计算机组成原理实验报告（P3）

各个部分的功能描述如下：

EXT:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 端口 | In/out | 名称 | 位数 | 描述 |
| 1 | imm | in | 立即数 | 16 | 指令的后16位，立即数 |
| 2 | ext\_sel | in | 选择 | 2 | 0为无符号扩展，以及不需要扩展的其他指令,1为有符号扩展，2为有符号扩展并且左移2位，3为左移16位 |
| 3 | out | out | 输出 | 32 | 扩展后的32位 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | Zero\_ext | 无符号扩展成32位 |
| 2 | Sign\_ext | 有符号扩展成32位 |
| 3 | Immi||0 16 | 左移16位 |

DM：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 端口 | In/out | 名称 | 位数 | 描述 |
| 1 | addr | in | 写入地址 | 32 | 写入的地址 |
| 2 | in | in | 写入数据 | 32 | 准备写入的数据 |
| 3 | memwr | in | 写使能 | 1 | 0时不能写入，1时可以 |
| 4 | reset | in | 重置 | 1 | 异步复位，清空RAM |
| 5 | out | out | 读出值 | 32 | 读出的数据 |
| 6 | wr\_addr | out | 写入地址 | 32 | 给评测机输出写入地址，其实不需要 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 复位信号有效时，所有数据变为0x00000000 |
| 2 | 读 | 根据地址读出数据 |
| 3 | 写 | 根据地址写入数据 |

ALU：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 端口 | In/out | 名称 | 位数 | 描述 |
| 1 | busa | in | 输入a | 32 | 第一个输入 |
| 2 | busb | in | 输入b | 32 | 第二个输入 |
| 3 | Alu\_ctr | in | 选择 | 2 | 0时加，1时减，2时或,3时输出输入b |
| 4 | Alu\_out | out | 输出 | 32 | Alu的计算结果 |
| 5 | zero | out | zero | 1 | 两输入相等时为1，否则为0 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 加 | 无符号加（不判断溢出） |
| 2 | 减 | 无符号减（不判断溢出） |
| 3 | 或 | 或 |
| 4 | 比较 | 比较是否相等 |

IFU：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 端口 | In/out | 名称 | 位数 | 描述 |
| 1 | reset | in | 重置 | 1 | 异步复位，复位pc |
| 2 | b\_in | in | offest | 32 | 偏移量 |
| 3 | npc\_sel | in | 选择 | 2 | 0时执行+4,1时执行+4+offest |
| 4 | zero | in | zero | 1 | B指令时判断rs，rt是否相等，相等是为1，否为0 |
| 5 | out | out | 输出 | 32 | 读到的指令 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 复位信号有效时pc变为0x00000000 |
| 2 | 取指令 | 根据pc从IM中取出指令 |
| 3 | 计算下一条指令地址 | 如果是b指令，根据zero判断是否加offest；如果不是，pc=pc+4 |

GRF：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 端口 | In/out | 名称 | 位数 | 描述 |
| 1 | rt | in | rt | 32 | 指令中的rt |
| 2 | rs | in | rs | 32 | 指令中的rs |
| 3 | rd | in | rd | 32 | 指令中的rd |
| 4 | reg\_dst | in | 写选择 | 1 | 0时回写寄存器为rt，1时为rd |
| 5 | reset | in | 重置 | 1 | 异步复位，清空所有寄存器 |
| 6 | write sth | in | 回写数据 | 32 | 回写的数据 |
| 7 | reg\_write | in | 写使能 | 1 | 回写使能，1时可以回写，0时不行 |
| 8 | busa | out | 输出a | 32 | 第一个输出（rs） |
| 9 | busb | out | 输出b | 32 | 第二个输出（rt） |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 复位信号有效时，所有寄存器清零 |
| 2 | 读 | 将rs，rt寄存器的值读出 |
| 3 | 写 | 根据写使能决定是否写寄存器；  根据写选择决定写rt还是rd寄存器 |

