计算机组成原理实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Logisim 实现的简单 MIPS 单周期处理器，支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, nop}。为了实现这些功能，CPU 主要包含了 IFU、GRF、DM、ALU、EXT、Controller。总体工作方式为：IFU 取指，将指令分解为若干部分；然后由 Controller 判断指令，控制其他部件工作；GRF 读出数据，在 ALU 中进行运算；将数据存入 DM 或从 DM 中读出数据写入 GRF。

（二）关键模块定义

1. GRF

GRF 负责读写 32 个寄存器，编号为 0~31，其中 0 号寄存器的值始终保持为 0，其它寄存器的初始值为 0。GRF 可以随时读出 2 个不同寄存器的内容，并在 WE 信号的控制下在时钟上升沿将指定内容写入指定寄存器，也具有异步复位的功能。其端口设置如表 1 所示。

表 1 GRF 端口描述

端口名称	方向	功能描述
Clk	I	时钟信号
Reset	I	异步复位信号，将 32 个寄存器中的值全部清零 0：无效 1：复位
WE	I	写使能信号 0：无效 1：向 GRF 写入
A1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的

		数据读出到 RD1
A2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3	I	5 位地址输入信号，指定 32 个寄存器中的一个，作为写入的目标寄存器
WD	I	32 位数据输入信号，表示需要写入 A3 指定的寄存器的数据
RD1	O	32 位数据输出信号，表示 A1 指定的寄存器中的数据
RD2	O	32 位数据输出信号，表示 A2 指定的寄存器中的数据

GRF 内部，为了方便起见，我运用了分治思想，用了 5 层子电路管理 32 个寄存器。寻找寄存器时在每个子电路中按对应二进制位的值进入下一层对应的子电路，直到定位到单个寄存器。Reset 信号的异步复位功能直接用 Losigim 寄存器的自带功能实现。对于 0 号寄存器，我在最外层用 MUX 判断了：如果写入 0 号寄存器，则写入数据置为 0x00000000。

2. DM

GRF 负责在 32bit*32 大小的 RAM 中读写数据，所有数据初始值为 0，起始地址为 0x00000000。具有异步复位功能。其端口设置如表 2 所示。

表 2 DM 端口描述

端口名称	方向	功能描述
Clk	I	时钟信号
Reset	I	异步复位信号，将 RAM 中的数据全部清零 0：无效 1：复位
WE	I	写使能信号 0：无效 1：向 DM 写入
A	I	32 位地址输入信号，表示将要读入或输出的数据地址（以字节为单位）
WD	I	32 位数据输入信号，表示需要写入 RAM 的数据
RD	O	32 位数据输出信号，表示 A 指定的地址的数据（字）

本题实际上只用到 32*32bit 大小的 RAM，但 DM 中 A 为 32 位地址，这是为了预留出空间方便修改、增加指令。

3. ALU

ALU 提供 32 位加、减、或、左移 16 位（lui）运算功能（也可进行大小比较，即两数相减判断正负以及 0）。目前没有支持判断溢出的功能。其端口设置如表 3 所示。

表 3 ALU 端口描述

端口名称	方向	功能描述
ALUOp	I	3 位输入信号，代表需要进行的运算操作 0：SrcA+SrcB 1：SrcA-SrcB 2：SrcA|SrcB 3：SrcB<<16
SrcA	I	32 位数据输入信号，表示要进行运算的第一个数
SrcB	I	32 位数据输入信号，表示要进行运算的第二个数
ALUResult	O	32 位数据输出信号，表示运算结果

ALU 内部使用一个 MUX，根据 ALUOp 选择对应的运算结果。不同运算使用 Losigim 自带部件完成。ALUOp 预留了一位，便于增加指令

4. Controller

Controller 根据机器码里的 Op 和 Funct 两部分判断当前 CPU 需要进行什么指令，根据不同指令的需求通过输出信号控制 CPU 各部分的运行。其端口定义如表 4 所示。每个指令对应的输出信号如表 5 所示。

表 4 Controller 端口描述

端口名称	方向	功能描述
Op	I	6 位输入信号，表示 Instr[31:26]
Funct	I	6 位输入信号，表示 Instr[5:0]
MemtoReg	O	寄存器写入数据是否来自于 DM 0：否（结果为立即数）

		1：是（结果为 DM 中的数）
MemWrite	O	DM 写使能信号 0：无效 1：向 DM 写入
Branch	O	是否需要修改 PC 值 0：否 1：是（PC 值要根据偏移量改变）
ALUControl	O	3 位输出信号，与 ALU 中 ALUOp 定义相同
ALUSrc	O	ALU 的 SrcB 来源 0：GRF 1：立即数
RegDst	O	写入的寄存器 0：rd（Instr[11:15]） 1：rt（Instr[16:20]）
RegWrite	O	GRF 写使能信号 0：无效 1：向 GRF 写入
ImmSign	O	立即数扩展方式 0：零扩展 1：符号扩展

