单周期 CPU（logisim 实现）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，支持的指令集包括{addu,subu,ori,beq,lw,sw,lui,nop}，为了实现这些功能，CPU 主要包括了 IFU、GRF、ALU、EXT、DM、Controller。

（二）关键模块定义

1. GRF

寄存器堆内部核心是 32 个寄存器，本模块包含一个数据写入端口和两个数据输出端口以满足运算。0 号寄存器始终为 0。

表格 1 GRF 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器的值全部清零 1：复位 0：无效
RegWrite	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
A1[5:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中储存的数据读到 RD1
A2[5:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中储存的数据读到 RD2
A3[5:0]	I	写寄存器地址
WD[31:0]	I	32 位写入数据
RD1[31:0]	O	输出 A2 指定的寄存器中的 32 位数据 A
RD2[31:0]	O	输出 A3 指定的寄存器中的 32 位数据 B

2. IFU

内部包括 PC（程序计数器）、IM（指令储存器）及相关逻辑。

PC 保存着指令在 ROM 的地址，本质上是一个具有置位功能的 32 位寄存器。

IM 起始地址为 0x00000000.本设计中 IM 使用 ROM 实现，容量为 32bit*32，因而只有 5 位的地址，故 PC 寄存器中地址映射到 ROM 中时只需要取 16 位立即数的 2-6 位，即 imm[2:6]。

模块的具体定义与功能见下表。

表格 2　IFU 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器的值全部清零 1：复位 0：无效
NPC[31:0]	I	PC 更新地址
toNPC[31:0]	O	当前 PC 地址加 4
Instr[31:0]	O	输出当前要执行的 32 位指令

3.　ALU

提供 32 位加、减、或运算以及比较等功能。

表格 3　ALU 模块端口定义

信号名	方向	描述
Src1[31:0]	I	32 位输入 A
Src2[31:0]	I	32 位输入 B
Aluop[2:0]	I	控制信号 000:加 001:减 010:或运算 011:(lui)加载到高位
Zero	O	当 Src1 与 Src2 传入数据相等时输出 1，否则为 0
Result[31:0]	O	32 位输出数据

4.　EXT

将 16 位立即数扩展为 32 位。

表格 4　EXT 模块端口定义

信号名	方向	描述

Imm[15:0]	I	待扩展 16 位输入
Extop	I	控制信号 0:无符号扩展 1:有符号扩展
Result[31:0]	O	32 位数据输出

5. DM

使用 RAM 实现，容量为 32bit*32。具有异步复位功能，复位值为 0x00000000。起始地址为 0x00000000。RAM 使用双端口模式。

表格 5 DM 模块端口定义

信号名	方向	描述
MemAddr[5:0]	I	5 位写入地址
Data[31:0]	I	32 位输入数据
Clk	I	时钟信号
Reset	I	（异步）复位信号
MemWrite	I	写使能信号 1：可以向 DM 写入数据 0：无效
RD[31:0]	O	32 位输出数据

