P3 logisim 单周期 CPU 设计文档

19373490-吴昊天

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 logisim 设计的单周期 CPU, 支持的指令集包括{addu,subu,ori,lw,sw,beq,sltu,lui,nop,jal,jr}。为了实现功能，主要包括 IFU，GRF，ALU，DM，Controller 以及 MUX 若干。处理器采用内置时钟信号，顶层有效驱动信号只有 reset。

（二）关键模块定义

1、IFU（取指令单元）：内部包括 PC（程序计数器）、NPC（次地址计算单元）、IM（指令存储器）等相关逻辑。

（1）PC 用寄存器实现，具有异步复位功能。起始地址 0x0000_0000。

（2）IM 用 ROM 实现，容量为 32 bit *32。IM 实际地址宽度为 5 位，而在本 CPU 中，以字为单位取值，PC 每次加 4，因此取 PC 的 2-6 位来当做地址线。

表 1、PC

功能描述	32 位可复位寄存器 reset 高电平有效，寄存器初值为 0x0000_0000	
信号名	方向	描述
clk	I	MIPS-C 处理器时钟
reset	I	复位信号
DI[31:0]	I	32 位输入
DO[32:0]	O	32 位输出

（3）NPC 相当于一个组合逻辑，计算次地址的单元。

表 2、NPC

信号名	方向	描述
PC[31:0]	I	当前地址
Imm[31:0]	I	16 位立即数

Instr_index[25:0]	I	26 位地址
NPCOp[2:0]	I	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 beq 地址 010：计算 jal 地址 011：计算 jr 地址
zero	I	rs 和 rt 相等判断 1：相等 0：不相等
RA[31:0]	I	32 位输入地址
NPC[31:0]	O	32 位次地址
PC4[31:0]	O	32 位 PC+4

（4）其中本 CPU 为排线整洁便于区分，单独将分线元件归置到一个名为 IMSplitter 的模块中，与 ROM 结合在一起充当 IM 的作用。

表 3、IMSplitter

信号名	方向	描述
Instr[31:0]	I	32 位指令
rs[4:0]	O	5 位寄存器编号
rt[4:0]	O	5 位寄存器编号
rd[4:0]	O	5 位寄存器编号
Imm[15:0]	O	16 位立即数
Instr_index[25:0]	O	26 位地址

2、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

用具有写使能的寄存器实现，寄存器总数为 32 个，具有异步复位功能。

表 4、GRF

信号名	方向	描述
clk	I	时钟
reset	I	复位信号
RFWr	I	使能信号，时钟上升沿

		时为 1 有效
A1[4:0]	I	第一个读出寄存器编号
A2[4:0]	I	第二个读出寄存器编号
A3[4:0]	I	回写寄存器编号
RD1[31:0]	O	第一个读出的寄存器值
RD2[31:0]	O	第二个读出的寄存器值
WD[31:0]	I	回写寄存器的值

3、ALU（算术逻辑单元）

提供 32 位加、减、或运算及比较大小功能，控制信号为 ALUOp。

表 5、ALU

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
ALUOp[2:0]	I	ALU 功能选择 000：A+B 001：A-B 010：A or B 011：SLT 其余保留
C[31:0]	O	32 位计算结果
Zero	O	A 和 B 相等比较结果 1：A＝B 0：A≠B

其中 ALU 内部控制信号模块为 ALUF，输入为 ALUOp，输出为控制信号，控制信号及真值表如下。

表 5(1)、 ALU 内部控制

ALUOp	M1Sel	C
000	0	0
001	1	1

010	0	0
011	0	0

4、DM（数据存储器）使用 RAM 实现，容量为 32bit * 32。具有异步复位功能，复位值为 0x0000_0000。

表 6、DM

信号名	方向	描述
clk	I	时钟
reset	I	复位信号
A[4:0]	I	5 位地址
DMWr	I	使能信号，时钟上升沿为 1 有效
DI[31:0]	I	写入数据
DO[31:0]	O	读出数据

5、EXT（扩展单元）用 logisim 自带的 Bit Extend 实现，其中控制信号为 EXTOp 以选择功能。

表 7、EXT

信号名	方向	描述
Imm[15:0]	I	16 位立即数
EXTOp[1:0]	I	EXT 功能选择信号 00：符号扩展 01：无符号扩展 10：加载立即数至高位 11：保留
EXTDM[31:0]	O	输出数据

（三）控制器设计及控制单元

CPU 中 Controller 包含两个部分，分别为 Controller_AND 和 Controller_OR，分别对应和逻辑和或逻辑。其中 Controller_AND 的功能是识别出相应的指令，而 Controller_OR 的功能是据输入的指令的不同，生成对应的控制信号逻辑表达式。

