P3设计文档

——Logisim单周期

1. 指令集
2. ADDU：不支持溢出的加法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编码 | Opcode  000000 | rs | rt | rd | Shamt  00000 | Func  100001 |
| 6 | 5 | 5 | 5 | 5 | 6 |
| 描述 | GPR[rd] ← GPR[rs] + GPR[rt] | | | | | |

1. SUBU：不支持溢出的减法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编码 | Opcode  000000 | rs | rt | rd | Shamt  00000 | Func  100011 |
| 6 | 5 | 5 | 5 | 5 | 6 |
| 描述 | GPR[rd] ← GPR[rs] - GPR[rt] | | | | | |

1. ORI：或立即数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | Opcode  001101 | rs | rt | immediate |
| 6 | 5 | 5 | 16 |
| 描述 | GPR[rt] ← GPR[rs] OR zero\_extend(immediate) | | | |

1. LW：加载字

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | Opcode  100011 | base | rt | offset |
| 6 | 5 | 5 | 16 |
| 描述 | Addr ← GPR[base] + sign\_extend(offset)  GPR[rt] ← memory[Addr] | | | |

1. SW：存储字

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | Opcode  101011 | base | rt | offset |
| 6 | 5 | 5 | 16 |
| 描述 | Addr ← GPR[base] + sign\_extend(offset)  memory[Addr] ← GPR[rt] | | | |

1. BEQ：相等时跳转

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | Opcode  000100 | rs | rt | offset |
| 6 | 5 | 5 | 16 |
| 描述 | If (GPR[rs] == GPR[rt])  PC ← PC + 4 + sign\_extend(offset||0²)  Else  PC ← PC + 4 | | | |

1. LUI：立即数加载至高位

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | Opcode  001111 | 0  00000 | rt | immediate |
| 6 | 5 | 5 | 16 |
| 描述 | GPR[rt] ← immediate||0^16 | | | |

1. 模块规格
2. IFU：取指令单元

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 异步复位信号 |
| NPCIMM | [31:0] | I | 计算NPC的立即数信号 |
| NPCOp |  | I | 计算NPC的选择信号 |
| IM | [31:0] | O | 当前指令信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 计算地址 | 计算下一条指令在IM（指令存储器）中的地址 |
| 2 | 取指令 | 从IM（指令存储器）中取出32位指令并通过IM端口输出 |
| 3 | 复位 | reset信号为1时，复位PC（指令地址）至初始状态 |

1. GRF： 寄存器堆

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 异步复位信号 |
| WE |  | I | 写使能信号 |
| A1 | [4:0] | I | 第一个地址输入信号 |
| A2 | [4:0] | I | 第二个地址输入信号 |
| A3 | [4:0] | I | 第三个地址输入信号 |
| WD | [31:0] | I | 写入数据信号 |
| RD1 | [31:0] | O | A1所对应的寄存器的数据信号 |
| RD2 | [31:0] | O | A2所对应的的寄存器的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 读数据 | 读出A1，A2地址对应寄存器数据并通过RD1，RD2端口输出 |
| 2 | 写数据 | Clk上升沿到来时，若WE信号为1，则向A3地址对应寄存器写入数据WD（0号寄存器不能被写入） |
| 3 | 复位 | reset信号为1时，复位所有寄存器至初始状态 |

1. ALU：计算模块

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| ALUA | [31:0] | I | 参与运算的第一个数据信号 |
| ALUB | [31:0] | I | 参与运算的第二个数据信号 |
| ALUOp | [2:0] | I | 运算方式的选择信号 |
| ALU | [31:0] | O | 运算结果的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 不支持溢出加法 | 若ALUOp==001，计算A+B并通过ALU端口输出 |
| 2 | 不支持溢出减法 | 若ALUOp==010，计算A-B并通过ALU端口输出 |
| 3 | 按位或运算 | 若ALUOp==011，计算A|B并通过ALU端口输出 |

