**计算机组成原理**

**课内大作业报告**

**学 号\_\_\_\_\_\_\_\_18020208\_\_\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_\_丛熙平\_\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师\_\_\_\_\_\_\_\_\_朱文军\_\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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

[一.总体数据通路结构设计 4](#_Toc40227048)

[二.全部模块详细描述 6](#_Toc40227049)

[1. 18020208-CONTROL 6](#_Toc40227050)

[(1)基本描述 6](#_Toc40227051)

[(2)模块接口 6](#_Toc40227052)

[(3)功能定义 7](#_Toc40227053)

[2. 18020208-ALU 7](#_Toc40227054)

[(1)基本描述 7](#_Toc40227055)

[(2)模块接口 7](#_Toc40227056)

[(3)功能定义 8](#_Toc40227057)

[3. 18020208-32bit\_sub 8](#_Toc40227058)

[(1)基本描述 8](#_Toc40227059)

[(2)模块接口 8](#_Toc40227060)

[(3)功能定义 8](#_Toc40227061)

[4. 18020208-32bit\_add 8](#_Toc40227062)

[(1)基本描述 8](#_Toc40227063)

[(2)模块接口 9](#_Toc40227064)

[(3)功能定义 9](#_Toc40227065)

[5. 18020208-16bit\_add 9](#_Toc40227066)

[(1)基本描述 9](#_Toc40227067)

[(2)模块接口 9](#_Toc40227068)

[(3)功能定义 9](#_Toc40227069)

[6. 18020208-Carry 10](#_Toc40227070)

[(1)基本描述 10](#_Toc40227071)

[(2)模块接口 10](#_Toc40227072)

[(3)功能定义 10](#_Toc40227073)

[7. 18020208-4bit\_add 11](#_Toc40227074)

[(1)基本描述 11](#_Toc40227075)

[(2)模块接口 11](#_Toc40227076)

[(3)功能定义 11](#_Toc40227077)

[8. 18020208-DM 11](#_Toc40227078)

[(1)基本描述 11](#_Toc40227079)

[(2)模块接口 12](#_Toc40227080)

[(3)功能定义 12](#_Toc40227081)

[9. 18020208-GPR 12](#_Toc40227082)

[(1)基本描述 12](#_Toc40227083)

[(2)模块接口 12](#_Toc40227084)

[(3)功能定义 13](#_Toc40227085)

[10. 18020208-EXT 13](#_Toc40227086)

[(1)基本描述 13](#_Toc40227087)

[(2)模块接口 13](#_Toc40227088)

[(3)功能定义 14](#_Toc40227089)

[11. 18020208-IFU 14](#_Toc40227090)

[(1)基本描述 14](#_Toc40227091)

[(2)模块接口 14](#_Toc40227092)

[(3)功能定义 15](#_Toc40227093)

[三.机器指令描述 15](#_Toc40227094)

[四.测试程序 15](#_Toc40227095)

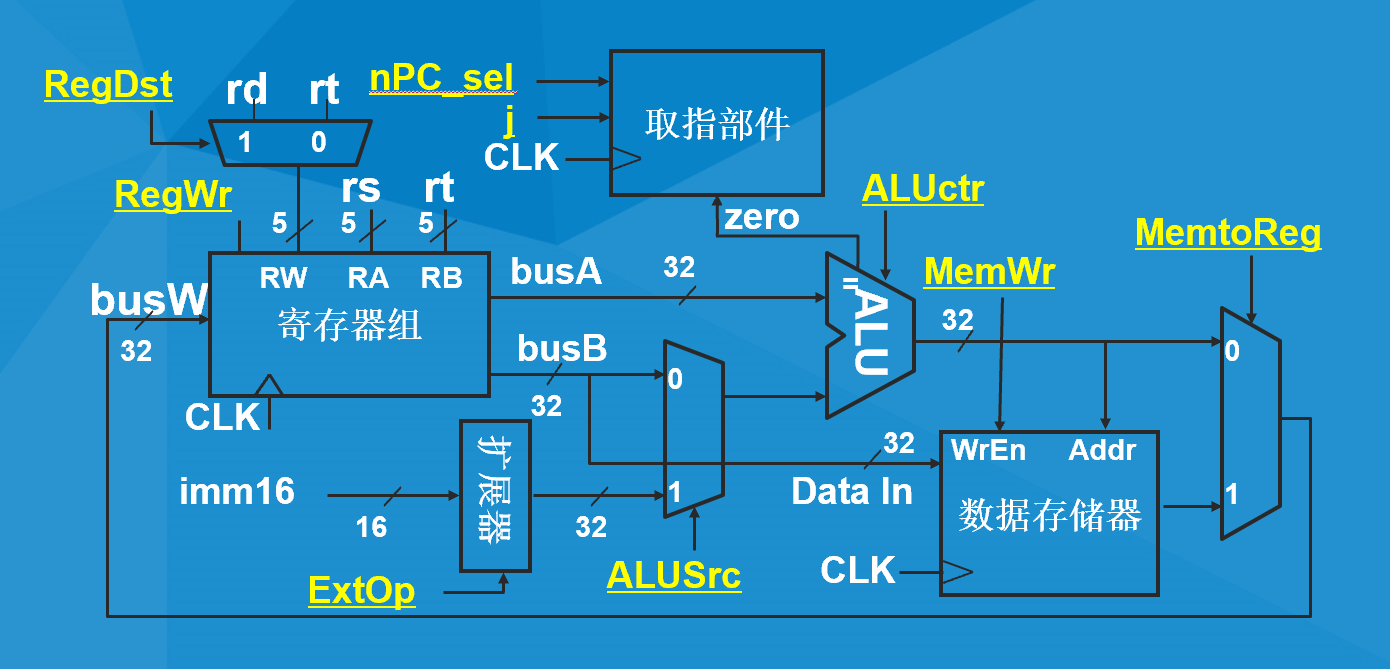
[五.测试结果 16](#_Toc40227096)

