实验**6**：单周期**CPU**设计与测试

姓名： 学号：

**一、实验目的**

1. 掌握RV32I 控制器的设计方法。

2. 掌握单周期CPU中的时序设计。

3. 掌握RISC-V汇编语言程序的基本设计方法。

4. 理解汇编语言程序与机器语言代码之间的对应关系。

**二、实验环境**

Logisim：https://github.com/Logisim-Ita/Logisim

RISC-V模拟器工具RARS：https://github.com/thethirdone/rars

**三、实验内容**

CPU中控制指令执行的部件是控制器。控制器输入的是指令操作码op和功能码，输出的是控制信号。控制器的主要设计步骤如下。

(1)根据每条指令的功能，分析控制信号的取值，并在表中列出。

(2)根据列出的指令和控制信号之间的关系，写出每个控制信号的逻辑表达式。

(3)实现取指令部件，设计时序信号，接连模块，实现CPU的综合。

**在实现CPU的过程中需要对每一个环节进行详细的测试才能够保证系统整体的可靠性。**

**1. 控制器设计实验**

RV32I 指令集中包含47条基础指令，涵盖了整数运算、存储器访问、控制转移和系统控制几个大类。本次实验中需要实现除了系统控制类的10条指令外的37条指令，分为整数运算指令（21条）、控制转移指令（8条）和存储器访问指令（8条）。在确定具体指令后生成每个指令对应的控制信号，来控制数据通路部件进行对应的动作。控制信号生产部件根据指令代码中的操作码opcode、功能码func3和功能码func7来生成对应的控制信号的。信号表如下：

