**计算机组成原理**

**大作业报告**

**学 号\_\_\_\_\_\_\_\_\_21071003\_\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_\_\_\_高立扬\_\_\_\_\_\_\_\_\_\_\_**

**指导教师\_\_\_\_\_\_\_\_\_\_\_高明霞\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_\_\_\_\_2023/05/05\_\_\_\_\_\_**

**成绩评价表**

|  |  |  |
| --- | --- | --- |
| **报告内容** | **报告结构** | **报告最终成绩** |
| **□丰富正确**  **□基本正确**  **□有一些问题**  **□问题很大** | **□完全符合要求**  **□基本符合要求**  **□有比较多的缺陷**  **□完全不符合要求** |  |
| **报告与Project功能一致性** | **报告图表** | **总体评价** |
| **□完全一致**  **□基本一致**  **□基本不一致** | **□符合规范**  **□基本符合规范**  **□有一些错误**  **□完全不正确** |  |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

目录

[计算机组成原理 1](#_Toc991)

[一． 总体数据通路结构设计图 3](#_Toc2980)

[1.基本描述 3](#_Toc20522)

[2.数据展示 3](#_Toc12811)

[二．模块详细描述 3](#_Toc30012)

[1.IFU模块定义 3](#_Toc19358)

[2.ALU模块定义 5](#_Toc6474)

[3.EXT模块定义 9](#_Toc27582)

[4.GPR模块定义 10](#_Toc30763)

[5.IM模块定义 12](#_Toc25849)

[6.DM模块定义 13](#_Toc20476)

[7.Controller模块定义 14](#_Toc923)

[三．验证 16](#_Toc12281)

[1. 给出设计的所有机器指令描述（列出指令操作码助记符、操作码的机器指令代码及指令功能的对应关系表）。 16](#_Toc9634)

[1.1. addu指令描述 16](#_Toc8970)

[1.2. subu指令描述 16](#_Toc25494)

[1.3. ori指令描述 17](#_Toc15321)

[1.4. j指令描述 17](#_Toc22733)

[1.5. lw指令描述 17](#_Toc30311)

[1.6. sw指令描述 18](#_Toc20260)

[1.7. beq指令描述 18](#_Toc17276)

[1.8. lui指令描述 18](#_Toc28986)

[2. 给出编写的测试程序，对每条机器指令添加注释。 18](#_Toc29797)

[2.1 homework-beq1.asm 19](#_Toc32673)

[2.2 homework2-beq2.ams 19](#_Toc12769)

[3. 给出与测试程序相对应的测试结果截图，并附以文字说明。 20](#_Toc10422)

[3.1 homework-beq1.asm代码测试 20](#_Toc16168)

[3.2 homework-beq2.asm代码测试 21](#_Toc24559)

[四．总结 22](#_Toc4719)

# 总体数据通路结构设计图

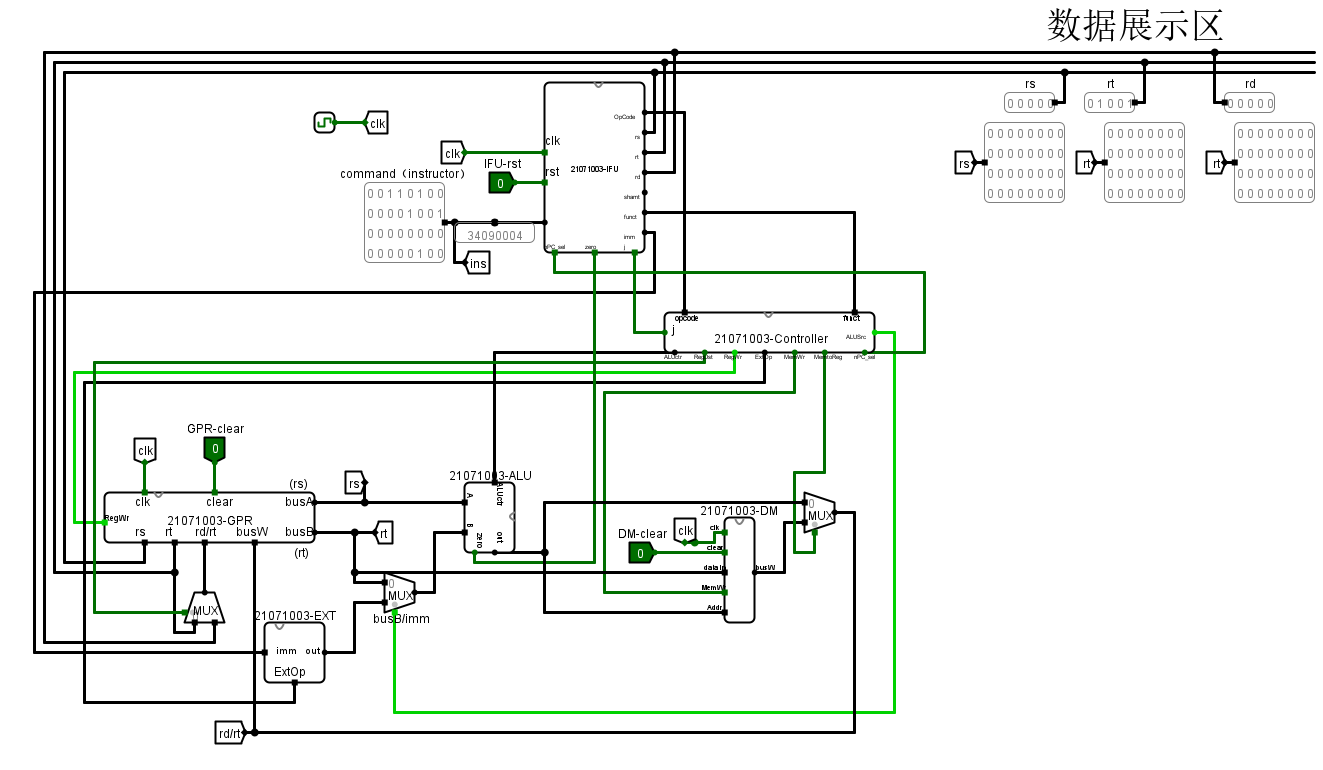


图1.总体数据通路结构设计图

1.基本描述

总体数据通路由IFU（含PC/DM）、Controller、GPR、EXT、ALU、IM以及一些logisim内部基本组件组成。本项目是单周期数据通路，IM内的指令用小端序存储，所有16进制指令均用MARS生成（已编译，无错误）。ALU中实现组内组间全并行的32位并行加法器。

2.数据展示

如图1最右侧，和IFU（中上）的左下角，会对32位指令，rs、rt、rd的地址和地址在GPR中存储的数据进行展示和打印，便于观察每一条命令执行的时候，数据是否正确传输或存储；也可以用logisim自带的基础库，点击每一根数据线，即可实时观察数据；从总结构点击进入DM或GPR，也可以试试观察其数据的存储，本项目的DM与GPR通过集成设计，一目了然，观察数据简单快捷，详细设计请看下文。