控制器

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| fun | op | 指令 | Ext\_sel | Alu\_src | Alu\_ctr | Npc\_sel | memwr | memtoreg | regdst | regwr |
| 100000 | 000000 | addu | x | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 100011 | 000000 | subu | x | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|  | 001101 | ori | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 1 |
|  | 100011 | lw | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|  | 101011 | sw | 1 | 1 | 0 | 0 | 1 | 0 | x | 0 |
|  | 000100 | beq | 1 | 0 | x | 1 | 0 | 0 | x | 0 |
|  | 001111 | lui | 3 | 1 | 3 | 0 | 0 | 0 | 0 | 1 |
| 000000 | 000000 | nop | x | 0 | x | 0 | 0 | 0 | x | 0 |

注：实际操作时x都为0

思考题

1. 若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

30位pc的优势：节省资源，在本题中，pc并没有用到32位，而且在大部分情况下，pc都不会用到前2位，所以设为30位能节省一点点资源。

劣势：b指令有可能会需要对pc进行额外修改，表示偏移量的数据是32位的，转换起来比较麻烦。

1. 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。

合理。ROM只能读，不能写，可以用在IM里。RAM可读可写，用在DM里；GRF需要比较快的速度，所以虽然比较昂贵，但还是要用寄存器。

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。）

首先assign{o5,o4,o3,o2,o1,o0}=op;assign{f5,f4,f3,f2,f1,f0}=func;

Regdst=f5 + f0 f1 f5

Alu\_src=o0 o2 o3 + o0 o1 o5 + o0 o1 o3 o5 + o0 o1 o2 o3

Memtoreg=o0 o1 o5

Regwr=f5 + f0 f1 f5 + o0 o2 o3 + o0 o1 o5 + o0 o1 o2 o3

Npc\_sel=o2

Ext\_sel[0]=o0 o1 o5 + o0 o1 o3 o5 + o2 + o0 o1 o2 o3

Ext\_sel[1]=o0 o1 o2 o3

1. 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。

将x当成0即可，我已经这么做了。

Regdst=f5 + f0 f1 f5

Alu\_src=o0 o2 o3 + o0 o1 o5 + o0 o1 o3 o5 + o0 o1 o2 o3

Memtoreg=o0 o1 o5

Regwr=f5 + f0 f1 f5 + o0 o2 o3 + o0 o1 o5 + o0 o1 o2 o3

Npc\_sel=o2

Ext\_sel[0]=o0 o1 o5 + o0 o1 o3 o5 + o2 + o0 o1 o2 o3

Ext\_sel[1]=o0 o1 o2 o3

1. 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。

Nop指令没有在或逻辑里起作用，没有也行，还能省一点资源。

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。
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。

如果指令很多，靠编写程序来测试就显得很困难而且容易遗漏，形式验证能穷尽所有可能，是理想且严谨的方法。

测试代码如下（循环部分将执行2次）

ori $s2 $s2 0x0001

loop:

addu $s1 $s1 $s2

subu $s3 $s2 $3

sw $s1 0($s0)

lw $s5 0($s0)

beq $s1 $s2 loop

nop

lui $s6 0x1111

单步的期望

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 次数\输出值 | regwr | Regaddr | Regdata（部分省略） | memwr | memaddr | memdata |
| 0 | 1 | 10010 | 1 | 0 | 0 | 0 |
| 1 | 1 | 10001 | 1 | 0 | 0 | 1 |
| 2 | 1 | 10011 | 1 | 0 | 0 | 0 |
| 3 | 0 | 10001 | 0 | 1 | 0 | 1 |
| 4 | 1 | 10101 | 1 | 0 | 0 | 0 |
| 5 | 0 | 10010 | 2 | 0 | 0 | 1 |
| 6 | 1 | 10001 | 2 | 0 | 0 | 1 |
| 7 | 1 | 10011 | 1 | 0 | 0 | 0 |
| 8 | 0 | 10001 | 0 | 1 | 0 | 2 |
| 9 | 1 | 10101 | 2 | 0 | 0 | 1 |
| 10 | 0 | 10010 | 3 | 0 | 0 | 1 |
| 11 | 0 | 00000 | 0 | 0 | 0 | 0 |
| 12 | 1 | 10110 | 00010001000100010000000000000000 | 0 | 0 | 0 |

最终期望：$s1=2 $2=1 $s3=1 $s5=2 $6=0x11110000

地址为0处存入2