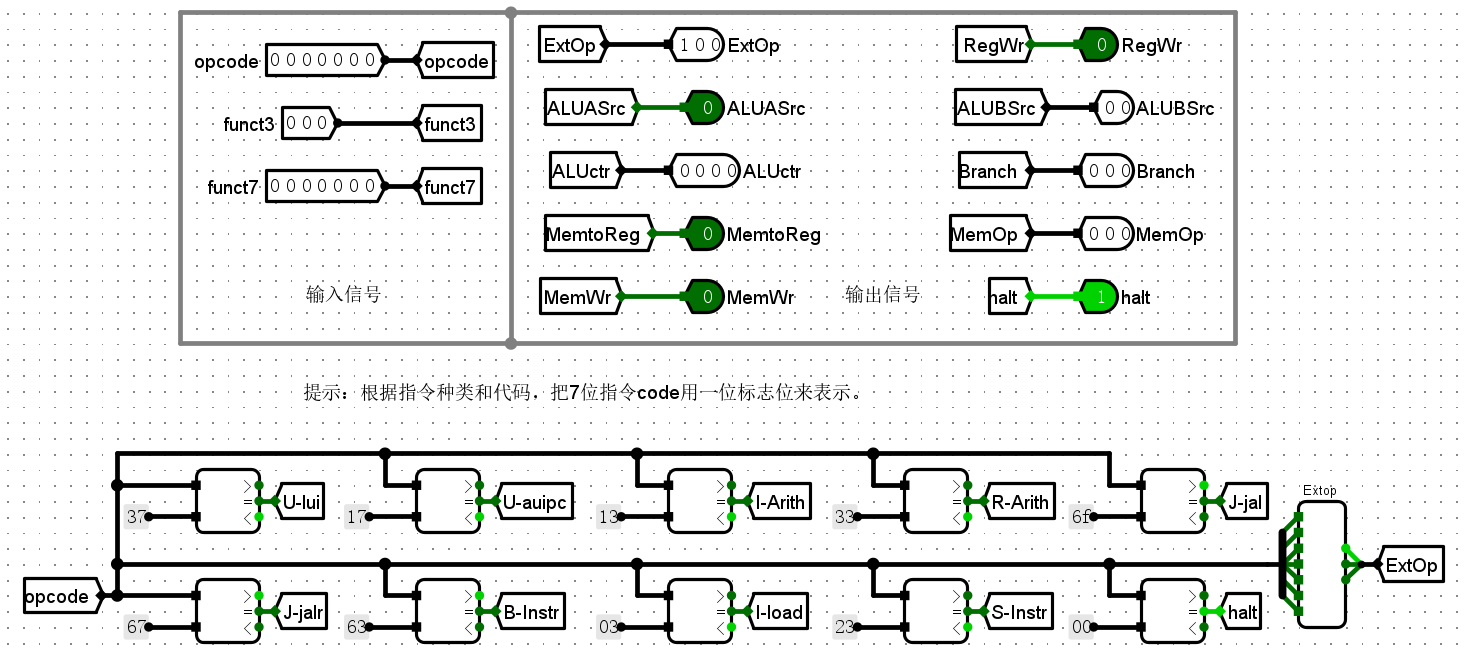
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **类型** | **op[6:0]** | **func3** | **func7[5]** | **ExtOp** | **RegWr** | **ALUASrc** | **ALUBSrc** | **ALUctr** | **Branch** | **MemtoReg** | **MemWr** | **MemOp** |
| **lui** | U | 0110111 | × | × | 001 | 1 | × | 10 | 1111 | 000 | 0 | 0 | × |
| **auipc** | U | 0010111 | × | × | 001 | 1 | 1 | 10 | 0000 | 000 | 0 | 0 | × |
| **addi** | I | 0010011 | 000 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 0 | 0 | × |
| **slti** | I | 0010011 | 010 | × | 000 | 1 | 0 | 10 | 0010 | 000 | 0 | 0 | × |
| **sltiu** | I | 0010011 | 011 | × | 000 | 1 | 0 | 10 | 0011 | 000 | 0 | 0 | × |
| **xori** | I | 0010011 | 100 | × | 000 | 1 | 0 | 10 | 0100 | 000 | 0 | 0 | × |
| **ori** | I | 0010011 | 110 | × | 000 | 1 | 0 | 10 | 0110 | 000 | 0 | 0 | × |
| **andi** | I | 0010011 | 111 | × | 000 | 1 | 0 | 10 | 0111 | 000 | 0 | 0 | × |
| **slli** | I | 0010011 | 001 | 0 | 000 | 1 | 0 | 10 | 0001 | 000 | 0 | 0 | × |
| **srli** | I | 0010011 | 101 | 0 | 000 | 1 | 0 | 10 | 0101 | 000 | 0 | 0 | × |
| **srai** | I | 0010011 | 101 | 1 | 000 | 1 | 0 | 10 | 1101 | 000 | 0 | 0 | × |
| **add** | R | 0110011 | 000 | 0 | × | 1 | 0 | 00 | 0000 | 000 | 0 | 0 | × |
| **sub** | R | 0110011 | 000 | 1 | × | 1 | 0 | 00 | 1000 | 000 | 0 | 0 | × |
| **sll** | R | 0110011 | 001 | 0 | × | 1 | 0 | 00 | 0001 | 000 | 0 | 0 | × |
| **slt** | R | 0110011 | 010 | 0 | × | 1 | 0 | 00 | 0010 | 000 | 0 | 0 | × |
| **sltu** | R | 0110011 | 011 | 0 | × | 1 | 0 | 00 | 0011 | 000 | 0 | 0 | × |
| **xor** | R | 0110011 | 100 | 0 | × | 1 | 0 | 00 | 0100 | 000 | 0 | 0 | × |