# 二．模块详细描述

1.IFU模块定义

1. 基本描述

IFU主要功能是完成取指令功能。IFU内部包括PC、IM(指令存储器)以及其他相关逻辑。IFU除了能执行顺序取指令外，还能①根据BEQ指令的执行情况决定顺序取指令还是转移取指令②根据J指令的imm，决定跳转到哪条指令。下面是本模块的设计图：

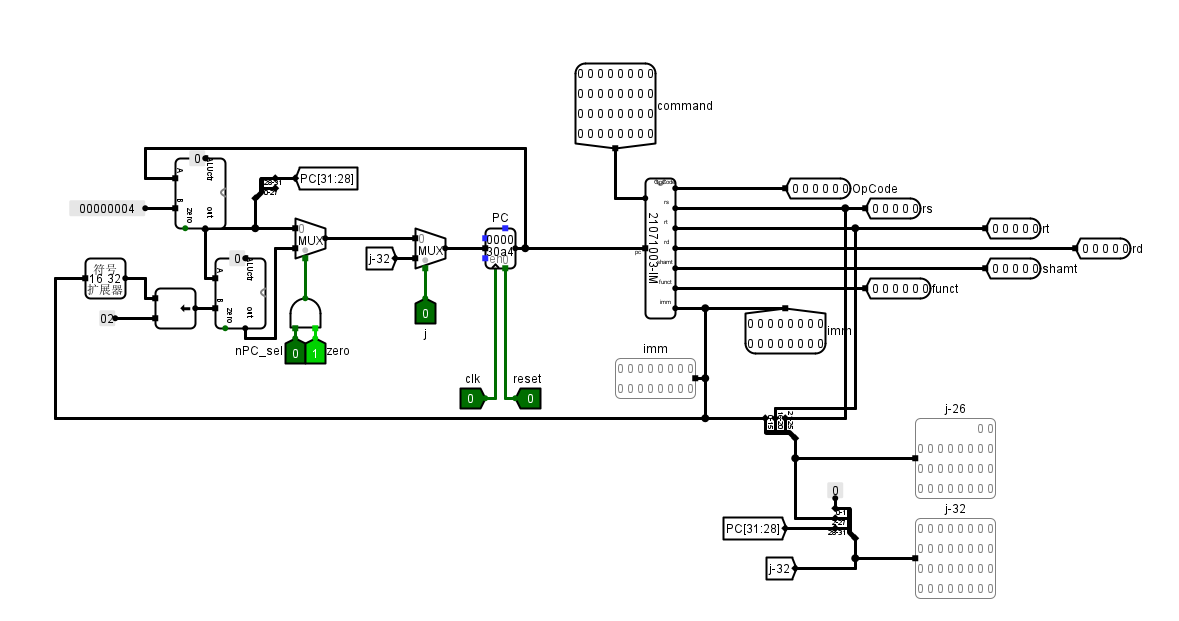


图2.IFU模块设计图

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| npc\_sel | I | 当前指令是否为beq指令标志  1：当前指令为beq  0：当前指令非beq |
| zero | I | ALU计算结果为0标志  1：计算结果为0  0：计算结果非0 |
| j | I | 当前指令是否为j指令标志  1：当前指令为j  0：当前指令非j |
| clk | I | 时钟信号 |
| rst | I | 复位信号  1：复位  0：无效 |
| command | O | 32位MIPS指令 |
| OpCode | O | 6位OpCode码 |
| rs | O | 5位rs码 |
| rt | O | 5位rt码 |
| rd | O | 5位rd码 |
| shamt | O | 5位shamt码 |
| funct | O | 6位funct码 |
| imm | O | 16位imm码 |

1. 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 | 编译指令 | rd、shamt、funct通过分线器合成imm备用；除了OpCode,剩下的指令合成26位j跳转立即数，并按照({PC[31:28],j[25:0],2’b00)拼接，备用。 |
| 4 | 计算下一条指令地址 | 如果当前指令不是beq指令，则PC🡨PC+4  如果当前指令是beq指令，并且zero为0，则PC🡨PC+4  如果当前指令是beq指令，并且zero为1，则PC🡨PC+4+(sign\_ext(ins[15:0])<<2)  如果当前是j指令，则PC🡨({PC[31:28],j[25:0],2’b00) |

2.ALU模块定义

1. 基本描述

ALU的主要功能是输入两个待操作数，通过ALUctr信号决定这两个数进行三种运算中的哪一种运算，这三种运算分别是：加法、减法、或运算。运算结果通过out端口输出。特别地，如果运算结果为0，那么ALU会输出zero信号。本报告的ALU设计路线为：组内并行加法器->组间并行加法器->16位并行加法器->32位并行加法器->ALU模块。其底层原理是与运算、异或运算和或运算。

