Logisim 单周期 CPU 实验报告

一、CPU 设计文档

（一）模块规格

1、IFU（取指令单元）

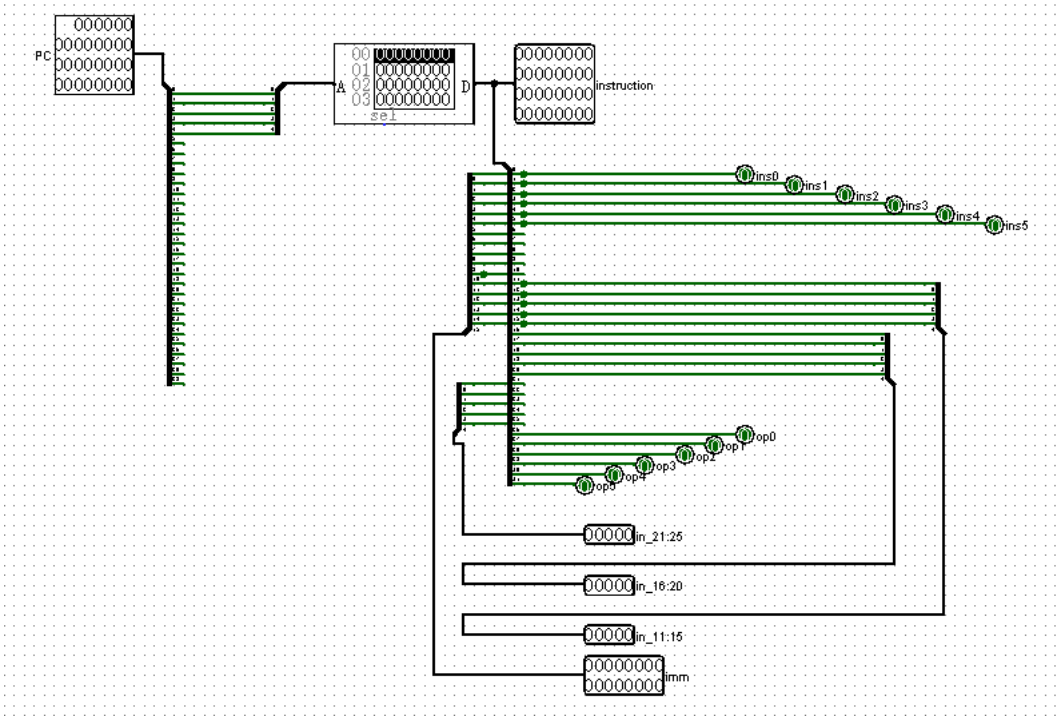

图 1

表 1

序号	端口	功能描述
1	PC	当前 PC 值，指向指令存储器，作为地址取出指令
2	instruction	32 位指令，作为输出端口直接输出
3	其他（指令的不同位）	为后面控制器，寄存器堆，扩展单元所用

2、GRF（通用寄存器组）

表 2

序号	端口	功能描述
1	clk	时钟信号，clk 为 1 且使能端为 1 时存入数据
2	ALUSrc	见控制器
3	RegDst	见控制器
4	in_21:25	Instr<25:21>号寄存器
5	in_16:20	Instr<20:16>号寄存器
6	in_11:15	Instr<15:11>号寄存器
7	WD3	回写的数据
8	RegWrite	见控制器
9	reset	重置信号，当有效时，所有寄存器清零
10	RD1	输出的第一个操作数
11	RD2	输出的第二个操作数
12	A3	回写的寄存器编号