| **srl** | R | 0110011 | 101 | 0 | × | 1 | 0 | 00 | 0101 | 000 | 0 | 0 | × |
| **sra** | R | 0110011 | 101 | 1 | × | 1 | 0 | 00 | 1101 | 000 | 0 | 0 | × |
| **or** | R | 0110011 | 110 | 0 | × | 1 | 0 | 00 | 0110 | 000 | 0 | 0 | × |
| **and** | R | 0110011 | 111 | 0 | × | 1 | 0 | 00 | 0111 | 000 | 0 | 0 | × |
| **jal** | J | 1101111 | × | × | 100 | 1 | 1 | 01 | 0000 | 001 | 0 | 0 | × |
| **jalr** | I | 1100111 | 000 | × | 000 | 1 | 1 | 01 | 0000 | 010 | 0 | 0 | × |
| **beq** | B | 1100011 | 000 | × | 011 | 0 | 0 | 00 | 0010 | 100 | × | 0 | × |
| **bne** | B | 1100011 | 001 | × | 011 | 0 | 0 | 00 | 0010 | 101 | × | 0 | × |
| **blt** | B | 1100011 | 100 | × | 011 | 0 | 0 | 00 | 0010 | 110 | × | 0 | × |
| **bge** | B | 1100011 | 101 | × | 011 | 0 | 0 | 00 | 0010 | 111 | × | 0 | × |
| **bltu** | B | 1100011 | 110 | × | 011 | 0 | 0 | 00 | 0011 | 110 | × | 0 | × |
| **bgeu** | B | 1100011 | 111 | × | 011 | 0 | 0 | 00 | 0011 | 111 | × | 0 | × |
| **lb** | I | 0000011 | 000 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 1 | 0 | 101 |
| **lh** | I | 0000011 | 001 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 1 | 0 | 110 |
| **lw** | I | 0000011 | 010 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 1 | 0 | 000 |
| **lbu** | I | 0000011 | 100 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 1 | 0 | 001 |
| **lhu** | I | 0000011 | 101 | × | 000 | 1 | 0 | 10 | 0000 | 000 | 1 | 0 | 010 |
| **sb** | S | 0100011 | 000 | × | 010 | 0 | 0 | 10 | 0000 | 000 | × | 1 | 101 |
| **sh** | S | 0100011 | 001 | × | 010 | 0 | 0 | 10 | 0000 | 000 | × | 1 | 110 |
| **sw** | S | 0100011 | 010 | × | 010 | 0 | 0 | 10 | 0000 | 000 | × | 1 | 000 |

