使用Logisim的单周期MIPS CPU设计文档

0. 综述

本CPU为Logisim实现的单周期CPU，支持的指令集包含`{addu,subu,ori,lw,sw,beq,lui,nop,sll,j,jal,jr,addiu}`。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、Ext、Control模块。

一、模块规格

1. IFU模块（取指单元）

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
PCSrc	Input	1	是否为分支/跳转指令(1是0否)
Imm	Input	[31:0]	分支/跳转指令所需的数
Instruction	Output	[31:0]	当前指令

具体功能：

功能	描述
输出下一条指令	在clk上升沿时， ① 当PCSrc为0时，$PC \leftarrow PC+4$; ② 当PCSrc为1时，$PC \leftarrow PC+4+sign\_ext(offset\|\|0^2)$
复位	reset信号变为1时，PC清零

2. GRF模块（寄存器堆）

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能端
Read1	Input	[4:0]	读寄存器编号1
Read2	Input	[4:0]	读寄存器编号2
Write	Input	[4:0]	写寄存器编号
WriteData	Input	[31:0]	写入数据
RData1	Output	[31:0]	读寄存器值1
RData2	Output	[31:0]	读寄存器值2

具体功能：

功能	描述
复位	当reset为1时，所有寄存器的值清零
读取数据	RData1的值是寄存器编号为Read1的寄存器的值；RData2的值是寄存器编号为Read2的寄存器的值
写入数据	当写使能WE为1时，向编号为Write的寄存器写入WriteData

3. ALU模块（算术逻辑单元）

端口定义：

端口名	方向	位宽	功能描述
A	Input	[31:0]	输入数据1
B	Input	[31:0]	输入数据2
ALUOp	Input	[2:0]	选择ALU功能
Shamt	Input	[4:0]	左移位数
Zero	Output	1	运算结果是否为0
Result	Output	[31:0]	运算结果

具体功能：

功能	描述
逻辑左移	`ALUOp=3'b0,Result=B<<Shamt`
与	`ALUOp=3'b001,Result=A|B`
或	`ALUOp=3'b010,Result=A&B`
减（比较）	`ALUOp=3'b011,Result=A-B`；若 `A==B,Zero=0`
加	`ALUOp=3'b100,Result=A+B`

4. DataMemory模块

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能信号
Addr	Input	[4:0]	写入地址
WD	Input	[31:0]	写入数据
RD	Output	[31:0]	读取数据

具体功能：

功能	描述
复位	Reset 为 1 时，数据清 0
写入数据	当时钟上升沿时，如果写使能信号WE有效，就将WD写入到地址为Addr处
读取数据	RD为地址Addr处的数据的值

5. Ext（数据扩展器）

端口定义：

端口名	方向	位宽	功能描述
Imm16	Input	[15:0]	输入的16位立即数
EXtCtrl	Input	[1:0]	控制信号
Imm32	Output	[31:0]	扩展结果

具体功能：

功能	描述
0扩展	`Imm32={{16{0}},Imm16}`
符号扩展	`Imm32={{16{Imm16[15]}},Imm16}`
把数加载到高位	`Imm32={Imm16,{16{0}}}`
1扩展	`Imm32={{16{1}},Imm16}`