3、ALU（算数逻辑单元）

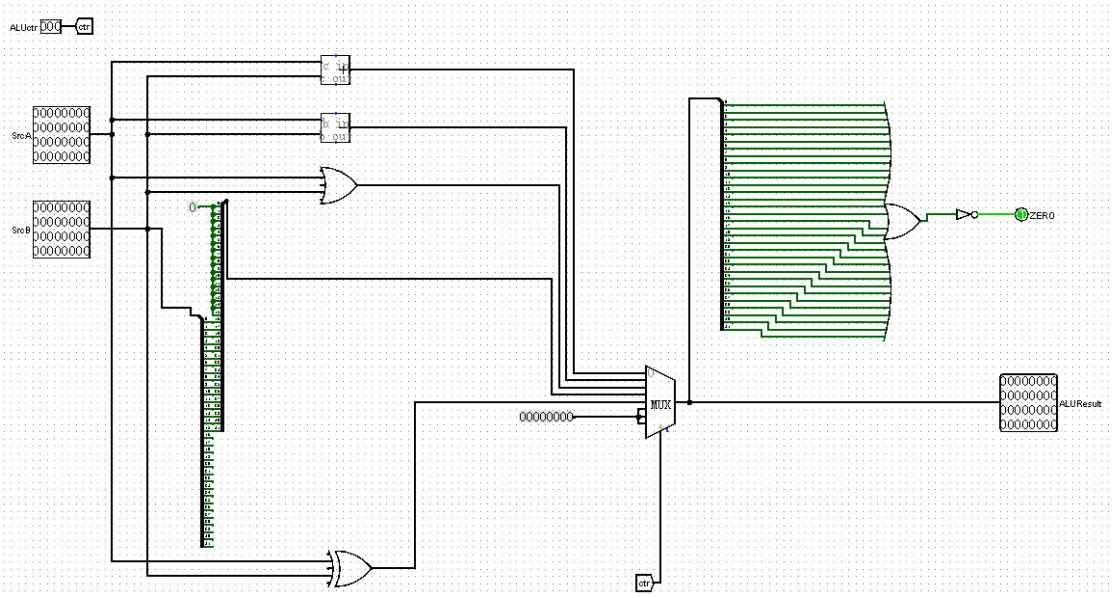

图 2

表 3

序号	端口	功能描述		
1	ALUctr	ALU 控制信号	000	加法运算
			001	减法运算
			010	或运算
			011	将 SrcB 逻辑左移 16 位
			100	异或运算
2	SrcA	第一个操作数		

3	SrcB	第二个操作数		
4	zero	PC 是否需要额外跳转，结合 nPC_sel 使用，当 nPC_sel 为 0 时有效	0	否
			1	是
5	ALUResult	ALU 计算结果		

4、DM（数据存储器）

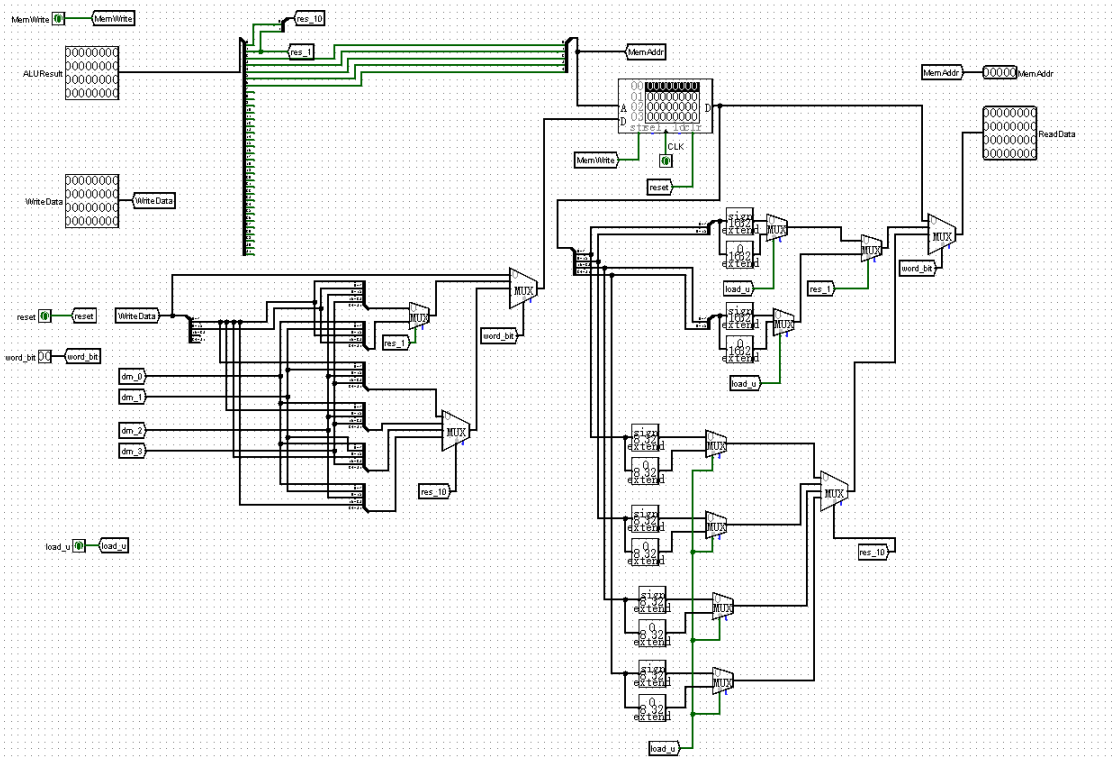

图 3

表 4

序号	端口	功能描述
1	MemWrite	见控制器
2	ALUResult	ALU 计算结果
3	WriteData	写入数据存储器的数据
4	reset	重置信号，当有效时（即为 1）存储器清零
5	CLK	时钟信号，上升沿时可以存入数据
6	word_bit<1:0>	见控制器
7	load_u	从 DM 中读取数据后是否有符号扩展
8	MemAddr	读出或写入数据的地址
9	ReadData	读出数据

