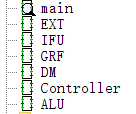
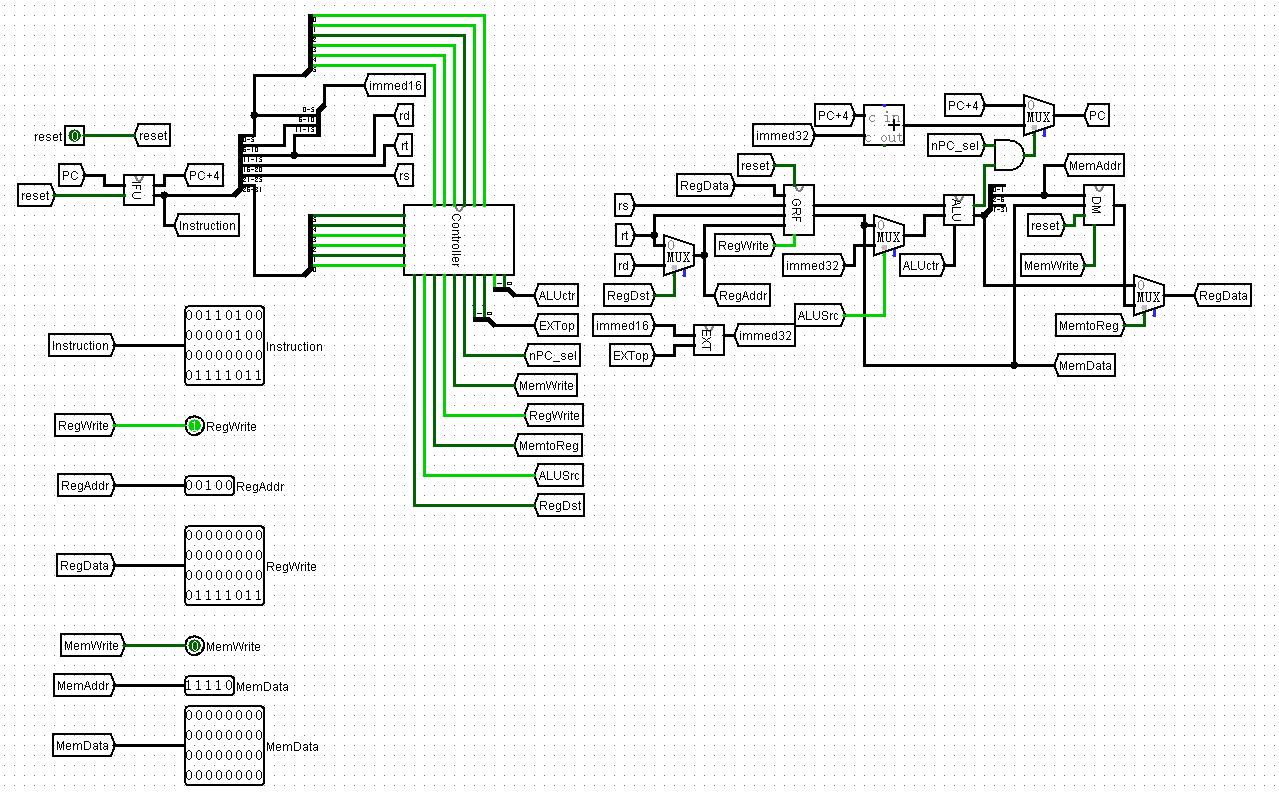
计算机组成原理实验报告

