计算机组成原理实验报告

一、**模块规格**

1. IFU（取指令单元）

PC 用寄存器实现，IM用 ROM 实现，容量为 32bit \* 32。

**IFU端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | PC | Input | [4:0] | 输入需要取指令的地址 |
| 2 | Instr | Output | [31:0] | 根据PC提供的指令地址，输出指令 |

1. GRF（寄存器堆）

用32具有写使能的寄存器实现

**GRF端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | A1 | Input | [4:0] | 读端口，输入地址，用于指定32个寄存器中的一个作为读入寄存器 |
| 2 | A2 | Input | [4:0] | 读端口，输入地址，用于指定32个寄存器中的一个作为读入寄存器 |
| 3 | A3 | Input | [4:0] | 写端口，输入地址，用于指定32个寄存器中的一个作为写入寄存器 |
| 4 | WD3 | Input | [31:0] | 写端口，输入数据，将数据写入到寄存器 |
| 5 | CLK | Input | 1 | 时钟输入信号 |
| 6 | WE3 | Input | 1 | 写入使能信号，若为1则写入 |
| 7 | RD1 | Output | [31:0] | 读入寄存器的数据输出 |
| 8 | RD2 | Output | [31:0] | 读入寄存器的数据输出 |
| 9 | Reset | Input | 1 | 重置使能，若为1，清空所有寄存器 |

1. ALU（算术逻辑单元）

**ALU端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | SrcA | Input | [31:0] | 写端口，将操作数传入到ALU |
| 2 | SrcB | Input | [31:0] | 写端口，将操作数传入到ALU |
| 3 | ALUOp | Input | [1:0] | ALU控制信号，通过信号判断执行指令 |
| 4 | Funct | Input | [5:0] | 写入funct指令段 |
| 5 | Op | Input | [6:0] | 写入Op指令段 |
| 6 | ALUResult | Output | [31:0] | 输出ALU操作结果 |
| 7 | Zero | Output | 1 | PC跳转使能信号 |

1. DM（数据存储器）

用RAM实现，容量为 32bit \* 32

**DM端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | A | Input | [4:0] | 写端口，输入地址，用于指定DM的数据地址 |
| 2 | WD | Input | [31:0] | 写端口，写入数据到DM |
| 3 | WE | Input | 1 | DM写入使能，若为1，则写入 |
| 4 | Clk | Input | 1 | 时钟输入信号 |
| 5 | Regwrite | Input | 1 | DM读出使能，若为1，则读出 |
| 6 | Reset | Input | 1 | 重置使能，若为1，清空DM |
| 7 | RD | Output | [31:0] | DM读出数据 |