在 Controller_AND 逻辑中主要分为两部分，一部分通过 opcode[5:0]来判断指令，对于 R 型指令，其 opcode 为 000000，因此用一个 Rtype 信号来标志 R 型指令，其他位信号以逻辑与生成指令信号。

在 Controller_OR 逻辑中，对应指令信号线横置，用逻辑或生成对应信号。

在 CPU 中总共有 8 个控制信号，功能区分如下表：

表 8、控制信号功能

信号名	描述
NPCOp[2:0]	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 beq 地址 010：计算 jal 地址 011：计算 jr 地址
M1Sel[1:0]	回写寄存器选择： 00：回写 rt 01：回写 rd 10：回写$ra 11：保留
M2Sel[1:0]	回写数据选择： 00：ALU 计算所得数据 01：DM 读出数据 10：PC+4 11：保留
RFWr	寄存器写使能信号 1：上升沿写入数据 0：写入无效
EXTOp[1:0]	扩展单元功能选择： 00：符号扩展 01：无符号扩展 10：加载立即数至高位

	11：保留
M3Sel	ALU 计算数选择： 0: 寄存器读出数据二 1：扩展单元所得立即数
ALUOp[2:0]	ALU 功能选择 000：A+B 001：A-B 010：A or B 011：SLT 其余保留
DMWr	DM 写使能信号： 1：上升沿数据写入 0：写入无效

表 9、指令与信号对应关系

指令\信号名	NPCOp	M1Sel	M2Sel	RFWr	EXTOp	M3Sel	ALUOp	DMWr
addu	000	01	00	1	00	0	000	0
subu	000	01	00	1	00	0	001	0
ori	000	00	00	1	01	1	010	0
lw	000	00	01	1	00	1	000	0
sw	000	00	00	0	00	1	000	1
beq	001	00	00	0	00	0	001	0
lui	000	00	00	1	10	1	000	0
sltu	000	01	00	1	00	0	011	0
jal	010	10	10	1	00	0	000	0
jr	011	00	00	0	00	0	000	0

二、测试方案

（一）测试程序 1：

```
ori $a0,$0,345
```

```
ori $a1,$a0,123
lui $t0,0xffff   ###t0 负数
lui $t1,123     ###t1 正数
ori $t0,$t0,0x12
ori $t1,$t1,0x34
addu $a2,$t0,$0
addu $t2,$t1,$a0
addu $t3,$t0,$a0
addu $t4,$t0,$t0
subu $t5,$t1,$a0
subu $t6,$t0,$a0
subu $t7,$t3,$t4
ori $t0,$0,0
sw $a0,0($t0)
sw $a1,4($t0)
sw $a2,8($t0)
lw $s0,8($t0)
lw $s1,4($t0)
lw $s2,0($t0)
sltu $a0,$s0,$s1
sltu $a1,$s0,$s0
```

2、期望结果：

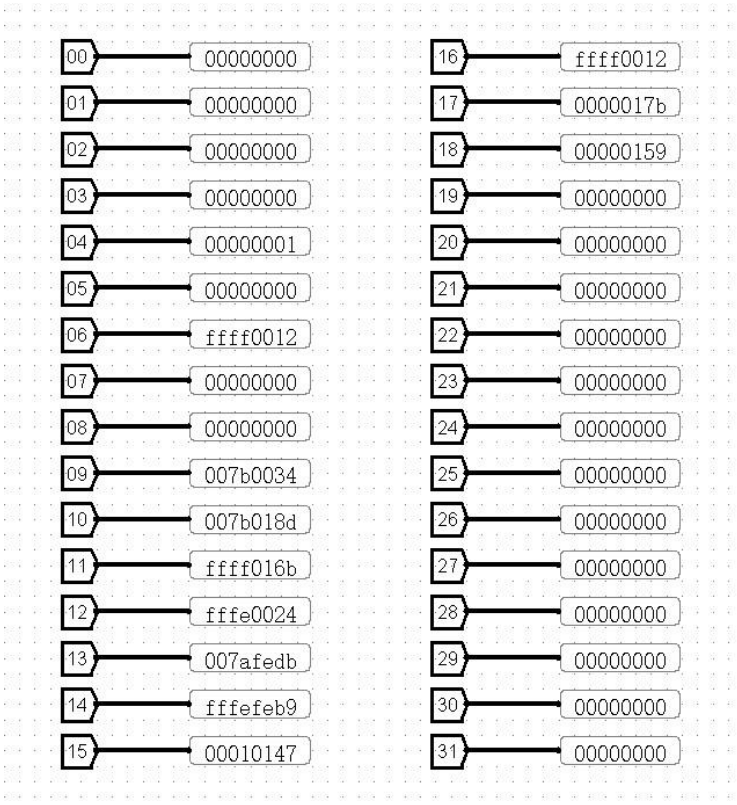
GRF:

