Logisim 单周期 CPU 设计文档

一、 CPU 设计方案综述

（一）总体设计概述

使用 Logisim 开发一个简单的单周期 CPU，总体概述如下：

1. 此 CPU 为 32 位 CPU
2. 此 CPU 为单周期设计
3. 此 CPU 支持的指令集为：

   {addu, subu, ori, lw, sw, beq, lui, nop}
4. nop 机器码为 0x00000000
5. addu, subu 不支持溢出

（二）关键模块定义

1. IFU

（1） 端口说明

表 1-IFU 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号，将 PC 值置为 0x00000000 0：无效 1：复位
3	beq?	I	指令是不是 beq 0：不是 1：是
4	eq?	I	A1、A2 对应 GRF 两个寄存器中的值是否相等 0：不相等 1：相等
5	imm32[31:0]	I	beq 指令中 0-15 位的 32 位符号扩展
6	instr[31:0]	O	将 IM 中，要执行的指令输出

（2） 功能定义

表 2-IFU 功能定义

序号	功能	描述

1	复位	reset 有效时，PC 置为 0x00000000
2	更新 PC 的值	当 eq?和 beq 皆为 1 时，PC <= PC + 4 + (imm32<<2) 否则，PC <= PC +4
3	输出指令	根据 PC 的值，取出 IM 中的指令

2. GRF

（1） 端口说明

表 3-GRF 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号，将 32 个寄存器中全部清零 1：清零 0：无效
3	WE	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
4	A1[4:0]	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
5	A2[4:0]	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
6	A3[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，作为 RD 的写入地址
7	WD[31:0]	I	32 位写入数据
8	RD1[31:0]	O	输出 A1 指定的寄存器的 32 位数据
9	RD2[31:0]	O	输出 A2 指定的寄存器的 32 位数据

（2） 功能定义

表 4-GRF 功能定义

序号	功能	描述
1	异步复位	reset 为 1 时，将所有寄存器清零
2	读数据	将 A1 和 A2 地址对应的寄存器的值分别通过 RD1 和 RD2 读出
3	写数据	当 WE 为 1 且时钟上升沿来临时，将 WD 写入到 A3 对应的寄存器内部

3. ALU

(1)端口说明

表 5-ALU 端口说明

序号	信号名	方向	描述

1	A[31:0]	I	参与运算的第一个数
2	B[31:0]	I	参与运算的第二个数
3	ALUop[2:0]	I	决定 ALU 做何种操作 000：无符号加 001：无符号减 010：与 011：或
4	eq?	O	A 与 B 是否相等 0：不相等 1：相等
5	res	O	A 与 B 做运算后的结果

(2)功能定义

表 6-ALU 功能定义

序号	功能	描述
1	加运算	res = A + B
2	减运算	res = A - B
3	与运算	res = A & B
4	或运算	res = A | B

4. DM

(1)端口说明

表 7-DM 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号 0：无效 1：内存值全部清零
3	WE	I	写使能信号 0：禁止写入 1：允许写入
4	A[4:0]	I	读取或写入信号地址
5	WD[31:0]	I	32 为写入数据
6	RD[31:0]	O	32 位读出数据

(2)功能定义

表 8-DM 功能定义

序号	功能	描述
1	异步复位	当 reset 为 1 时，DM 中所有数据清零

2	写入数据	当 WE 有效时，时钟上升沿来临时，WD 中数据写入 A 对应的 DM 地址中
3	读出数据	RD 永远读出 A 对应的 DM 地址中的值

5. EXT

(1)端口说明

表 9-EXT 端口说明

序号	信号名	方向	描述
1	imm16[15:0]	I	代扩展的 16 位信号
2	sign?	I	无符号或符号扩展选择信号 0：无符号扩展 1：符号扩展
3	imm32[31:0]	O	扩展后的 32 位的信号

(2)功能定义

表 10-EXT 功能定义

序号	功能	描述
1	无符号扩展	当 sign?为 0 时，将 imm16 无符号扩展输出
2	符号扩展	当 sign?为 1 时，将 imm16 符号扩展输出

6. Controller

(1)端口说明

表 11-Controller 端口说明

序号	信号名	方向	描述
1	op[5:0]	I	instr[31:26] 6 位控制信号
2	func	I	instr[0:5] 6 位控制信号
3	AluOp[2:0]	O	ALU 的控制信号
4	WeGrf	O	GRF 写使能信号 0：禁止写入 1：允许写入
5	WeDm	O	DM 的写入信号 0：禁止写入 1：允许写入
6	beq?	O	instr 是否为 beq 信号 0：不是 1：是
7	AluSrc	O	参与 ALU 运算的第二个数，来自 GRF 还是 imm 0：来自 GRF 1：imm
8	WhichtoReg [1:0]	O	将何种数据写入 GRF? 00：ALU 计算结果 01：DM 读出信号