1. CPU设计文档
2. 总体设计

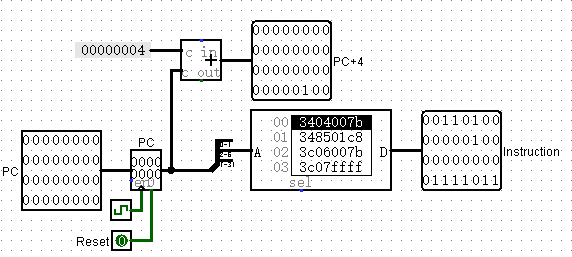


图表 1模块设计



图表 2总体设计

1. 模块规格
2. IFU（取指令单元）



图表 3IFU

1. 端口说明

表格 1 IFU端口说明

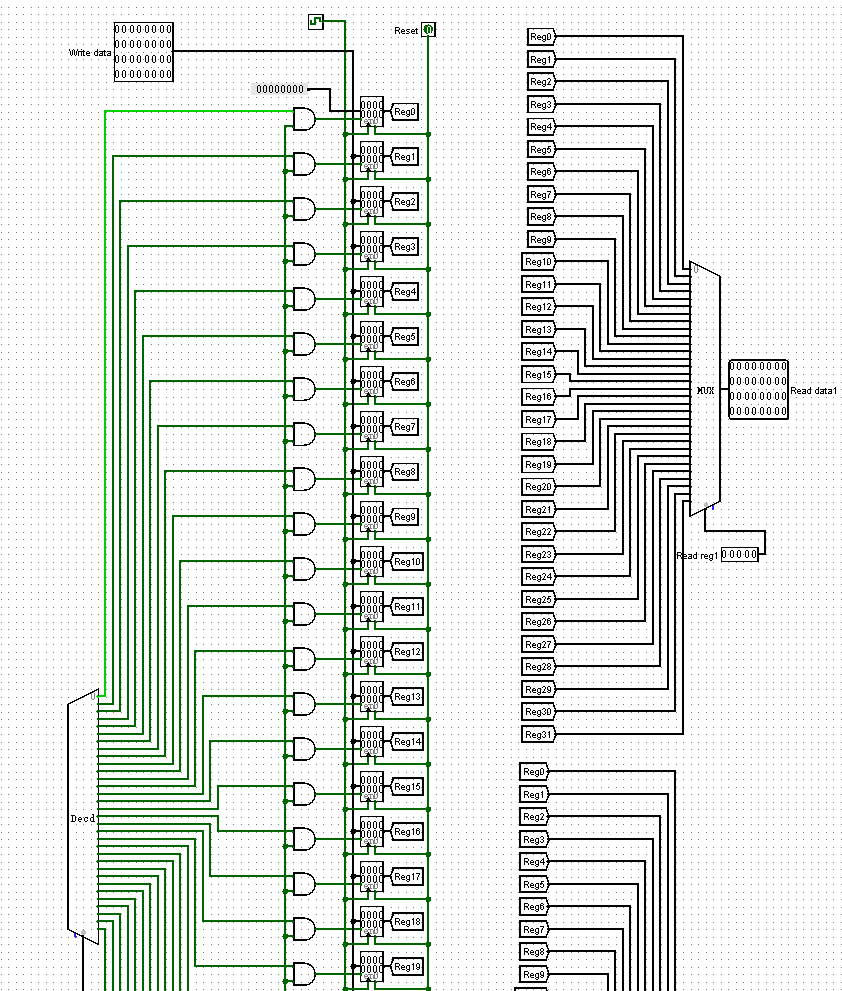
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC[31:0] | I | 当前指令地址 |
| Reset | I | 复位信号  1：复位  0：无效 |
| PC+4[31:0] | O | 下一条指令地址 |
| Instruction[31:0] | O | 32位MIPS指令 |

1. 功能定义

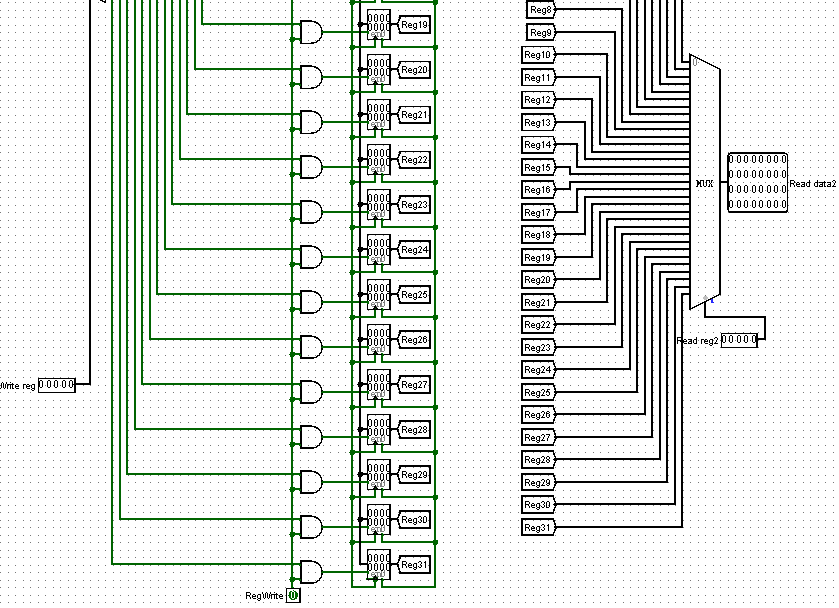
表格 2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，PC被置为0x00000000 |
| 2 | 取指令 | 根据PC从IM中取出指令 |
| 3 | 计算下一条指令地址 | PCPC+4 |