5、EXT（扩展单元）

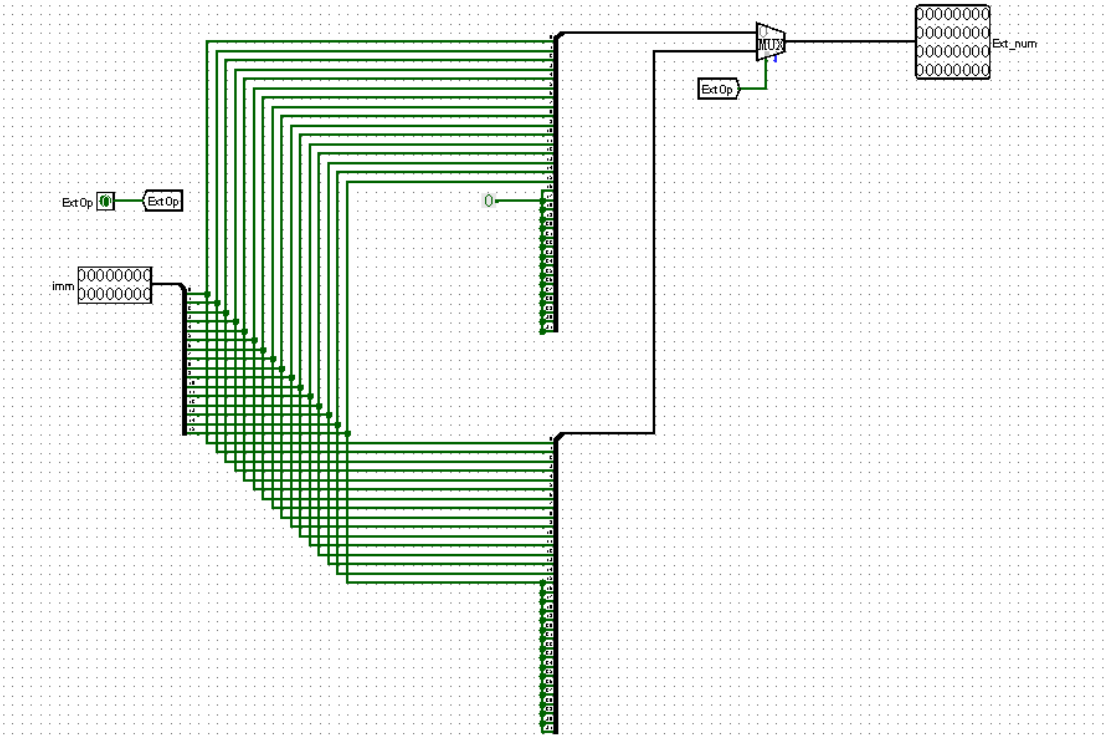

图 4

表 5

序号	端口	功能描述		
1	ExtOp	扩展信号	0	无符号扩展
			1	有符号扩展
2	imm	Instr<15:0>立即数		
3	Ext_num	扩展之后得到的数		

（二）控制器设计

表 6

func	100001	100011	n/a					
op	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst<1:0>	01	01	00	00	xx	xx	00	xx
ALUSrc	0	0	1	1	1	0	1	X
MemtoReg	00	00	00	01	xx	xx	00	xx
RegWrite	1	1	1	1	0	0	1	0
MemWrite	0	0	0	0	1	0	0	0

nPC_sel	0	0	0	0	0	1	0	0
j_zero<1:0>	xx	xx	xx	xx	xx	01	xx	xx
ExtOp<1:0>	xx	xx	00	01	01	01	00	xx
ALUctr<2:0>	000	001	010	000	000	001	011	xxx

表 7

序号	端口	功能描述		
1	输入端口（op5、op4 等）	输入指令 op 段，func 段识别指令		
2	RegDst<1:0>	确定回写寄存器——A3 的位置	00	Instr<20:16>号寄存器对应的数
			01	Instr<15:11>号寄存器对应的数
			10	回写寄存器为$31
3	ALUSrc	确定第二个操作数	0	Instr<20:16>号寄存器对应的数
			1	imm（立即数）扩展之后的数
4	MemtoReg<1:0>	确定回写什么	00	ALUResult
			01	数据存储器里的数
			10	PC+4
5	RegWrite	寄存器文件使能端	0	寄存器值不改变
			1	寄存器值会改变
6	MemWrite	数据存储器使能端	0	不会写入数据
			1	会写入数据
7	nPC_sel	是否有 PC 的改变	0	无，直接 PC+4 即可
			1	有，（具体见 ALU 模块）
8	ExtOp<1:0>	如何将立即数进行扩展	00	16 位无符号扩展为 32 位
			01	16 位有符号扩展为 32 位
			10	jal 特殊扩展为 32 位
9	ALUctr<2:0>	具体见 ALU 模块		
10	j_zero	如何决定 zero	00	Zero=1
			01	SrcA-SrcB=0 时，zero=1
10	word_bit<1:0>	存储或取出数据的位数	00	32 位（4 个字节）
			01	26 位（2 个字节）
			10	8 位（1 一个字节）
11	load_u	存入 load 指令是否有符号扩展	0	有符号
			1	无符号
12	j_src	如何确定跳转类指令 PC 值	0	PC+4+Ext_num
			1	Ext_num
			2	从寄存器里读出来的值