二、控制信号

1. 指令编码

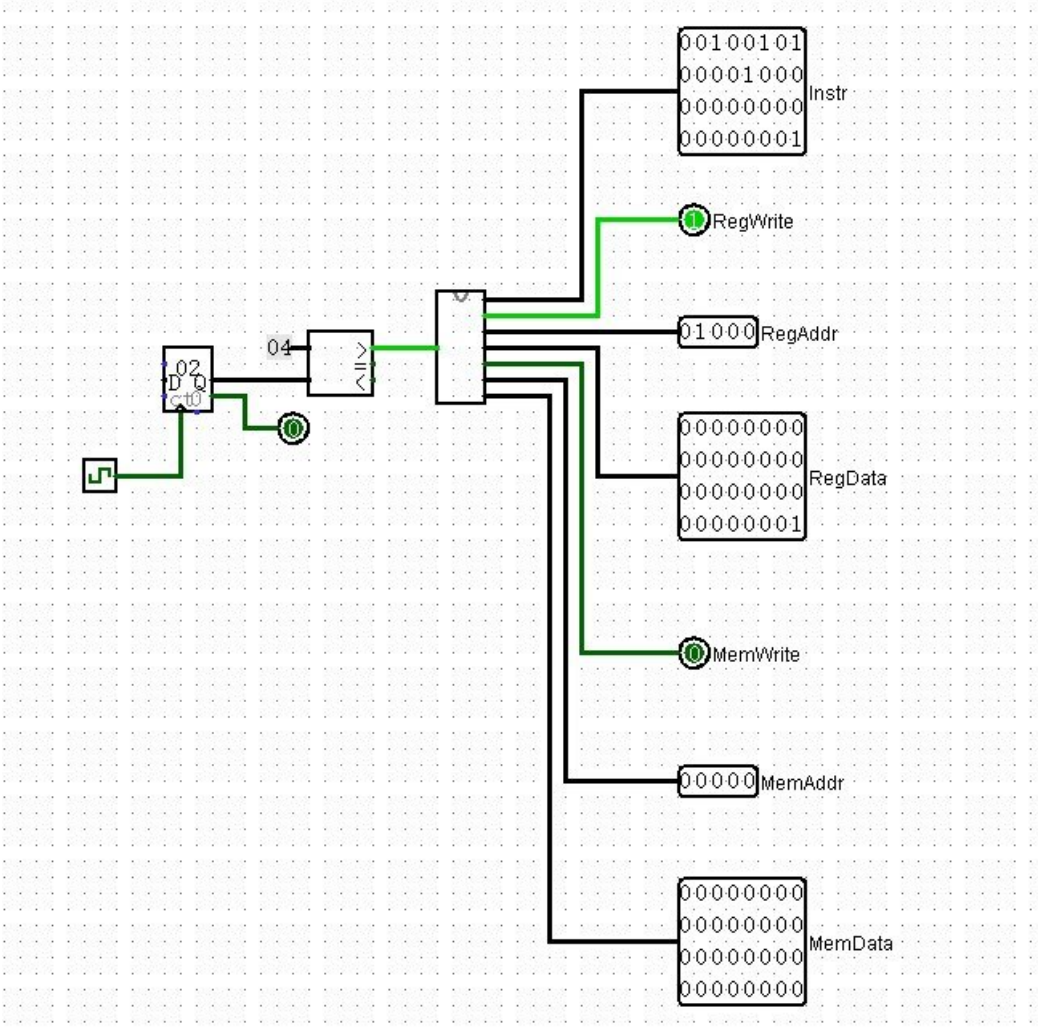
	addu	subu	ori	lw	sw	beq	lui	sll
op	`000000`	`000000`	`001101`	`100011`	`101011`	`000100`	`001111`	`000000`
func	`100001`	`100011`	`n/a`	`n/a`	`n/a`	`n/a`	`n/a`	`000000`

2. 控制信号真值表

	RegDst	RegWrite	EXTCtrl	ALUSrc	ALUOp	WE	MemToReg	Branch
作用	MUX选择写回寄存器	GRF写使能	扩展器控制	MUX选择ALU端口B的输入	ALU功能	DM写使能	MUX选择写回数据	是否是`beq`指令
addu	1	1	x	0	`3'b100`	0	0	0
subu	1	1	x	0	`3'b011`	0	0	0
ori	0	1	00	1	`3'b010`	0	0	0
lw	0	1	01	1	`3'b100`	0	1	0
sw	x	0	01	1	`3'b100`	1	x	0
beq	x	0	x	0	`3'b011`	0	x	1
lui	0	1	10	1	`3'b100`	0	0	0
nop	x	0	x	x	x	0	x	0

