单周期CPU设计文档

零、简要需求

CPU支持的指令集：{addu subu ori lui lw sw beq nop}

CPU 结构：单周期

指令起始地址：0x00003000

数据起始地址：0x00000000

一、模块规格

1、取指令单元IFU

IFU包含程序计数器PC、指令存储器IM、次地址计算单元NPC

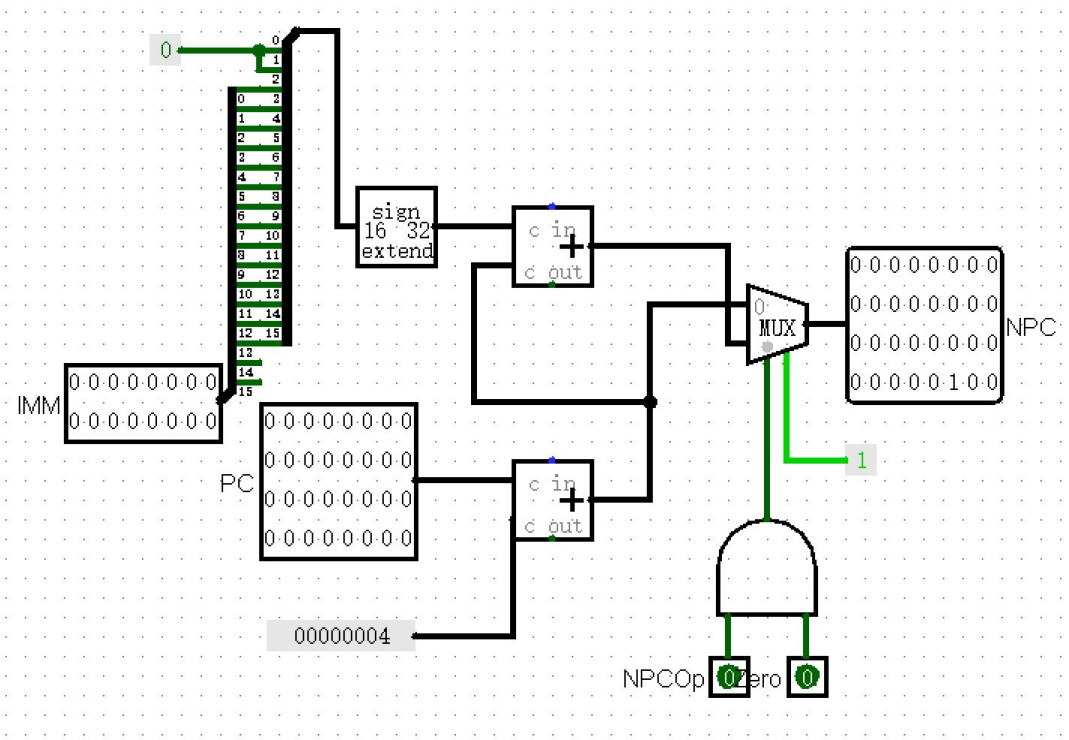
（异步复位功能：复位不受时钟控制，复位信号生效即可复位）

（1）PC的规格

功能描述	**32位可异步复位寄存器，起始地址为0x00000000**	
信号名	方向	功能
Clk（内置）	I	CPU时钟
Reset	I	异步复位信号，将PC内的值赋值为0x00000000
Di[31:0]	I	32位输入
Do[31:0]	O	32位输出

（2）NPC的规格

功能描述	**32位计算单元，根据指令要求决定下一条指令地址为PC+4还是分支地址**	
信号名	方向	功能
PC[31:0]	I	来自PC的32位地址输入
IMM[25:0]	I	26位立即数，16位来自beq指令，需要与PC相加；26位来自jal指令，左移两位后与PC相加
NPCOp[1:0]	I	beq标志，10表示jal，01表示为beq，00表示顺序执行
Zero	I	1表示rs==rt，0表示rs!=rt
NPC[31:0]	O	次地址输出



