**1 基础实验：原型机**

**实验1.1 原型机I**

【实验目的】

1. 了解冯诺伊曼体系结构；
2. 理解指令集结构及其作用；
3. 理解计算机的运行过程，就是指令的执行过程，并初步掌握调试方法。

【实验准备】

1. 阅读教材，掌握冯诺伊曼体系的相关内容；
2. 学习课程《最小系统与原型机I》。

【实验步骤】

1. 进入终端，使用cd hnuvm进入code目录，有两个目录，32bit和64bit，分别对应Ubuntu 32位（Intel CPU），Ubuntu 64位（Intel CPU），请进入相应目录；
2. 使用.**/vm32 1.config（或./vm64 1.config)**来运行原型机I的模拟器，其中**1.config**文件为配置文件，一共有四行，其格式如下：

4

3

0011

a.txt

第一行的4，表示原型机的地址为4位，即内存有16个字节

第二行的3，表示数据段为3个字节——0000，0001，0010

第三行的0011，表示从第四个字节，也就是0011开始存放指令

第四行的a.txt，表示指令内容在a.txt中

其中的a.txt中，指令的内容为：

1

5 R0 0000

4 1 R2

2 R0 R1

3 R2 R0

6 -2

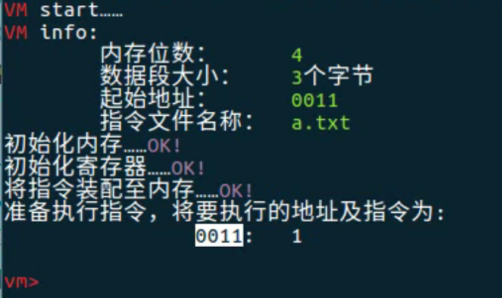
5 R1 0001

8 R1

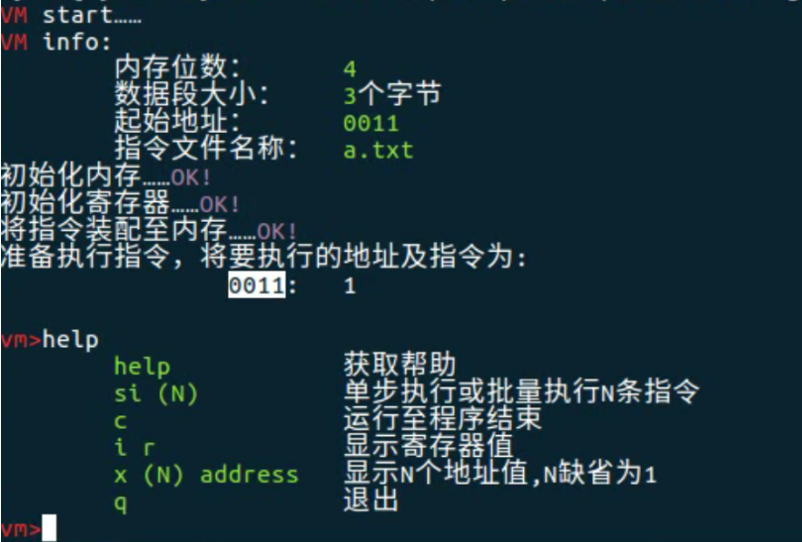
0

每一行指令代表的意义及整体执行结果在《最小系统与原型机I》中已经进行了详细说明。另请注意：**编写指令时，请不要增加空格（例如指令结尾处），否则代码执行时有可能出错。**

