P3-Logisim 单周期实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Logisim 实现的单周期 MIPS - CPU，处理器为 32 位处理器，支持的指令集包含{ addu, subu, ori, lw, sw, beq, lui, nop,j}。为了实现这些功能，CPU 主要包含了 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件，这些模块分成 3 级。

（二）关键模块定义

1. GRF

（可使用表格进行端口说明）

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号
we	I	可写入信号
A1	I	读出地址 1
A2	I	读出地址 2
A3	I	写入地址
WD	I	写入数据
RD1	O	读出数据 1
RD2	O	读出数据 2

2. DM

信号名	方向	描述
Clk	I	时钟信号

reset	I	复位信号
MemWrite	I	可写入信号
MemRead	I	可读出信号
A	I	地址
D1	I	写入数据
D2	O	读出数据

3. ALU

该模块包括 add,sub,or,and 子模块，用于进行运算

信号名	方向	描述
In1	I	运算数 1
In2	I	运算数 2
OP	I	操作信号
Zero	I	运算结果是否为 0
Out	I	运算结果

运算符对应表(四位操信号备用)

0000	加
0001	减
0010	或
0011	与
0100	将 in2 左移 16 位（用于 lui）

4. IFU

该模块保证每个时钟周期 PC 根据信号决定自增 4 或进行 beq/q 的跳转

并且持续输出读取到的指令

该模块包括 PC_add 用于自增 4，sel26 用于获取 2~6 位信号

信号名	方向	描述
Clk	I	时钟信号

reset	I	复位信号
26bits	I	26 位立即数用于 beq
PCSrc	I	是否 beq
PCJump	I	是否 j
Out	O	指令

5. EXT

用于拓展信号 16->32

0：无符号拓展 1：有符号拓展

6. sel26

用于获取 2~6 位信号

（三）控制器 Controller 定义

该模块对指令的 op 和 func 进行解码，输出对应的信号。

利用和逻辑和或逻辑识别指令然后生成信号

信号名	方向	描述
op	I	指令的 6 位 op 部分
func	I	指令的 6 位 func 部分
RegDst	O	寄存器写入选择信号 0：使用 rt 写入 1：使用 rd 写入
RegWrite	O	寄存器写入信号
ALUSrc	O	ALU 运算数选择信号 0：grf 1：imm
branch	O	PC 变更选择信号 0：自增 4 1：beq
PCJump	O	PC 变更选择信号

		0：branch 1：j
MemRead	O	内存写信号
MemWrite	O	内存读信号
MemtoReg	O	写入寄存器信号 0：ALU 结果 1：内存

指令信号对应表

	RegDst	RegWrite	ALUSrc	branch	PCJump	MemRead	MemWrite	MemtoReg	extsel	ALU
addu	1	1	0	0	0	0	0	0	x	0000
subu	1	1	0	0	0	0	0	0	x	0001
ori	0	1	1	0	0	0	0	0	0	0010
lw	0	1	1	0	0	1	0	1	1	0000
sw	x	0	1	0	0	0	1	x	1	0000
beq	x	0	0	1	1	0	0	x	x	0001
lui	0	1	1	0	0	0	0	0	0	0100
j	x	0	x	1	0	0	0	x	x	x
nop	x	x	x	x	x	x	x	x	x	x

二、测试方案

（一）典型测试样例

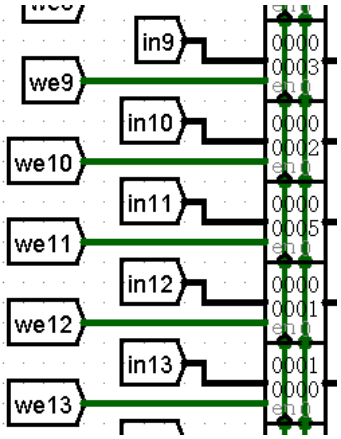
1. ALU 功能测试

```
1
2   ori $t1,3
3   ori $t2,2
4   addu $t3,$t1,$t2
5   subu $t4,$t1,$t2
6   lui $t5,1
```