表 5 Controller 种每个指令对应的输出信号

输出信号	指令							
	addu	subu	ori	lw	sw	beq	lui	nop
MemtoReg	0	0	0	1	X	X	0	0
MemWrite	0	0	0	0	1	0	0	0
Branch	0	0	0	0	0	1	0	0
ALUControl	0	1	2	0	0	1	3	0
ALUSrc	0	0	1	1	1	0	1	0

RegDst	0	0	1	1	X	X	1	X
RegWrite	1	1	1	1	0	0	1	0
ImmSign	X	X	0	1	1	1	X	X

在 Controller 中，使用与或门阵列构造控制信号。

5. IFU

IFU 包括了 PC、IM 和相关逻辑，用来取出每一步的指令。其中 PC 为寄存器，具有异步复位功能，起始地址为 0x00000000。IM 用 ROM 实现，容量为 32bit*32。其端口定义如表 6 所示。

表 6 IFU 端口描述

端口名称	方向	功能描述
Clk	I	时钟信号
Reset	I	异步复位信号，将 PC 值清零 0：无效 1：复位
Offset	I	32 位输入信号，表示偏移量
Instr	O	32 位输出信号，表示当前指令

PC 每个周期自增 4，另外会加上 Offset，作为下一个周期的指令地址。从 PC 值中取[6:2]作为访问 ROM 的真正地址。

6. EXT

使用两个 Logisim 自带的 Bit Extender，分别实现立即数的 0 扩展以及符号扩展。随后输出作为 ALU 的运算数或 PC 偏移量。

（三）重要机制实现方法

1. 跳转

CPU 目前仅支持 beq 跳转指令。

在 ALU 中判断两个寄存器的数值是否相等，若相等则将立即数进行符号扩展、左移的处理，作为偏移量加到 PC 值上。

判定单元位于顶层，计算单元位于 IFU 中，协同工作支持 beq 的跳转机

制。若加入 j 指令，也可在 Controller 中新增一个 PCeq 信号，让 PC 值直接等于处理后的立即数即可。

二、测试方案

（一）典型测试样例

1. ALU 与 GRF 功能测试

将只涉及 ALU 和 GRF 的指令多次修改寄存器和立即数，重复测试。特别注意了寄存器 0 和立即数大小为边界的例子。例如：

```
.text
  ori $0,$0,100
  lui $1,0xffff
  ori $2,$2,0x0f0f
  ori $3,$2,0xf0f0
  ori $4,$5,100
  ori $5,$4,200
  addu $6,$4,$5
  subu $7,$5,$4
  lui $8,0
```

2. DM 功能测试

主要测试 lw 与 sw 指令，和上面的指令相结合。例如：

```
.data
  A: .space 1000
.text
  ori $0,$0,100
  lui $1,0xffff
  ori $2,$2,0x0f0f
  ori $3,$2,0xf0f0
  ori $4,$5,100
```

```
ori $5,$4,200
addu $6,$4,$5
subu $7,$5,$4
lui $8,0
ori $9,$0,0
ori $10,$0,4
lw $11,A($9)
sw $1,A($9)
sw $5,A($10)
lw $11,A($9)
lw $12,A($10)
```

3. beq 指令测试

用 beq 多构造一些向前、向后的情况，同时也可以检查其他指令的操作是否正确。例如：

```
.data
  A: .space 1000
.text
  f1:
  lui $1,100
  ori $2,$0,100
  ori $3,$0,200
  addu $4,$4,$2
  sw $4,A
  lw $5,A
  beq $3,$4,f2
  beq $4,$5,f1
  f2:
  nop
```

4. 异步复位测试

在上面的过程中时不时使用 reset，检查是否复位。

（二）自动测试工具

暂时没有使用到。

三、思考题

（一）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。

答：合理。在我们的 CPU 中，没有写入 IM 的要求，所以 IM 可以使用 ROM。DM 需要写入和读出，所以用 RAM。GRF 需要写入和读出，而且要同时读出 2 个寄存器的值，所以必须使用 32 个 Register。

（二）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。

答：因为不加入真值表时，无法识别出 nop 是什么指令，所有的与门输出均为 0，或门输出也为 0，所有控制信号均为 0，CPU 不会执行任何修改 GRF、DM 的操作，只会空着运算一次 0+0，没有任何影响。

（三）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。

答：如果.text 从 0x00000000 开始，.data 从 0x00002000 开始，那么 PC 不变，DM 的实际地址相当于是当前地址异或 0x00002000。那么可以用 2 个 RAM，按第 14 个二进制位的 0/1 分类，分别存储数据。将第 14 个二进制位作

为片选信号，控制两个 RAM 的使用，注意进位即可。

我采用的方法是，按.text 开始和.data 开始分别导出两次，对于每个指令选择从 0 开始的版本即可。这也可以使用脚本实现。

（四）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。

答：形式验证的优点有：可以对所有可能的情况进行验证，更加全面，可以检查所有边界情况。

形式验证的缺点有：建立新的数学模型验证需要较多时间；面对 NPC 等问题，不能有效验证所有情况。