(三) 控制器设计

控制器输出的各个信号是控制 CPU 数据通路的重要模块。

MIPS 处理器通过 op 字段和 func 字段来判断具体是哪一条指令，将指令中的信息转化为 CPU 各部分的控制信号。真值表如下图所示。

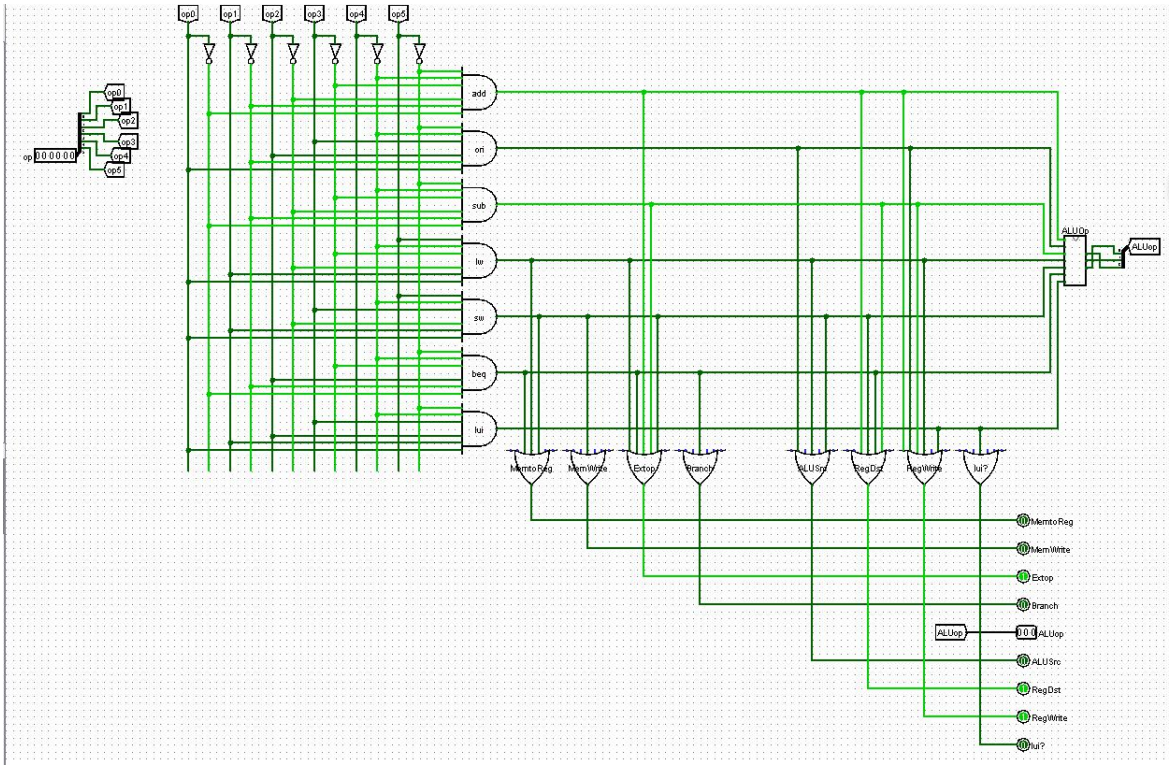
表格 6 控制器真值表

func	100000		100010							
op	000000	001101	000000	100011	101011	000100	001111			
	Add	ori	sub	lw	sw	beq	lui			
MemtoReg	0	0	0	1	x	x	0			
MemWrite	0	0	0	0	1	0	0			
Branch	0	0	0	0	0	1	0			
ALUop	add	ori	sub	add	add	cmp				
ALUSrc	0	1	0	1	1	0	1			
RegDst	1	0	1	0	x	x	0			
RegWrite	1	1	1	1	0	0	1			
Extop	x	0	x	1	1	x	0			

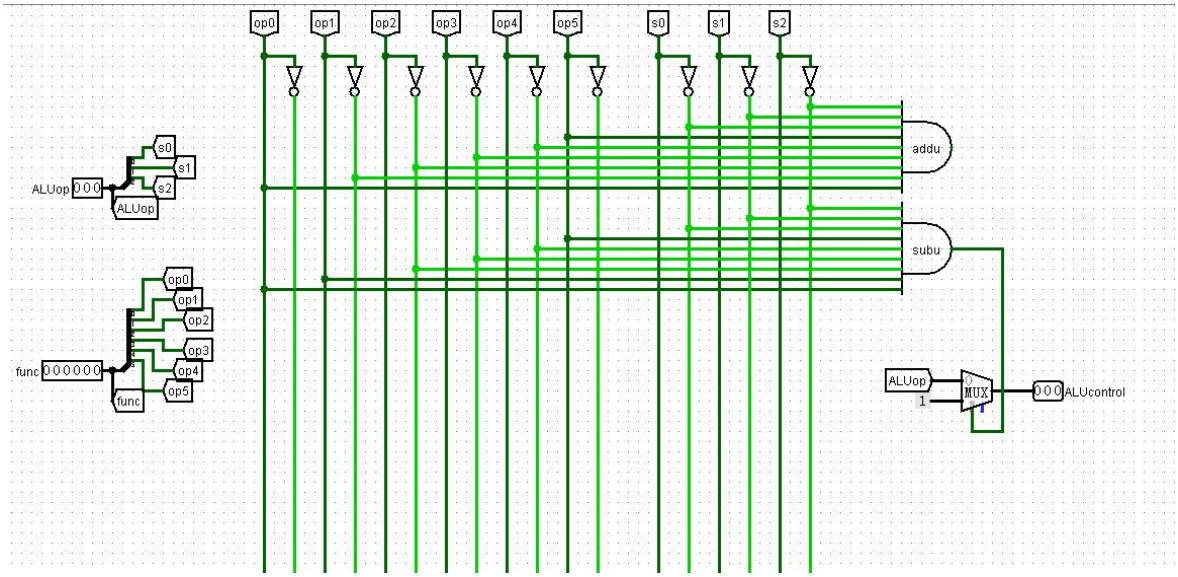
Lui?	0	0	0	0	0	0	1			

表格 7　Controller 模块端口定义

信号名	方向	描述
Op[5:0]	I	Instr[31:26]
Func[5:0]	I	Instr[5:0]
RegWrite	O	写使能信号 1：可向 GRF 中写入数据 0：不能向 GRF 中写入数据
ALUop[2:0]	O	控制信号 000:加 001:减 010:或运算 011:比较运算 100:取决于 func
RegDst	O	RW 选择信号 0：GRF 写入的寄存器地址为(Instr[16:20]) 1：GRF 写入的寄存器地址为(Instr[25:21])
ALUsrc	O	ALUsrcB 输入选择信号 0：SrcB 输入来自 GRF 的 RD2 输出 1：SrcB 输入来自 16 位 imm 扩展后的 32 位数
Extop	O	EXT 扩展类型选择信号 0：无符号扩展 1：有符号扩展
MemWrite	O	写使能信号 1：可以向 DM 写入数据 0：无效
MemtoReg	O	DM 读出控制信号、GRF 写入选择信号 0：GRF 的 WD 写入数据为 ALU_result，DM 输出使能无效 1：GRF 的 WD 写入数据为从 DM 输出 RD
Branch	O	IM 地址更新选择信号 0：PC <= PC + 4 1：beq 指令，PC <= PC + 4 + sign_extend(offset||00)