1. GRF（通用寄存器组）



图表 4GRF



图表 5GRF

1. 端口说明

表格 3GRF端口说明

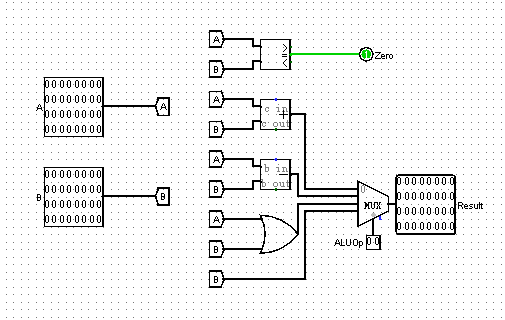
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Write data[31:0] | I | 写入的32位数据 |
| Reset | I | 复位信号  1：复位  0：无效 |
| Read reg1[4:0] | I | 读寄存器1编号 |
| Read reg2[4:0] | I | 读寄存器2编号 |
| Write reg[4:0] | I | 写寄存器编号 |
| RegWrite | I | 写控制信号  1：写入  0：无效 |
| Read data1[31:0] | O | 32位寄存器1输出 |
| Read data2[31:0] | O | 32位寄存器2输出 |

1. 功能定义

表格 4GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，32个寄存器被置为0x00000000 |
| 2 | 写寄存器 | 写寄存器控制信号有效时，把32位数据写入寄存器 |
| 3 | 读寄存器 | 根据输入的地址读出两个寄存器中的值 |

1. ALU（算术逻辑单元）



图表 6ALU

1. 端口说明

表格 5ALU端口说明

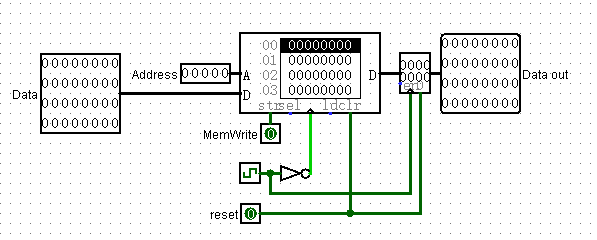
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 32位写入数据1 |
| B[31:0] | I | 32位写入数据2 |
| ALUOp[1:0] | I | 控制信号  00：加运算  01：减运算  10：或运算  11：输出写入数据2 |
| Zero | O | 相等信号  1：相等  0：不相等 |
| Result[31:0] | O | 32位输出数据 |

1. 功能定义

表格 6ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | A+B |
| 2 | 减运算 | A-B |
| 3 | 或运算 | A|B |
| 4 | 输出写入数据2 | B |
| 5 | 比较运算 | A==B |

1. DM（数据存储器）



图表 7DM

1. 端口说明

表格 7DM端口说明

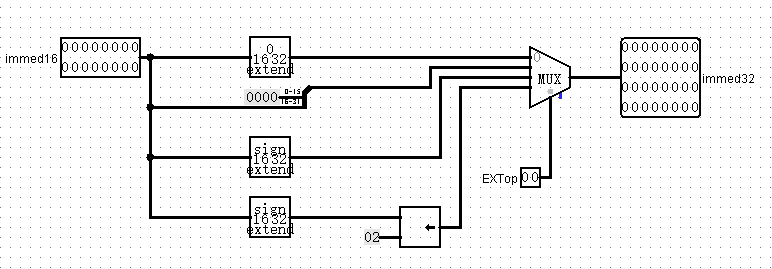
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Address[4:0] | I | 5位写入内存地址 |
| Data[31:0] | I | 32位写入数据 |
| reset | I | 复位信号  1：复位  0：无效 |
| MemWrite | I | 写内存控制信号  1：写入  0：无效 |
| Dataout[31:0] | O | 32位输出数据 |

1. 功能定义

表格 8DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，内存和读出内存的寄存器被置为0x00000000 |
| 2 | 写内存 | 写内存控制信号有效时，根据输入的地址写入32位数据 |
| 3 | 读内存 | 根据输入的地址读出内存数据 |