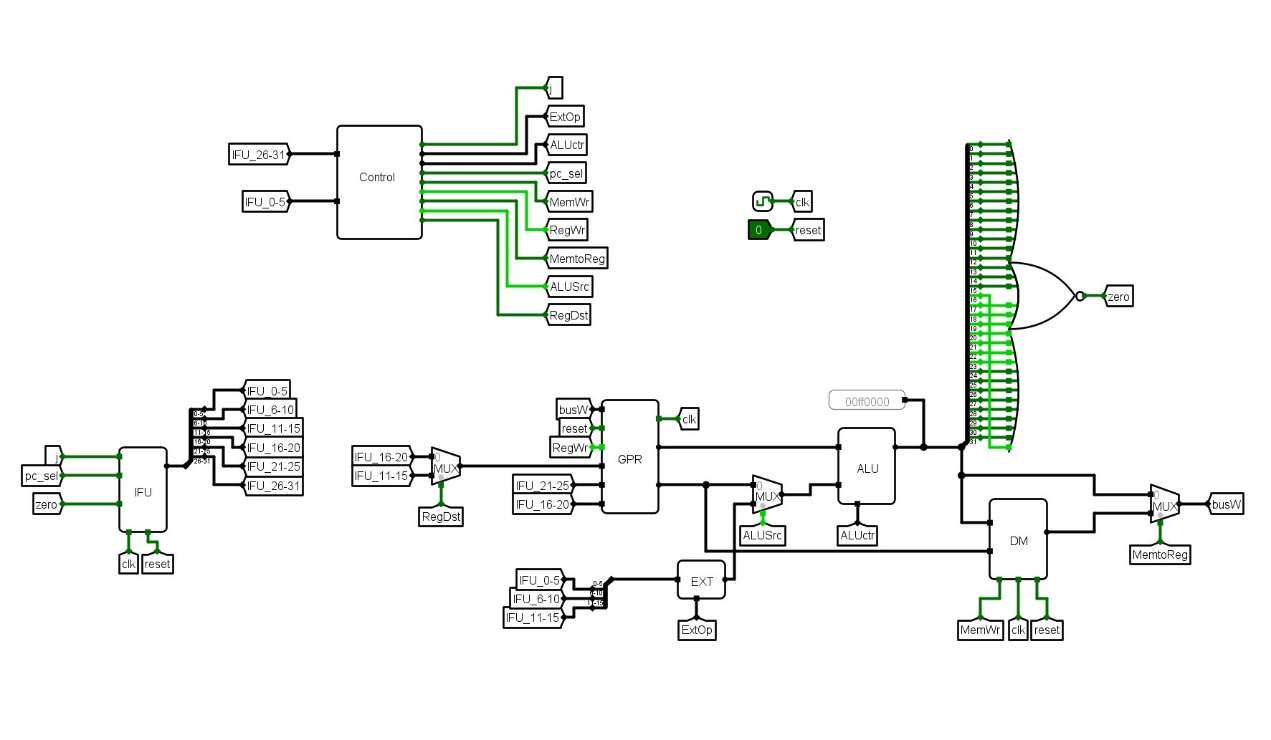
[1.测试结果截图 16](#_Toc40227097)

[2.测试结果说明 18](#_Toc40227098)

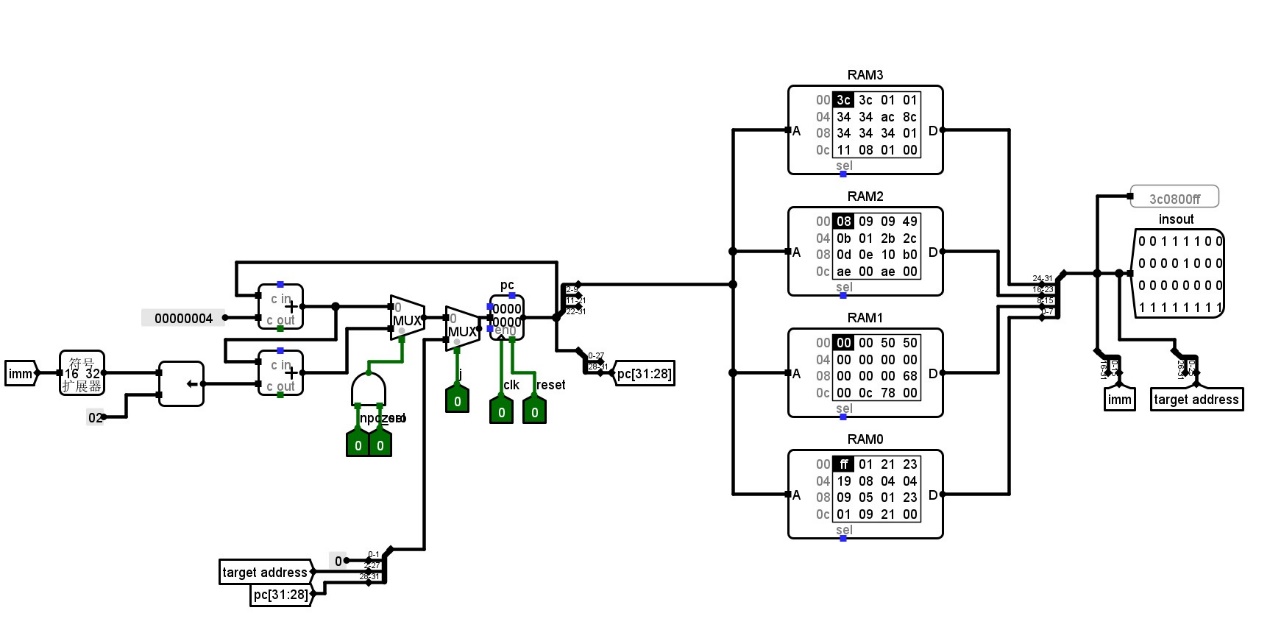
[六.心得体会 18](#_Toc40227099)