组内并行设计公式：

G0\*=G3+P3G2+P3P2G1+P3P2P1G0 P0\*=P3P2P1P0

G1\*=G7+P7G6+P7P6G5+P7P6P5G4 P1\*=P7P6P5P4

G2\*=G11+P11G10+P11P10G9+P11P10P9G8 P2\*=P11P10P9P8

G3\*=G15+P15G14+P15P14G13+P15P14P13G12 P3\*=P15P14P13P12

C4 = G3+P3G2+P3P2G1+P3P2P1G0+P3P2P1P0C0

组间并行设计公式：

C4= G0\*+P0\*C0

C8= G1\*+P1\*C4= G1\*+P1\*G0\*+P1\*P0\*C0

C12= G2\*+P2\*C8=G2\*+P2\*G1\*+P2\*P1\*G0\*+P2\*P1\*P0\*C0

C16= G3\*+P3\*C12=G3\*+P3\*G2\*+P3\*P2\*G1\*+P3\*P2\*P1\*G0\*+P3\*P2\*P1\*P0\*C0

当Gi\*、Pi\*及C0形成后，C4、C8、C12和C16便可同时产生。

具体设计图如下图所示：

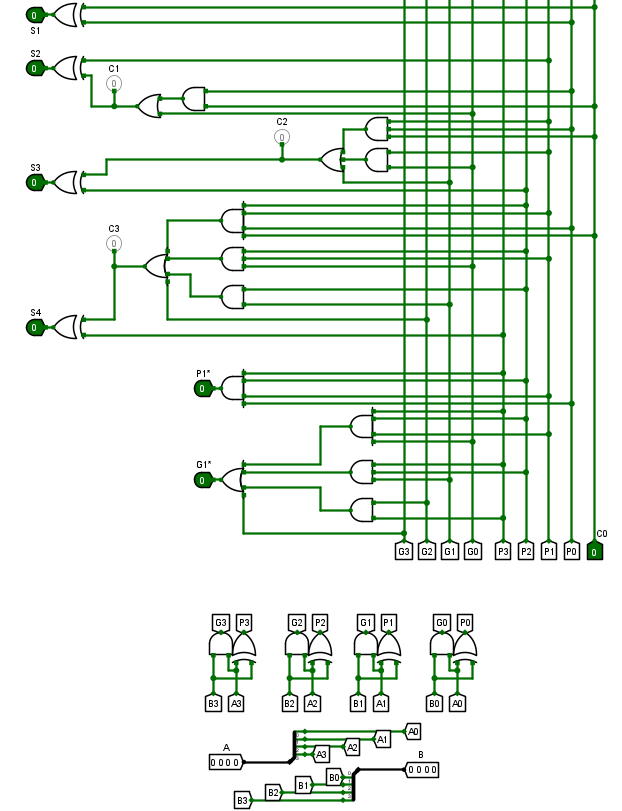


图3.组内并行加法器

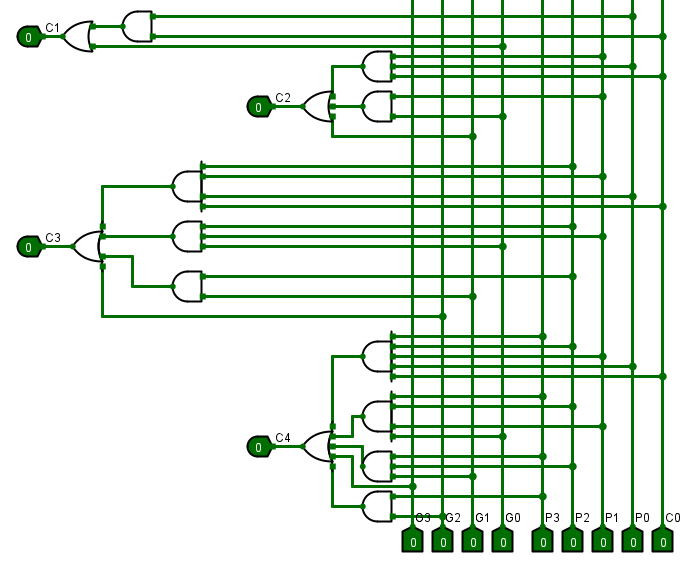


图4.组间并行加法器

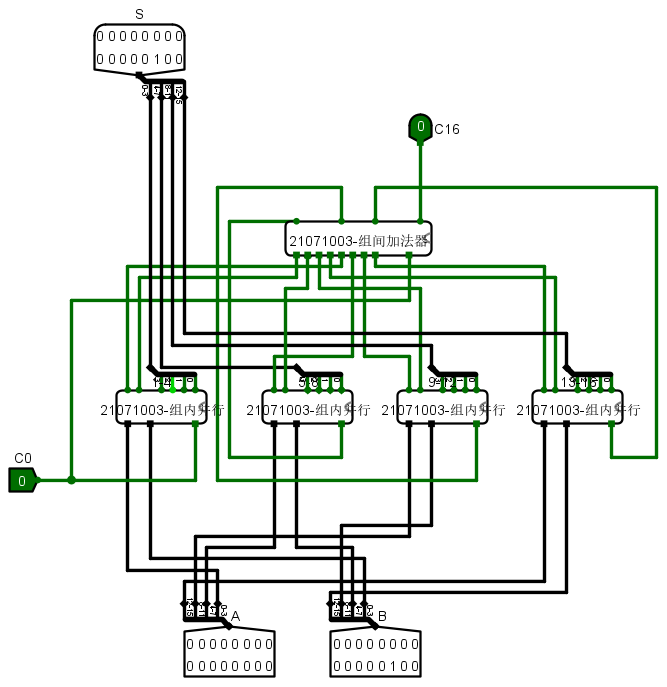


图5.16位并行加法器

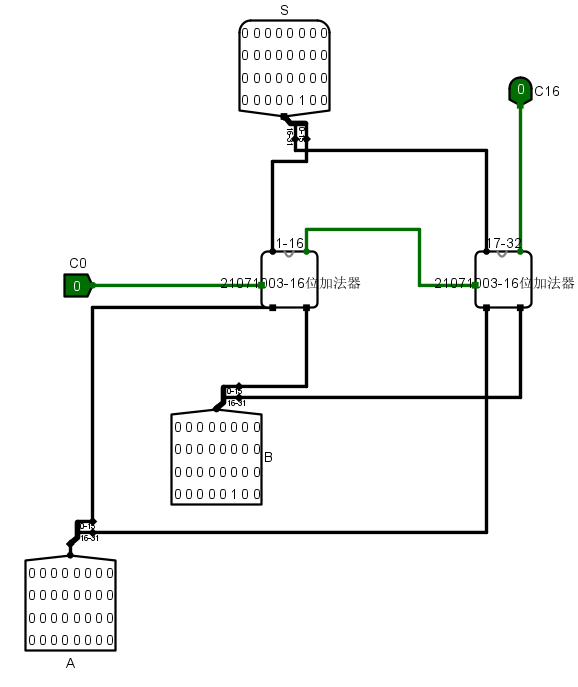


图6.32位并行加法器

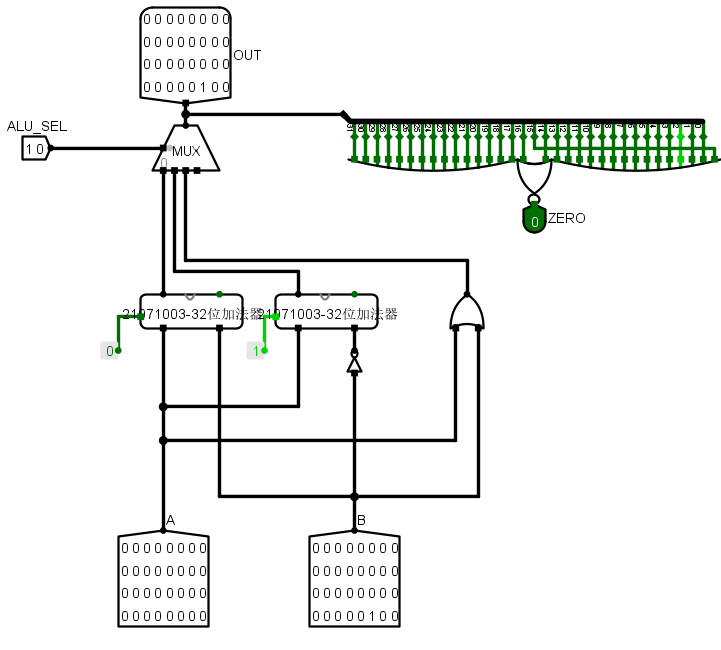


图7.ALU

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALUctr | I | 2位信号，功能如下：  00：加法操作  01：减法操作  10：或操作 |
| A | I | 待运算数A，32位 |
| B | I | 待运算数B，32位 |
| zero | O | 若运算结果全0，zero才为1，否则一直为0 |
| out | O | 运算结果输出，32位 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取模式 | 接收ALUctr，从而决定ALU进行哪种运算：  00：加法  01：减法  10：或运算 |
| 2 | 读取AB | 分别读取32位A和B |
| 3 | 运算并输出 | 选择好运算模式后，对AB进行相应的运算操作并选择输出 |
| 4 | Zero信号输出 | 通过或非门和分线器，对输出结果进行是否为全0的判断，如果是全0，输出zero信号为1，否则为0 |