图片 1 与或门阵列



图片 2 ALU 译码器

（四）重要机制实现方法

1. 跳转

Controller 模块的 branch 信号与 ALU 的 Zero 信号通过与运算后决定是否使用跳转地址，跳转地址的计算由位扩展器将指令码低 16 位扩展为 32 位数后，与当前的 PC 值相加得到。

2. 访存

DM 地址为 ALU 输出的 2-6 位，输入数据为 GRF 的 RD2，当指令为 lw 时，MemtoReg 为 1，多路选择器输出数据存储器的输出。寄存器的编码选择指令码 20:16 位，RegWrite 为 1. 当指令码为 sw 时，MemWrite 为 1，向数据存储器写入数据。

3. lui 实现

在本人设计的 CPU 中，并未将 lui 放入 ALU，而是在顶层设计中加入了左移模块，并通过 lui?信号与多路选择器来控制输出。

二、CPU 测试方案

1. ALU 及 lui 功能测试

Mips 代码：

```
ori $t0, $zero, 123
ori $t1, $zero, 456
addu $t2, $t1, $t0
subu $t3, $t1, $t0
lui $t4, 4
```

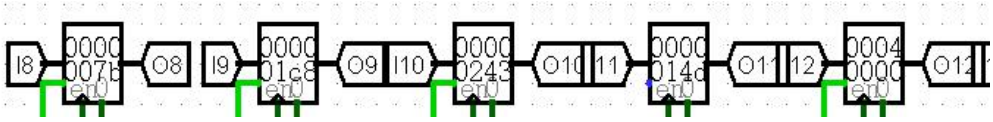
操作码：

```
3408007b
340901c8
01285021
01285823
3c0c0004
```

期望输出：

$t0	8	0x0000007b
$t1	9	0x000001c8
$t2	10	0x00000243
$t3	11	0x0000014d
$t4	12	0x00040000

实际输出：



符合期望

2. DM 测试

MIPS 代码：

```
ori $t0, $zero, 4
ori $t1, $zero, 8
sw $t1, ($t0)
lw $t2, ($t0)
```

操作码：

34080004

34090008

ad090000

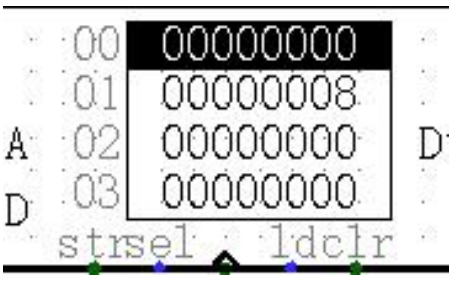
8d0a0000

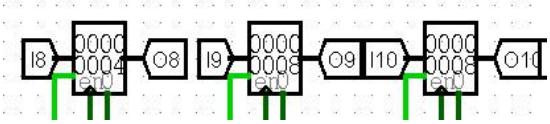
期望输出：

Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00000000	0x00000008	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000

$t0	8	0x00000004
$t1	9	0x00000008
$t2	10	0x00000008

实际输出：





符合期望

3. beq 测试

MIPS 代码：

```
ori $t1, $zero, 123
beq $t1, $t1, if
addu $t1, $t1, $t1
if:
addu $a0, $t1, $t1
```

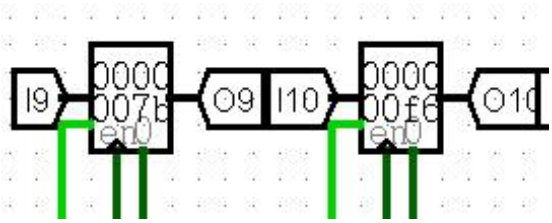
操作码：

```
3409007b
11290001
01294821
01292021
```

期望输出：

$t1	9	0x0000007b
$t2	10	0x000000f6

实际输出：



符合期望

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。

合理。RAM 为随机存储，速度快于 ROM，可以迅速的对 RAM 进行修改，设备掉电后 RAM 会被清空，ROM 通常用作硬盘以存储指令，掉电后并不会清空，而是需要人为进行修改。效率最高的方法是直接使用寄存器，每一条指令的实现都需要用到寄存器，所以在需要频繁调用的 GRF 模块应当使用 Register。

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。

因为当指令为 nop 时，RegWrite、MemWrite、MemtoReg 均为 0，无论 ALU 是否进行相关的计算，都无法改变 GRF 和 DM 中的值。

3. 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。

加一个模块，当地址大于 0x00003000 时减去 0x00003000

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。

在计算机硬件（特别是集成电路）和软件系统的设计过程中，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。由于仿真对于超大规模设计来说太耗费时间，形式验证就出现了。形式验证使用严格的数学推理来证明待测试设计的正确性，由于其静态、数学的特性，避免了对所有可能测试向量的枚举，而且能够达到无漏洞的检测。