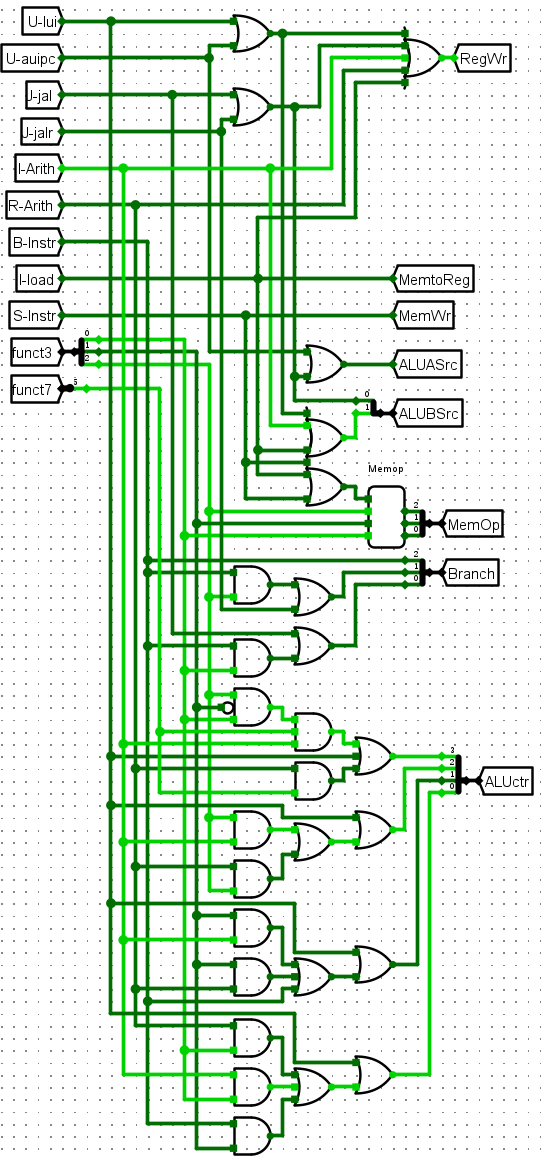
根据上表的定义，列出每个控制信号的逻辑表达式

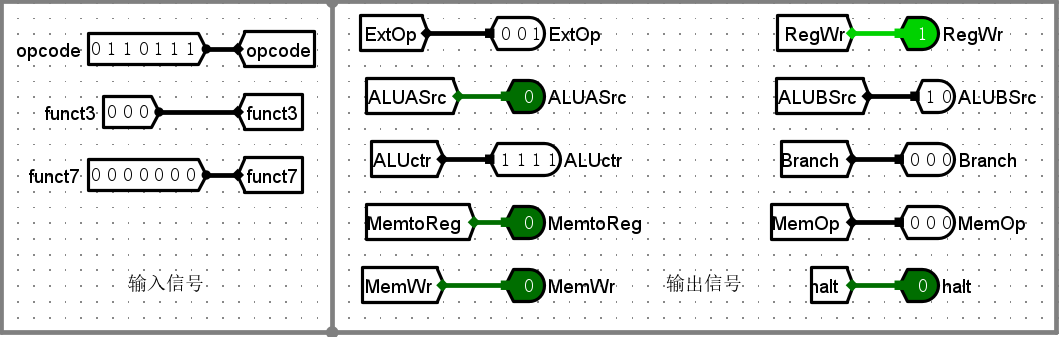
**实验步骤如下：**

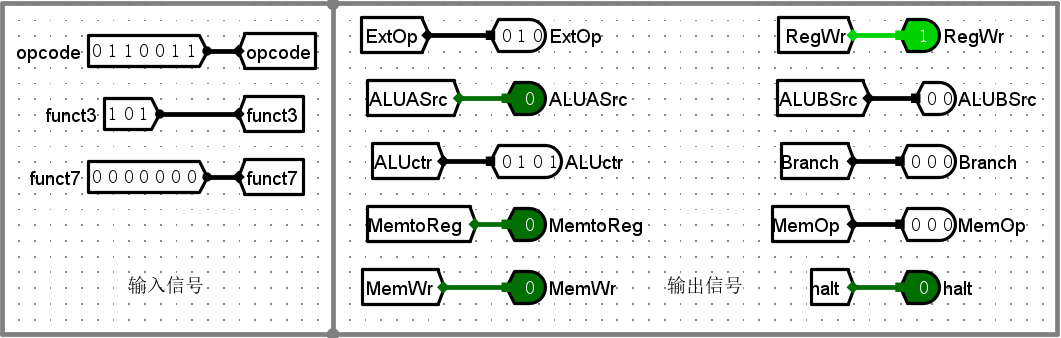
1） 创建子电路。在Logisim中添加一个名为“控制器”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。

2） 设计子电路。在工作区中添加所需的逻辑门、输入/输出引脚。根据逻辑表达式进行线路连接，添加标识符和电路功能描述信息，得到完整的控制器电路。为了方便程序的结束执行，定义了一个输出信号halt，当指令操作码opcode=0000000时，赋值为1，反相输出到PC寄存器的使能端，中止程序的执行。可以根据不同类型指令操作码进行比较输出1位标志位。设计电路如下：

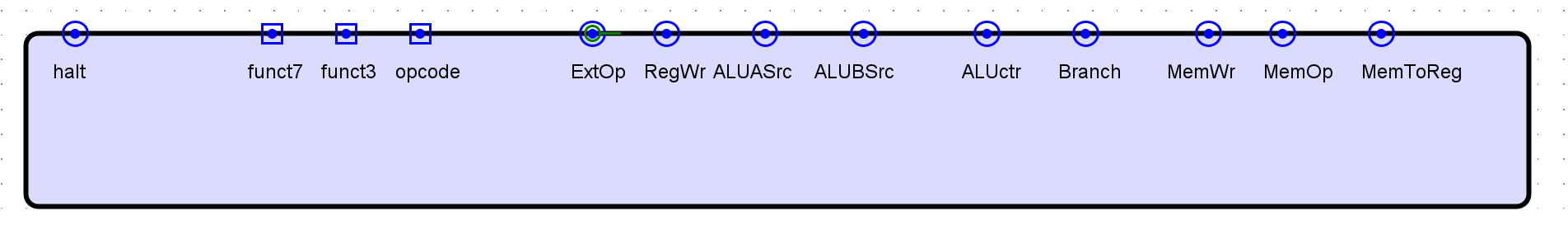




3）仿真测试输入与输出是否正确对应。为从简，列出部分信号 



4）封装电路如下：



**2.单周期CPU设计实验**

在单周期CPU 中，每条的指令都需要在一个时钟周期内完成。本次实验中，以时钟下降沿为每个时钟周期的开始，写入操作在时钟下降沿时同步实现；读取操作异步实现，只要输入有效地址后，立即输出对应数据。