运行后界面如下图所示。



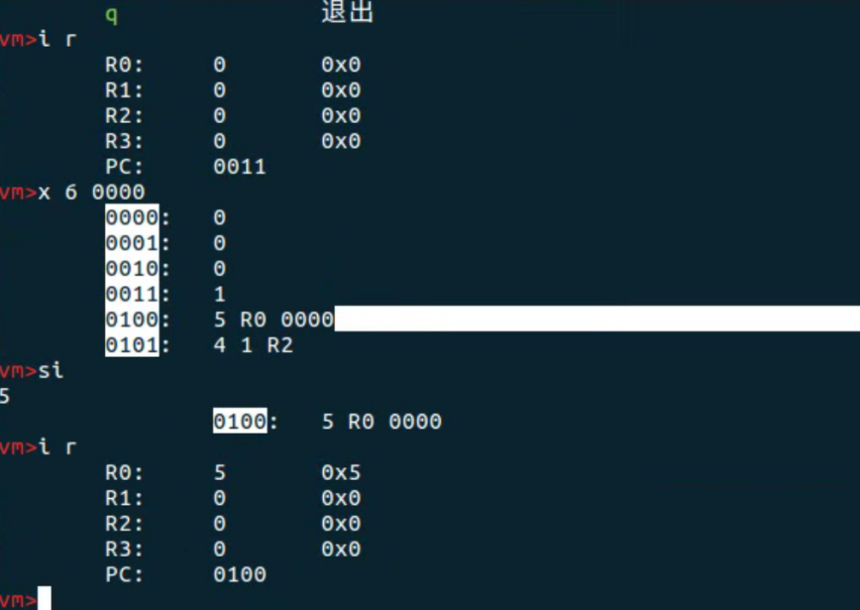
1. 在运行后，提示将要执行的指令地址及内容，在本例中，提示要执行位于内存**0011**处的指令“**1**”，即等待输入一个整数值。此时输入**si**则表示执行此指令，同时也可以输入其他的指令，使用help可以查看此模拟器支持的命令：



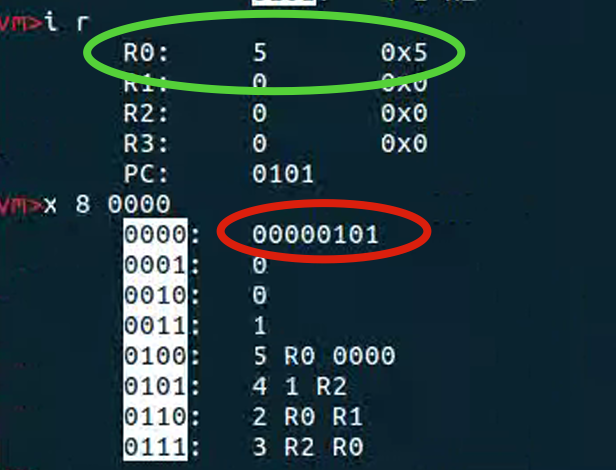
1. 此时输入**i r**，可以查看各个寄存器的值，而输入**x 6 0000**则表示查看从**0000**开始的连续6个内存地址值，结果如图所示。



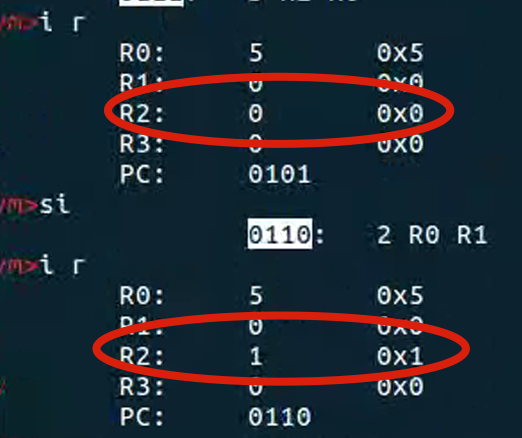
1. 输入**si**，则表示运行一条指令，例如此时运行的指令是“**1**”，表示等待输入，输入一个值5，在输入完成后将此数值存至**R0**寄存器，运行完成后，再运行**i r**指令，就可以看到输入的值5确实是已经存在**R0**寄存器中，每个寄存器的值都用十进制和十六进制表示，如下图所示。



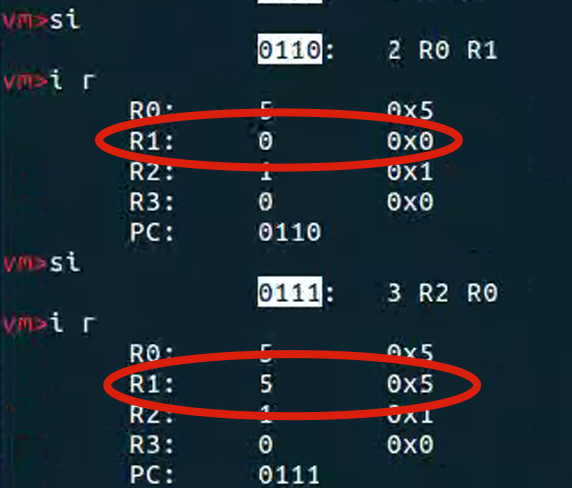
1. 继续使用**si**指令单步执行指令 5 R0 0000， 表示将R0寄存器中的值5传送到内存地址0000处，如下图红色圆圈处以二进制显示了值5。