（3）IM的规格

32位地址中，仅有低5位有效！本CPU全部为word操作，不需要详细到byte

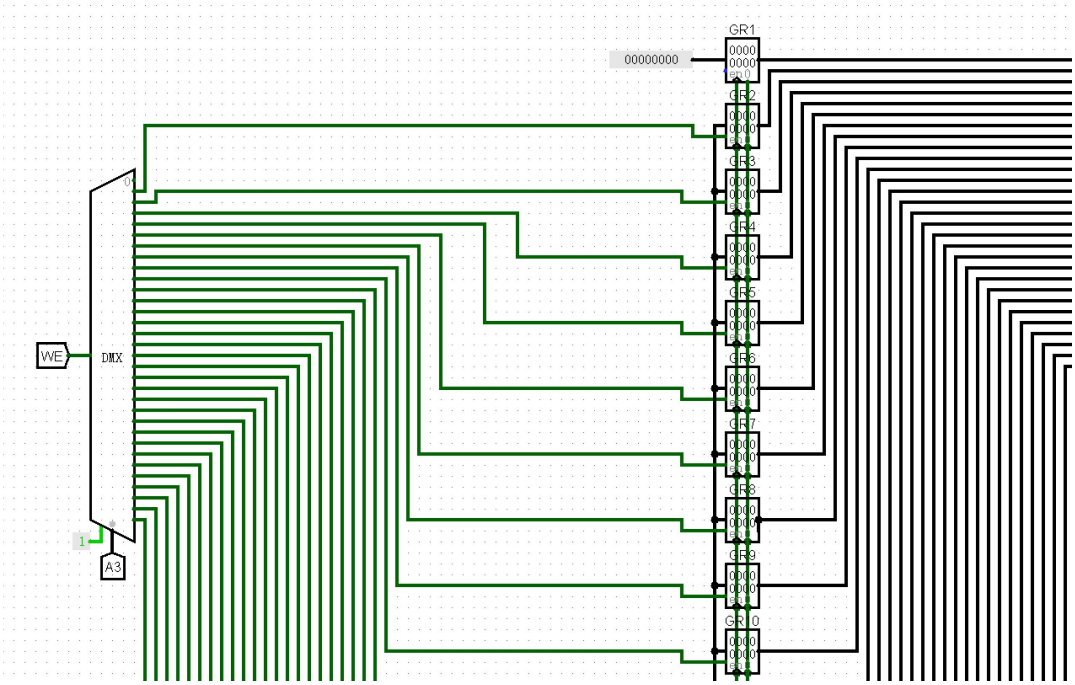
logisimROM需要5位信号控制，因为它要控制到byte级别。

功能描述	**ROM实现，大小为32*32bit，根据输入地址，输出指令数据**	
信号名	方向	功能
A[31:0]	I	32位地址输入
D[31:0]	O	32位指令输出