1. EXT：立即数拓展模块

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| IMM | [15:0] | I | 参与拓展的立即数信号 |
| EXTOp | [2:0] | I | 拓展方式的选择信号 |
| EXT | [31:0] | O | 拓展完成后的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 零拓展 | 若EXTOp==001，将立即数加载至输出信号低位并用0填充输出信号的高16位 |
| 2 | 符号拓展 | 若EXTOp==010，将立即数加载至输出信号低位并用其最高位填充输出信号的高16位 |
| 3 | 补00符号拓展 | 若EXTOp==011，将立即数末尾补两个0后进行符号拓展 |
| 4 | 加载至高位 | 若EXTOp==100，将立即数加载至输出信号高位并用0填充输出信号的低16位 |

1. DM：数据存储器

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 异步复位信号 |
| DMA | [31:0] | I | 存取的地址信号 |
| DMD | [31:0] | I | 存取的数据信号 |
| DMOp |  | I | 存数据使能信号 |
| DM | [31:0] | O | 取出的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 存数据 | clk上升沿到来时，若DMOp信号为1，则向DMA对应的地址中写入DMD数据 |
| 2 | 取数据 | 将DMA地址对应的数据通过DM端口输出 |
| 3 | 复位 | reset信号为1时，复位DM（数据存储器）至初始状态 |

1. 控制器设计
2. 控制信号说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 位数 | 描述 |
| 1 | NPCOp |  | 作为计算NPC时的选择信号 |
| 2 | GRFWE |  | 作为GRF的写使能信号 |
| 3 | ALUOp | [2:0] | 作为ALU的计算方式选择信号 |
| 4 | EXTOp | [2:0] | 作为EXT的拓展方式选择信号 |
| 5 | DMOp |  | 作为DM的写使能信号 |
| 6 | GRFA3\_MUXOp |  | 作为GRF的A3端口输入信号的选择信号 |
| 7 | GRFWD\_MUXOp | [1:0] | 作为GRF的WD端口输入信号的选择信号 |
| 8 | ALUB\_MUXOp |  | 作为ALU的B端口输入信号的选择信号 |

1. 指令与控制信号真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 |  |  |  |  |  | 000000 |
| opcode | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 |
| 指令名 | addu | subu | ori | lw | sw | beq | lui | nop |
| NPCop | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| GRFWE | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| ALUOp | 001 | 010 | 011 | 001 | 001 | 010 | 000 | 000 |
| EXTOp | 000 | 000 | 001 | 010 | 010 | 011 | 100 | 000 |
| DMOp | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| GRFA3\_MUXOp | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| GRFWD\_MUXOp | 00 | 00 | 00 | 01 | 00 | 00 | 10 | 00 |
| ALUB\_MUXOp | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

1. 数据通路
2. 信号连接表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | IFU | GRF | | | |
| 输入信号 | NPCIMM | A1 | A2 | A3 | WD |
| addu |  | IM.IM[25:21] | IM.IM[20:16] | IM.IM[15:11] | ALU.ALU |
| subu |  | IM.IM[25:21] | IM.IM[20:16] | IM.IM[15:11] | ALU.ALU |
| ori |  | IM.IM[25:21] |  | IM.IM[20:16] | ALU.ALU |
| lw |  | IM.IM[25:21] |  | IM.IM[20:16] | DM.DM |
| sw |  | IM.IM[25:21] | IM.IM[20:16] |  |  |
| beq | EXT.EXT | IM.IM[25:21] | IM.IM[20:16] |  |  |
| lui |  |  |  | IM.IM[20:16] | EXT.EXT |
| nop |  |  |  |  |  |
| ALL | EXT.EXT | IM.IM[25:21] | IM.IM[20:16] | IM.IM[15:11]  IM.IM[20:16] | ALU.ALU  DM.DM  EXT.EXT |
|  | EXT | ALU | | DM | |
| 输入信号 | EXTIMM | ALUA | ALUB | DMA | DMD |
| addu |  | GRF.RD1 | GRF.RD2 |  |  |
| subu |  | GRF.RD1 | GRF.RD2 |  |  |
| ori | IM.IM[15:0] | GRF.RD1 | EXT.EXT |  |  |
| lw | IM.IM[15:0] | GRF.RD1 | EXT.EXT | ALU.ALU |  |
| sw | IM.IM[15:0] | GRF.RD1 | EXT.EXT | ALU.ALU | GRF.RD2 |
| beq |  | GRF.RD1 | GRF.RD2 |  |  |
| lui | IM.IM[15:0] | GRF.RD1 |  |  |  |
| nop |  |  |  |  |  |
| ALL | IM.IM[15:0] | GRF.RD1 | GRF.RD2  EXT.EXT | ALU.ALU | GRF.RD2 |

