

华中科技大学计算机科学与技术学院

《C 语言程序设计》课程设计实验报告

题目： 模拟器和汇编程序设计

专业： 计算机科学与技术

班级： ACM12 级

学号： U201214946

姓名： 赵子昂

成绩：

指导教师： 卢萍副教授

完成日期： 2013 年 2 月 27 日

目录

一、系统功能结构设计	3
二、数据结构设计	4
三、程序结构	6
四、各函数原型及功能	8
五、实验结果	14
六、实验体会	20
七、参考文献	23
八、程序清单	23

一、系统功能结构设计

该课程设计主要分为两部分。

第一部分为汇编程序。程序的主要设计思路主要是对汇编语言信息的存储。由于指令集格式的规范性，已给出的 32 条指令均可用相应的四字节十六进制数来表示。相应较为复杂的伪指令则另外用链表储存其相关信息。此外，对于跳转标号的处理也用链表进行储存。所以，该部分大体设计思路如下，先扫描一遍指令，处理伪代码和标号的信息并储存。第二遍扫描处理普通指令并储存。在译码中变量信息的储存和普通译码的储存顺序可以互换。

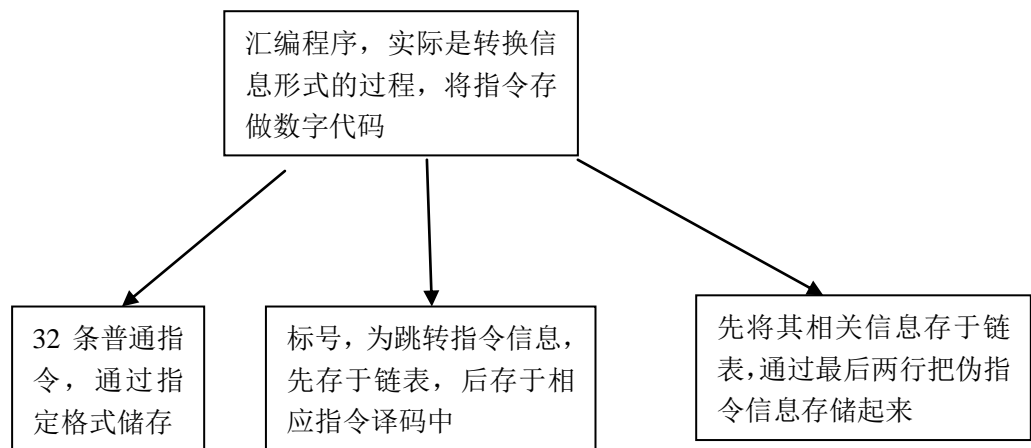


图 1. 汇编程序模块结构图

第二部分为模拟器设计。模拟器设计主要分为如下几个部分：先向系统申请空间模拟内存，然后配置好 CS, DS, ES, SS 几个段的 address，之后可以照课设书上的例子运用好几个寄存器写好 32 个函数即可。这一部分需要注意的重点是代码段、数据段、堆栈段，附加段等的配置，还有 32 个函数的可行性。