（三）测试程序

```
1  ori $26, $23, 2 # $26 = 2
2  ori $2, $2, 1 # $2 = 1
3  nop
4  sw $26, 8($0)   #存$26=2到地址为300的位置
5  lw $6, 8($0)    #取地址为300对应的数到$6
6  lab:
7  addu $3, $2, $3 # $3 ++
8  subu $4, $3, $2    #$4 = $3 - 1
9  lui $5, 1    # 加载立即数1至高位，存入$5
10 nop
11 beq $3, $2, lab  # 当$3 != $2 时跳转
```

图 5

将代码导入 logisim 的 IFU 中，运行结束后，测试期望为：

$2 存入 1，$3 存入 2，$4 存入 1，$5 存入 0x00010000，$6 存入 2，$26 存入 2

地址 0x00000008 存入 2

二、思考题

（一）、模块规格

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

如果是指令按照字边界对齐，那么有 32 位地址线只需要 30 位的 PC 就能标识出不同的指令，不需要 32 位。

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。

合理，IM 存的是指令，只需要读出，不需要写入，用 ROM 很合适，DM 存的是数据，不仅需要读出，还需要写入，只能用 RAM，GRF 只能存一个临时变量且需要有很快的存储速度，用寄存器很合适。

（二）、控制器设计

1、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。）

$$RegDst=\overline{o_5o_4o_3o_2o_1o_0}f_5\overline{f_4f_3f_2f_0}$$

$$ALUSrc=\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_3o_2o_1o_0}+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$$

$$=\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_2o_1o_0}$$

$$MemtoReg=o_5\overline{o_4o_3o_2o_1o_0}$$

$$RegWrite=\overline{o_5o_4o_3o_2o_1o_0}f_5\overline{f_4f_3f_2f_0}+\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_3o_2o_1o_0}$$

$$nPC\_sel=o_5o_4o_3\overline{o_2}o_1o_0$$

$$ExtOp=o_5\overline{o_4o_3o_2o_1o_0}+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$$

$$=o_5\overline{o_4o_2o_1o_0}$$

图 6

2、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。

将 X 按照方便化简的方式化成 0 或 1

$$RegDst=\overline{o_5o_4o_3o_2o_1o_0}f_5\overline{f_4f_3f_2f_0}$$

$$ALUSrc=\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_3o_2o_1o_0}+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$$

$$=\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_2o_1o_0}$$

$$MemtoReg=o_5\overline{o_4o_3o_2o_1o_0}$$

$$RegWrite=\overline{o_5o_4o_3o_2o_1o_0}f_5\overline{f_4f_3f_2f_0}+\overline{o_5o_4o_3o_2}\overline{o_1}o_0+o_5\overline{o_4o_3o_2o_1o_0}$$

$$nPC\_sel=o_5o_4o_3\overline{o_2}o_1o_0$$

$$ExtOp=o_5\overline{o_4o_3o_2o_1o_0}+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$$

$$=o_5\overline{o_4o_2o_1o_0}$$

图 7

3、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 指令不对 DM 和 GRF 造成任何改变，其使能端都为 0，当不把它加入控制信号真值表中时，那根据与逻辑，它的使能端都为 0，符合要求。

（三）、测试程序

1、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。

假设 DM 有 256MB 容量，在 0x30000000-0x3fffffff 区间，那么只需要把高四位与 0x3 进行比较，比较结果就是 DM 的片选信号，之前的 DM 存满后就从 0x30000000-0x3fffffff 开始存储数据，这次的 DM 最多存到 0x00000080，所以不需要片选信号。

2、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。

形式验证的优点：

1、由于形式验证技术是借用数学上的方法将待测验证电路和功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量。

2、形式验证是对指定描述的所有可能情况进行验证，而不仅仅对其中的一个在积极进行多次验证，因此有效克服了测试验证的不足。

3、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、尽快发现并改正电路设计中的错误，有可能缩短设计周期。

形式验证缺点： 验证到目前为止仍然不能有效的验证电路的性能，如电路的时延和功耗等。