期望值：

$t1	9	3
$t2	10	2
$t3	11	5
$t4	12	1
$t5	13	65536

实际值：



2. DM 功能测试

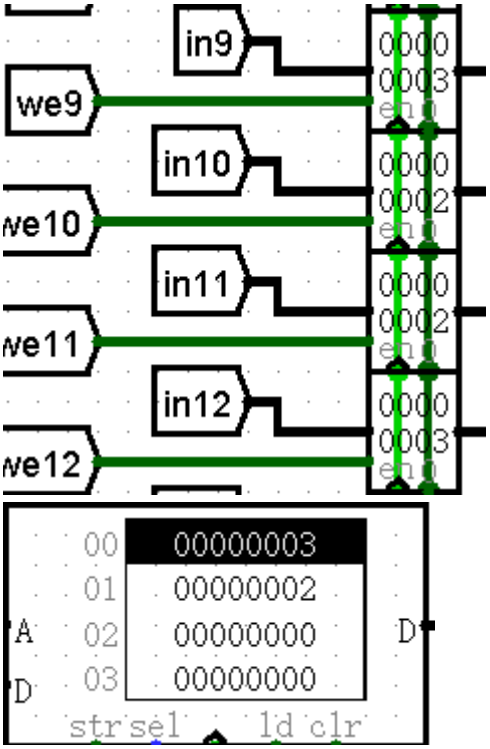
```
1
2   ori $t1,3
3   ori $t2,2
4   sw $t1,0($0)
5   sw $t2,4($0)
6   lw $t3,4($0)
7   lw $t4,0($0)
```

期望值：

$t1	9	3
$t2	10	2
$t3	11	5
$t4	12	1
$t5	13	65536

Address	Value (+0)	Value (+4)
0x00000000	3	2

实际值：



3. 其他指令综合测试

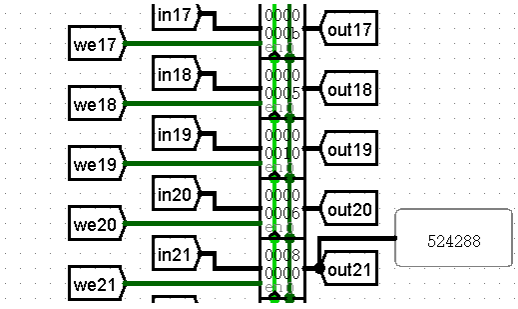
```
ori $s1,$s1,11
ori $s2,$s2,5
addu $s3,$s1,$s2
j this2
this1:
nop
this2:
nop
subu $s4,$s1,$s2
lui $s5,8
sw $s3,0($0)
beq $t1,$t2,that
lw $s6,4($0)
lw $s3,4($0)
that:
lw $s6,4($0)
beq $s1,$s2,this1
```