3.EXT模块定义

1. 基本描述

EXT主要功能是对立即数进行扩展。其内部包括logisim自带的扩展器，分线器和MUX多路选择器。通过ExtOp信号决定本模块对立即数进行三种扩展方式中的哪一种扩展，这三种扩展方式分别为：无符号扩展、带符号扩展、lui低16位放入高16位，低16位补零。最后对扩展的立即数进行输出。其设计图如下：

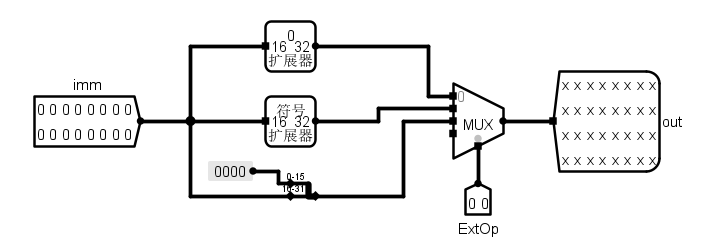


图8.EXT模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm | I | 16位立即数输入 |
| ExtOp | I | 2位信号，功能如下：  00：无符号扩展  01：带符号扩展  10：lui指令，低16位存入高16位，低16位补零 |
| out | O | 扩展结果输出 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取模式 | 接收ExtOp，从而决定扩展器的扩展模式：  00：无符号扩展  01：带符号扩展  10：lui指令，低16位存入高16位，低16位补零 |
| 2 | 读取立即数 | 读取16位imm立即数 |
| 3 | 扩展并输出 | 选择好扩展模式后，对立即数进行相应的扩展操作并选择输出 |

4.GPR模块定义

1. 基本描述

GPR是单周期数据通路的32个32位寄存器堆，可以进行寄存器地址选择、数据读取、数据清空操作。其中数据的写入还受到RegWr信号的控制。本模块主要由寄存器、多路选择器和解码器等组成。具体设计图如下：

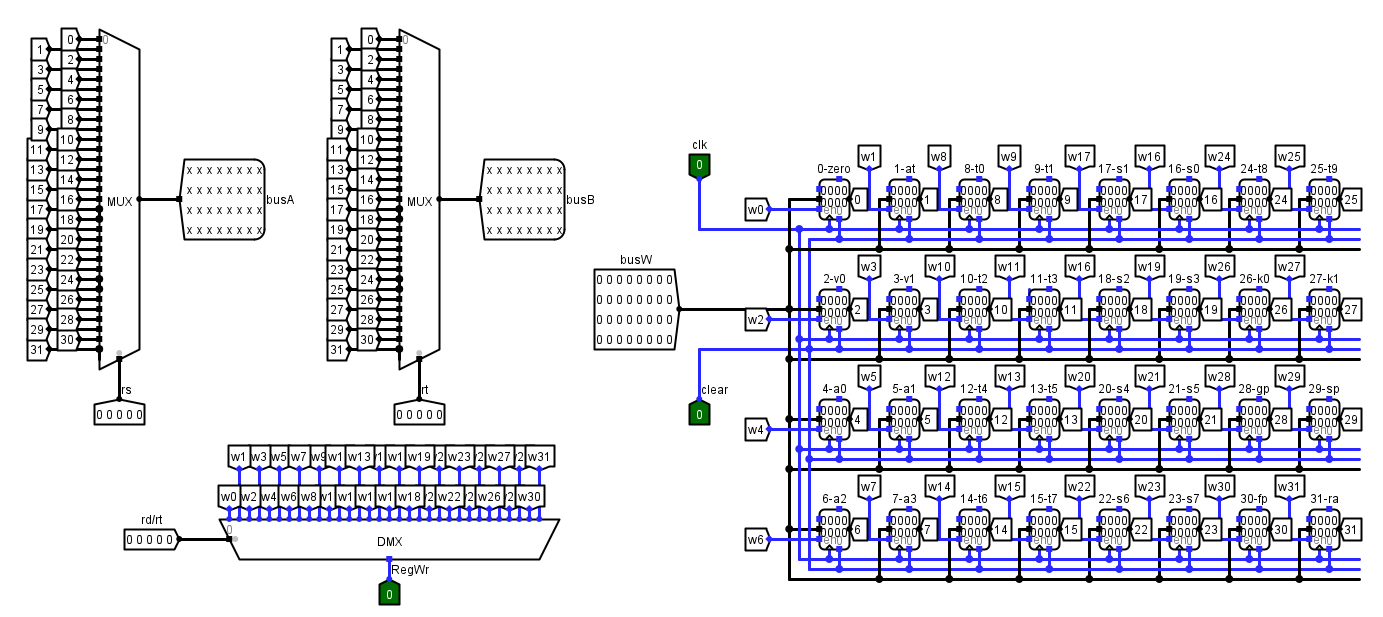


图9.GPR模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| clear | I | 异步清零信号，1代表清除GPR中所有寄存器的数据，0则无操作 |
| RegWr | I | 寄存器堆写入权限，1代表可写，0代表不可写 |
| rs | I | 5位输入，rs的地址 |
| rt | I | 5位输入，rt的地址 |
| rd/rt | I | 5位输入，当前指令为R型指令的时候，传入的是rd的地址，否则是rt的地址 |
| busW | I | 待写入寄存器堆的数据，32位 |
| busA | O | 从rs读取出的数据，32位 |
| busB | O | 从rt读取出的数据，32位 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 写入权限 | 接收rd/rt地址和RegWr写入权限，并通过解码器对其进行解码，得到32个寄存器的写入权限读取。当RegWr为1的时候，有且只有一个寄存器的写入权限为1，其余为0；当RegWr为0的时候，全部寄存器写入权限都是0 |
| 2 | 读取地址与输出 | 读取5位rs和rt的地址，通过多路选择器译码，选择某一个需要输出的寄存器，并输出32位的busA和busB |
| 3 | 数据写入 | 读取32位busW，将其接入寄存器堆的数据写入端，当某一寄存器的Wr为1，clk上升沿的时候，数据就会写入对应的寄存器。 |
| 4 | 异步清零 | Clear信号无视时钟，只要其值为1，就会清空寄存器堆所有寄存器存储的数据 |

5.IM模块定义

1. 基本描述

IM的主要功能是存储指令，并通过PC进行读取操作，指令通过小端序存储。主要由4个256B只读存储器ROM组成。本模块还可以对指令进行译码进行输出。其具体设计图如下：