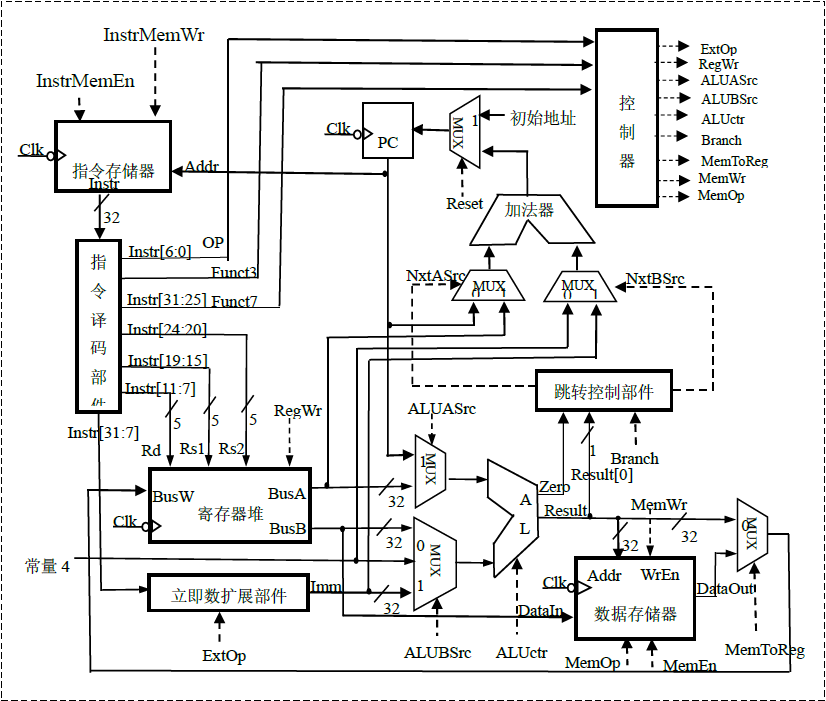
单周期CPU需要考虑的时序信号，有复位信号Reset、片选信号Sel、中止信号halt等。

复位信号Reset，用来初始化CPU中各个部件的控制信号，设置PC寄存器的起始址。

指令存储器和数据存储器的片选信号Sel，高电平有效。只要CPU的复位信号撤销，该信号设置恒为1。数据存储器的清零信号可与复位信号相连。写使能信号MemWr，高电平有效，在指令执行过程中赋值。

程序执行结束后，中止信号halt有效，PC寄存器使能端无效，停止输出。

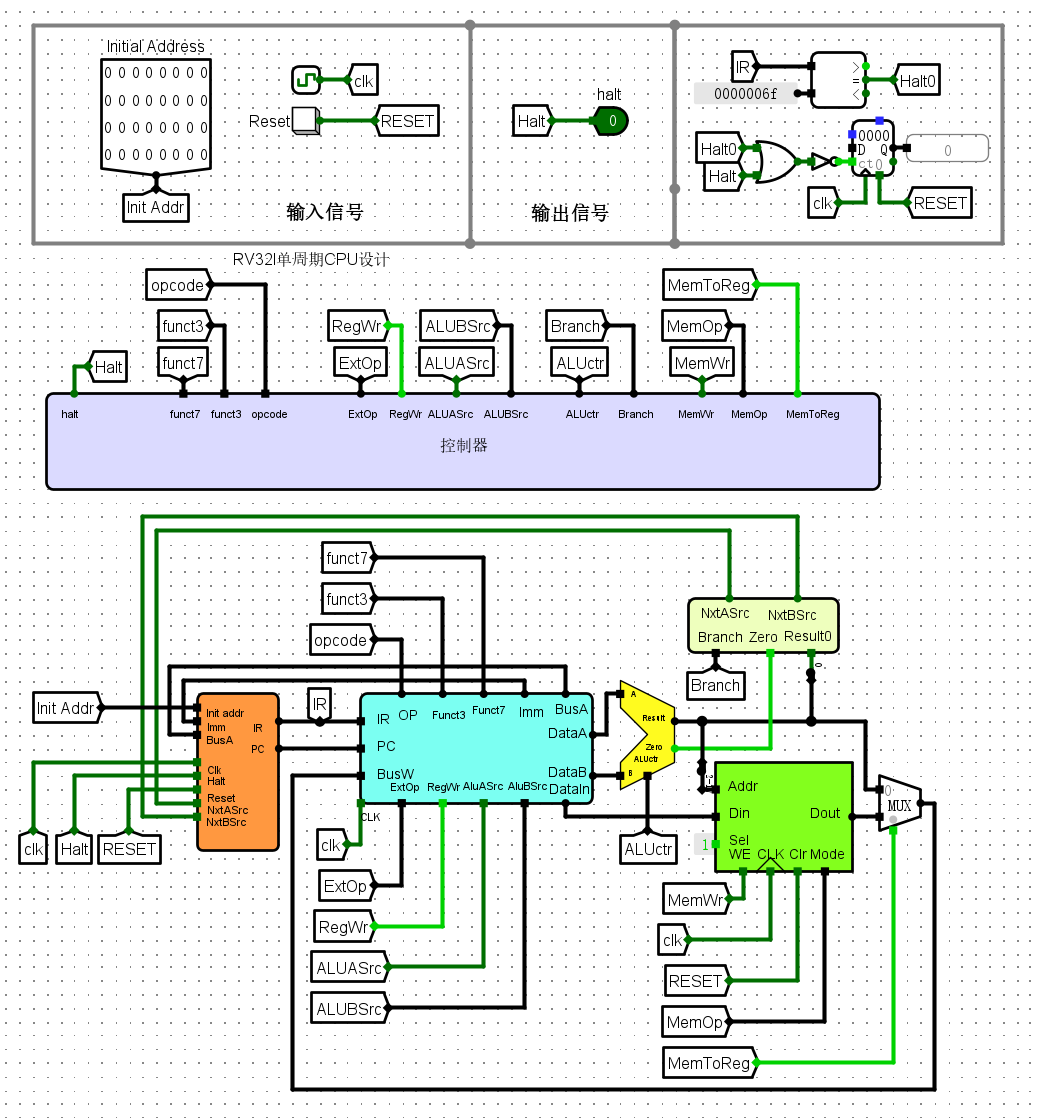
CPU由不同模块构成，分工协作完成功能。在前期的实验中已经完成全部模块的设计。lab3：寄存器堆模块；lab4：ALU模块；lab5：数据存储器、取指令部件和数据通路模块。本次实验完成了控制器模块。

