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 实现的单周期 MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、sll、nop}。为了实现这些功能，CPU 主要包含了 IFU、GRF、ALU、DM、控制器、移位器等模块。

（二）关键模块定义

1. GRF

表 1　GRF 模块接口

信号名	方向	描述
in0[31:0]	I	要读取寄存器的第一个地址
In1[31:0]	I	要读取寄存器的第二个地址
ALUOp[1:0]	I	要写入寄存器的地址
Zero	I	写入数据
Out[31:0]	I	写入使能
Clk	I	时钟
Reset	I	异步复位信号
RD1[31:0]	O	第一个地址寄存器的值
RD2[31:0]	O	第二个地址寄存器的值

表 2　GRF 功能定义

序号	功能名称	功能描述
1	读寄存器	RD1 输出 A1 地址所寻址的寄存器 RD2 输出 A2 地址所寻址的寄存器
2	写寄存器	WE 有效且 Reset 不为 1 时，在时钟上升沿将 WD 的数据写入 A3 地址所寻址的寄存器

3	异步复位	Reset 信号有效时，将 32 个 GPR 单元清零

2. IFU

表 3 IF 模块接口

信号名	方向	描述
NextPC[31:0]	I	下一条指令的地址
Reset	I	异步复位信号
clk	I	时钟
PC[31:0]	O	当前指令地址
Instr[31:0]	O	当前指令
rs[4:0]	O	当前指令的 rs 字段
rt[4:0]	O	当前指令的 rt 字段
rd[4:0]	O	当前指令的 rd 字段
Shamt[4:0]	O	当前指令的 Shamt 字段
imm16[15:0]	O	当前指令的 16 位立即数字段
OpCode[5:0]	O	当前指令的操作码字段
Func[5:0]	O	当前指令的 Func 字段

IFU 模块功能为存储指令，在时钟上升沿且 Reset=0 时从 NPC 更新 PC，并取出当前执行的指令，将其各字段分解后分别输出以供后续模块使用。当 Reset=1 时执行异步复位，将 PC 的值清零。

3. ALU

表 4 ALU 模块接口

信号名	方向	描述
In0[31:0]	I	第一个输入数据
In1[31:0]	I	第二个输入数据
ALUOp[1:0]	I	ALU 控制信号
Zero	O	输入数据是否相等
Out[31:0]	O	计算结果

表 5　ALU 功能定义

序号	功能名称	功能描述
1	加法	Out 输出两个输入数据无符号加法结果
2	减法	Out 输出两个输入数据无符号减法结果
3	按位或	Out 输出两个输入数据按位或结果
4	判断相等	Zero 输出是否相等结果

4. DM

表 6　DM 模块接口

信号名	方向	描述
AddrIn[4:0]	I	用于寻址的内存地址
DataIn[31:0]	I	要写入的数据
MemWrite	I	写使能
clk	I	时钟
Reset	I	RAM 异步复位信号
DataOut[31:0]	O	从 AddrIn 的地址读取的数据

表 7　DM 功能定义

序号	功能名称	功能描述
1	读取	DataOut 输出 RAM 的 AddrIn 地址存储的数据
2	写入	向 RAM 的 AddrIn 地址写入 DataIn 的数据
3	复位	Reset=0 时清空内存

5. 控制器

表 8　控制器模块接口

信号名	方向	描述
Func[5:0]	I	指令的 Func 字段
OpCode[5:0]	I	指令的操作码字段
RegDst	O	寄存器堆写入地址选择信号

RegWrite	O	寄存器堆写使能
ALUOp[1:0]	O	ALU 操作信号
ALUSrc	O	ALU 数据来源选择信号
MemWrite	O	DM 写使能
Branch	O	分支跳转信号
RegWriteSel[1:0]	O	寄存器堆写入数据选择信号
ShiftSel	O	移位模块选择信号
Signed	O	立即数有符号扩展信号

控制器的功能就是将操作码解码后转换为各个控制信号。其中采用了与逻辑模块和或逻辑模块。与逻辑模块用于将操作码译为对应的指令，或逻辑模块用于将与逻辑模块译码得到的指令转换为对应操作信号的通断。

与一般设计不同的是，RegWriteSel[1:0]信号用于从 ALU 运算结果、DM 读取结果、移位器运算结果三个来源中选择一个送给寄存器堆写入端；ShiftSel 信号用于控制移位模块执行 sll 指令中的左移 Shamt 位还是执行 lui 指令中的固定左移 16 位；Signed 信号用于控制对 16 位立即数做符号扩展还是零扩展。

表 9　控制信号真值表

	addu	**subu**	**ori**	**lw**	**sw**	**beq**	**lui**	**sll**
RegDst	1	1	0	0	x	x	0	1
RegWrite	1	1	1	1	0	0	1	1
ALUOp[1:0]	00	01	10	00	00	00	x	x
ALUSrc	0	0	1	1	1	0	x	X
MemWrite	0	0	0	0	1	0	0	0
Branch	0	0	0	0	0	1	0	0
RegWriteSel[1:0]	00	00	00	01	x	x	10	10
ShiftSel	x	x	x	x	x	x	1	0
Signed	x	x	0	1	1	x	0	x

6. 移位器

表 10　DM 模块接口

信号名	方向	描述
GPR[rt][31:0]	I	来自寄存器堆的输入
imm32[31:0]	I	来自指令中立即数的输入
Shamt[4:0]	I	来自指令中的移位数
ShiftSel	I	选择固定移位或可变移位
Res[31:0]	O	移位结果