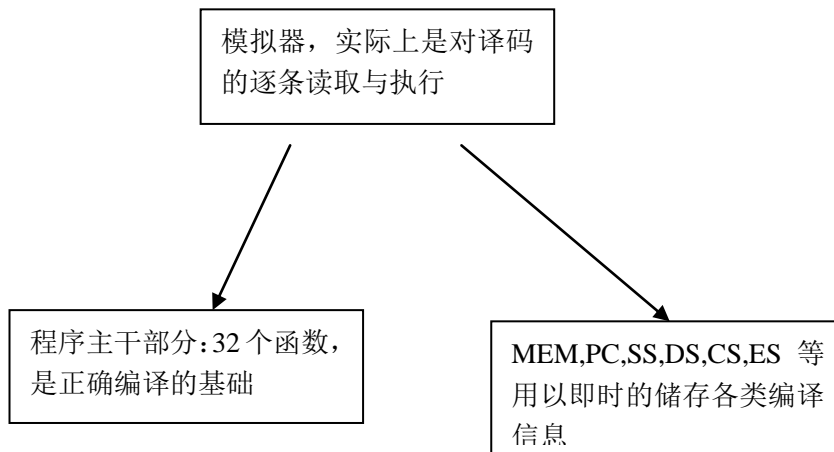


图 2.模拟器模块结构图

二、数据结构设计

数据结构有如下几种：

- 1、**哈希表**：此表主要记录指令集和对应的操作码，便于第二遍读取时操作码的快速查找。

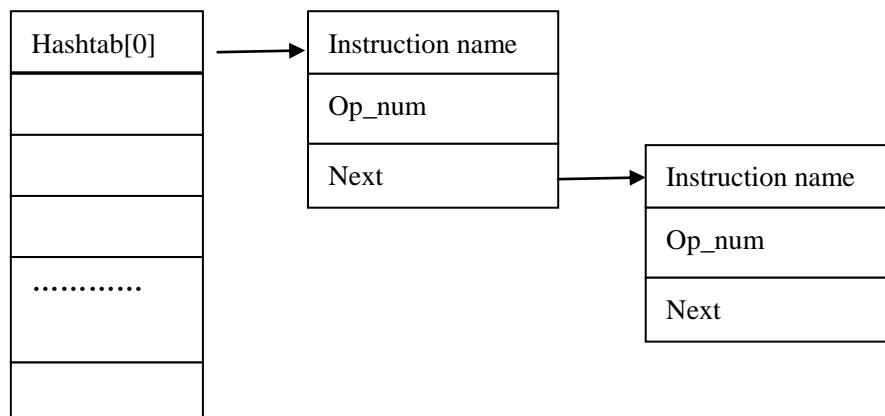


图 3.哈希表数据结构示意图

其中 **next** 为指向下一指针域的指针，**name** 为指令名，**op_num** 为操作码。

2、链表：(存储标号和变量信息的链表)