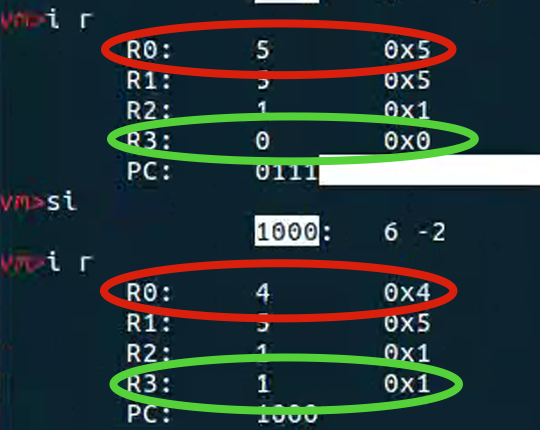
1. 继续si执行下一条指令 4 1 R2，表示将立即数1传送到寄存器R2中，结果如下图所示，注意观察寄存器R2值的变化情况。



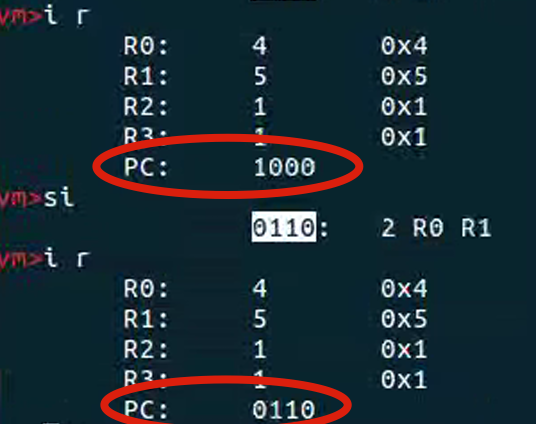
1. 继续si执行下一条指令 2 R0 R1，表示将R0与R1的值相加，结果放于R1中，注意观察寄存器R1值的变化情况。



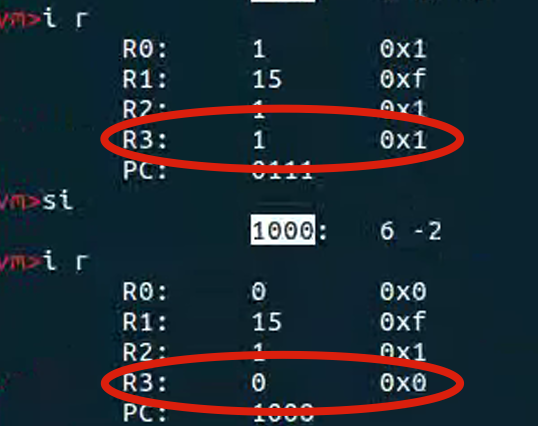
1. 继续si执行下一条指令 3 R2 R0，表示将R0的值减去R2的值，结果放于R0中，注意观察寄存器R0值与R3值的变化情况。



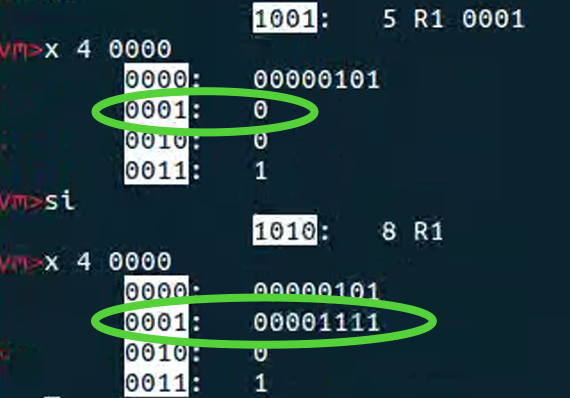
1. 继续si执行下一条指令 6 -2，即有条件跳转，如果R3的值为1，则需要向前或向后跳转，此时跳转的值为-2，则表示需要向前跳转两条指令，再去执行2 R0 R1指令。从整体上来看，我们的任务是要执行累加和5+4+3+2+1，截止到目前，我们已经完成了5+4+3+2+1中的第一步计算，并保存结果到了R1中，现在准备去执行+4这一操作，执行完6 -2这一操作后，PC值发生变化，在下图中观察到了PC值的改变。