1. EXT（位扩展器）



图表 8EXT

1. 端口说明

表格 9EXT端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Immed16[15:0] | I | 16位写入立即数 |
| EXTop[1:0] | I | 扩展控制信号  00：无符号扩展  01：低16位补0  10：有符号扩展  11：有符号扩展后逻辑左移两位 |
| Immed32[31:0] | O | 32位输出立即数 |

1. 功能定义

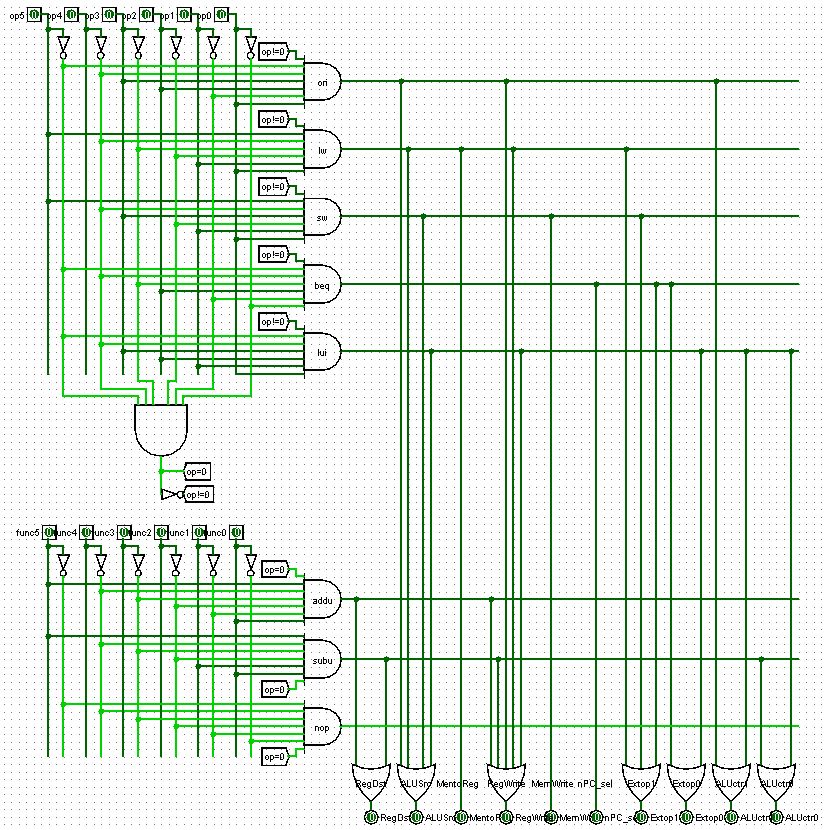
表格 10EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | 高16位补0 |
| 2 | 低16位补0 | 低16位补0 |
| 3 | 有符号扩展 | Immed[15]为1时高16位补1，为0时高16位补0 |
| 4 | 有符号扩展后逻辑左移两位 | Immed[15]为1时高16位补1，为0时高16位补0，再左移两位，溢出舍去，低2位补0 |

1. Controller（控制器）

见第二章控制器设计。

1. 控制器设计



图表 9Controller

1. 端口说明

表格 11Controller端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op[5:0] | I | 6位指令码op |
| func[5:0] | I | 6位功能码func |
| RegDst | O | GRF写寄存器决定信号  1：rd  0：rt |
| ALUSrc | O | ALU输入数据B决定信号  1：32位立即数  0：GRF寄存器2输出值 |
| MemtoReg | O | GRF写入数据决定信号  1：DM输出数据  0：ALU输出数据 |
| RegWrite | O | GEF写寄存器信号  1：写寄存器  0：无效 |
| MemWrite | O | 写内存DM信号  1：写入内存  0：无效 |
| nPC\_sel | O | 跳转信号  Zero为1时：  1：跳转  0：无效  Zero为0时：  无效 |
| EXTop[1:0] | O | 扩展控制信号  00：无符号扩展  01：低16位补0  10：有符号扩展  11：有符号扩展后逻辑左移两位 |
| ALUctr[1:0] | O | ALU控制信号  00：加运算  01：减运算  10：或运算  11：输出写入数据2 |

1. 真值表

表格 12Controller真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | N/A | | | | | |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 |
|  | addu | subu | ori | lw | sw | beq | lui | nop |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 | 0 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| MemtoReg | 0 | 0 | 0 | 1 | x | x | x | 0 |
| 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 |
| EXTop | x | x | 00 | 10 | 10 | 11 | 01 | 00 |
| ALUctr | 00 | 01 | 10 | 00 | 00 | x | 11 | 00 |