标号链表：第一遍扫描时，存储了标号名、标号所在行数，便于第二遍扫描时对跳转指令的译码地址的存储。



图 4.标号链表数据结构示意图

其中 name 为标号名，line_num 为行数，next 为指向下一指针域的指针。

变量链表：储存变量的信息，保存在文件尾部。

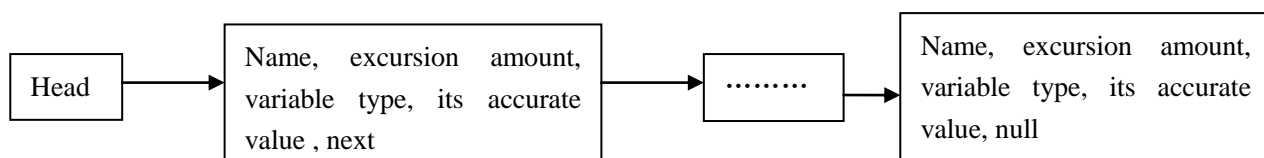


图 5.变量信息链表数据结构示意图

其中 name 为变量名，excursion 为偏移量，variable type 为数据类型，value 为变量值，next 为指向下一指针域的指针。

3、函数指针数组：在模拟器程序中，用来存放 32 个函数的指针，以便于指令对应函数的调用。

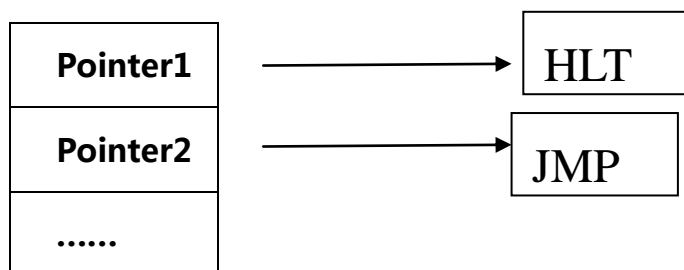


图 6.函数指针数组数据结构示意图

其中 pointer 为相应函数指针，HLT、JMP 为函数。

三、程序结构