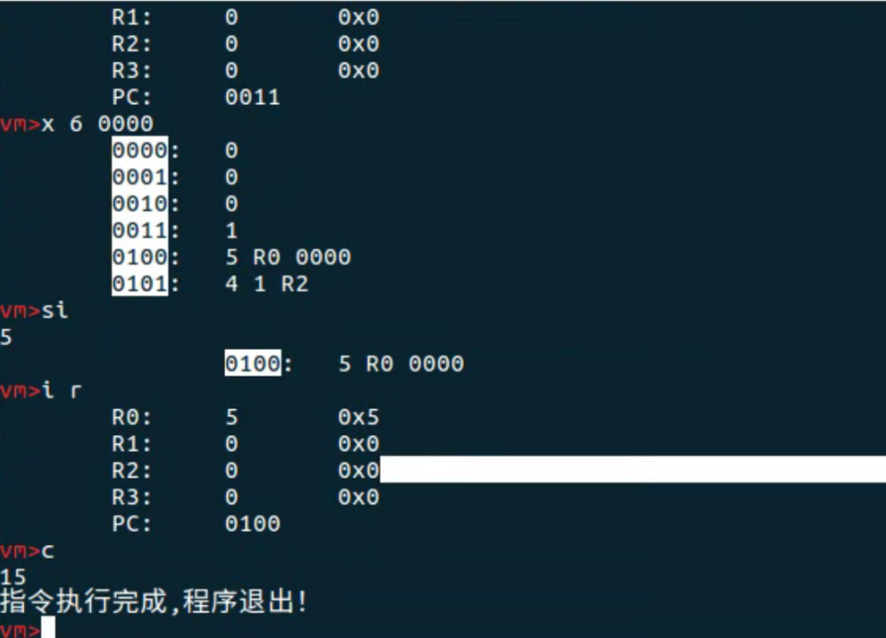
1. 在计算完5+4+3+2+1后，这时候R0的值也变成了1，执行 3 R2 R0时，R0的值变为0，而R3的值也变成了0，表示刚才相减的结果为0。



1. 这时候执行下一条指令 6 -2时，由于R3的值为0，不满足跳转的条件，因此会执行下一条语句5 R1 0001，表示将寄存器的值保存到内存中，以便于其他的代码调用。



1. 继续si或者使用si 4等来批量执行指令，或者是使用**c**来执行指令直到程序结束。由于最后我们使用了8 R1来输出最后结果，因此会打印出15这一结果。



1. 程序执行完毕后，可以使用q退出。

## 三、练习思考

### 1.练习内容

1. 按照上述的实验步骤，完成相关操作；
2. 在目录下还有**2.config，3.config**，其对应的指令代码文件分别为**b.txt**和**c.txt**，请运行并调试，并对这些代码所做的工作进行解释；
3. 完成实验报告。

### 2.思考问题

（1) 如果基于这些指令实现两个整数的乘法与除法？

（2） 原型机I的指令集是否完备？如果是，那么如何证明（提示：搜索并阅读“可计算性理论”）？如果不是，那么要增加哪些指令？

**实验1.2 原型机II-扩充指令集**

【实验目的】

1. 理解指令集结构及其作用；
2. 理解计算机的运行过程，对指令集进行修改；

【实验准备】

1. 阅读教材，掌握冯诺伊曼体系的相关内容；
2. 学习《最小系统与原型机I》内容，完成实验1.1

【实验步骤】

1. 进入相应目录；
2. 使用nano cpu.c或gedit cpu.c来打开cpu.c文件；



1. cpu.c是原型机的核心模块，其中

1)前面的变量申明，例如machine\_info,memory,R0,R1,R2,R3等均为原型机的硬件配置信息，因为其在另外一个文件中已经定义，因此采用extern方式引入；