1. EXT

用logisim自带的bitextender实现

**EXT端口功能表**

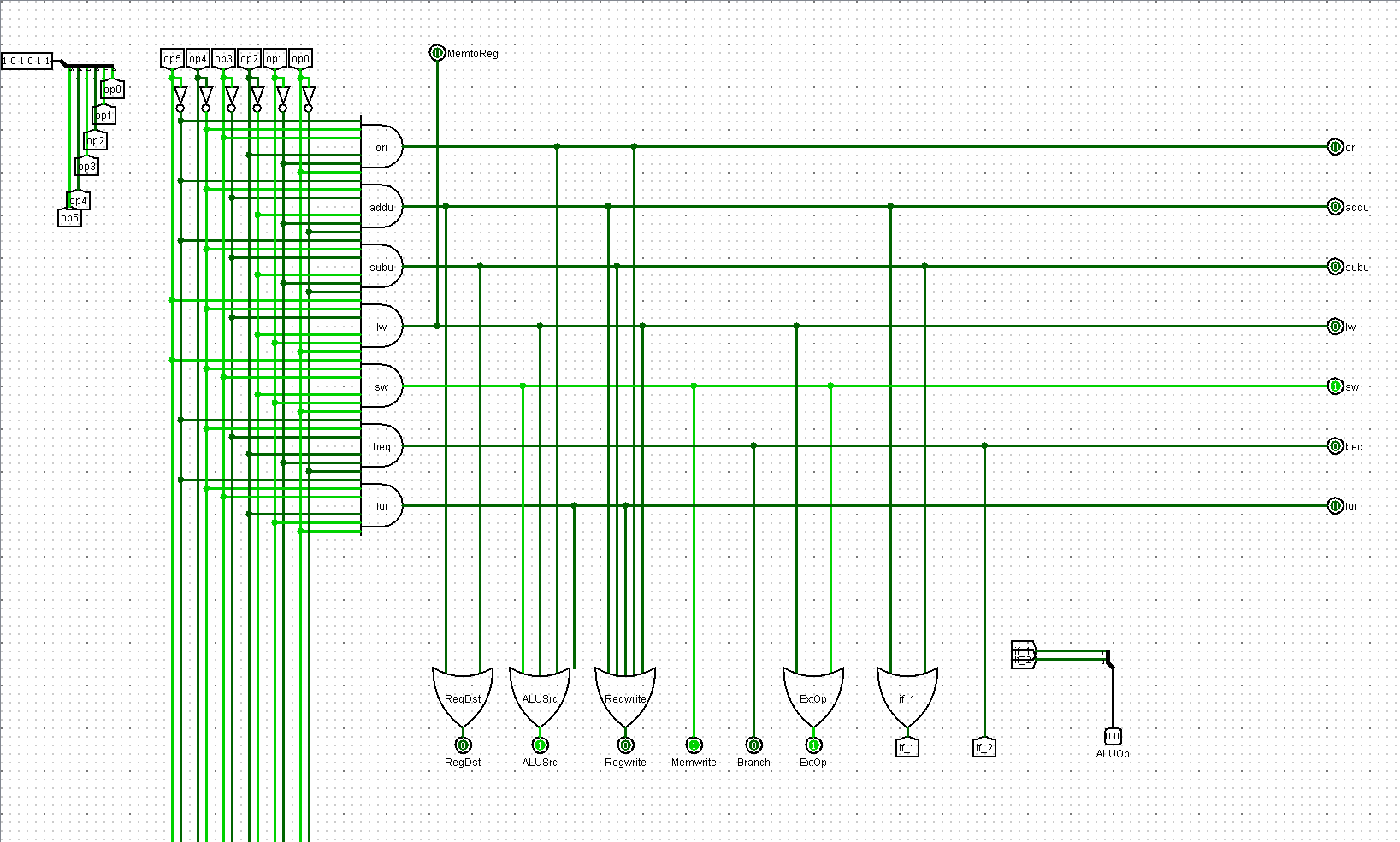
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | immediate | Input | [16:0] | 输入要扩展的数据 |
| 2 | ExtOp | Input | 1 | 控制信号输入，选择无符号扩展或符号扩展 |
| 3 | EXTimme | Output | [31:0] | 输出已扩展的数据 |

二、**控制器设计**

使用与或门阵列控制信号

**控制信号端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | Op | Input | [6:0] | 写入Op指令段 |
| 2 | MemtoReg | Input | 1 | 内存写入寄存器使能信号 |
| 3 | RegDst | Output | 1 | 选择目的寄存器由rt字段或rd字段控制 |
| 4 | ALUsrc | Output | 1 | 选择SrcB来自RD2或signimm |
| 5 | Regwrite | Output | 1 | 输出寄存器写入使能信号 |
| 6 | Memwrite | Output | 1 | 输出内存写入使能信号 |
| 7 | Branch | Output | 1 | 输出是否跳转信号 |
| 8 | ALUOp | Output | 1 | 输出ALU选择信号 |
| 9 | ExtOp | Output | 1 | 选择扩展immediate |



**控制信号实现图**

**控制信号真值表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | n/a | | | | | 000000 |
| op | 000000 | 000000 | 001101 | 001111 | 100011 | 101011 | 000100 | 000000 |
| 名称 | addu | subu | ori | lui | lw | sw | beq | nop |
| RegDst | 1 | 1 | 0 | 0 | 0 | X | X | X |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 1 | 0 | X |
| MemtoReg | 0 | 0 | 0 | 0 | 1 | X | X | X |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 0 | 0 | X |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 | 0 | X |
| PCSrc | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X |
| ExtOp | X | X | 0 | X | 1 | 1 | X | X |
| ALUctr  <2.0> | 00 | 01 | 10 | 00 | 00 | 00 | 01 | XX |