# 一.总体数据通路结构设计

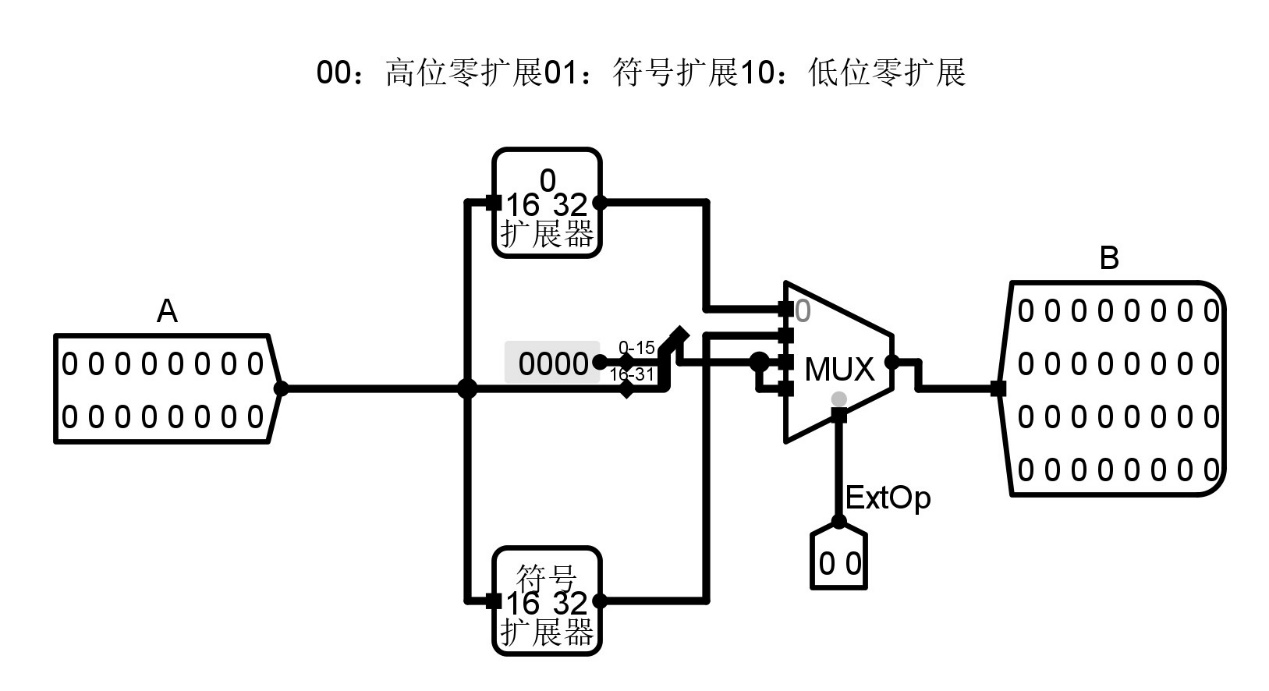




在原有整体数据通路的基础上，增加了j指令与lui指令的设计，整体main中的数据通路如图，具体调整为，ifu模块加入j使能端，ExtOp扩展为2bit选择端，EXT扩展器中加入低位0扩展功能。



如图所示，ifu部件中的改动为在pc前多增加了多路选择器，用j选择信号实现控制输入pc中的地址为beq指令中的跳转地址，正常pc+4（由npc\_sel，zero共同控制）或j指令跳转地址，实现方法为在取指拼接后的输出线路中加入分线器，将j指令0-25位的target address取出，与pc+4，低两位0共同拼接成[pc+4，target address，00]共32位地址，输入pc读取相应的指令。



如图所示，EXT扩展器部件中增加低位0扩展功能，并将ExtOp选择段扩展为2bit，具体实现方法：将输入的16位立即数作为32位输出的16-31位，其余0-15位置0，实现lui高16位复制，低16位清零功能。

# 二.全部模块详细描述

## 1. 18020208-CONTROL

### (1)基本描述

CONTROL模块主要用于根据不同指令生成总体数据通路中RegDst，ALUsrc，MemtoReg，RegWrite等控制端与选择端信号。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| opcode | I | 32bit操作数 |
| function | I | 32bit操作数 |
| RegDst | O | 控制GPR的rw端选择rt/rd  0：rt  1：rd |
| ALUsrc | O | 控制输入ALU的第二操作数为扩展后的16bit立即数/寄存器中所读取的数据  0：寄存器中所读取的数据  1：扩展后的16bit立即数 |
| MemtoReg | O | 控制写回GPR模块busw端的数据为ALU运算器的输出值/DM中所读取的数据  0：ALU运算器的输出值  1：DM中所读取的数据 |
| RegWrite | O | GPR模块写入使能端  0：不允许数据写入寄存器  1：允许数据写入寄存器 |
| MemWrite | O | 控制DM模块写入使能端  0：不允许数据写入内存  1：允许数据写入内存 |
| npc\_sel | O | 与zero控制端配合，控制ifu模块中beq/其他指令的取指模式  0：其他指令取值  1：beq指令取值 |
| ExtOp | O | 2bit控制端，用于控制EXT模块对输入进行不同的扩展操作  00：高位0扩展  01：符号扩展  10/11：低位0扩展 |
| ALUCtr | O | 2bit控制端，用于控制ALU模块对输入两操作数执行不同运算  00：加法运算  01：减法运算  10：按位或运算 |
| j | O | 用于控制ifu模块输入到pc中的跳转指令地址为正常pc+4，beq，j  0：pc+4，beq2选1（由npc\_sel，zero联合控制）  1：j指令跳转地址 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 生成控制信号 | 根据不同指令的opcode与function生成总体数据通路中控制端与选择端信号。 |