			11：upperImm
9	RegDst	O	GRF 写入地址选择信号 0：Rd 1：Rt
10	SignExt?	O	是否对 imm16 进行符号扩展 0：不进行符号扩展 1：进行符号扩展

(2)真值表

端口	addu	subu	ori	lw	sw	lui	beq
op	000000	000000	001101	100011	101011	001111	000100
func	100001	100011					
AluOp	000	001	011	000	000	000	000
WeGrf	1	1	1	1	0	1	0
WeDm	0	0	0	0	1	0	0
beq?	0	0	0	0	0	0	1
AluSrc	0	0	1	1	1	0	0
WhichtoReg	00	00	00	01	00	10	00
RegDst	0	0	1	1	1	1	1
SignExt?	0	0	0	1	1	0	1

二、 测试方案

（1）测试代码：

```
.text
   ori      $t1, $v0, 100  # t1: 100
   ori      $t2, $v0, 250  # t2: 250
   ori      $t3, $v0, 200  # t3: 200
   addu     $t4, $t1, $t3
   # t4 = t1 + t3 (t4 = 300)
   subu     $t5, $t4, $t2
   # t5 = t4 - t2 (t5 = 50)
   sw    $t5, 4($v0)
   lw    $t6, 4($v0)
Beq:
   lui      $t7, 100
   beq      $t5, $t1, Beq
   beq      $t6, $t5, Beq
```

（2）MARS 中运行结果

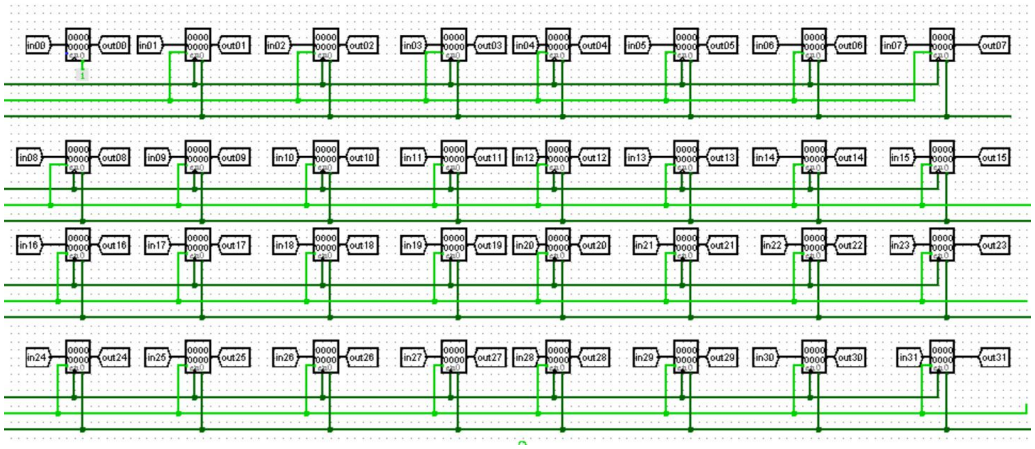
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	100
$t2	10	250
$t3	11	200
$t4	12	300
$t5	13	50
$t6	14	50
$t7	15	6553600
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	6144
$sp	29	12284
$fp	30	0
$ra	31	0
pc		12324
hi		0
lo		0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0	0	50	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0

（3）该 CPU 运行结果



00	00000000	00000032	00000000	00000000	00000000	00000000	00000000	00000000
08	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
18	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

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。

合理。
IM 只需被读取，ROM 只有读取功能；
DM 既要进行读取，又要进行写入，但是一个周期只会进行读取和写入之一，RAM 的单一地址和各一个的读写端口满足了这种要求，
当然，用寄存器也能实现 DM，但是 DM 需要较大的空间，使用寄存器太“浪费”；
GRF 需要读写，且其与 ALU 直接连接，需要高速地读写，故使用寄存器堆搭建合理。
抱歉，没有改进意见。

（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，整个 CPU 只执行 PC<=PC+4 指令，一段指令中，加与不加，执行结果没有区别。

（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。

如果很不幸，寄存器中存储的 DM 的地址被映射在 0x3000_0000 到 0x3fff_ffff 间，而我们的 DM 起始地址是 0，那么，我们可以将输入地址直接减去 0x3000_0000，再作为 DM 的地址输入。

假如我们不确定寄存器中的存储的 DM 地址的起始值，我们可以将其与 0x3000_0000 比较，得到片选信号。

（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。

形式验证的优点如下：

（1） 所有可能的情况进行验证，覆盖率达到了100%。

（2） 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。

形式验证的缺点如下：

（1） 形式验证只能检验电路设计的正确性，却无法检验其它方面如电路能耗等的优劣。