2)然后是各种操作的具体执行过程，例如对于加法操作，其参数为

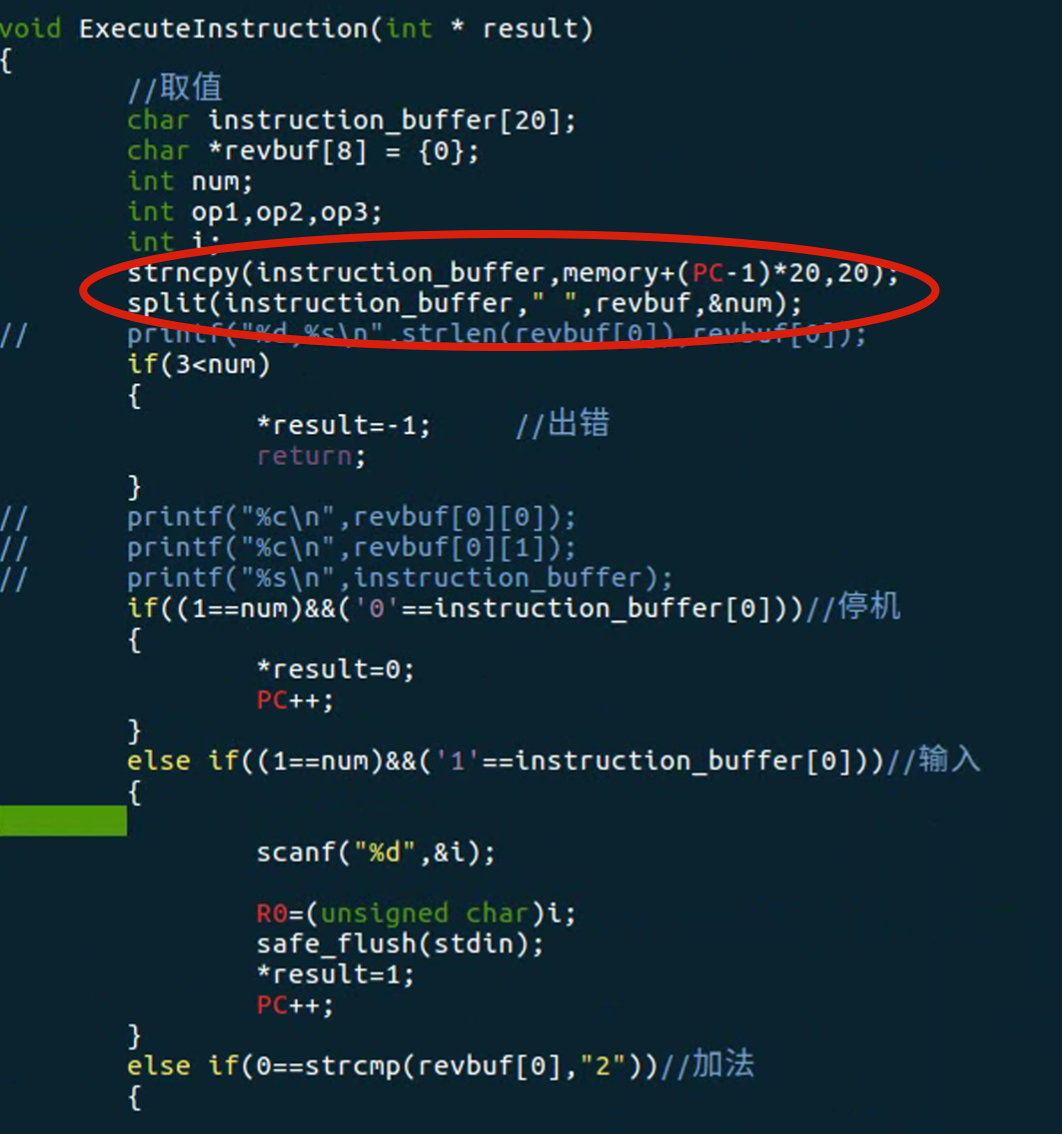
源寄存器：source[]

目的寄存器:dest[]