## 2. 18020208-ALU

### (1)基本描述

用32bit-add，32bit-sub，和一个或门（logisim自带）实现ALU，主要功能为实现两个32bit操作数的加法，减法，或操作。其中，有ALUctr控制端，00为加法，01为减法，10为按位或运算。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 32bit操作数 |
| B | I | 32bit操作数 |
| ALUctr | I | 2bit运算控制端：  00：两操作数加法  01：两操作数减法  10：两操作数按位或运算 |
| result | O | 两32bit操作数运算结果 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 32bit操作数加法 | 两32bit操作数加法 |
| 2 | 32bit操作数减法 | 两32bit操作数减法 |
| 3 | 32bit操作数或操作 | 两32bit操作数按位或运算 |

## 3. 18020208-32bit\_sub

### (1)基本描述

用两个16bit-add并行连接，将第二个操作数输入之后进行取反操作，实现两个32bit操作数的减法。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 32bit操作数 |
| B | I | 32bit操作数 |
| res | O | 两32bit操作数减法结果 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 32bit操作数减法 | 两32bit操作数减法 |

## 4. 18020208-32bit\_add

### (1)基本描述

用两个16bit-add并行连接，实现两个32bit操作数的加法。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 32bit操作数 |
| B | I | 32bit操作数 |
| res | O | 两32bit操作数加法结果 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 32bit操作数加法 | 两32bit操作数加法 |

## 5. 18020208-16bit\_add

### (1)基本描述

用四个4bit-add与carry模块并行连接，实现两个16bit操作数的加法。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| C0 | I | 上一组件的进位信号 |
| A | I | 16bit操作数 |
| B | I | 16bit操作数 |
| res | O | 两16bit操作数加法结果 |

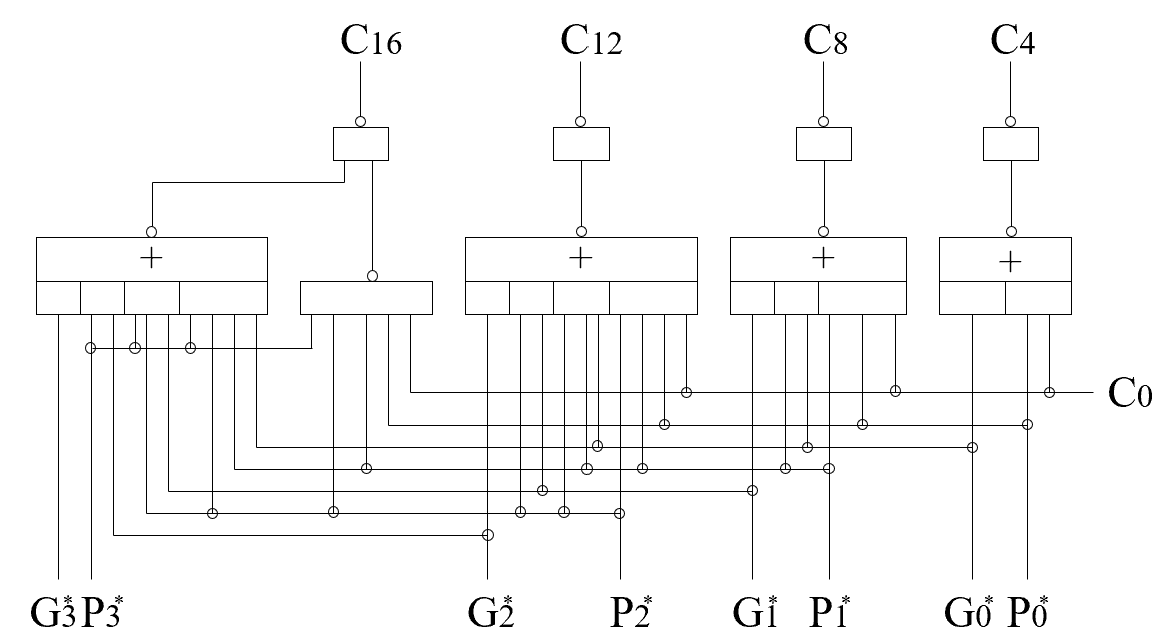
### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 16bit操作数加法 | 两16bit操作数加法 |

## 6. 18020208-Carry

### (1)基本描述

16位加法器采用组内并行，组间并行的进位结构，模块用于产生小组间的四个进位。下图为原理图。



### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| C0 | I | 上一组件的进位信号 |
| G0\*-G3\* | I | 不同小组产生的进位 |
| P1\*-P3\* | I | 小组进位的传递条件(决定于低位小组进位能否传送至高位小组） |
| C4\*-C16\* | O | 组间进位信号 |