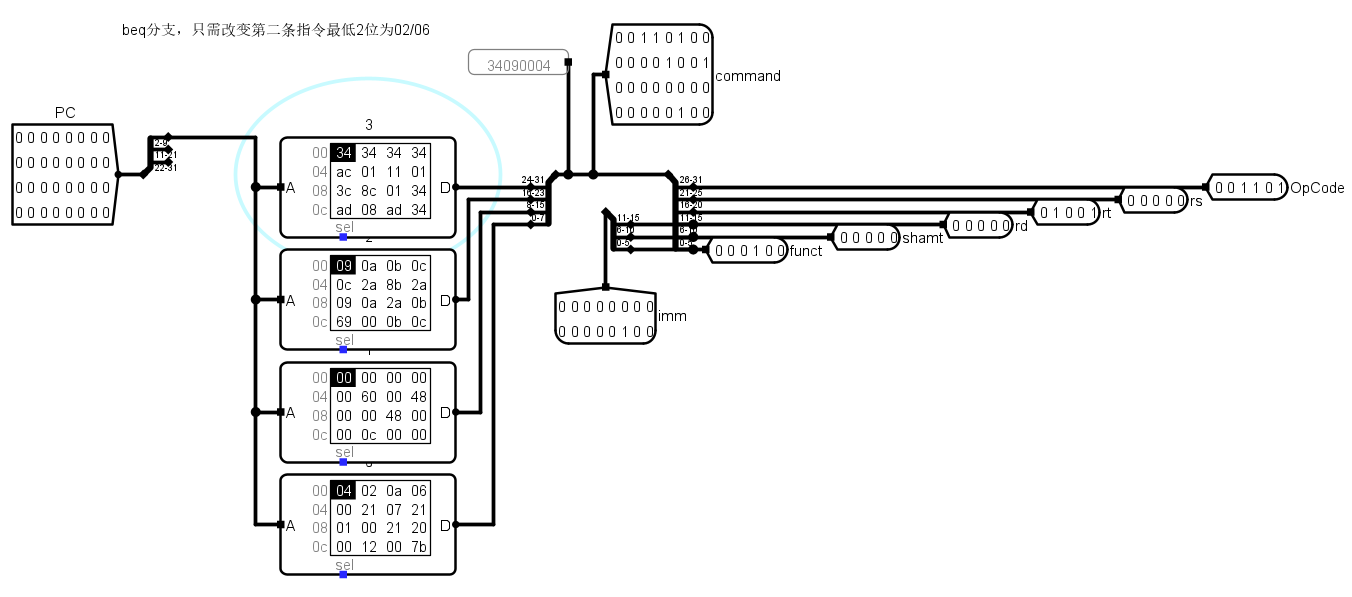


图10.IM模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc | I | 32位指令地址输入 |
| command | O | 32位MIPS指令 |
| OpCode | O | 6位OpCode码 |
| rs | O | 5位rs码 |
| rt | O | 5位rt码 |
| rd | O | 5位rd码 |
| shamt | O | 5位shamt码 |
| funct | O | 6位funct码 |
| imm | O | 16位imm码 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 指令读取 | 读取PC的[9:2]，传入4个ROM进行指令读取 |
| 2 | 指令译码 | 对读取出的32位指令进行译码，得到OpCode,rs,rt,rd,shamt,funct,imm |
| 3 | 指令展示 | 为了便于观测指令，本模块支持32位指令直接输出，便于用探针观测 |

6.DM模块定义

1. 基本描述

DM的主要功能是存储数据，以供寄存器组进行读写操作。它由4个256B的随机存储器RAM组成，存储模式为小端序，每个RAM的数据接口模式为“分离的加载和存储引脚”，从而将需要写入DM的值和写入地址进行分开读取操作。具体设计如下：

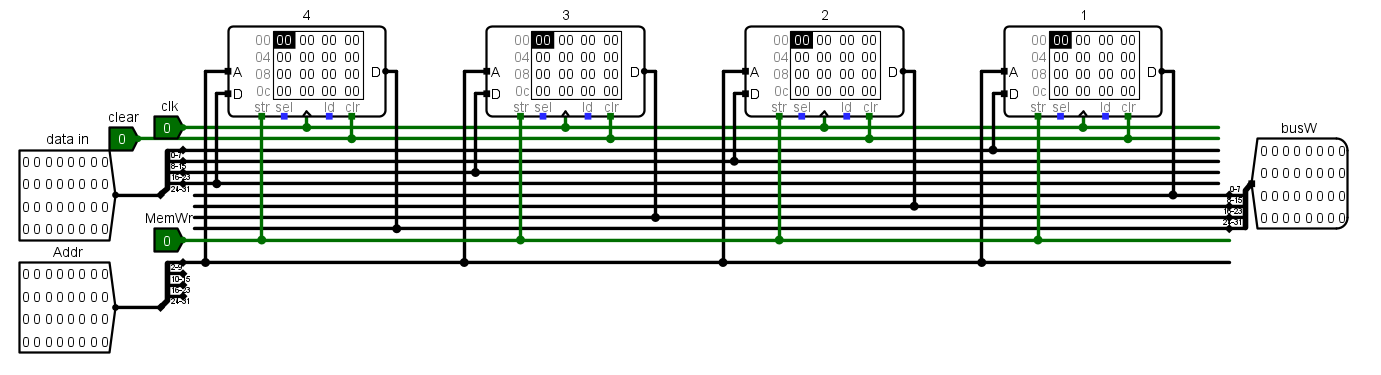


图11.DM模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| clear | I | 异步清零信号，1为清除DM中所有数值，0则无操作 |
| data in | I | 32位待写入数据 |
| MemWr | I | 1位信号，决定是否对DM进行写入操作，1代表可写，0代表不可写 |
| Addr | I | 32位地址 |
| busW | O | 32位输出 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据写入 | 读取data in和Addr[9:2]，当MemWr为1且clk上升沿的时候，对DM中4个RAM对应地址进行数据写入。 |
| 2 | 数据读取 | 根据Addr[9:2]，寻找到DM中4个RAM对应地址的数据并且输出到busW引脚 |
| 3 | 异步清零 | Clear信号无视时钟，只要其值为1，就会清空DM所有RAM存储的数据 |

7.Controller模块定义

1. 基本描述

Controller模块的主要功能是对OpCode和funct进行译码，从而输出一系列对数据通路非常重要的信号，比如上文提到的ALUctr，ExtOp等，这些信号将会在（2）详细列出。模块主要由分线器、与或非逻辑门组成。具体设计如下：