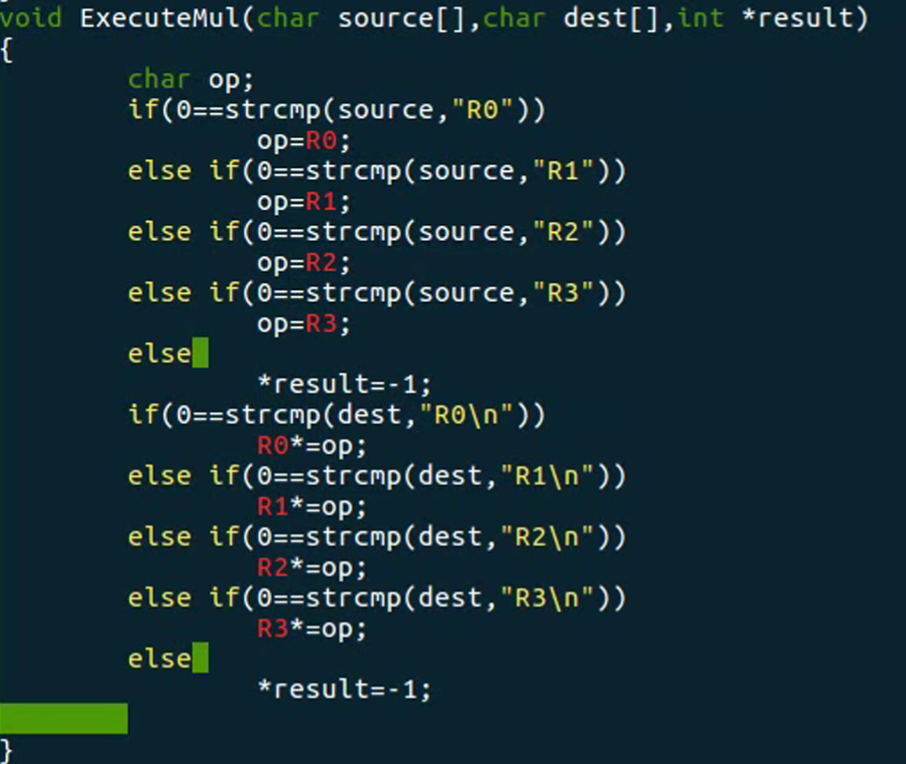
操作标志：\*result（-1表示出错）



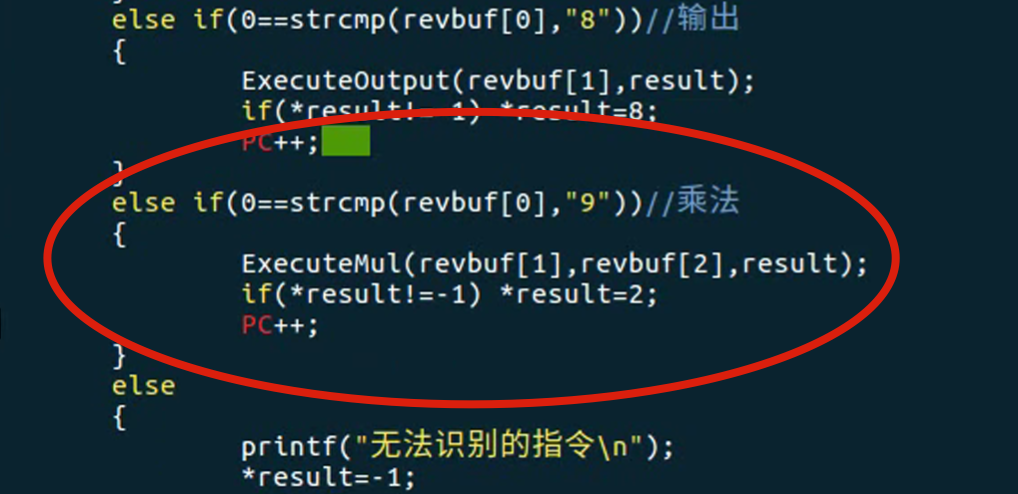
3)最后的一个函数ExecuteInstruction即为CPU的指令执行过程，其中红圈处的strncpy即表示从内存中取出一条指令（每条指令为字符串型，占20个字节），然后用split函数来对指令进行分割，再根据分割后的结果进行判断指令类型，调用前面的各种函数来进行操作。



1. 在原型机I的基础上，我们对指令集进行扩充，增加一条乘法指令，其格式为 9 Ra Rb，即将寄存器Ra的值与寄存器Rb的值相乘，结果放在Rb寄存器中，因此需要增加一个ExecuteMul函数。