三、测试CPU

1. 测试电路



2. 测试集

1.

```
ori         $t0, $zero, 1   # t0=1
ori         $t1, $zero, 2   # t1=2
ori         $t3, $zero, 4   # t3=4
ori         $t2, $t0, 1     # t2=1
sw          $t2, 8($zero)
sw          $t1, 12($zero)
lw          $t2, 8($t3)     # t2=2
eq:
lw          $t1, 8($zero)   # t1=1
addu        $t4, $t1, $t0   # t4=2
addu        $1, $1, $t1     # $1=1
sw          $1, 4($zero)
lw          $2, 4($zero)    # $2=1
beq         $t4, $t0, eq    # not equal
subu        $t4, $t4, 1     # t4=1
beq         $t4, $t0, eq    # equal
```

复合指令，主要用于观测指令能否正常运行。运行过程中寄存器堆中的寄存器是否正确存入数据，DM是否正确储存值，PC是否如预期跳转(-6和-10)

2.

```
ori     $t1, $0, 32
ori     $t2, $0, 0
ori     $t0, $0, 0
nop
nop
for_begin:
beq     $t0, $t1, for_end
sw      $t0, 0($t2)
ori     $s0, $0, 4
addu    $t2, $t2, $s0
ori     $s0, $0, 1
addu    $t0, $s0, $t0
beq     $0, $0, for_begin
for_end:
nop
nop
```

期望：DM中地址0~31分别填入0~31

3.

```
# test ori
ori $a0, $0, 123
ori $a1, $a0, 456

# test lui
lui $a2, 123
lui $a3, 0xffff
ori $a3, $a3, 0xffff    # $a3 = -1

#test add
add $s0, $a0, $a2   # ++
add $s1, $a0, $a3   # +-
add $s2, $a3, $a3   # --

# test sw
ori $t0, $0, 0x0000
sw  $a0, 0($t0)
sw  $a1, 4($t0)
sw  $a2, 8($t0)
sw  $a3, 12($t0)
sw  $s0, 16($t0)
sw  $s1, 20($t0)
sw  $s2, 24($t0)

# test lw
lw  $a0, 0($t0)
lw  $a1, 12($t0)
sw  $a0, 28($t0)
sw  $a1, 32($t0)
```

```
# test beq
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, ne
beq $a0, $a2, eq

ne:
sw  $a0, 36($t0)
eq:
sw  $a1, 40($t0)
```

4.
```
# test sll,j
ori     $t0, $0, 3
nop
test:
sll     $t0, $t0, 2
nop
sll     $t1, $t0, 2
nop
j       test
```

四、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。

IM使用ROM在现阶段合理，因为在logisim中可以直接导入数据。但实际应用中，ROM大多用于BIOS存储器，图形卡、硬盘控制器等，向指令这样不是一成不变的需要经常改写，因此用的是RAM。DM使用RAM， GRF使用Register是合理的。

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指令为`32'b0`，相当于在IFU中没有加载指令时的状态。此时的寄存器堆写使能端、数据存储器写使能端都是0，其他信号如何都不会对寄存器堆与数据存储器的值进行修改。因此不会造成影响，无需将它加入控制信号真值表。`sll $0, $0, 0`对应的指令码是 0x0000_0000，也被认为是 NOP(空操作指令)。若指令集支持`sll`,便无需将`nop`加入控制信号真值表

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。

片选信号是选定芯片的信号。`.text base address`为`0x00003000`，我们需要判断地址前16位是否为`0x0003`。

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。

形式验证优点：

- 形式验证的方法有等价性检查、模型检查、定理证明等，可以覆盖所有可能的输入情况下检查是否与给定的规范一致，覆盖率达到了100%。而软件测试只能证明在当前数据集的输入下，输出与给定规范一致，并不代表被测程序绝对正确。
- 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。

缺点：

- 相较于测试，形式验证更为复杂，并且有一定的限制。