依据如下原理图，综合利用这上述模块，设计单周期CPU。

**实验步骤如下：**

1）创建子电路。在Logisim中添加一个名为“单周期CPU”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。

2）设计子电路。在logisim的project菜单下添加logisim库文件，把实验3、实验4和实验5的设计文件添加到lab6的项目中。（为方便修改调试，本人在lab6中复写了部分元件） 在工作区中放置取指令部件IFU、指令存储器ROM、数据通路IDU、ALU、数据存储器模块和控制器部件，以及输入输出引脚和隧道，布局引脚。根据模块中定义信号进行连接。设置指令存储器和数据存储器的属性，片选信号都定义为高电平有效。指令存储器的地址位宽设置为16位，数据位宽设置为32位。实验电路设计如下：



3）仿真测试：加载lab5中的测试文件并测试结果。为从简，报告中略去此部分，由接下来的部分验证CPU功能。

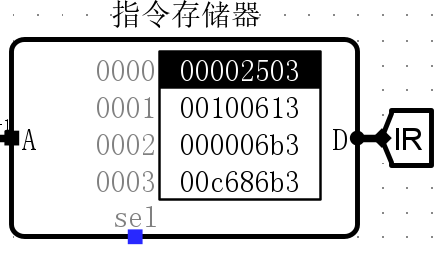
**3.用累加和程序验证CPU设计**

（本报告略过机器码编译过程，直接使用编译好的机器码文件验证CPU功能）

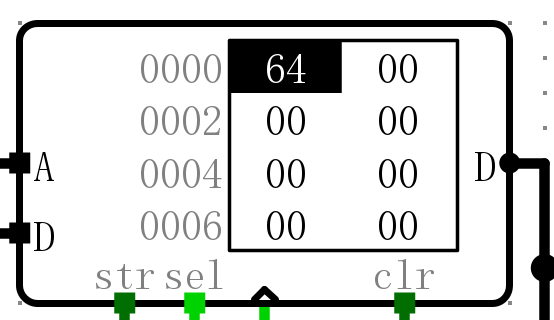
在单周期CPU的指令存储器（ROM）中装载计算累加和可执行机器代码镜像文件sum.hex，在数据存储器地址0单元中设置初始参数，然后启动时钟信号，执行程序，观察输出结果，验证CPU设计的正确性。

实验步骤如下：

1）在Logisim中，打开单周期CPU电路图，用鼠标右键点击指令存储器（ROM组件），选择装载镜像（load image）命令，将镜像文件sum.hex加载到指令存储器起始地址0单元中。