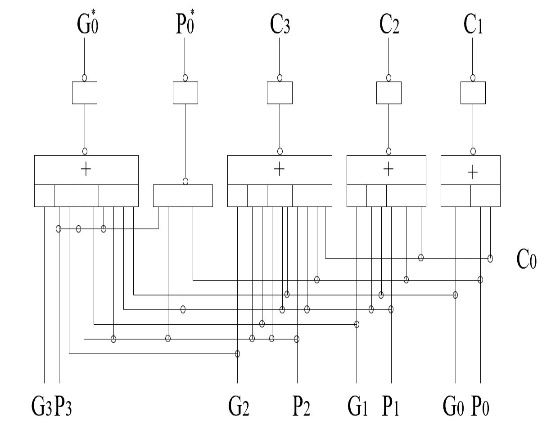
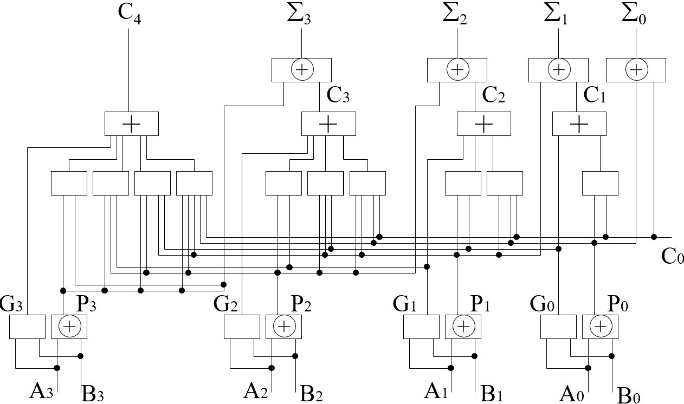
### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 进位器 | 16bit加法器有4个4bit加法器并行连接，通过4bit加法器输出的Gi\*，Pi\*，C0产生组间的进位信号 |

## 7. 18020208-4bit\_add

### (1)基本描述

4bit-add为4位组内并行加法器，下图为原理图



### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| C0 | I | 并行进位信号 |
| A0-A3 | I | 4bit操作数1 |
| B0-B3 | I | 4bit操作数2 |
| Sigma0-Sigma3 | O | 两4bit操作数加和后结果 |
| Gi\* | O | 本小组产生的进位 |
| Pi\* | O | 小组进位的传递条件决定于低位小组进位能否传送至高位小组） |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 4bit操作数加法 | 两4bit数的并行加法 |

## 8. 18020208-DM

### (1)基本描述

DM位mips处理器的内存部分，容量为1KB，分为4个RAM实现，主要功能为存储和读取相应位置的数据。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| MemWr | I | 控制存储的使能信号：  0：无法存储  1：从输入端存储数据 |
| reset | I | 重置使能，当为1时，异步重置内存为全0 |
| addr | I | 需要访问数据的地址 |
| data\_in | I | 即将写入的数据值 |
| clk | I | 时钟信号 |
| data\_out | O | 读取数据输出 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据写入 | 当MemWr写使能信号为1时，addr需要访问数据的地址配合data\_in输入数据进行内存的写入。 |
| 2 | 数据读取 | 当MemWr写使能信号为0时，addr需要访问数据进行内存的读取，data\_out进行输出 |
| 3 | 清零 | 当reset控制端为1时，异步重置内存为全0 |

## 9. 18020208-GPR

### (1)基本描述

GPR的主要功能为寄存器组，在mips结构中，有32个32位的寄存器，以这些具有写使能的寄存器为基础，辅以多路选择器与输入输出，实现寄存器组。功能包括置零，选择，写入，输出。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RegWr | I | 控制写入的使能信号：  0：无法写入  1：允许写入 |
| rw | I | 写入寄存器的多路选择  5位位宽，分别对应31个寄存器单元,$s0不能写入 |
| busw | I | 写入的数据，32位位宽 |
| clk | I | 时钟信号 |
| ra | I | 输出寄存器数据的多路选择  5位位宽，分别对应32个寄存器单元 |
| rb | I | 输出寄存器数据的多路选择  5位位宽，分别对应32个寄存器单元 |
| busa | O | 寄存器数据输出 |
| busb | O | 寄存器数据输出 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 寄存器数据写入 | 当RegWr写使能信号为1时，rw寄存器多路选择配合busw输入数据进行相应寄存器数据的写入。 |
| 2 | 清零 | Reset使能信号为1时，全部寄存器数据清零。 |
| 3 | 寄存器数据读取 | ra，rb寄存器多路选择配合busa，busb32位输出相应寄存器内的数据。 |

## 10. 18020208-EXT

### (1)基本描述

EXT的主要功能是对立即数进行扩展，EXT内包括扩展器，功能包括立即数的高位零扩展，符号扩展以及低位零扩展，低位零扩展针对lui指令。

### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 输入立即数 |
| ExtOp | I | 多路选择控制  00：高位零扩展  01：符号扩展  10：低位零扩展 |
| B | O | 输出不同扩展操作过后的立即数 |

### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 高位零扩展 | 当ExtOp信号为00时，使用16-32的0扩展器对立即数进行位扩展。 |
| 2 | 符号扩展 | 当ExtOp信号为01时，使用16-32的符号扩展器对立即数进行位扩展。 |
| 3 | 低位零扩展 | 当ExtOp信号为01时，使用分线器，16-31位为立即数，0-15为置为常量0 |

## 11. 18020208-IFU

### (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，还能根据BEQ、J指令的执行情况决定顺序取指令还是转移取指令。

### (2)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 |
| clk | I | 时钟信号 |
| reset | I | 复位信号。  1：复位  0：无效 |
| J | I | 控制信号  0：其余指令地址传输  1：J指令[pc[31:28],targetaddress,00]地址传输 |
| insout[31:0] | O | 32位MIPS指令 |

