Logisim 单周期CPU实验报告

# CPU设计文档

## （一）模块规格

### 1、IFU（取指令单元）

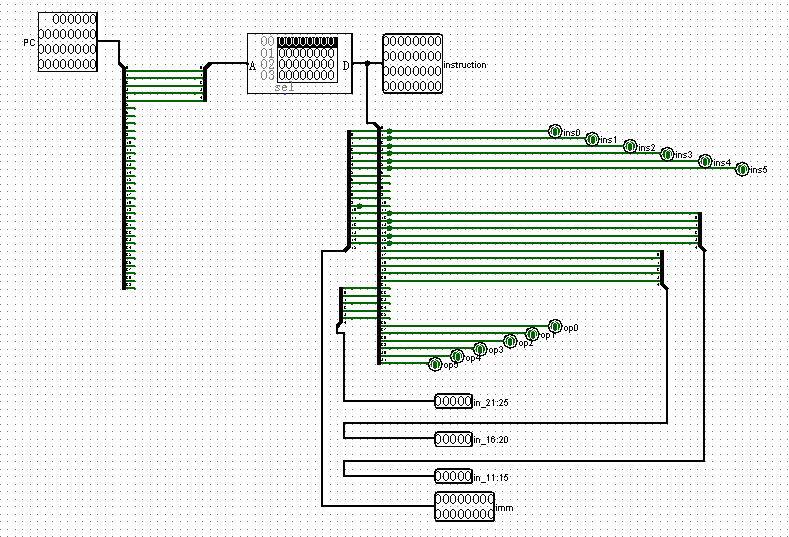


图1

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）

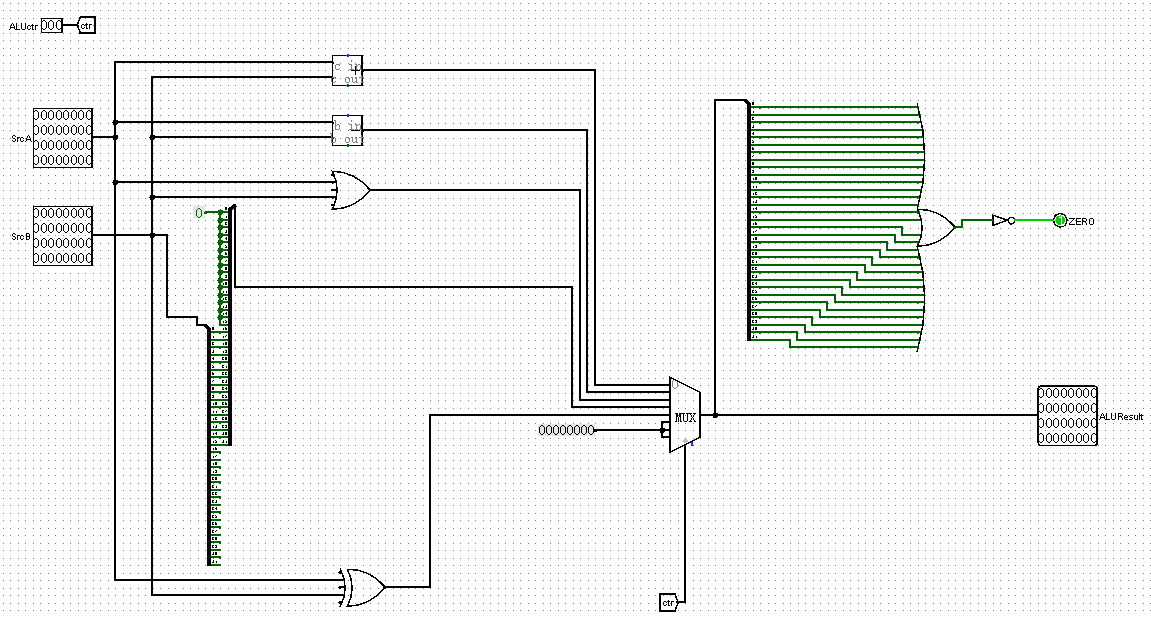


图2

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）

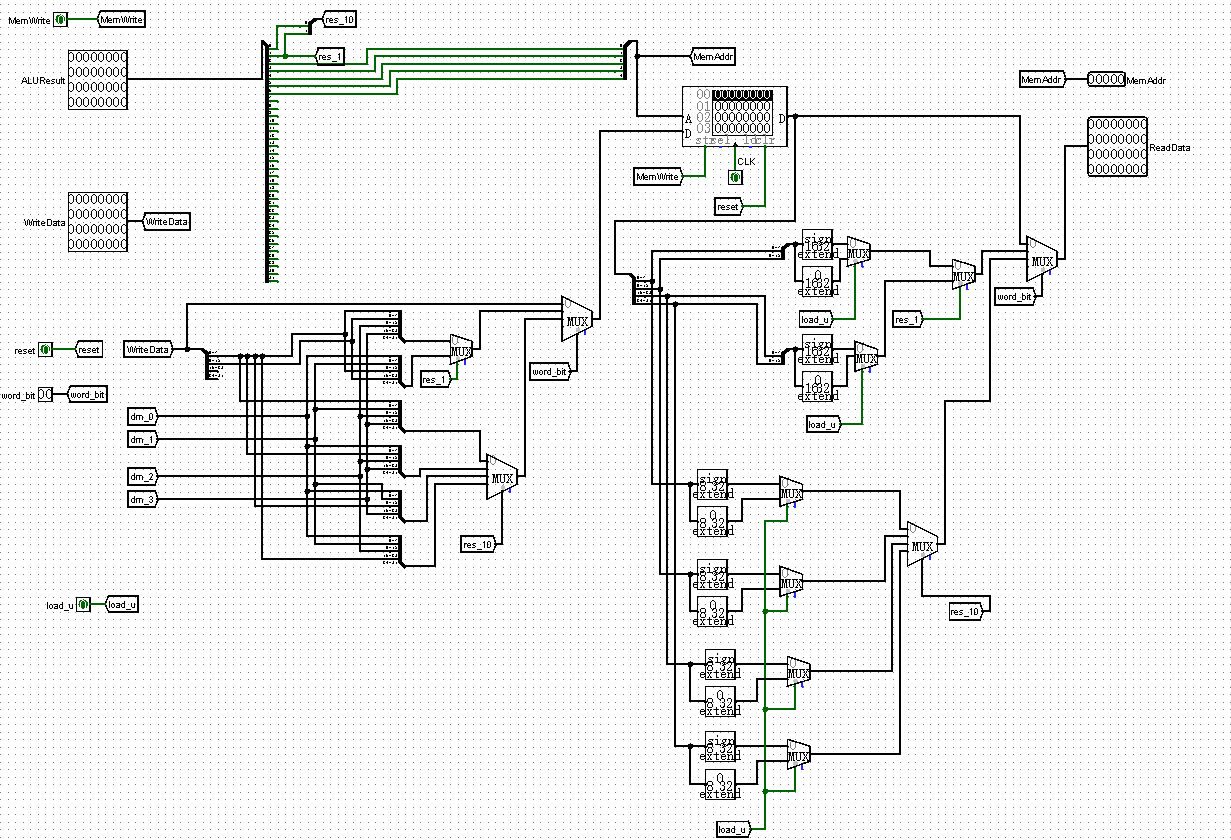