1. 测试程序
2. 测试程序

.data

.text

#测试ori指令

#第三个立即数是无符号扩展，不存在负数的情况

ori $a0, $0, 123 #测试与0进行or运算

ori $a1, $a0, 456 #测试两个非0数的or运算

#测试lui指令

lui $a2, 123 #测试，构造正数

lui $a3, 0xffff #测试，构造负数

#测试addu指令

#无符号相加，不存在负数情况

addu $s0, $a0, $a2 #测试正数相加

#测试subu指令

#无符号相减，不存在负数情况

subu $s1, $a2, $a0 #测试正数相减

#测试sw指令

ori $t0, $t0, 0x0000 #构造0

sw $a0, 0($t0) #测试0111\_1011存入内存

sw $a1, 4($t0) #测试1\_1111\_1011存入内存

#测试lw指令

lw $s0, 0($t0) #测试0111\_1011写入寄存器

lw $s1, 4($t0) #测试1\_1111\_1011入寄存器

#测试beq指令

ori $a0, $0, 1 #构造1

ori $a1, $0, 2 #构造2

ori $a2, $0, 1 #构造1

beq $a0, $a1, loop1 #测试不跳转

nop

beq $a0, $a2, loop2 #测试跳转

nop

loop1:ori $a0, $0, 0

loop2:ori $a1, $0, 0

1. 机器码

3404007b

348501c8

3c06007b

3c07ffff

00868021

00c48823

35080000

ad040000

ad050004

8d100000

8d110004

34040001

34050002

34060001

10850003

00000000

10860002

00000000

34040000

34050000

1. 期望输出

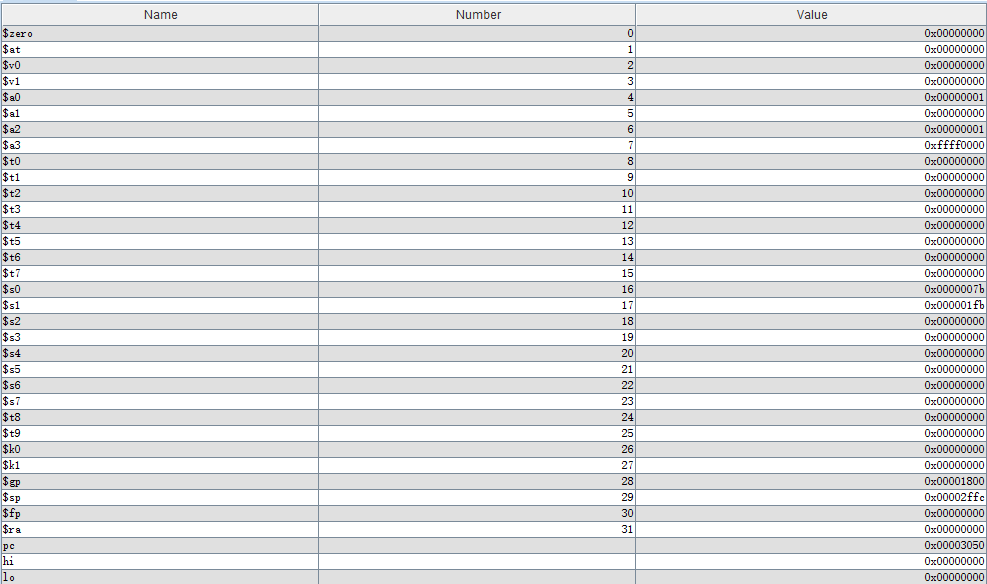
$a0 0x00000001

$a2 0x00000001

$a3 0xffff0000

$s0 0x0000007b

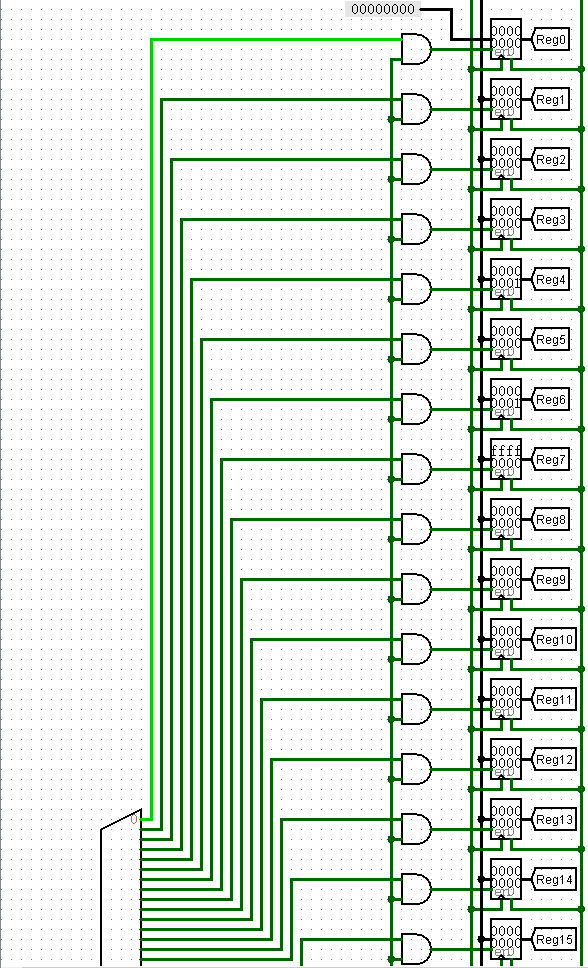
$s1 0x000001fb