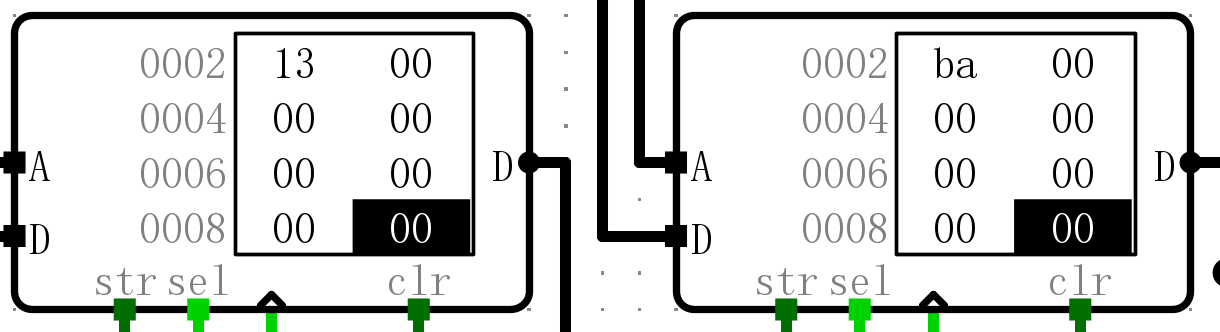
2）在CPU电路图中选中数据存储器组件，进入子电路，在最低字节的RAM组件地址0x0000处，用16进制编辑器写入参数0x64。



3）返回CPU设计主电路窗口。在Logisim的仿真菜单下，选择时钟频率1kHz，然后选中使用“时钟连续”快捷键{Ctrl+K}，CPU开始自动执行机器代码。

当程序始终执行最后一行（第8行）指令时，说明程序执行已经结束，可按Ctrl+K取消“时钟连续”，以暂停/中止程序执行。

程序结束后，进入数据储存器查看结果：

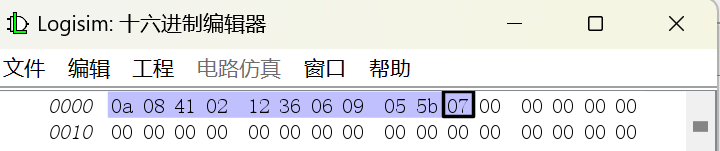
0x13ba(5050) 确是1至100累加和

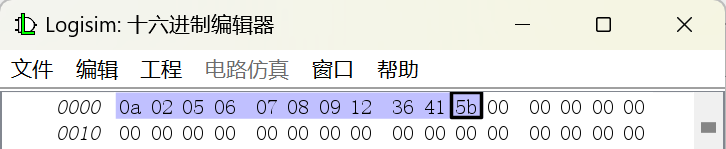
**4.排序程序测试**

本报告综合了小、大两次冒泡排序和快速排序三个排序程序，均在此展示。

**1）（小）冒泡排序：**

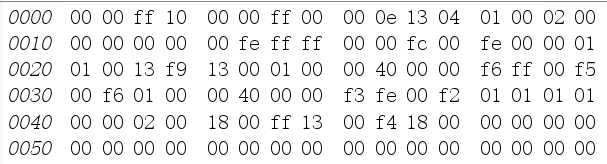
加载指令

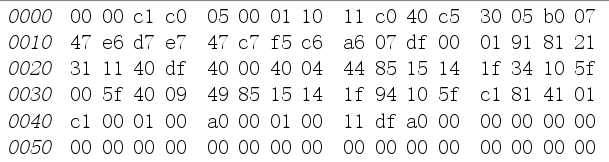
加载数据

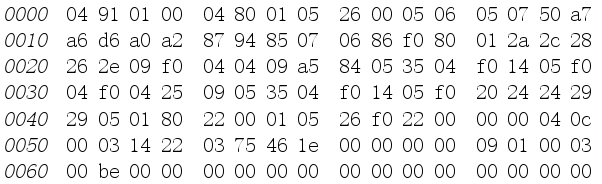
排序结果

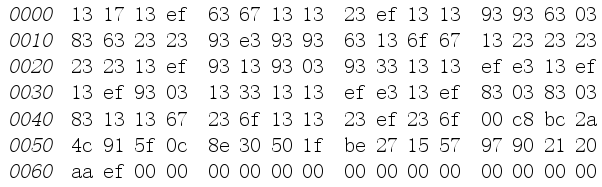
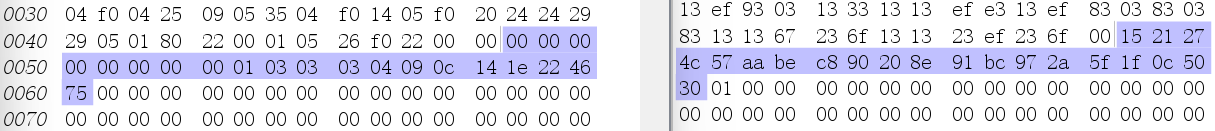
**2）（大）冒泡排序：**

