P3 Logisim 单周期 CPU 设计文档

1. 整体结构

本次 CPU 设计为 Logisim 单周期 CPU（32 位）设计，基本思路是通过 MUX、Splitter 等内置器件将 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件连接成数据通路，可支持的指令集：**{addu, subu, ori, lw, sw, beq, lui, nop}**。

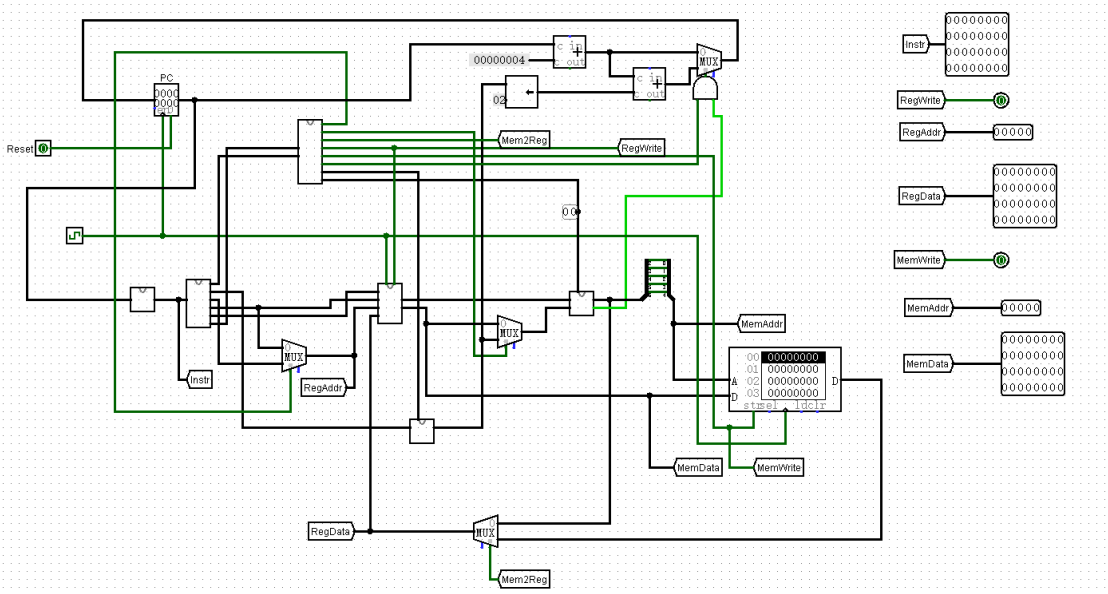

图 1 顶层设计图

2. 模块规格

由于 Controller 较为特殊，在 CPU 中占有核心地位，将在下一章节描述设计，以下为其他基本元件的设计描述。

ALU 功能定义与端口说明：

表 1 ALU 功能定义

序号	功能名称	功能描述
1	计算	根据选择信号（ALUOp[1:0]）对输入的 A, B 进行计算，输出结果 C。 00: C = A + B 01: C = A – B 10: C = A | B

表 2 ALU 端口说明

序号	端口名称	功能描述
1	ALUOp[1:0] (I)	ALU 运算选择信号
2	A[31:0] (I)	立即数 A 的输入端口
3	B[31:0] (I)	立即数 B 的输入端口
4	Clk (I)	时钟信号
5	C[31:0] (O)	运算结果 C 的输出端口
6	Zero (O)	Zero = A==B?1:0

GRF 功能定义与端口说明：

表 3 GRF 功能定义

序号	功能名称	功能描述
1	使能	当使能信号有效且在时钟上升沿时将写入数据
2	存储数据	将数据 写入指定寄存器
3	读取数据	从指定寄存器读取所需数据

表 4 GRF 端口说明

序号	端口名称	功能描述
1	RA1 (I)	待读寄存器的地址 address1
2	RA2 (I)	待读寄存器的地址 address2
3	WA (I)	待写寄存器的地址 address3
4	WD (I)	待写入地址为 address3 寄存器的值
5	Clk (I)	时钟信号
6	RD1 (O)	地址为 address1 的寄存器的值 data1 输出端口
7	RD2 (O)	地址为 address2 的寄存器的值 data2 输出端口

IFU 功能定义与端口说明：

表 5 IFU 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00000000
2	取指令	根据 PC 从 IM 中取出指令
3	计算下条指令地址	根据不是跳转指令 PC = PC + 4，如果是 beq，计算跳转后指令地址

表 6 IFU 端口说明

序号	端口名称	功能描述
1	PC (I)	输入当前的 PC 值
2	Reset (I)	输入复位信号
3	Clk (I)	时钟信号
4	Instr (O)	输出取出的指令

Splitter 功能定义与端口说明：

表 7 Splitter 功能定义

序号	功能名称	功能描述
1	分离	从 Instr 中分离出 func、immediate、Rd、Rt、Rs、op

表 8 Splitter 端口说明

序号	端口名称	功能描述
1	Instr (I)	指令的输入端口
2	func (O)	R 型指令的 function 输出
3	immediate (O)	I 型指令的立即数输出
4	Rd (O)	寄存器编号输出
5	Rt (O)	寄存器编号输出
6	Rs (O)	寄存器编号输出
7	op (O)	指令的 op 输出

EXT 功能定义与端口说明：

表 9 EXT 功能定义

序号	功能名称	功能描述
1	立即数扩展	根据 Extop[1:0]信号对输入立即数 A 进行扩展 00: 对输入立即数 A 进行零扩展 01: 对输入立即数 A 进行符号扩展 10: 对输入立即数 A 进行加载至高位运算