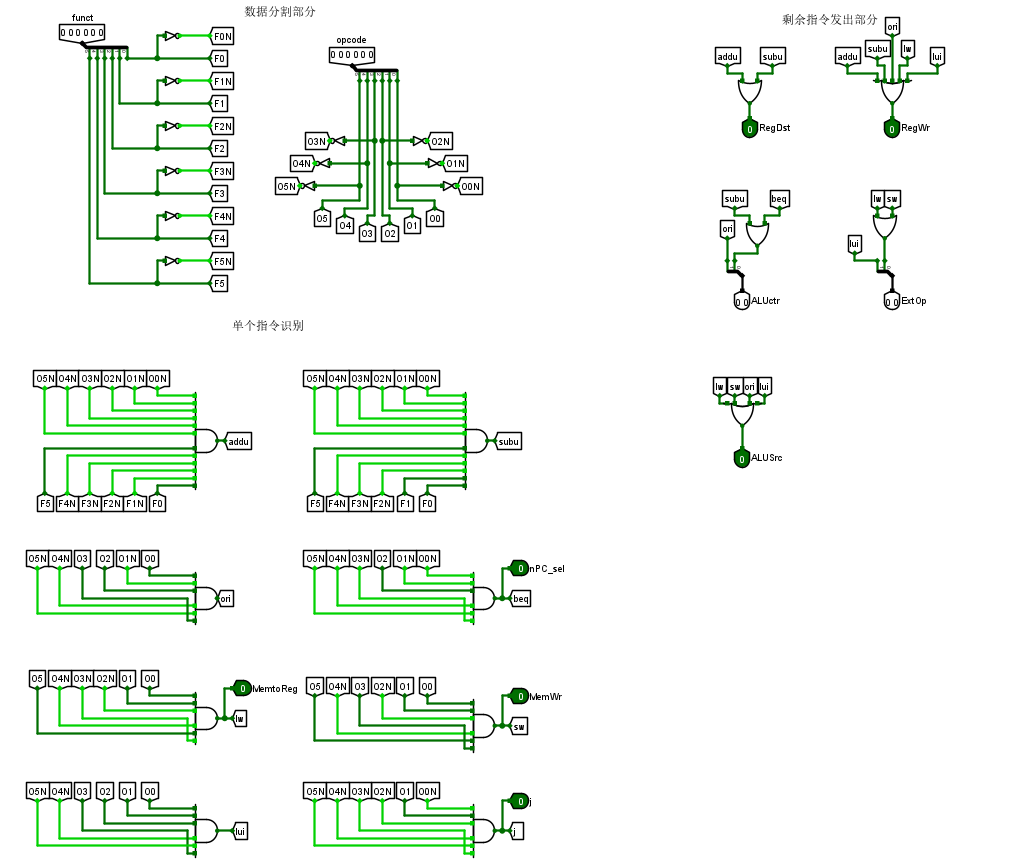


图12.Controller模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| OpCode | I | 6位OpCode |
| funct | I | 6位funct |
| j | O | 判断当前指令是否为j指令  0：不是j指令  1：是j指令 |
| ALUctr | O | 控制ALU进行哪种运算  00：加法  01：减法  10：或运算  对于MIPS-Lite，ALUctr = {ori, (subu | beq)} |
| RegDst | O | 根据当前指令判断是不是R型，在总体设计中输入到MUX  是：输出1，使得rd传入GPR的rd/rt输入端  否：输出0，使得rt传入GPR的rd/rt输入端  对于MIPS-Lite，RegDst = (addu | subu) |
| RegWr | O | 根据当前指令判断是否涉及到寄存器堆的写入操作  是：输出1，寄存器堆指定寄存器可写  否：输出0，寄存器堆不可写  对于MIPS-Lite，RegWr = (addu | subu | ori | lw | lui) |
| ExtOp | O | 根据当前指令设置EXT模块的扩展模式  00：无符号扩展  01：带符号扩展  10：lui指令，低16位存入高16位，低16位补零  对于MIPS-Lite，ExtOp = {lui, (lw | sw)} |
| MemWr | O | 根据当前指令判断DM是否需要写入  1：需要写入DM  0：不需要  对于MIPS-Lite，MemWr = sw |
| MemtoReg | O | 根据当前指令判断从DM读取出的数据是否需要存入GPR  1：需要存入GPR  0：不需要  对于MIPS-Lite，MemtoReg = lw |
| nPC\_SEL | O | 判断当前指令是不是beq指令  1：是  0：不是 |
| ALUSrc | O | 根据当前指令判断是不是R型，从而决定ALU的B输入端输入的是busB还是imm  1：是R型，ALU的B端口输入imm  0：不是R型，ALU的B端口输入busB  对于MIPS-Lite，ALUSrc = (lw | sw | ori | lui) |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 指令译码 | 读取6位OpCode和funct，进行译码，获得（2）中提到的各个信号 |

# 三．验证

1. 给出设计的所有机器指令描述（列出指令操作码助记符、操作码的机器指令代码及指令功能的对应关系表）。
   1. addu指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 无符号加法 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. subu指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 无符号减法 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. ori指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 或运算 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. j指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 跳转 |
| 格式 |  |
| 描述 | 把4GB划分为16个256MB区域，j指令可以允许PC在当前PC所在的256MB的区域内进行跳转。如果跳转超出256MB，需要JR指令 |
| 操作 |  |

* 1. lw指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 读字 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. sw指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 存字 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. beq指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 判断相等 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

* 1. lui指令描述

|  |  |
| --- | --- |
| 编码 |  |
| 功能 | 高位扩展 |
| 格式 |  |
| 描述 |  |
| 操作 |  |

1. 给出编写的测试程序，对每条机器指令添加注释。

# 创建三个数字，DM存入6，判断前两个数相加是否等于第三者

# 相等则DM更新为第三个数,DM[7]存储123，计算123+4x7

# 否则两数相加，对1进行lui后，再加上DM的数，存到DM[9]

# 最后存入2023到某一寄存器

2.1 homework-beq1.asm

ori $t1, $0, 4 # a = 4;

ori $t2, $0, 6 # b = 6; # homework-beq2.asm这里的6改成了2，其他都和本程序相同

ori $t3, $0, 10 # c = 10;

ori $t4, $0, 6 # d = 6;

sw $t4, 0($0) # DM[0] = d;

addu $t4, $t1, $t2 # d = a +b;

beq $t4, $t3, then # if(c == d) ---> then

else:addu $t1, $t1, $t2 # else{

lui $t1,0x1 # 对0x1进行lui操作，存到a里

lw $t2, 0($0) # b = DM[0];

addu $t1, $t1, $t2 # a += b;

ori $t3, $0, 32 # c = 32;

sw $t1, 0($t3) # DM[c / 4] = a;}

j end # 跳转到end

then: sw $t3, 0($0) # then{ DM[0] = c

ori $t4, $0, 123 # d = 123;

ori $t3, $0, 28 # c = 28;

sw $t4, 0($t3) # DM[c / 4] = d;

addu $t4, $t4, $t3 # d += c;}

end:

ori $t5, $0, 2023 # e = 2023;

sw $t5, 36($0) # DM[9] = e;