### (3)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0x00003000。 |
| 2 | 取指令 | 根据PC从IM中取出指令。 |
| 3 | 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)  如果当前指令是j指令，PC🡨[PC[31:28],targetaddress,00] |

# 三.机器指令描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令助记符 | 指令操作码 | 指令功能码 | 指令类型 | 指令功能 |
| ori | 001101 | 无 | I | R[rt]←R[rs]|zero\_ext(imm16); |
| addu | 000000 | 100001 | R | R[rd]←R[rs]+R[rt]; |
| subu | 000000 | 100011 | R | R[rd]←R[rs]–R[rt]; |
| lw | 100011 | 无 | I | R[rt]←MEM[R[rs]+sign\_ext(imm16)]; |
| sw | 101011 | 无 | I | MEM[R[rs]+sign\_ext(imm16)]←R[rt]; |
| beq | 000100 | 无 | I | if (R[rs] == R[rt])  then PC←PC+4 + (sign\_ext(imm16) || 00)  else PC←PC+4 |
| lui | 001111 | 无 | I | R[rt]←[imm|16’b0] |
| j | 000010 | 无 | J | pc←[pc+4[31:28]|target address|00] |

# 四.测试程序

lui $t0,255 //高位复制指令，$t0 = 0x00ff0000

lui $t1,1 //高位复制指令，$t1结果为0x00010000

addu $t2,$t0,$t1 //加法运算指令，$t2 = $t0 + $t1，$t2 = 0x01000000

subu $t2,$t2,$t1 //减法运算指令，$t2 = $t2 - $t1，$t2 = 0x00ff0000

ori $t3,$0,25 //或运算指令，$t3 = $0 or 25，$t3 = 0x00000019

ori $at,$0,8 //或运算指令，$at = $0 or 8，$at = 0x00000008

sw $t3,4($at) //存储字指令，Mem[$at + 1] = 0x00000019

lw $t4,4($at) //读取字指令，$t4 = Mem[$at + 1] = 0x00000019

ori $t5,$0,9 //或运算指令，$t5 = $0 or 9，$t5 = 0x00000009

label\_1:ori $t6,$0,5 或运算指令，$t6 = $0 or 5，$t5 = 0x00000005

ori $s0,$0,1 //或运算指令，$s0 = $0 or 1，$t5 = 0x00000001

subu $t5,$t5,$s0 //减法运算指令，$t5 = $t5 - $s0(1)

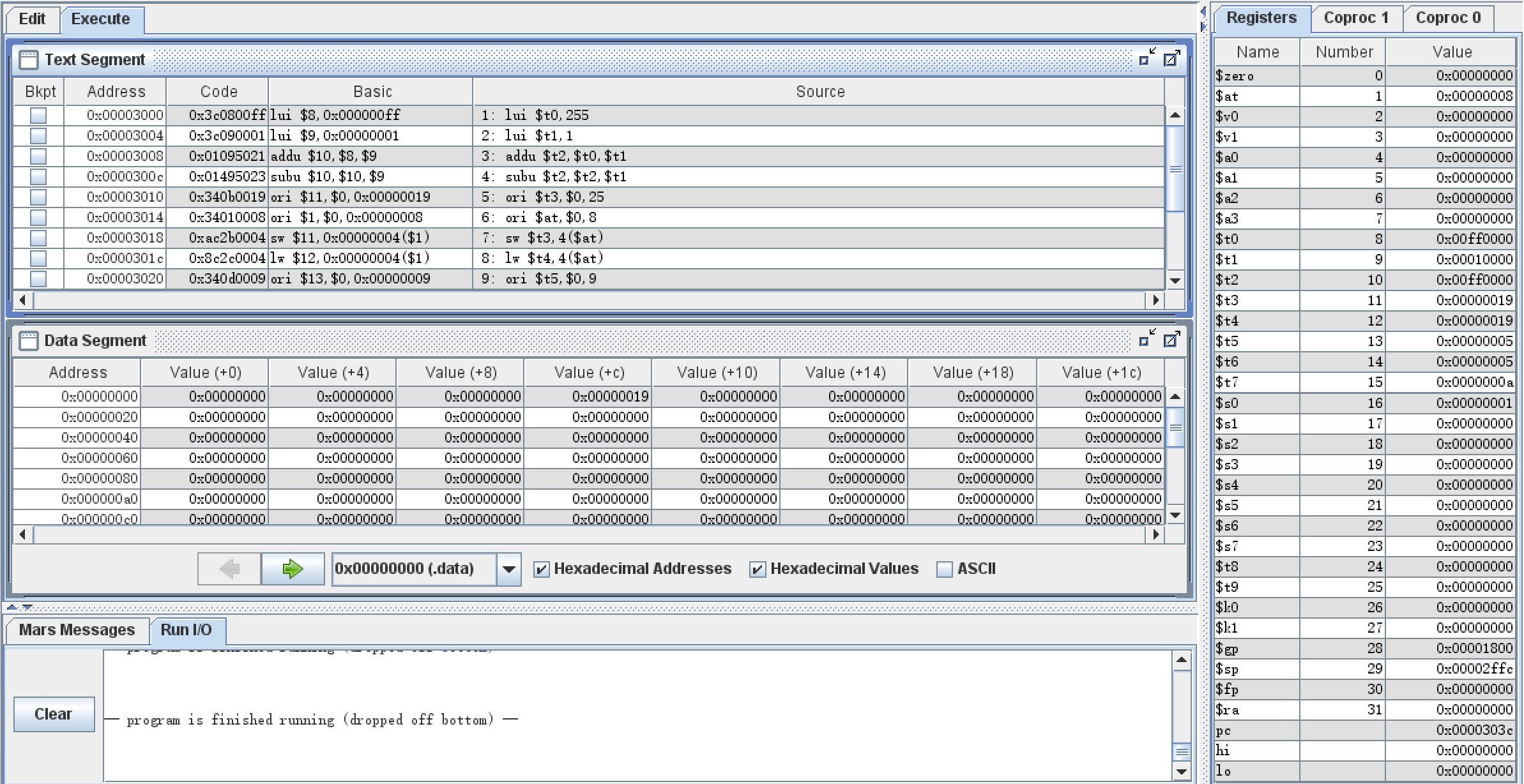
beq $t5,$t6,label\_2 //分支转移指令，若$t5 = $t6，跳转至label\_2