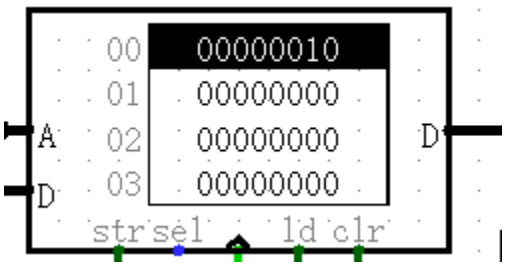
期望值：

$s1	17	11
$s2	18	5
$s3	19	16
$s4	20	6
$s5	21	524288

	Value (+0)
0x00000000	16

实际值：





（二）自动测试工具

1. 测试样例生成器

```
import os

import random

import re

with open("in.asm","w") as file:

    for i in range(2):

        x=random.randint(0,31)

        y=random.randint(0,31)

        z=random.randint(0,31)

        file.write("addu $%d,$%d,$%d\n"%(x,y,z))

    for i in range(2):

        x=random.randint(0,31)

        y=random.randint(0,31)

        z=random.randint(0,31)

        file.write("subu $%d,$%d,$%d\n"%(x,y,z))

    for i in range(2):

        x=random.randint(0,31)

        num=random.randint(0,10000)

        file.write("lui $%d,%d\n"%(x,num))

    for i in range(2):

        x=random.randint(0,31)

        y=random.randint(0,31)

        num=random.randint(0,10000)

        file.write("ori $%d,$%d,%d\n"%(x,y,num))

    for i in range(2):

        x=random.randint(0,31)
```

```
        y=random.randint(0,31)
        num=random.randint(0,100)
        file.write("lw $%d,%d($%d)\n"%(x,4*num,y))
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        num=random.randint(0,100)
        file.write("sw $%d,%d($%d)\n"%(x,4*num,y))
    # for i in range(2):
    #     x=random.randint(0,31)
    #     y=random.randint(0,31)
    #     num=random.randint(0,100)
    #     file.write("beq $%d,$%d,%d\n"%(x,y,4*num))
```

```
with open("in.asm","w") as file:
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        z=random.randint(0,31)
        file.write("addu $%d,$%d,$%d\n"%(x,y,z))
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        z=random.randint(0,31)
        file.write("subu $%d,$%d,$%d\n"%(x,y,z))
    for i in range(2):
        x=random.randint(0,31)
        num=random.randint(0,10000)
        file.write("lui $%d,%d\n"%(x,num))
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        num=random.randint(0,10000)
        file.write("ori $%d,$%d,%d\n"%(x,y,num))
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        num=random.randint(0,100)
        file.write("lw $%d,%d($%d)\n"%(x,4*num,y))
    for i in range(2):
        x=random.randint(0,31)
        y=random.randint(0,31)
        num=random.randint(0,100)
        file.write("sw $%d,%d($%d)\n"%(x,4*num,y))
```

2. 自动执行脚本

```
java -jar mars4_5.jar test.asm nc mc CompactTextAtZero a dump .text HexText rom.txt
echo v2.0 raw> test-logisim.txt
type rom.txt >> test-logisim.txt
```

```
os.system("java -jar mars4_5.jar in.asm nc mc CompactTextAtZero a dump .text HexText rom.txt")

with open("rom.txt","r") as file:

    content=file.read()

with open("CPU.circ","r") as file:

    circmy=file.read()

circmy=re.sub(r'addr/data: 5 32([\s\S]*)</a>',"addr/data: 5 32\n"+content+"</a>",circmy)

with open("CPU.circ","w") as file:

            circmy=file.write(circmy)
```

3. 正确性判定脚本

没有想到合适的提取 GRF 数值的方案，采取直接对比的方法。

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。

我认为合理。IM 中存只读的指令，因此用 ROM，DM 需要读取和存储，因此使用 RAM，GRF 为寄存器堆，应使用 Register，可存可取。

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？请给出你的理由。

Nop 指令机器码为全 0，此时由控制器信号全部为 0，决定了 DM 和 Register 都是不可存入的状态，仅 PC 会自增 4，CPU 在本周期等价于无操作。

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 中 PC 高位为 0x3...可以将 0x3 全部修改为 0x0，然而本项目中不会涉及到高位，仅需取出 2~6 位用作片选信号即可。我选择设置了 sel26 模块用于选取 ALU 和 PC 中的 2-6 位作为 DM 和 IM 的地址。

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。在集成电路设计中，形式验证是一种集成电路设计的验证方法，它的主要思想是通过使用形式证明的方式来验证一个设计的功能是否正确。形式验证可以分为三大类：抽象解释（Abstract Interpretation）、形式模型检查（Formal Model Checking，也被称作特性检查）和定理证明（Theory Prover）。

优：通过数学的方式系统性的检查电路，易于发现细小的漏洞。

劣：过程较为复杂，需要较多的知识背景。