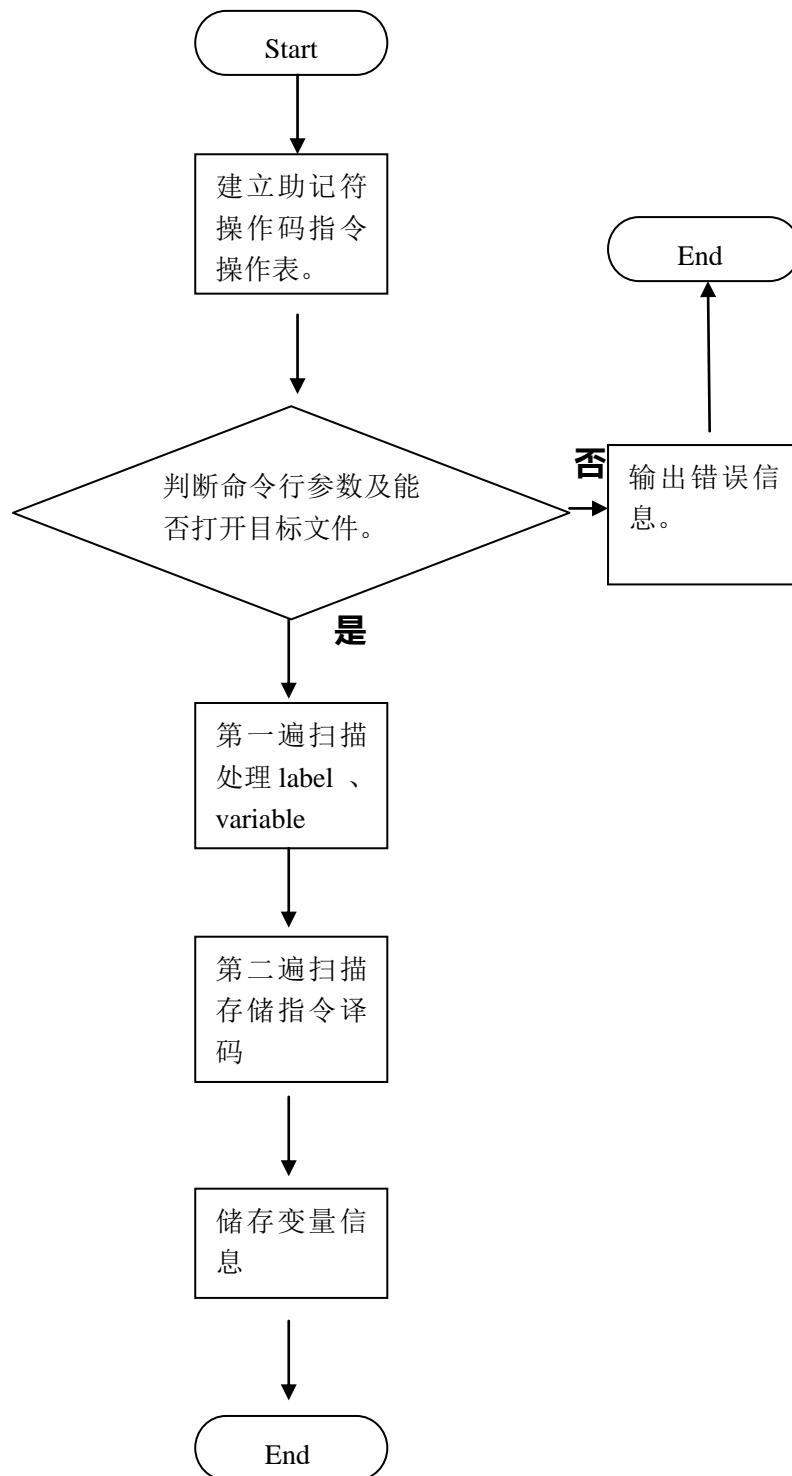


图 7.汇编程序执行流程图

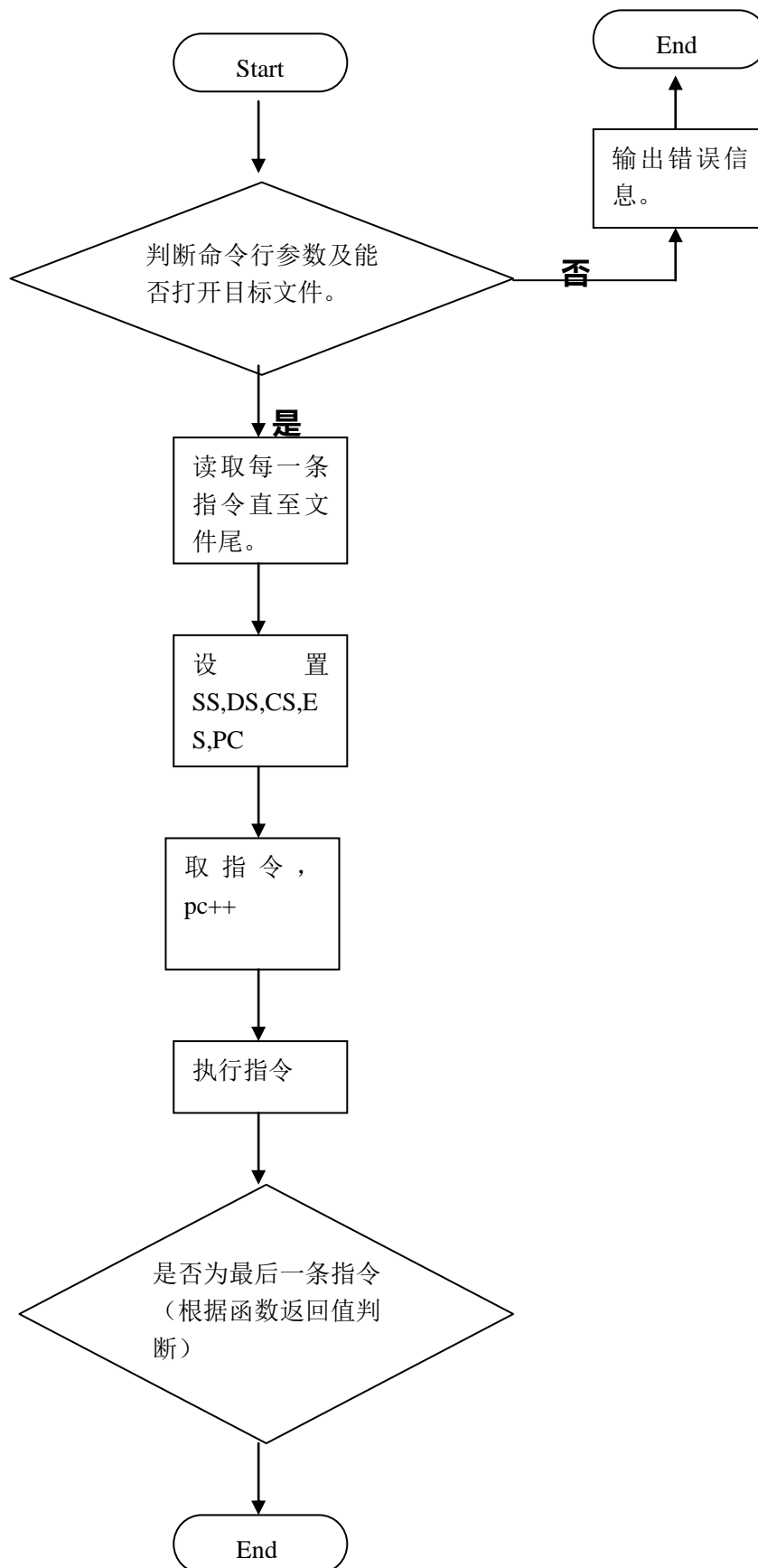


图 8.模拟器执行流程图

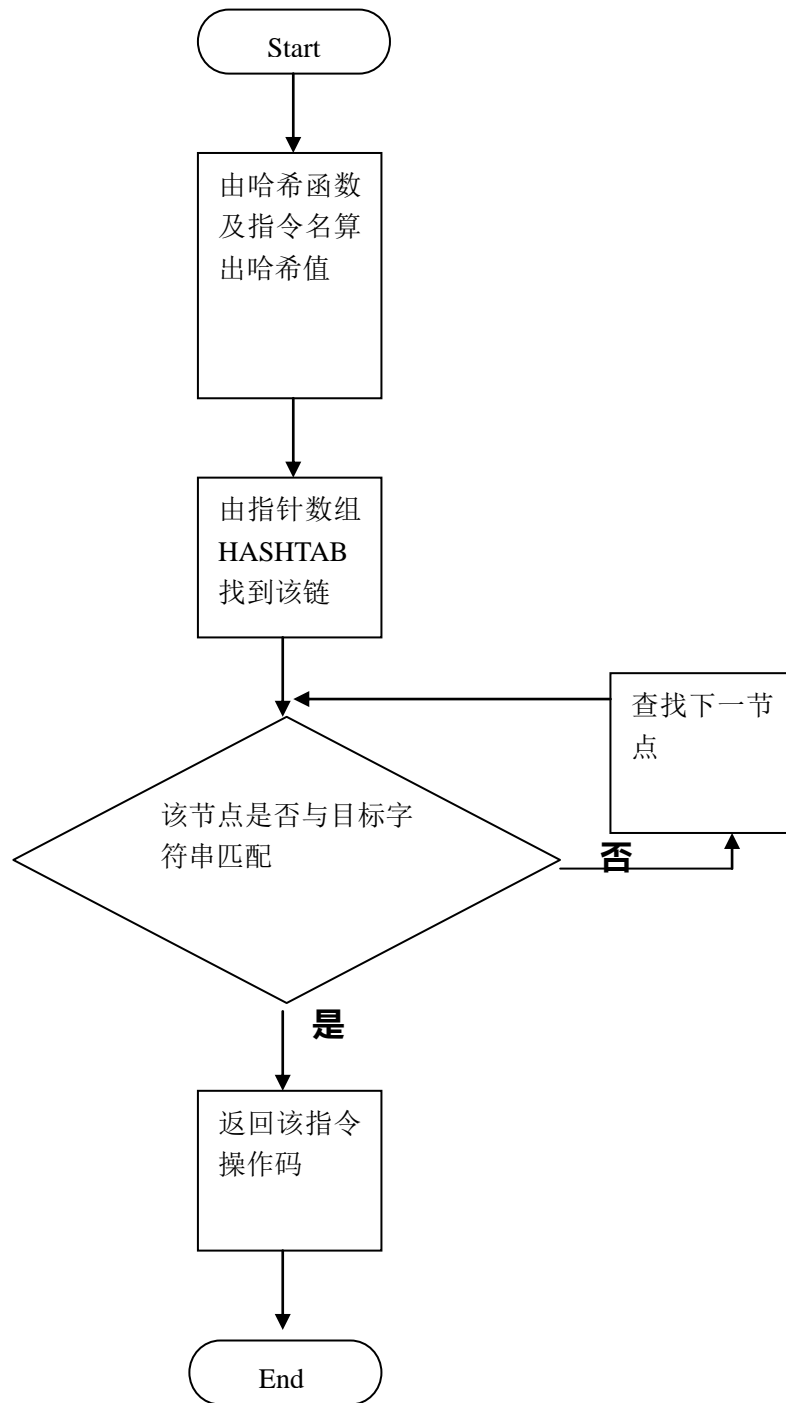


图 9.哈希表查找流程图

四、各函数原型及功能

1.int GetInstrCode(char *op_sym);