加载指令

加载数据（高2位）

加载数据（次高2位）

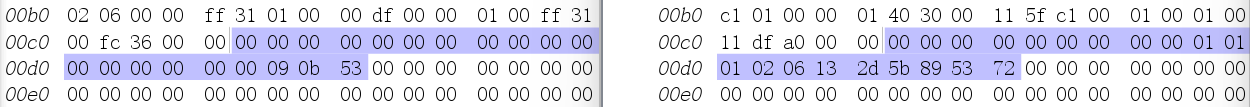
加载数据（次低2位）

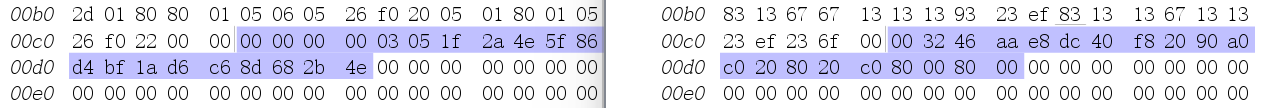
加载数据（低2位） 

排序结果（高4位全0，故不予展示）

**3）快速排序**

为从简，仅展示排序结果如下：

 高4位结果



低4位结果

（可以看到，高2，次高2，次低2和低2均按从小到大顺序排列，也就是排序数据全部按从小到大顺序排序，排序结果正确）

**四、思考题**

**1.在累加和计算程序中，添加溢出判读语句，并把最大的不溢出累计和以及累加序数保存到数据存储器中输出。**

#计算累加和 RV32I 汇编程序（红色为添加的语句）  
 main:  
 lw a0,0(x0) # 从数据存储器地址 0x0000 单元中读取参数 n 到寄存器 a0；  
 addi a2, x0,1 # 循环变量 i，存放在 a2，初值为 1  
 add a3,x0,x0 # 累计和存放在 a3，初值为 0  
 loop:  
 add a3, a3, a2 # 将 a3=a3+i

blt a3, a2, overflow #若累加和变小，发生溢出  
 beq a2, a0, finish # 若 i=n，则跳出循环  
 addi a2, a2, 1 # i++  
 jal x0, loop # 无条件跳转到 loop 执行

overflow：

sub a3,a3,a2 #减法还原溢出前结果

sub a2,a2,1 #减法还原溢出前循环变量（累加序数）  
 finish:  
 sw a3, 8(x0) # 将累加结果保存到数据存储器 0x0008 单元

sw a2 12(x0) # 将累加序数保存到数据存储器 0x000c 单元  
 end:  
 jal x0, end # 无条件跳转到 end

**2.如果分支跳转指令不在ALU内部使用减法运算来实现，而是在ALU外使用独立比较器来实现，说说单周期CPU的电路原理图中需要做哪些修改？**

添加额外的比较器，比较器接收busA和busB两个数据，输出0或1，输出替换原本的ALU zero送入branch部件。

**3.实现单周期CPU后，如何实现键盘输入、TTY输出部件等输入输出设备的数据访问，构建完整的计算机系统。**

通过存储器(缓冲区)建立CPU和输入输出部件之间的联系，在CPU中设置MAR（存储器地址寄存器）和MDR（存储器数据寄存器）。

例如，通过键盘输入时，键入的数据会转换为二进制暂存到键盘缓冲区，等待CPU从缓冲区中存取数据。

**4.如果需要实现5级流水线RV32I CPU，则如何在单周期CPU基础上进行修改？**

* 1. 将CPU的各部件按阶段划分，分别为，取指令段（IF），指令译码（ID），执行（EX）， 访存（M），写回（WB），在五个阶段之间添加流水段寄存器，用于存放当前流水段传输到后面所有流水段的信息。
  2. 重新设计控制信号，ID段的控制信号有ExtOp，EX段有ALUASrc，ALUBSrc， ALUctr，M段有MenWr， Branch， Jump，WB段有MemtoReg， RegWr 。
  3. 处理冒险，为转发和动态预测等冒险处理机制添加相应的电路。