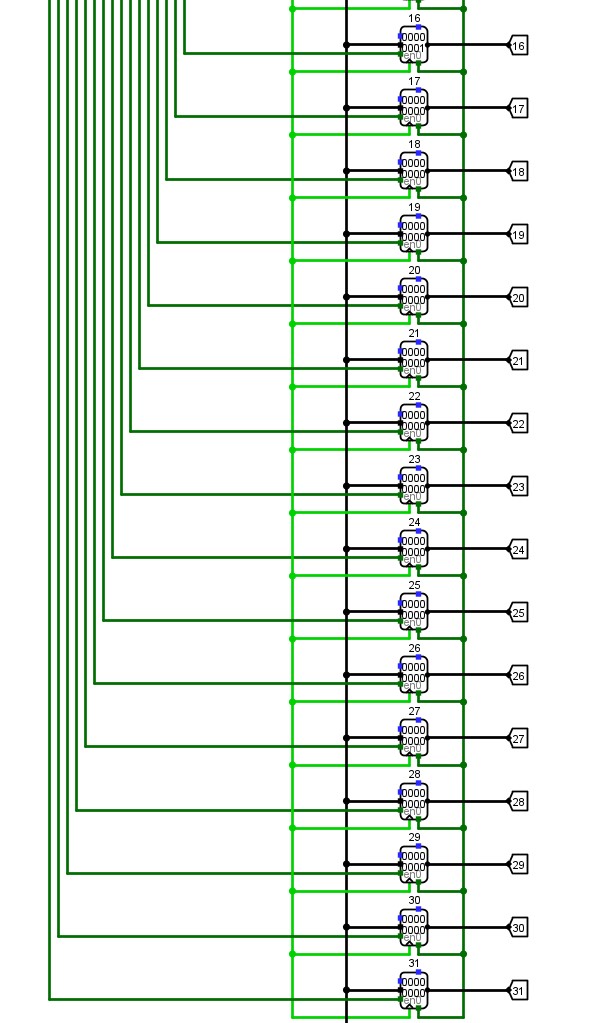
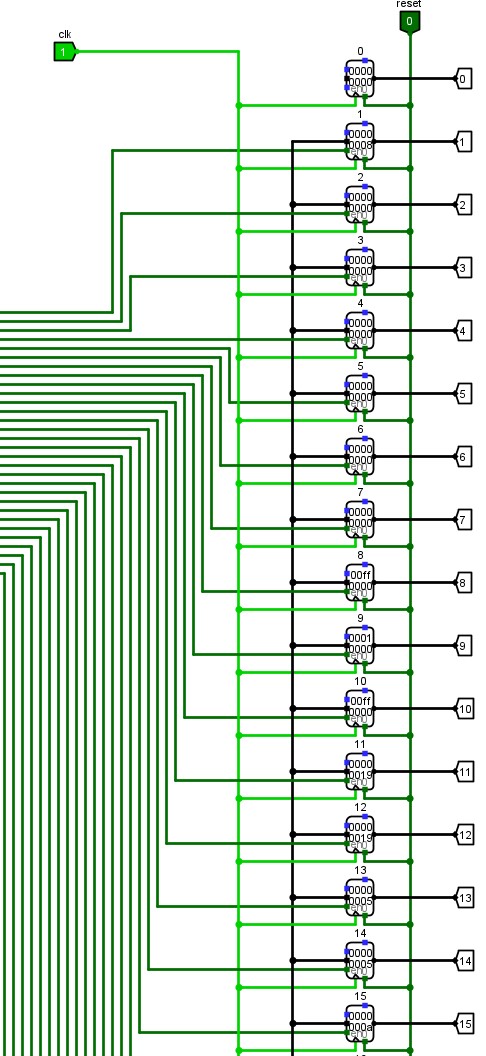
j label\_1 // 跳转指令，pc跳转至label\_1