表 11　DM 功能定义

序号	功能名称	功能描述
1	固定左移 16 位	ShiftSel=1 时，将 imm32 左移 16 位后从 Res 输出
2	左移 Shamt 位	ShiftSel=0 时，将 GPR[rt]左移 Shamt 位后从 Res 输出

（三）重要机制实现方法

1. nop 指令

设计了对 sll 指令的支持，因此自然而然地支持 nop 指令。

2. 立即数扩展

设计了专门的信号以决定对立即数进行有符号扩展或无符号扩展，以支持 ori 指令，且为之后添加其它涉及立即数的指令留下接口。

二、测试方案

（一）ori 测试代码

```
ori $2, $zero, 0x1234
ori $4, $zero, 0xffff
ori $8, $2, 0x436d
ori $13, $4, 0x2715
ori $25, $8, 0x0
```

（二）lui 测试代码

```
lui $0, 43
```

```
lui $1, 0xffff
lui $4, 0x9427
lui $15, 0x7219
ori $15, $15, 0x2618
lui $15, 0xabcd
lui $27, 0xecfa
ori $27, $15, 0x1235
```

（三）addu 和 subu 测试代码

```
lui $1, 0x1
ori $1, $1, 0x5f90
lui $2, 0xffff
ori $2, $2, 0xffff #li $2, -1
lui $3, 0x7fff
ori $3, $3, 0xffff #li $3, 2147483647
lui $4, 0x118
ori $4, $4, 0x6220 #li $4, 18375200
ori $5, $0, 3
addu $10, $1, $4
addu $11, $1, $2
addu $12, $2, $1
addu $13, $2, $2
subu $20, $1, $1
subu $21, $2, $2
subu $22, $1, $5
subu $23, $5, $4
subu $24, $4, $2
```

（四）lw 和 sw 测试代码

```
lui $t1, 0x3333
ori $t1, $t1, 0x3333
ori $t0, 4
```

```
addu $s0, $zero, $t0
addu $s0, $s0, $s0
sw $t1, 0($s0)
lui $t1, 0x2222
ori $t1, $t1, 0x2222
sw $t1, -4($s0)
lui $t1, 0x1111
ori $t1, $t1, 0x1111
sw $t1, -8($s0)
lui $t1, 0x4444
ori $t1, $t1, 0x4444
sw $t1, 4($s0)
lui $t1, 0x5555
ori $t1, $t1, 0x5555
sw $t1, 8($s0)
######## lw test ######
addu $s0, $zero, $zero
addu $s1, $s0, $t0
addu $s2, $s1, $t0
addu $s3, $s2, $t0
addu $s4, $s3, $t0
lw $t0, -8($s2)
lw $t1, -4($s2)
lw $t2, ($s2)
lw $t3, 4($s2)
lw $t4, 8($s2)
```

（五）sll 测试代码

```
lui $1, 0x9556
ori $1, $1, 0x592d
sll $2, $1, 1
```

```
sll $3, $1, 2
sll $4, $1, 4
sll $5, $1, 8
sll $6, $1, 15
sll $7, $1, 27
sll $8, $1, 30
sll $9, $1, 31
sll $10, $1, 0
sll $11, $0, 1
sll $12, $0, 0
nop
```

（六）beq 测试代码

```
ori $s2, $0, 4
ori $1, $0, 1
ori $2, $0, 2
ori $3, $0, 3
Switch:
beq $t1, $0, Case_3
beq $t1, $1, Case_1
beq $t1, $2, Case_2
beq $t1, $3, EndCase
Case_1:
  addu $s0, $1, $0
  sw $s0, 0($s1)
  addu $s1, $s1, $s2
  addu $t1, $t1, $1
  beq $t0, $zero, Switch
Case_2:
  addu $s0, $2, $0
  sw $s0, 0($s1)
```

```
    addu $s1, $s1, $s2
    addu $t1, $t1, $1
    beq $t0, $zero, Switch
Case_3:
    addu $s0, $0, $3
    sw $s0, 0($s1)
    addu $s1, $s1, $s2
    addu $t1, $t1, $1
    beq $t0, $zero, Switch
EndCase:
lui $s0, 0x9876
sw $s0, 0($s1)
```

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。

由于 IM 在 CPU 使用过程中只需要读取指令，不需要也不应该写入指令，所以 IM 使用 ROM 是合理的。DM 需要读写，而且需要存储较大的数据量，且对速度要求不高，因此采用 RAM 而非 Register 是合理的，可以降低成本。GRF 作为使用频率最高的存储元件，对速度有较高要求，且空间不大，用 Register 是合理的。

（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。

nop 指令只需要不修改寄存器、不修改 RAM 的值即可，则在控制信号真值表的或逻辑中，每一个信号都不必考虑 nop，这样自然不会有信号在 nop 到来时为高电平。

（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。

添加片选信号，只有当 sw 指令和 lw 指令被执行时才激活 DM 的片选信号即可。

（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。

相比于测试，形式验证可以极大地减轻验证的工作量，尤其是对于输入的组合较多，无法通过模拟仿真穷举得到所有输入的电路系统。形式验证通过证明定理或检验模型的正确性来完成对所有输入的验证，而模拟仿真一般很难做到全面，只能针对部分输入进行验证，容易遗漏。但是形式验证需要构造完备的模型，这对验证者的数学功底要求很高，验证难度也很高；而模拟仿真测试只需要不断构造样例，验证难度较低。