2.2 homework2-beq2.ams

ori $t1, $0, 4 # a = 4;

ori $t2, $0, 2 # b = 6; # homework-beq1.asm这里的2改成了6，其他都和本程序相同

ori $t3, $0, 10 # c = 10;

ori $t4, $0, 6 # d = 6;

sw $t4, 0($0) # DM[0] = d;

addu $t4, $t1, $t2 # d = a +b;

beq $t4, $t3, then # if(c == d) ---> then

else:addu $t1, $t1, $t2 # else{

lui $t1,0x1 # 对0x1进行lui操作，存到a里

lw $t2, 0($0) # b = DM[0];

addu $t1, $t1, $t2 # a += b;

ori $t3, $0, 32 # c = 32;

sw $t1, 0($t3) # DM[c / 4] = a;}

j end # 跳转到end

then: sw $t3, 0($0) # then{

ori $t4, $0, 123 # d = 123;

ori $t3, $0, 28 # c = 28;

sw $t4, 0($t3) # DM[c / 4] = d;

addu $t4, $t4, $t3 # d += c;}

end:

ori $t5, $0, 2023 # e = 2023;

sw $t5, 36($0) # DM[9] = e;

1. 给出与测试程序相对应的测试结果截图，并附以文字说明。

3.1 homework-beq1.asm代码测试

本段代码中的beq会进入then分支，根据程序设计，测试结果应当如下：

①t1存储0x4 ②t2存储0x6 ③t3存储0x1c ④t4存储0x97 ⑤t5存储0x7e7⑥DM[0] = 0xa ⑦DM[7] = 0x7b ⑧DM[9] = 0x7e7



图13.homework-beq1.asm MARS中运算最终结果 DM



图14.homework-beq1.asm MARS中运算最终结果 GPR

运行代码过程中，总设计图并无出现蓝线和红线错误；运行完代码，对GPR和DM的数据进行观察，如下图：

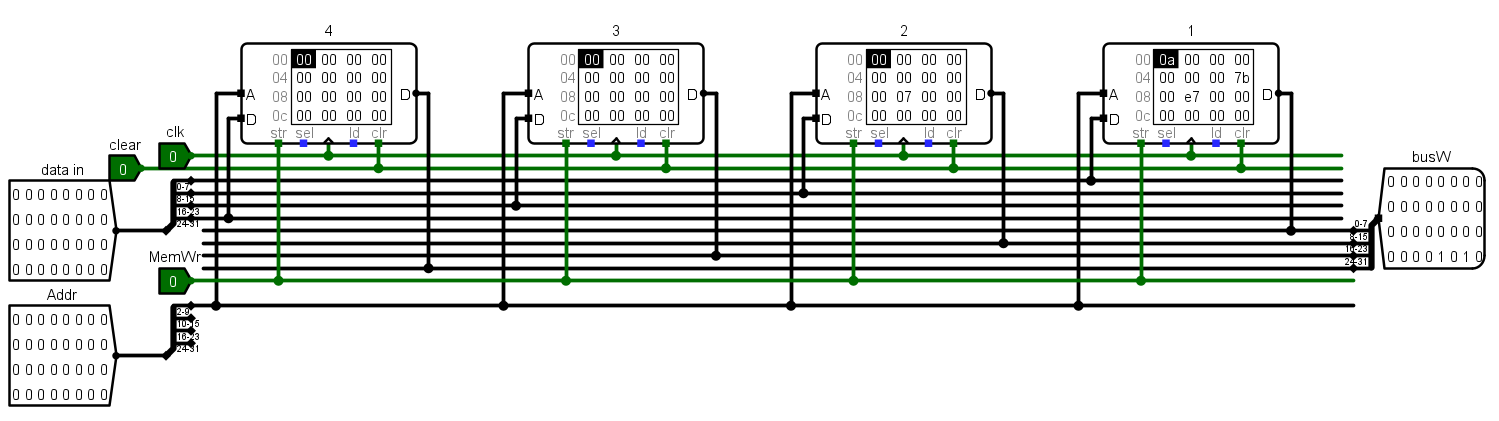


图15.homework-beq1.asm DM最终结果

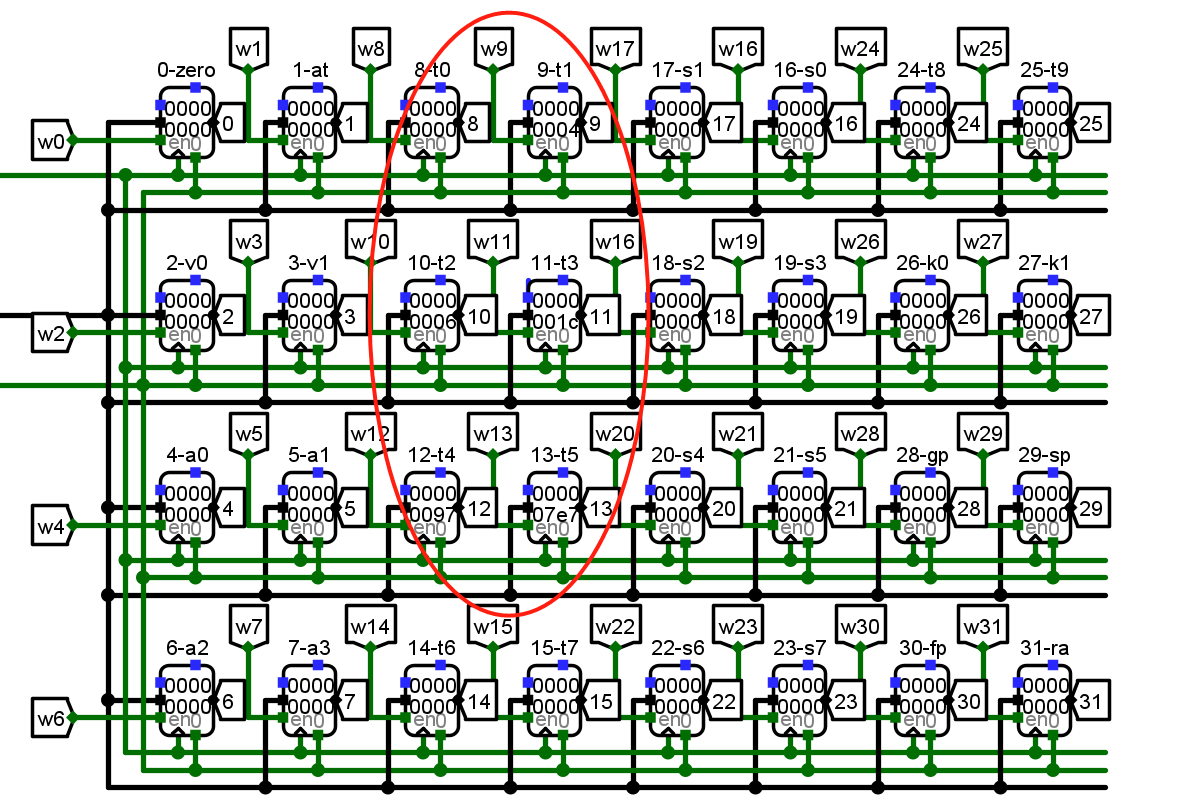


图16.homework-beq1.asm GPR最终结果

结果表明，测试程序对应的测试结果正确。

3.2 homework-beq2.asm代码测试

本段代码中的beq会进入else分支，根据程序设计，测试结果应当如下：

①t1存储0x6 ②t2存储0x6 ③t3存储0x20 ④t4存储0x26 ⑤t5存储0x7e7 ⑥DM[0] = 0x6 ⑦DM[8] = 0x0001\_0006