三、测试程序

 v2.0 raw

3c090001

3c0a0002

012b5021

012c5023

352d0064

adcd0000

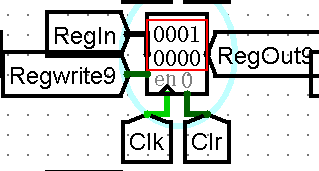
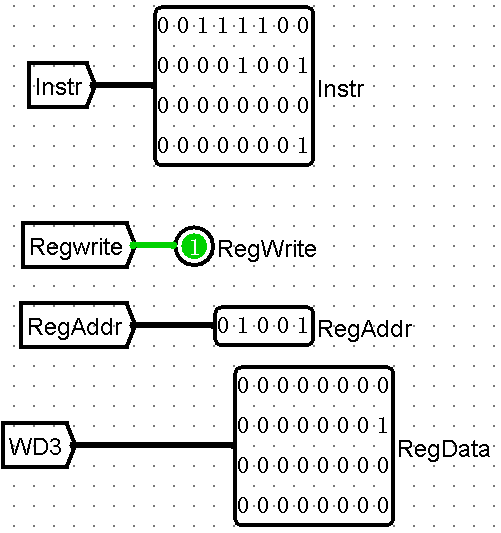
8dcf0000

1129fff8

执行代码：

1. lui $t1,1

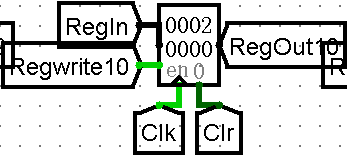
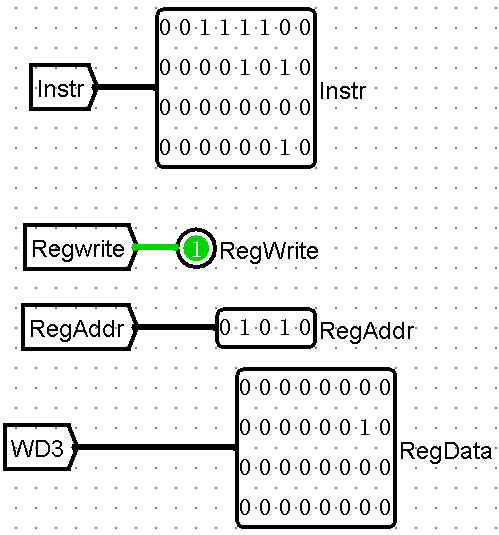
执行完后$t1寄存器结果为立即数1加载至高位即



**执行结果图**

1. lui $t2,2

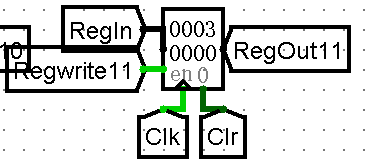
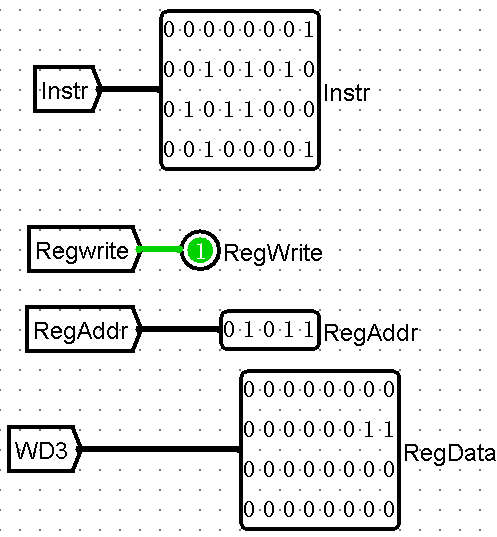
执行完后$t2寄存器结果为立即数2加载至高位即