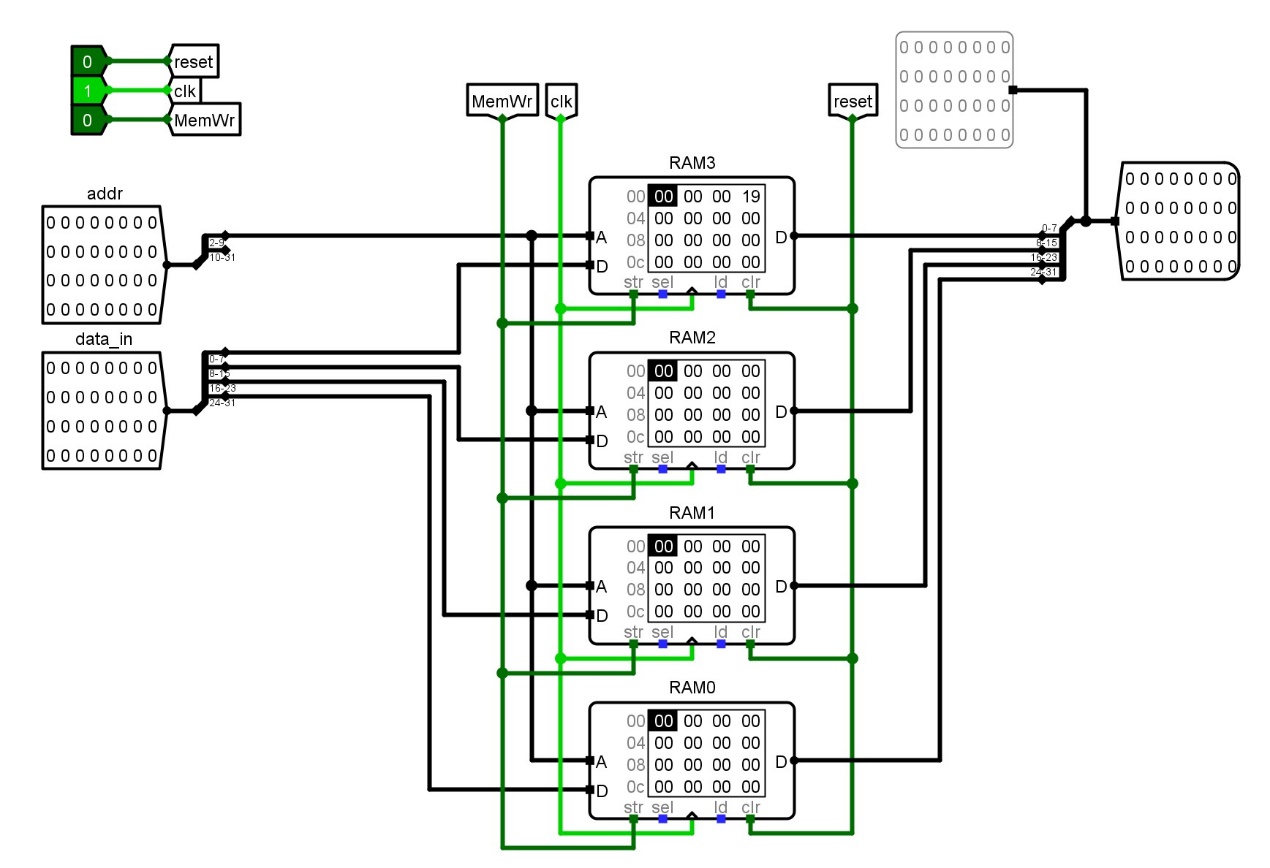
label\_2: addu $t7,$t5,$t6 //加法运算指令，$t7 = $t5 + $t6，$t2 = 0x0000000a

# 五.测试结果

## 1.测试结果截图







## 2.测试结果说明

测试程序如上述所示

检测结果：

1.

lui $t0,255 //高位复制指令，$t0 = 0x00ff0000

lui $t1,1 //高位复制指令，$t1结果为0x00010000

addu $t2,$t0,$t1 //加法运算指令，$t2 = $t0 + $t1，$t2 = 0x01000000

subu $t2,$t2,$t1 //减法运算指令，$t2 = $t2 - $t1，$t2 = 0x00ff0000

检测lui，addu，subu指令，结果在注释中已经标明

2.

ori $t3,$0,25 //或运算指令，$t3 = $0 or 25，$t3 = 0x00000019

ori $at,$0,8 //或运算指令，$at = $0 or 8，$at = 0x00000008

sw $t3,4($at) //存储字指令，Mem[$at + 1] = 0x00000019

lw $t4,4($at) //读取字指令，$t4 = Mem[$at + 1] = 0x00000019

检测ori，sw，lw指令，结果在注释中已经标明

3.

ori $t5,$0,9 //或运算指令，$t5 = $0 or 9，$t5 = 0x00000009

label\_1:ori $t6,$0,5 或运算指令，$t6 = $0 or 5，$t5 = 0x00000005

ori $s0,$0,1 //或运算指令，$s0 = $0 or 1，$t5 = 0x00000001

subu $t5,$t5,$s0 //减法运算指令，$t5 = $t5 - $s0(1)

beq $t5,$t6,label\_2 //分支转移指令，若$t5 = $t6，跳转至label\_2

j label\_1 // 跳转指令，pc跳转至label\_1

label\_2: addu $t7,$t5,$t6 //加法运算指令，$t7 = $t5 + $t6，$t2 = 0x0000000a

检测j，beq指令，本段测试程序解释：$t5赋值为9，$t6赋值为5，$s0赋值为1，每次将$t5中的值减一，并与$t6中数值进行比较，若相等，跳转至label\_2，$t7 = $t5 + $t6，若成功跳转，说明$t5，$t6中的数值相等，均为5，故$t7中数值应为10；若不想等，跳转至label\_1，再进行上述的减1比较步骤。

# 六.心得体会

通过计算机组成原理一学期的课程，我真正比较系统地了解了计算机内部结构，熟悉并学习到了计算机的基本设计理念和各部件协同工作进行运算的原理。我的认知由计算机很“高级”变到计算机实际很“蠢”，但也越来越由衷地敬佩计算机的设计者。相信计算机组成原理会为我之后的学习打下一个良好的基础。同时我十分感谢任课教师朱文军老师，本学期疫情的特殊导致一些课程上的问题没有那么容易得到解决，但朱老师都一一耐心地为我解答，甚至会单独给我开会议讲解，再次感谢老师的不厌其烦。