CPU设计

一、CPU设计方案综述

1.1 总体设计概述

本CPU为logisim实现的单周期CPU，支持的指令集包含{lw，sw，addu，subu，ori，beq，lui，j，sh，lh，slt，sllv，jr，bgt，jal}。

1.2 关键模块定义

1.2.1 PC相关

关于PC的下一个信号，有

信号	解释
PC + 4	下一条指令
branchAdd	当用b类指令的时候PC的值
jIndex	使用j和jal的PC值
jarIndex	使用jar的PC的值

1.2.2 ANALSIS

集成了一些组合逻辑电路，通过分析指令，来将指令拆分

信号	解释
offset	16位的偏移量
R2	寄存器R2的编号
R1	寄存器R1的编号（1,2是按照由高位到地位排列的）
opcode	指令码
SignImm	将offset有符号拓展为32位
func	函数码
R3	寄存器R3的编号
UnsignImm	将offset无符号拓展为32位
index	取j和jal指令的后26位

1.2.3 GRF

信号	解释
RegAddr1	读寄存器地址1
RegAddr2	读寄存器地址2
RegAddr3	写寄存器地址
RegDataOut1	读寄存器内容1
RegDataOut2	读寄存器内容2
RegWrite	置1时写入数据

1.2.4 ALU

可以实现**比较**和**计算**两种功能。

信号	解释
SrcA	第一个运算数
SrcB	第二个运算符
ALUResult	运算结果
greater	置1说明 A > B
equal	置1说明 A == B
less	置1说明 A < B
ALUOp	控制运算的类型

ALU的控制信号如下

编码	意义
000	or
001	左移16位
010	addu
011	
100	
101	
110	subu
111	

1.2.5 DM

端口如下

信号	解释
MemAdd	读取和存储的地址
MemDataIn	存储的数据
MemWrite	置1的时候写入数据
MemDataOut	读取的数据

1.2.6 control

分为CONTROLfront，用来确定是哪一条指令，CONTROLback，用来输出具体的控制信号。

控制信号解释如下：

信号	解释	编码
RegWrite	决定是否向寄存器中写入数据	0：不写，1：写
MemWrite	决定是否向存储写入数据	0：不写，1：写
RegSrc	决定写入寄存器中的数据来源	00：MemDataOut，01：ALUResult，10：nextPC，11：lh
ALUSrc	决定写入ALU的SrcB的数据来源	00：SignImm，01：寄存器RegDataOut2，10：UnsignImm
RegAddr3Src	决定写入寄存器的地址	00：寄存器R2，01：寄存器R3，10:31号
PCSrc（非控制器信号）	决定nextPC未加4之前的值	00：当前PC，01：BranchAdd，10：JIndex，11：jrIndex
MemSrc	决定写入存储的数据来源	0：RegDataOut2，1：sb

1.3 重要指令的实现方法

各个指令对应的控制信号如下：

指令	RegWrite	MemWrite	RegSrc	ALUSrc	RegAddr3Src	PCSrc	MemSrc	ALUOp
lw	1	0	00	00	00	00	0	010
sw	0	1	00	00	00	00	0	010
addu	1	0	01	01	01	00	0	010
subu	1	0	01	01	01	00	0	110
ori	1	0	01	10	00	00	0	000
beq	0	0	00	01	00	01	0	110
lui	1	0	01	00	00	00	0	001
j	0	0	00	00	00	10	0	000
jal	1	0	10	00	10	10	0	000
jr	0	0	00	00	00	11	0	000
sb	0	1	00	00	00	00	1	010
lh	1	0	11	00	00	00	0	010

二、测试方案

2.1 典型测试样例

2.1.1 j

```
ori $t0, $t0, 2
j B
addu $t1, $t1, $t0
B:
addu $t1, $t1, $t0
```

2.1.2 jal

```
ori $t0, $t0, 2
jal B
addu $t1, $t1, $t0
B:
addu $t1, $t1, $t0
```

2.1.3 jr

```
ori $t0, $t0, 2
jal B
j end

B:
addu $t1, $t0, $t0
jr $ra

end:
```

2.1.4 sb

```
ori $t0, $t0, 0x7f
ori $t1, $0, 1
sb $t0, 6($t1)
```

2.1.5 lh

```
ori $t0, $t0, 0x7f
ori $t1, $0, 1
sb $t0, 6($t1)
lh $t2, 5($t1)
```

2.1.6 lw, sw, ori, addu, subu, beq

```
ori $t0, $t0, 0x7f
ori $t1, $0, 4
A:
addu $t2, $t0, $t1
subu $t3, $t0, $t1
sw $t2, ($0)
sw $t3, ($t1)
lw $t4, ($0)
lw $t5, ($t1)
beq $t5, $t3, A
```

2.1.7 lui

```
lui $t0, 100
```

三、思考题

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

不合理。

寄存器文件通常由快速的静态随机读写存储器（SRAM）实现。只是因为logisim中RAM的端口过少，所以我们才不用他实现寄存器文件。在现实中，用寄存器过于浪费资源。

IM不知道需不需要只用一个例程，如果不是，那么用ROM对于不同程序的写入代价过大。

2.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?

因为nop为空指令，所以等效于各个控制信号的取值为0或X（不关心）。这就导致一些信号必须为零（比如写使能），而另一些信号无所谓，所以我们可以让不关心的信号也取零，这样达到统一简化的效果，也就等效于不把nop空指令加入控制信号真值表。

而机器码全为零实际对应的是sll左移零位，等效于无操作。

3.上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦。请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

MARS不能导出PC与DM地址均为0的机器码，一般能导出的情况有两种：一是PC从0x3000开始，data从0开始；第二种是PC从0开始，data从0x3000开始。

针对两种导出情况分别有不同的解决方法：

对于PC不从0x3000开始，这主要影响的是j、jal等伪直接跳转指令，可以将这些指令获得的pc减去0x3000即可解决这一问题；

对于data不从0x3000开始，主要采用片选的方式来解决问题。片选（Chip Select）也称为选片，是利用高位作为选择信号，来选择对哪一个RAM片进行操作，例如若选高4位为片选信号，则将整个内存区域分为16片，每一片对应的高四位相同。采用片选的方式，可以利用data高位的不同来选择不同的RAM，进而使得0x3000开始的data和0x0000开始的data在低位保持一致，以此解决问题。

4.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，简要阐述相比于测试，形式验证的优劣之处。

形式化验证是用数学的方法证明我们的系统是无Bug的，可以分为三大类：等价性检查、形式模型检查以及定理证明。

形式化验证的优点：

- 测试无法证明系统不存在缺陷，也不能证明它符合一定的属性，但是形式化验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此形式化验证克服了测试的局限性，达到了百分百覆盖率；
- 形式化验证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，因此减少了测试所必须的大量样例输入及输出。
- 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。

形式化验证的缺点：

- 对数学技能要求高，很难确定属性是否完备；
- **时间**成本太高，不适应当前发展迅速的软件开发。