**执行结果图**

1. addu $t3,$t1,$t2

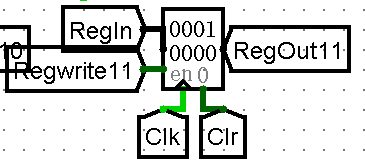
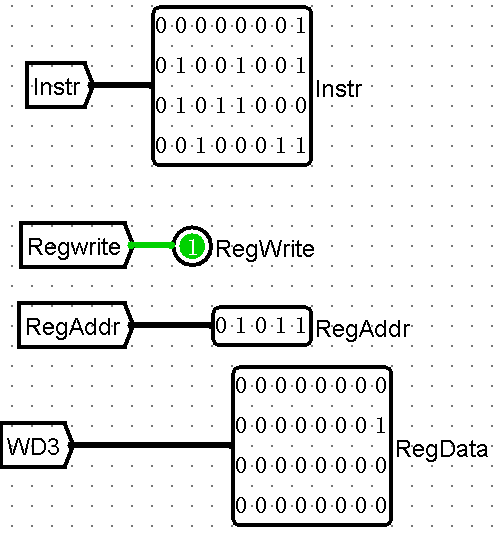
执行完后$t3寄存器结果为$t1与$t2的和



**执行结果图**

1. subu $t3,$t2,$t1

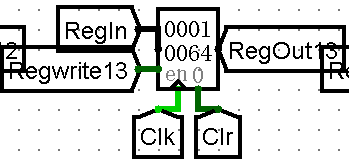
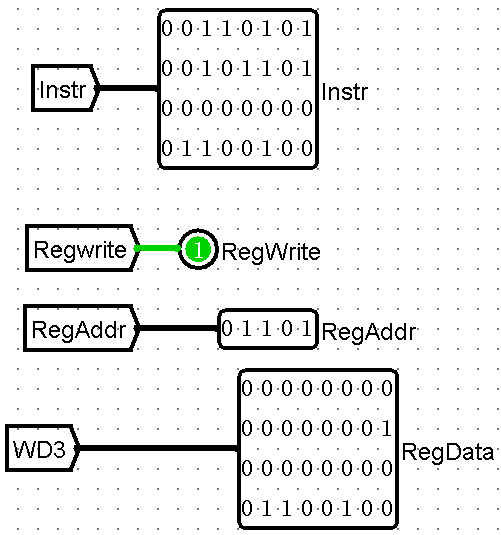
执行完后$t3寄存器结果为$t2与$t1的差



**执行结果图**

1. ori $t5,$t1,100

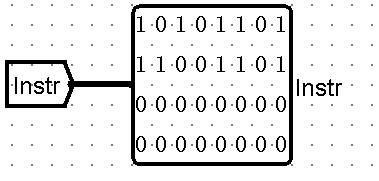
执行完后$t5寄存器结果为$t1与100或做或运算的结果

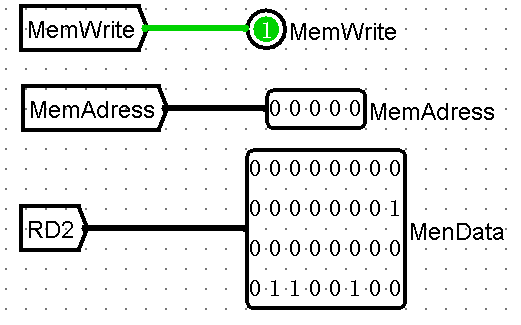
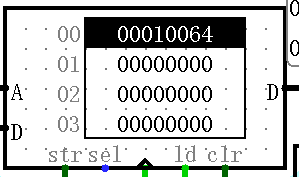


**执行结果图**

1. sw $t5,0($t6)