⑧DM[9] = 0x7e7



图17.homework-beq2.asm MARS中运算最终结果 DM

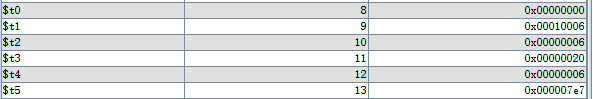
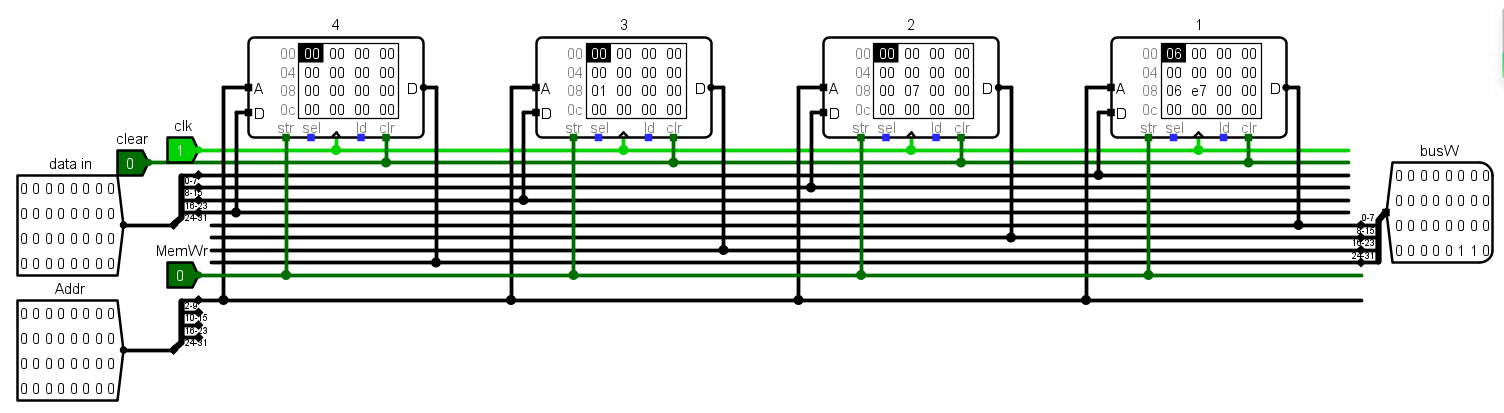
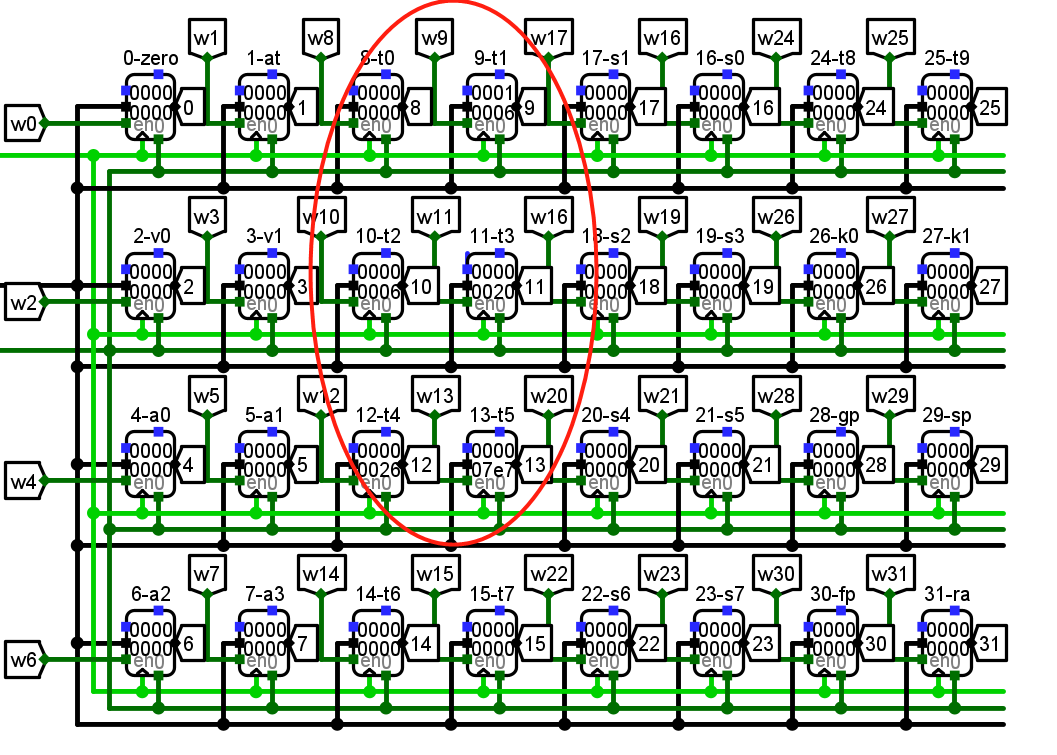


图18.homework-beq2.asm MARS中运算最终结果 GPR

运行代码过程中，总设计图并无出现蓝线和红线错误；运行完代码，对GPR和DM的数据进行观察，如下图：

图19.homework-beq2.asm DM最终结果

图20.homework-beq2.asm GPR最终结果

结果表明，测试程序对应的测试结果正确。

# 四．总结

在学习到计组第五章单周期数据通路的时候，猛然出现在眼前的数据通路图和大量的信号等，使我在课上和课下很难记住并学会它们的名称及功能，因为没有上手对数据通路进行设计和实践，所以对知识点的理解还停留在课上PPT中的抽象化文字图片内容。

通过本次《计算机组成原理》大作业的亲手编写设计，我不仅对单周期数据通路的理解更上一层楼，更是跳出了课本内容，基于课本内容自行设计规定的指令和任务，比如对j指令的处理。

用logism设计时，难点在于Controller和IFU的设计，幸好有上学期数逻大作业的基础，设计起来不算很难，花费时间也不多。但是用MARS设计验证程序的时候，我却卡住了很久，由于第一次接触本语言，而且它和高级程序语言不同之处太多，我一时难以上手，而且MIPS-Lite指令少，我总想写出一些有意义的程序，却发现有些难以办到。最后我选择了if-else分支进行验证。在验证的过程中，我发现了电路潜在的错误，并进行了改正，最终电路验证正确，开始写报告。

总体来说，大作业使得我对课堂的理论知识加以实践，使得我对理论的理解大大加深，而且大作业还占总分的分数占比，减小期末考试的分数占比，我很喜欢这种考核方式。