图3

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）

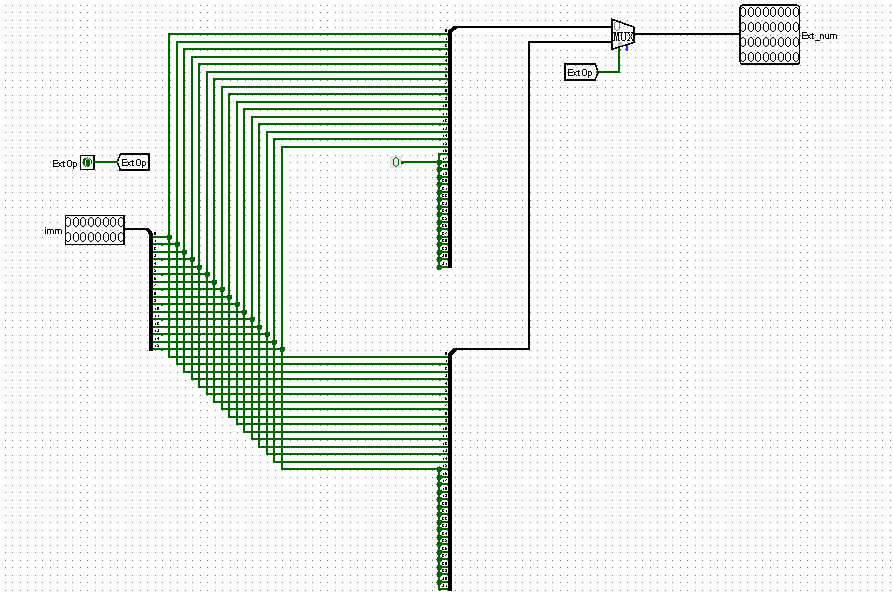


图4

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

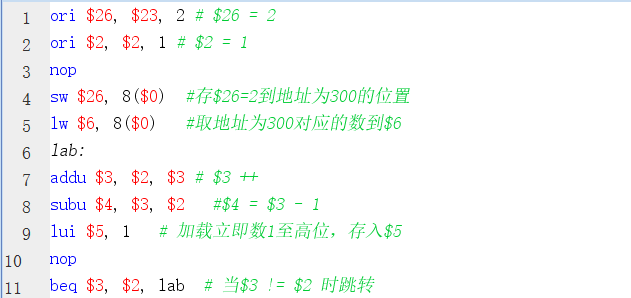


图5

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