表 10 EXT 端口说明

序号	端口名称	功能描述
1	A (I)	立即数 A(16bits)的输入端口
2	Extop[1:0] (I)	EXT 选择信号
3	B (O)	结果 B(32bits)的输出端口

DM 功能定义与端口说明：

表 11 DM 功能定义

序号	功能名称	功能描述
1	使能	当使能信号有效且在时钟上升沿时将写入数据
2	写入数据	将数据写在指定地址上
3	读取数据	将指定地址的数据读取出来

表 12 DM 端口说明

序号	端口名称	功能描述
1	A (I)	要写入数据的 RAM 地址
2	D (I)	写入 RAM 的数据
3	MemWrite (I)	写操作的使能信号
4	Clk (I)	时钟信号
5	RD (O)	RAM 读取的数据

3. 控制器设计

控制的本质就是一个译码的过程，将指令包含的信息转为 CPU 各部分的控制信号，由真值表通过与或门阵实现。

表 13 Controller 功能定义

序号	功能名称	功能描述
1	输出控制信息	根据输入的 6 位 op 和 6 位 func 得出各器件控制信息

表 14 Controller 端口说明

序号	端口名称	功能描述
1	Instr (I)	32 位指令的输入
2	RegDst (O)	判断指令类型: 1: R 0: I
3	ALUSrc (O)	选择输入 ALU 的立即数
4	Memtoreg (O)	选择从 DM 读取写入 GRF 的数据
5	RegWrite (O)	GRF 写使能信号
6	MemWrite (O)	DM 写使能信号
7	nPC_Sel (O)	Beq 指令的控制信号
8	ExtOp (O)	扩展立即数选择信号
9	ALUctr (O)	ALU 运算选择信号

表 15 Controller 真值表

op	000000	000000	001101	100011	101011	000100	001111
func	100001	100011	N/A				
	addu	subu	ori	lw	sw	beq	lui
RegDst	1	1	0	0	0	x	x
ALUSrc	0	0	1	1	1	0	1
Memtoreg	0	0	0	1	0	0	0
RegWrite	1	1	1	1	0	0	1
MemWrite	0	0	0	0	1	0	0
nPC_Sel	0	0	0	0	0	1	0
ExtOp	xx	xx	00	01	01	01	10
ALUctr	00	01	10	00	00	01	10

4. CPU 测试

测试代码：

```
lw $t0, 0($0)
lw $t1, 4($0)
loop:

lw $t2, 8($0)
lw $t3, 12($0)
addu $t4, $t0, $t1
subu $t5, $t2, $t3
sw $t4, 8($0)
sw $t5, 12($0)

beq $t5, $0, loop

ori $t6, $t5, 5
sw $t6, 8($0)
```

测试期望：由 DM 加载出来$t0=1,$t1=2,$t2=3,$t3=3,$t0+$t1=3 写入$t4,$t2-$t3=0 写入$t5,将$t4=3 写入地址 0x00000008,将$t5=0 写入地址 0x0000000c,之后$t5=$0=0,跳转到 loop, 地址 0x00000008 的数据加载至$t2=3, 地址 0x0000000c 的数据加载至$t3=0,$t0+$t1=3 写入$t4,$t2-$t3=3 写入$t5,$t5!=$0 不再跳转，$t5|5=7 写入$t6,再将$t6=7 写入地址 0x00000008。

最终，地址 0x00000000 数据为 1，地址 0x00000004 数据为 2，地址 0x00000008 数据为 7，地址 0x0000000c 数据为 3。

5. 思考题

1.若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。

答：若 PC 为 30 位则执行+1，若为 32 位则执行+4，本质上没有优劣之别。

2.现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，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：合理。IM 作为一个指令寄存器，可以直接导入，无需写操作。DM 作为一种内存可以用 RAM 实现，GRF 用寄存器实现速度较快。

3.结合上文给出的样例真值表，给出 RegDst， ALUSrc， MemtoReg，RegWrite, nPC_Sel, ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

答：RegDst = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

ALUSrc = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

Memtoreg = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0])

RegWrite = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

| (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

MemWrite = (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

nPC_Sel = (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ExtOp[0] = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

ExtOp[1] = (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

ALUctr[0] = (~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0]) | (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ALUctr[1] = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

4.充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式，请

给出化简后的形式。

答：RegDst = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

ALUSrc = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

Memtoreg = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0])

RegWrite = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

| (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

MemWrite = (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

nPC_Sel = (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ExtOp[0] = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

ExtOp[1] = (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

ALUctr[0] = (~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0]) | (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ALUctr[1] = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

5.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。

答：空指令为 8’h00000000，即不进行任何操作，控制信号并没有实际操作，所以不用加入真值表。

6.前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增

加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

答：对于 DM 输入地址 RA，取出实际地址减去 DM 起始地址接上 RA 即可。DM 片选信号决定输入地址是 RA 还是相减的结果。

7.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，简要阐述相比与测试，形式验证的优劣。

答：形式验证的优势：

仿真对于超大规模设计太耗费时间，当确认某个功能的仿真是正确的以后，设计实现的每个步骤的结果可以与上个步骤结果做形式比较，不必进行复杂的仿真。

形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了模拟验证的不足。

形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

形式验证的劣势：

不能有效的验证电路的性能，例如电路的延时和功耗，只能和模拟验证相互补充。