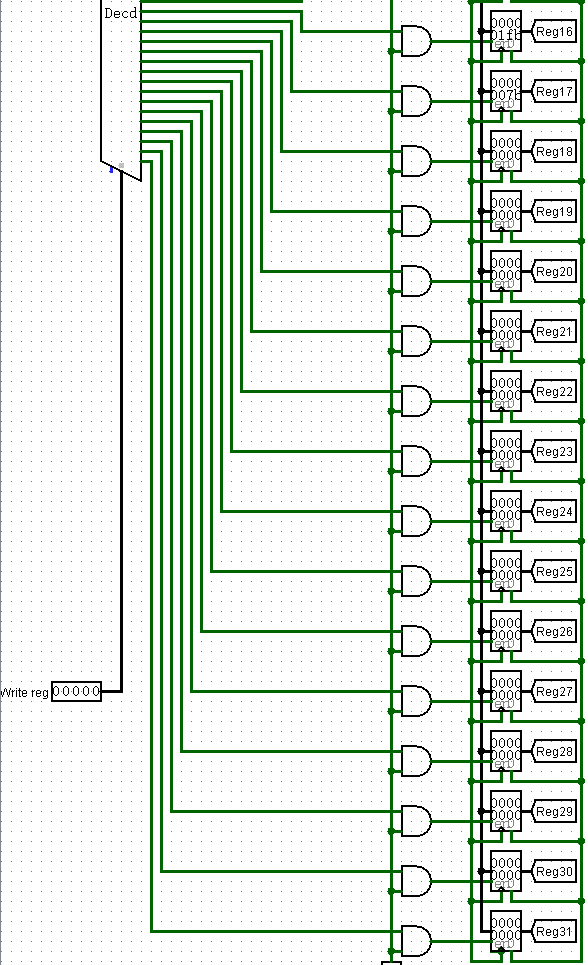
图表 10MARS输出

1. Logisim输出

GRF：

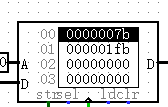


图表 11logisim输出1



图表 12logisim输出2

DM：



图表 13logisim输出3

1. 结论

期望输出与实际输出相同。

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

30位PC能寻址256M个字的位置，32位则能寻1GB字的位置，因此32位寻址范围更广。然而30位PC使用PC+1，取[4：0]位，32位PC使用PC+4，取[6:2]位，实际上是相同的。但因beq等指令中，跳转时需要相加的立即数为32位，采用30位PC非常不便，因此采用32位PC更方便。

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存储机器码，不应该改变，只需读出即可，因此使用ROM；DM为内存，其数据包含写入和读取，应使用RAM；GRF需要快速读写，以匹配CPU速度，因此用寄存器。

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。）
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。
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指令机器码为0x00000000，不进行任何有效行为，因此不会影响控制信号真值表。

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。

data base address[31:0]=0x00003000, 因此，data base address[12]=data base address[13]=1，其余位都为零，将第12或13位作为DM中RAM的str和ld信号之一控制ld，MemWrite 为1时控制str，只有当第12或13 位为1时才有可能对DM 进行读写。

1. 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：不用考虑输入具体数据，克服了不能穷举所有情况的问题，耗时较短，减小设计周期。

缺点：过于抽象，对抽象思维的要求高。