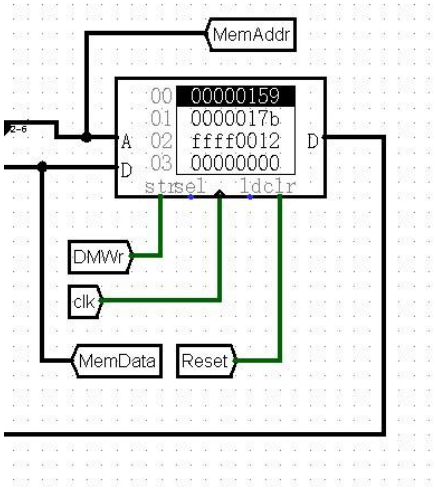
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0xffff0012
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x007b0034
$t2	10	0x007b018d
$t3	11	0xffff016b
$t4	12	0xfffe0024
$t5	13	0x007afedb
$t6	14	0xfffefeb9
$t7	15	0x00010147
$s0	16	0xffff0012
$s1	17	0x0000017b
$s2	18	0x00000159
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003058
hi		0x00000000
lo		0x00000000

DM:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000159	0x0000017b	0xffff0012	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

3、测试结果：





（二）测试程序 2：

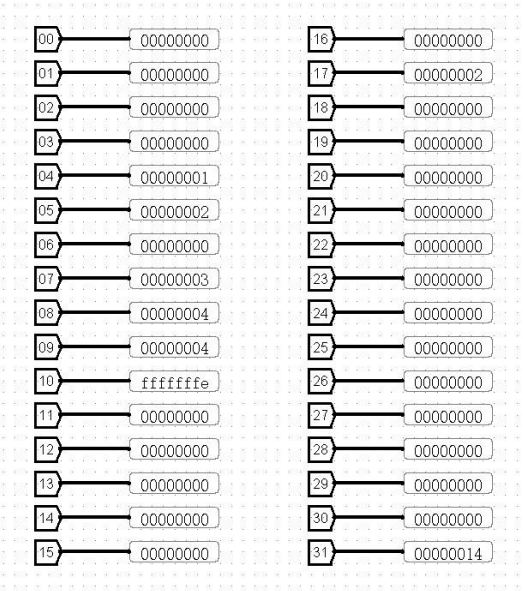
```
ori $a0,$0,1
ori $a1,$0,2
addu $a3,$a1,$a0
addu $t1,$a0,$a3
jal label
addu $t1,$a0,$a3
beq $t0,$t2,loop1
```

```
beq $t0,$t1,loop2
label:
addu $t0,$a0,$a3
subu $t2,$a0,$a3
jr $31
loop1:
ori $s0,$0,1
loop2:
ori $s1,$0,2
```

2、期望结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000001
$a1	5	0x00000002
$a2	6	0x00000000
$a3	7	0x00000003
$t0	8	0x00000004
$t1	9	0x00000004
$t2	10	0xfffffffe
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000002
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003014
pc		0x00003034
hi		0x00000000
lo		0x00000000

3、测试结果：



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。

IM 使用 ROM 可以导入指令且无需修改，每次只会读取一个内存单元；DM 需要对数据进行读写操作；GRF 可以写入、读出数据，同时可以清零。因此，IM 使用 ROM，DM 使用 RAM，GRF 使用 Register 是合理的。但也有需要改进的地方，PC 是按字读入，每次更迭加 4，因此需要取地址的 2-6 位作为地址线。

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 nop 空指令对应的控制信号都为 0 或不相关 x,可简化为所有控制信号都为 0。对于控制信号逻辑或表达式，A+0=A，因此不需要将它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。

我们已经约定在 MARS 中选择“Compact，Data at Address 0”，因此 DM 的地址就是从零开始的，对于 sw、lw 指令，写入和读取地址时按字读取，因此 DM 可以取地址的 2-6 位。

对于 PC 起始不为 0 的问题，只在 jal 和 jr 指令中有影响。其中 jal 和 jr 采用

的是绝对地址，会因为起始地址非 0 的问题产生一个 0x00003000 的偏差。解决方案是在 NPC 中计算下一条地址时采用减法器减去偏移差 0x0000_3000，同时存入寄存器的也是修正后的地址，即可解决 PC 起始地址不为 0 的情况正常支持 jal 和 jr 指令。

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。

形式验证的优点如下：

(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了 100%。

(2)形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励。

(3)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：对于时序逻辑的验证，无法模拟出元件的物理特性以及延迟。