:0] | I | 扩展类型选择信号  00：零扩展  01：符号扩展  10：将立即数加载至高位，低位补零 |
| imm32[31:0] | O | 扩展后的32位立即数 |

b.功能定义

表 10

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零扩展 | imm32=zero\_ext(imm16) |
| 2 | 符号扩展 | imm32=signed\_ext(imm16) |
| 3 | 加载至高位，低位补零 | imm32=imm16||016 |

二.控制器设计

1.真值表

表 11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| func | 10 0001 | 10 0011 | n/a | | | | |
| op | 000000 | 00 0000 | 00 1101 | 10 0011 | 10 1011 | 000100 | 00 1111 |
|  | addu | subu | ori | lw | sw | beq | lui |
| RegDst | 1 | 1 | 0 | 0 | X | X | 0 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| MemtoReg | 0 | 0 | 0 | 1 | X | X | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ExtOp<1:0> | X | X | 00 | 01 | 01 | X | 10 |
| ALUctr<1:0> | 00 | 01 | 10 | 00 | 00 | 01 | 00 |

2.控制信号说明

RegDst:寄存器堆写入地址的选择信号（rd or rs）

0：选择rt作为写入地址

1：选择rd作为写入地址

ALUSrc:参与ALU运算的第二个数据的选择信号

0：选择rt对应寄存器的值

1：选择扩展后的立即数

MemtoReg:写入寄存器堆的数据的选择信号

0：选择ALU的输出

1：选择DM的输出

RegWrite:寄存器堆的写使能信号

0：不向寄存器堆写入数据

1：向寄存器堆写入数据

MemWrite:DM的写使能信号

0：不向DM写入数据

1：向DM写入数据

nPC\_sel: beq指令的判断信号

0：不是beq指令

1：是beq指令

ExtOp<1:0>:16位立即数位扩展选择信号

00：零扩展

01：符号扩展

10:将16位立即数加载至高位，低位补零

ALUctr<1:0>:ALU运算类型选择信号

00：加法

01：减法

10：或运算

三.测试程序

1.测试代码

（测试程序中的数据存储器的起始地址为0x00002000，指令存储器的地址为0x00000000）

1 ori $s0,$0,0x2000

2 ori $s1,$0,1

3 addu $s2,$s0,$s1

4 sw $s1,4($s0)

5 loop:

6 subu $t0,$s1,$0

7 lui $t1,3

8 nop

9 lw $t2,4($s0)

10 beq $s1,$t2,loop

2.测试期望

1.将0号寄存器的值和0x2000做或运算，得到的结果0x00002000写入16号寄存器。

2.将0号寄存器的值和1做或运算，得到的结果0x00000001写入17号寄存器。

3.对16号寄存器和17号寄存器的值做加法运算，得到结果0x00002001写入18号寄存器。

4.将17号寄存器的值0x00000001写入DM的地址为01的区域。

6.对17号寄存器和0号寄存器的值做减法运算，得到结果0x00000001写入8号寄存器。

7.将16位数据0x0003加载至高位，低位补零，得到的32位数据0x00030000写入9号寄存器。

8.不进行任何操作。

9.从DM的地址为01的区域取出数据0x00000001并写入10号寄存器。

10.17号和18号寄存器的值相等，跳转到指令subu $t0,$s1,$0。

思考题

一.模块规格

1.若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

答：30位PC优点：由于32位PC的低两位总为零，因此30位PC可以免去低两位的零。

32位PC优点：由于数据、指令都是32位的，所以使用32位PC可以免去新的硬件设计需求。

2.现在我们的模块中 IM使用ROM， DM使用RAM， GRF使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

答：合理。IM是指令存储器只需根据PC值取出指令即可，不需要写入的功能，所以IM使用ROM。而DM是数据存储器，既需要能读出相应地址中存储的数据，也需要具备将数据写入对应地址的功能，但不需要特别快的访问速度，所以DM使用RAM。GRF由于使用频率较高，需要较快的访问速度，所以使用寄存器。

二.控制器设计

1.结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

答：RegDst=op[5]’·op[4]’·op[3]’·op[2]’·op[1]’·op[0]’·

func[5]·func[4]’·func[3]’·func[2]’·func[0]’

ALUSrc= op[5]’·op[4]’·op[3]·op[2]·op[1]’·op[0]+

op[5]·op[4]’·op[2]’·op[1]·op[0]

MemtoReg=op[5]·op[4]’·op[3]’·op[2]’·op[1]·op[0]

RegWrite=(op[5]’+op[4]+op[3]’+op[2]+op[1]’+op[0]’)·

(op[5]+op[4]+op[3]+op[2]’+op[1]+op[0])

nPC\_sel=op[5]’·op[4]’·op[3]’·op[2]·op[1]’·op[0]’

ExtOp=op[5]·op[4]’·op[2]’·op[1]·op[0]

2.充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。

答：RegDst=op[5]’·op[4]’·op[3]’·op[1]’·op[0]’·

func[5]·func[4]’·func[3]’·func[2]’·func[0]’

ALUSrc=op[5]’·op[4]’·op[3]·op[2]·op[1]’·op[0]+

op[5]·op[4]’·op[2]’·op[1]·op[0]

MemtoReg=op[5]·op[4]’·op[2]’·op[1]·op[0]

RegWrite=(op[5]’+op[4]+op[3]’+op[2]+op[1]’+op[0]’)·

(op[5]+op[4]+op[3]+op[2]’+op[1]+op[0])

nPC\_sel=op[5]’·op[4]’·op[3]’·op[2]·op[1]’·op[0]’

ExtOp=op[5]·op[4]’·op[2]’·op[1]·op[0]

3.事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

答：nop指令不进行任何操作，也不对GRF和DM进行写入操作，即控制信号RegWrite和MemWrite为0，其他控制信号为X，根据其他指令设计的与或门阵列可以使nop指令输出符合要求的控制信号。

三.测试CPU

1.前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

答：在MARS中将data base address 设置为了0x00002000，而由于RAM为32bit\*32，只需要输入5位地址即可，因此可以取指令码计算出的写入地址的低位（2-6位）（MARS是按字节寻址，而RAM按字寻址），而不必修改指令码的数据偏移。

2.除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

答：优点：

1）测试者不必考虑如何去获得测试向量。

2）形式验证是对所有可能情况进行验证，而不是只对其中某些情况进 行验证，所以可以弥补测试验证的不足。

3）形式验证可以从系统级到门级验证，且验证时间短，可以尽快发现 和改正设计错误，从而缩短设计周期。

缺点：

形式验证目前不能有效验证电路的性能，如电路的时延和功耗。