2、GRF

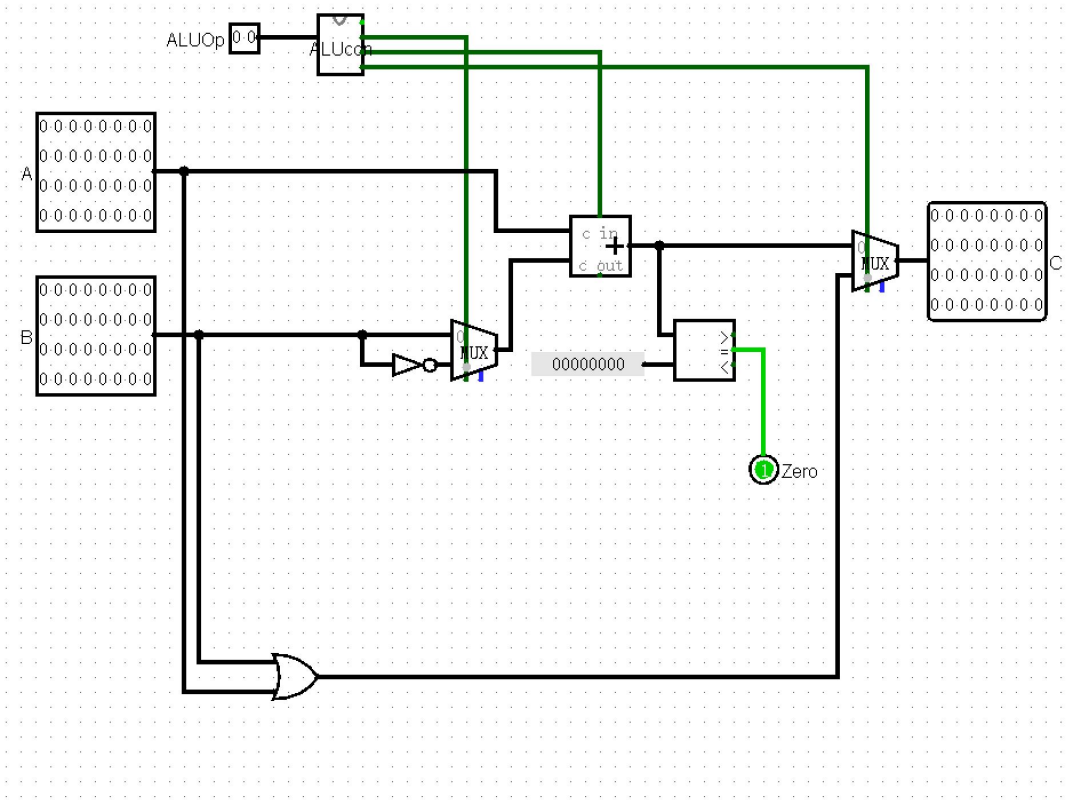
功能描述	**32个32位寄存器，具有异步复位功能，0号寄存器接地；读出时将A1和A2对应的寄存器值通过RD1和RD2输出；写入时如果Wr有效，则将WD输入写入A3寄存器**	
信号名	方向	功能
A1[4:0]	I	第一个读出寄存器的编号
A2[4:0]	I	第二个读出寄存器的编号
A3[4:0]	I	写入寄存器的编号
RD1[31:0]	O	第一个寄存器编号读出的寄存器值
RD2[31:0]	O	第二个寄存器编号读出的寄存器值
WD[31:0]	I	写入寄存器的值
RFWr	I	写入使能
Clk（内置）	I	CPU时钟
Reset	I	异步复位信号

部分结构如下：



3、ALU

电路图可以参考高小鹏书P122



（1）内部器件：ALU控制器、加法器、反相器、或运算器、bit-extend（借用外部的）

（2）功能分析：

	MIPS指令	addu	subu	ori	lw	sw	beq
	计算需求	**加法**	**减法**	**立即数或运算**	**加法**	**加法**	**相等比较**
功能分析	加法器B端选择	原	反相	-	原	原	反相
	加法器进位	0	1	-	0	0	1
	扩展方式	符号扩展	符号扩展	-	符号扩展	符号扩展	-

（3）输入输出信号：

功能描述	实现加、减、或、比较大小功能（检测溢出）	
信号名	**方向**	**功能**
ALUOp[1:0]	I	控制信号，00-加，01-减，10-（保留给比较），11-或
Zero	O	输出beq比较结果，1为等于，0为不等
A[31:0]	I	第一个运算数
B[31:0]	I	第二个运算数
C[31:0]	I	ALU运算结果

（4）ALU的控制器：

F[1:0]	**SExt**	**M1Sel**	**Cin**	**M2Sel**
00	1	0	0	0
01	1	1	1	0
10	-	-	-	--
11	-	-	-	1