执行完后DM中地址为$t6和0的和存储数据为$t5的值

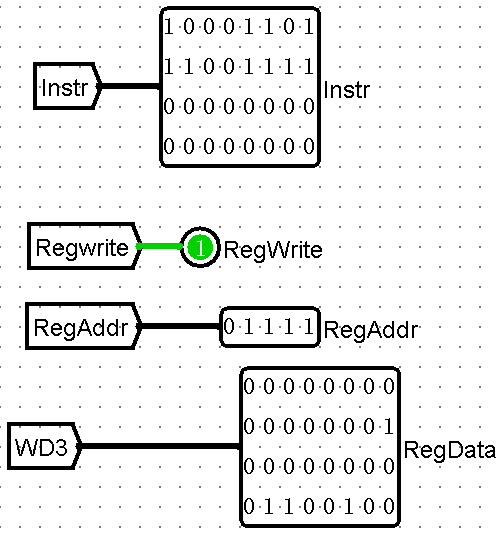
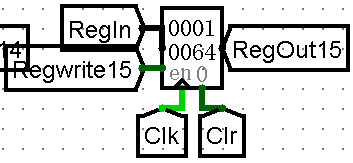


**执行结果图**

1. lw $t7,0($t6)

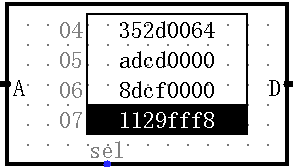
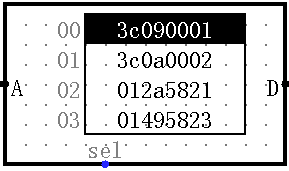
执行完后$t7寄存器的值为地址为$t6和0的和存储数据的值

**执行结果图**

1. beq $t1,$t1,s0

执行完后，代码跳转到s0继续执行



—>

**执行结果图**

四、思考题

T2：

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：30位PC相比32位PC可读的指令变少了且30位PC不需要扩展

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，用ROM（只读存储器）作为IM（存储指令器）来存储指令可以保证指令不会被改变。用RAM（内存）作为DM（数据存储器）容量够大，且多数功能可以直接用信号控制。GRF（寄存器堆）需要很快的读取速度，用寄存器可以做到。

T3：

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。）

答：

RegDst=‾o5‾o4‾o3‾o2‾o1‾o0 f5‾f4‾f3‾f2 f0

ALUSrc=‾o5‾o4 o3 o2‾o1 o0 + o5‾o4 o3‾o2 o1 o0 + o5‾o4 o3‾o2‾o1 o0

= ‾o5‾o4 o3 o2‾o1 o0 + o5‾o4‾o2‾o1 o0

Regwrite=‾o5‾o4‾o3‾o2‾o1‾o0 f5‾f4‾f3‾f2 f0 + ‾o5‾o4 o3 o2‾o1 o0 + o5‾o4‾o3‾o2 o1 o0

Memwrite=o5‾o4‾o3‾o2 o1 o0

nPc\_Sel=o5 o4 o3‾o2 o1 o0

ExtOp=o5‾o4‾o3‾o2 o1 o0 + o5‾o4 o3‾o2 o1 o0

=o5‾o4‾o2 o1 o0

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

答：

将X按照方便化简的原则当成0或1

RegDst=‾o5‾o4‾o3‾o2‾o1‾o0 f5‾f4‾f3‾f2 f0

ALUSrc=‾o5‾o4 o3 o2‾o1 o0 + o5‾o4‾o2‾o1 o0

Regwrite=‾o5‾o4‾o3‾o2‾o1‾o0 f5‾f4‾f3‾f2 f0 + ‾o5‾o4 o3 o2‾o1 o0 + o5‾o4‾o3‾o2 o1 o0

Memwrite=o5‾o4‾o3‾o2 o1 o0

nPc\_Sel=o5 o4 o3‾o2 o1 o0

ExtOp=o5‾o4‾o2 o1 o0

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。

答：nop指令并没有对电路中的原件做任何操作，所以对电路没有影响，不需要加入控制信号真值表

T4：

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。

答：片选信号就是对指令发出的数据存储器地址的高位分析，假设DM有256MB容量，并且映射在0x3000\_0000～0x3FFF\_FFFF区间。那么只需要把高4位地址与0x3进行比较，比较结果就是DM的片选信号。

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。

答：所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。

形式验证与测试相比的优点：

(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。

形式验证与测试相比的缺点：

形式验证到目前为止仍然不能有效地验证电路性能，如电路的延时和功能。