1. 在ExecuteInstruction增加一个判断分支，从而能够识别此条指令。



1. 增加一个d.txt文件，其中包括有乘法指令

1

5 R0 R1

1

9 R0 R1

8 R1

0

1. 修改1.config

4

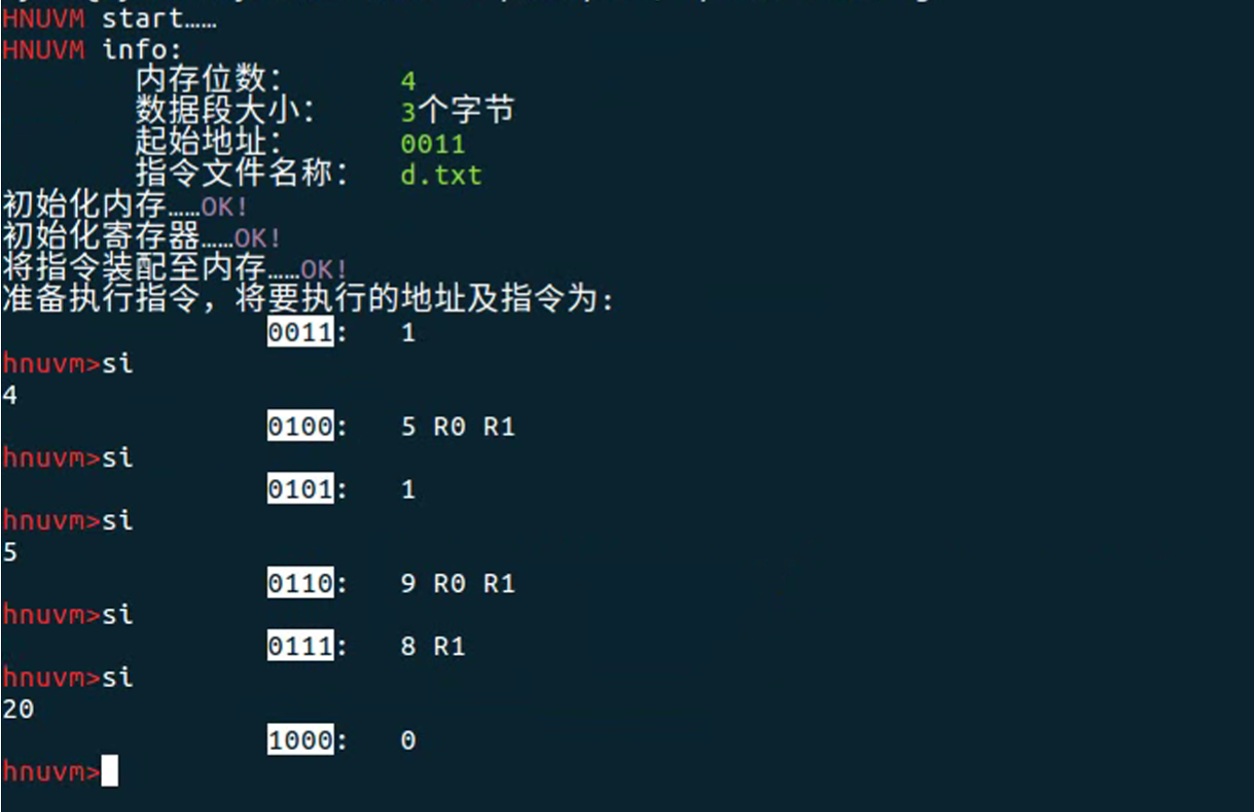
3

0011

d.txt

1. 输入make 生成可执行文件

然后使用 ./vm32 1.config（或./vm64 1.config)来运行程序。



1. 增加一个e.txt文件，基于原型机I的指令完成了两个数的乘法操作，其基本思路是将乘法分解为加法，例如对于5\*6，执行6次加5的操作：5\*6=5+5+5+5+5+5

1

5 R0 0000 //输入乘数，并保存在内存地址0000中

1 //输入另一个乘数，保存在R0中

4 1 R2 //将R2寄存器赋值为1

5 0000 R3 //从地址0000中取出乘数的值5

2 R3，R1 //将R1的值与R3的值相加，结果保存在R1中

3 R2,R0 //被乘数减1

6 -3 //如果被乘数不为0，则结果还需要再加一次乘数

8 R1 //被乘数已经为0，此时R1中即为两数相乘的结果，输出此结果

0 //停机

1. 在原型机II上执行上述代码。

## 三、练习思考

### 1.练习内容

1. 按照上述的实验步骤，完成相关操作；
2. 为原型机II增加整除指令，并基于原型机I的指令写出两数整除的代码，进行对比；
3. 完成实验报告。

### 2.思考问题

（1) 原型机I与原型机II完成乘法和除法操作的方式有何不同？

（2） 在指令集中增加乘法、除法等指令时，原型机中需要增加代码，那么硬件实现上需要增加什么样的部件？

（3） 如果一台计算机只支持加法、减法操作，那么能否计算三角函数，对数函数？（提示：搜索并阅读“泰勒级数展开”等内容）

（4）对于某个需要完成的功能，如果既可以通过硬件上增加电路来实现，也可以通过其他已有指令的组合来实现，那么如何判断哪一种比较合适？（提示：搜索并阅读RISC与CISC）。