4、DM

RAM调为separate模式，str为Wr信号，sel和ld信号可以全置1

功能描述	读写存储器	
信号名	**方向**	**功能**
A[31:0]	I	要写入或读出的地址（需要转换为5位地址）
WD[31:0]	I	写入的32数据
RD[31:0]	O	读出的32位数据
Wr（为RAM separate模式的str信号）	I	写入使能
Clk（内置）	I	CPU时钟
Reset	I	异步复位信号

5、EXT

需要选择是zero还是sign

功能描述	实现符号扩展和无符号扩展	
信号名	**方向**	**功能**
EXTOp[1:0]	I	控制信号，00为无符号扩展，01为有符号扩展,10为lui的扩展
Imm[15:0]	I	待扩展的16位立即数
Ext[31:0]	O	输出32位扩展的数字

二、数据通路

1、指令分析

数据通路

控制部件	PC	NPC		IM	GRF				EXT	ALU	
输入信号	DI	PC	Imm	A	A1	A2	A3	WD	Imm	A	B
addu	NPC.NPC	PC.Do		PC.Do	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD
subu	NPC.NPC	PC.Do		PC.Do	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD
ori	NPC.NPC	PC.Do		PC.Do	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[15:0]	RF.RD1	EXT.Ex
lw	NPC.NPC	PC.Do		PC.Do	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:0]	RF.RD1	EXT.Ex
sw	NPC.NPC	PC.Do		PC.Do	IM.D[25:21]	IM.D[20:16]			IM.D[15:0]	RF.RD1	EXT.Ex
beq	NPC.NPC	PC.Do	IM.D[25:0] (实际16位)	PC.Do	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD
lui	NPC.NPC	PC.Do		PC.Do			IM.D[20:16]	EXT.Ext	Im.D[15:0]		
jal	NPC.NPC	PC.Do	IM.D[25:0]	PC.Do			31	PC.Do8			

数据通路

控制部件	EXT	ALU		DM	
输入信号	Imm	A	B	A	WD
addu		RF.RD1	RF.RD2		
subu		RF.RD1	RF.RD2		
ori	IM.D[15:0]	RF.RD1	EXT.Ext		
lw	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C	
sw	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C	RF.RD2
beq		RF.RD1	RF.RD2		
lui	Im.D[15:0]				
jal					

需要的MUX：GRF.A3——GRF.A3Sel（IM.D[15:11]-00，IM.D[20:16]-01，31-10），GRF.WD——GRF.WDSel（ALU.C-00，DM.RD-01，EXT.Ext-10，PC.Do8-11），ALU.B——ALU.BSel，

控制信号矩阵

指令	NPCOp	RFWr	EXTOp	ALUOp	DMWr	GRF.A3Sel	GRF.WDSel	ALU.BSel
addu	00	1	-	00	0	00	00	0
subu	00	1	-	01	0	00	00	0
ori	00	1	00	11	0	01	00	1
lw	00	1	01	00	0	01	01	1
sw	00	0	01	00	1	--	--	1
beq	01	-	--	01	0	--	--	0
lui	00	1	10	--	0	01	10	-
jal	10	1	--	--	0	10	11	-

思考题：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：我觉得DM和GRF使用的设备完全合理，IM使用的设备比较合理。美中不足的地方是IM使用ROM之后，只能一次性写入指令存储器，而不能多次写入程序，CPU的复用性不强。如果能用RAM并克服控制困难，那将会是更具有泛化性的。