# 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。

1. 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。）

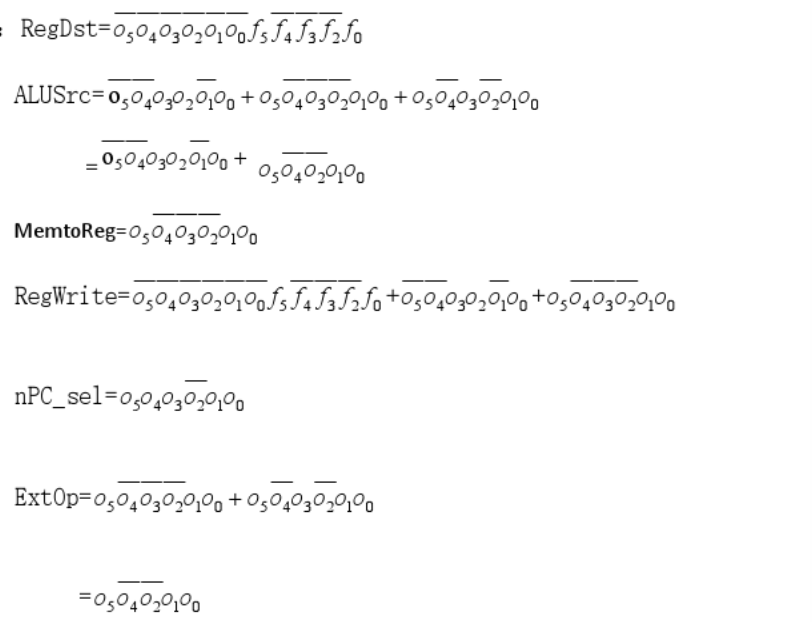


图6

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



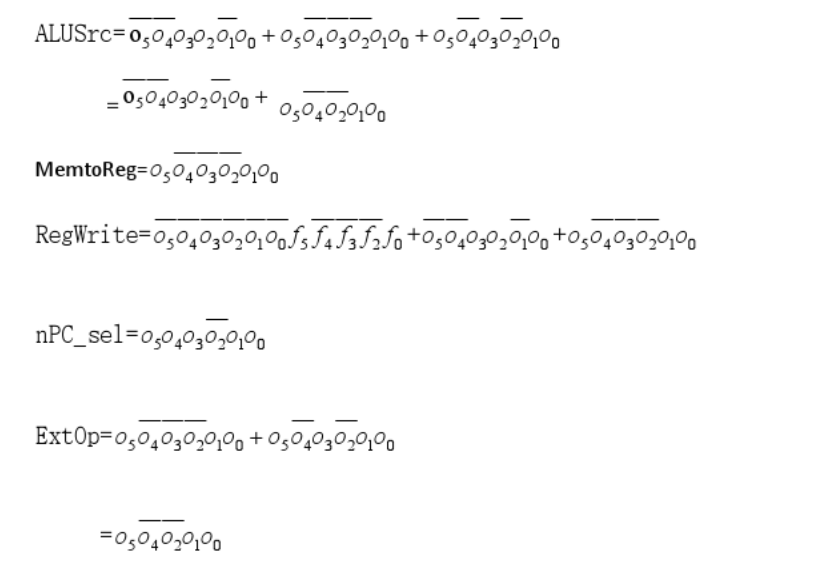


图7

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。