1. 冲突选择表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GRFA3 | | | | |
| 选择信号GRFA3\_MUXOp | 0 | | 1 | |
| 连接信号 | IM.IM[20:16] | | IM.IM[15:11] | |
| GRFWD | | | | |
| 选择信号GRFWD\_MUXOp | 00 | 01 | 10 | 11 |
| 连接信号 | ALU.ALU | DM.DM | EXT.EXT |  |
| ALUB | | | | |
| 选择信号ALUB\_MUXOp | 0 | | 1 | |
| 连接信号 | EXT.EXT | | RD2.RD2 | |

1. 测试样例

ori $t1,$0,1 # t1 = 0x00000001

lui $t2,1 # t2 = 0x00010000

ori $t3,$0,0xffff # t3 = 0x0000ffff

lui $t4,0xffff # t4 = 0xffff0000

beq $t1,$t2,end

nop

addu $s1,$t1,$t2 # s1 = 0x00010001

addu $s2,$t2,$t4 # s2 = 0x00000000

subu $s3,$t2,$t1 # s3 = 0x0000ffff

subu $s4,$t1,$t3 # s4 = 0xffff0002

ori $t6,$0,4 # t6 = 0x00000004

sw $s3,0($t6) # dm[4] = 0x0000ffff

sw $s4,4($t6) # dm[8] = 0xffff0002

ori $t5,$0,8 # t5 = 0x00000008

lw $s5,0($t5) # s5 = 0xffff0002

addu $t7,$t6,$0 # t7 = 0x00000004

addu $t8,$t1,$0 # t8 = 0x00000001

begin:

addu $t7,$t7,$t1 # t7 = t1 +1

beq $t7,$t5,end # if (t7 = 0x00000008) jump to end

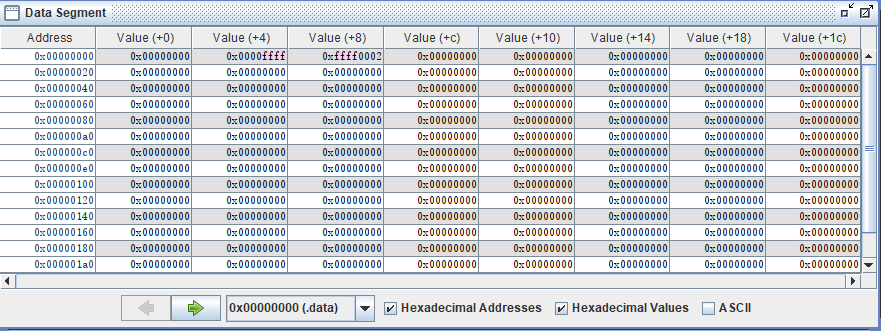
addu $t8,$t8,$t8 # t8 = t8 + t8

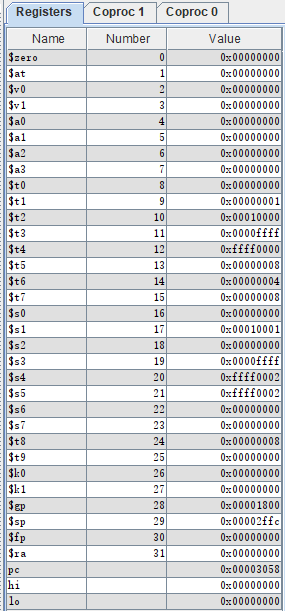
beq $0,$0,begin # jump to begin

end:

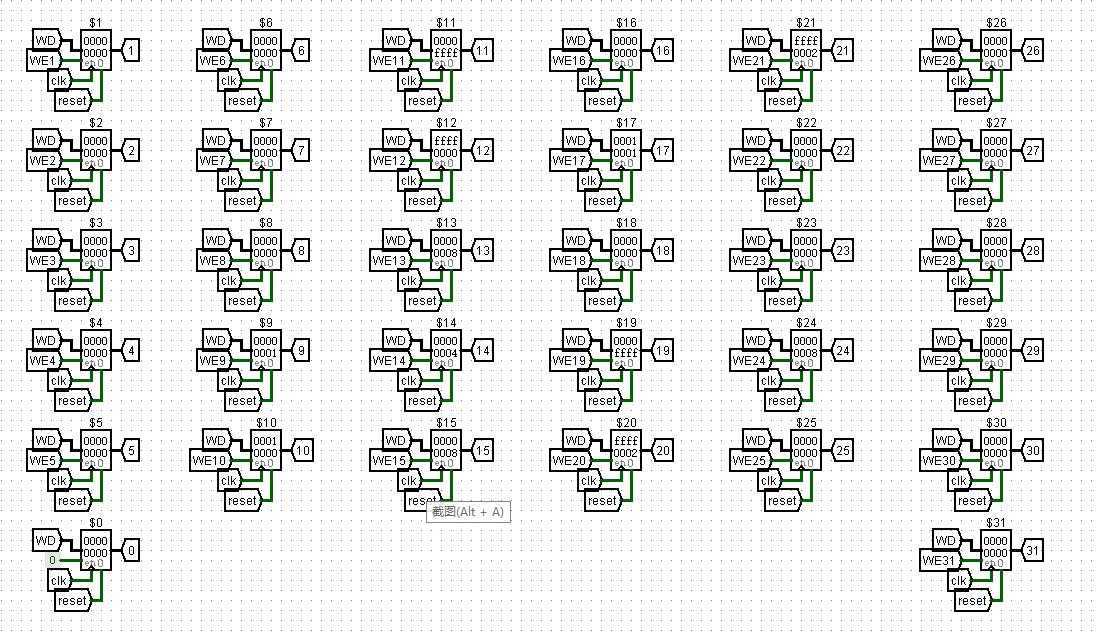
nop

MARS模拟结果：





本地运行结果：



e8c2fc266f54e377448d5c17aeef960

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。

32位PC在进行NPC的计算时不需要进行拓展，但是其最低两位在系统中没有作用，造成一定的浪费。

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。

这种做法是合理的，IM作为指令存储器，不需要对其进行写操作，适合用只读的ROM实现，DM作为数据存储器，需要进行读写操作且存储空间要足够大，适合用双端口且存储空间大的RAM实现，GRF作为数据临时存储器，需要存取速度快，适合用存取速度快的寄存器实现。

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）

add=!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func1&&!func2&&!func3&&!func4&&func5

sub= !op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&func1&&!func2&&!func3&&!func4 &&func5

ori=op0&&!op1&&op2&&op3&&!op4&&!op5

lw=op0&&op1&&!op2&&!op3&&!op4&&op5

sw=op0&&op1&&!op2&&op3&&!op4&&op5

beq=!op0&&!op1&&op2&&!op3&&!op4&&!op5

RegDst=add||sub

ALUSrc=ori||lw||sw

MemtoReg=lw

RegWrite=add||sub||ori||lw

nPC\_Sel=beq

EXTOp=lw||sw

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。

RegDst=!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func2&&!func3&&!func4&&func5

ALUSrc=(op0&&!op1&&op2&&op3&&!op4&&!op5)||(op0&&op1&&!op2&&!op4&&op5)

MemtoReg=op0&&op1&&!op2&&!op3&&!op4&&op5

RegWrite=(!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func2&&!func3&&!func4&&func5)||(op0&&!op1&&op2&&op3&&!op4&&!op5)||(op0&&op1&&!op2&&!op3&&!op4&&op5)

nPC\_Sel=!op0&&!op1&&op2&&!op3&&!op4&&!op5

EXTOp=op0&&op1&&!op2&&!op4&&op5

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指令机器码为0x00000000，无论将其视作何种指令，其Opcode，rs，rt，rd，Shamt，func，immediate/offset段全为0，不会对DM，GRF等部件造成实际影响，因此不需要考虑nop指令对控制信号的控制。

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。

若有多个片中能够存储数据，则需要将计算或拓展出的地址进行片选，具体方法是将所得地址与各片地址作比较，得出其所在的片段并做相应运算后存入所对应的位置。

1. 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。

与测试相比，形式验证的优点如下：

(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了100%。

(2)形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励。

(3)形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

其缺点为：

（1）形式验证到目前为止仍然不能有效地验证电路的性能，如电路的时延与功效等。