三、控制器设计

需要控制的信号：见上图，控制信号矩阵。

1、和逻辑部分

R型指令：addu、subu

I型指令：ori、lw、sw、beq、lui

和逻辑第一层：识别opcode，输出结果包含：R、ori、lw、sw、beq、lui

和逻辑第二层：根据funct，将R型指令具体划分，得到输出结果addu、subu。

附每条指令的机器码：

addu 000000 100001

subu 000000 100011

ori 001101

lw 100011

sw 101011

beq 000100

lui 001111

nop 000000 000000

jal 000011

2、或逻辑部分

输入为以上七条指令信号，输出为控制信号矩阵，以此法设计真值表即可。

思考题：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？

答：nop指令的编码为32位的0，前六位0会意味着它进入R型指令范围，而末6位0意味着它既不属于addu也不属于subu，也就是和逻辑的所有输出结果都是0.这也就意味着，controller中所有控制信号将均为未激活状态。因此，在这个时钟周期内，CPU什么都不会执行，达到了nop指令的效果。

四、测试部分

我选择了逐条指令测试。测试时主要方法为常规数据+各种能想到的极端情况。测试时观察的指标主要是32个寄存器的值，以及各个控制信号的值。

对于每条指令，测试了以下数据：

1、addu

- addu 31寄存器，30寄存器（初值为-2），29寄存器（初值为1）——机器码为00000011110111011111100000100001
- addu 31寄存器，31寄存器（初值为-1），29寄存器（初值为1）——机器码为00000011111111101111100000100001

2、subu

- subu 31寄存器，31寄存器（初值为-1），29寄存器（初值为1）——机器码为00000011111111101111100000100011
- subu 18寄存器，17寄存器（初值为0xffff9880），16寄存器（初值为0x9288fd5a）——机器码为00000010010100001000100000100011

3、lui

- lui 30寄存器，-1——机器码为00111100000111101111111111111111

4、ori

- ori 30寄存器，30寄存器，-2——机器码为00110111110111101111111111111110
- ori 29寄存器，29寄存器，1——机器码为00110111101111010000000000000001

5、beq

- 跳转情况：beq 0寄存器，2寄存器（寄存器内值为0），2
- 不跳转情况：beq 0寄存器，2寄存器（寄存器内值为-1），2

6、lw

- 由于单独测试lw指令，无法在存储器内写入非0值，因此采用了在下述测试中进行综合测试的方法进行测试

7、sw

- 理由同上

8、jal

- jal 0x00000002——机器码00001100000000000000000000000010

目前状态：通过！

五、综合测试

使用Mars构建包含所有测试集指令的MIPS程序，导出机器码，令CPU执行，将寄存器和存储器的值与Mars执行结果进行对比。效果如下：

代码为（li指令会被拆分为lui和ori）

```
lui $18, 0x0000ffff
ori $18, $18, 0x00008901
lui $17, 0x000098d8
ori $17, $17, 0x000092ab
subu    $19, $18, $17
lui $16, 0x00008324
ori $16, $16, 0x0000a987
addu    $15, $16, $18
jal label
label:  subu    $14, $19, $16
addu    $13, $31, $18
jal label2
label2: addu    $20, $31, $16
```

机器码为

```
v2.0 raw
3c12ffff
36528901
3c1198d8
363192ab
02519823
3c108324
3610a987
02127821
0c000c09
02707023
03f26821
0c000c0c
03f0a021
```

执行后，检查每个寄存器的数值，除了$sp等特殊寄存器之外，其它寄存器的数值均与Mars执行结果相同。

思考题：上文提到，MARS 不能导出 PC 与 DM 起始地址均为 000 的机器码。实际上，可以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。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：对于第一个问题，我的处理方法是这样的：注意到我们的指令集中只有jal指令需要在指令机器码中写入指令地址，因此单独针对jal指令进行修改。当确定指令为jal时，我将传入寄存器的26位立即数先扩展两位后，提前减掉0x00003000的值，供PC和NPC使用；而将当前指令地址写入寄存器前，我把写入值再加回来一个0x00003000，这样就在总体效果上看上去没有了问题。

对于第二个问题，形式验证的优点在于验证时间短，操作方便等；而相比于测试，形式验证最大的缺点是并没有落在硬件层面，难以确保理论上没问题的事情，在硬件层面依旧没问题。