功能 :由指令助记符得到指令操作码 ,参数为字符型指针。返回值为相应操作码数组标。

2.unsigned long TransToCode(char *line, int num , label *head, varia *head);

功能 :将指令翻译成目标代码 ,即译码。参数为指令行数 ,指令 ,标号链表头指针 ,变

量链表头指针，返回值为十六进制代码。

3.int GetRegNum(char *, char *);

功能：由寄存器名得到寄存器编号。参数为该行指令和寄存器名。返回值为其编号。

4.int SaveVaria(char *, int , int , varia *, int);

功能：保存变量信息。参数为该行指令，链表头指针，变量类型，偏移量，变量个数。

5.void PrintVaria(FILE *, varia *);

功能：输出变量信息。参数：输出的目标文件指针，变量信息的头指针。

6.int SaveLabel(char *, int , label *);

功能：保存标号信息。参数：标号名，行数，标号链表头指针。

7.unsigned hash(char *);

功能：计算 hash 值。参数：指令名。返回值为相应 hashvalue。

8.struct Instr_list *lookup(char *);

功能：哈希表查找。参数：指令名。返回值为结构指针或 NULL。

9.char *strddup(char *);

功能：字符串分配存储空间。返回字符指针。

10.struct Instr_list *install(char *,int);

功能：创建 hash 表。参数:指令名，操作码。返回值为头指针。

11. int HLT(void);

功能：终止程序运行。

12.int JMP(void);

功能：将控制转移至标号 label 处，执行标号 label 后的指令。

13.int CJMP(void);

功能：如果程序状态字中比较标志位 c 的值为 1(即关系运算的结果为真)，则将控制转移至标号 label 处，执行标号 label 后的指令；否则，顺序往下执行。

14.int OJMP(void);

功能：如果程序状态字中比较标志位 o 的值为 1(即算术运算的结果发生溢出)，则将控制转移至标号 label 处，执行标号 label 后的指令；否则，顺序往下执行。

15.int CALL(void);

功能：将通用寄存器 A~G、程序状态字 PSW、程序计数器 PC 中的值保存到 ES，然后调用以标号 label 开始的子程序，将控制转移至标号 label 处，执行标号 label 后的指令。

16.int RET(void);

功能：将 ES 中保存的通用寄存器 A~Z、程序状态字 PSW 和程序计数器 PC 的值恢复，控制转移到子程序被调用的地方，执行调用指令的下一条指令。

17.int PUSH(void);

功能：将通用寄存器 reg0 的值压入堆栈 SS，reg0 可以是 A~G 和 Z 八个通用寄存器之一。

18.int POP(void);

功能：从堆栈 SS 中将数据出栈到寄存器 reg0，reg0 可以是 A~G 七个通用寄存器之一，但不能是通用寄存器 Z。

19.int LOADB(void);

功能：从字节数据或字节数据块 symbol 中取一个字节的数存入寄存器 reg0，所取的字节数据在数据块 symbol 中的位置由寄存器 G 的值决定。用 C 的语法可将此指令的功能描述为：

$$\text{reg0} = \text{symbol}[\text{G}]$$

例如，假设用伪指令定义了以下字节数据块 num：

```
BYTE    num[10] = {5,3,2,8,6,9,1,7,4,0}
```

如果要将字节数据块 num 中第 5 个单元的值(即下标为 4 的元素)取到寄存器 C，指令如下：

```
LOADI    G    5
LOADB    C    num
```

后面的指令 LOADW、STOREB 和 STOREW 在操作上与此指令类似。

20.int LOADW(void);

功能 从双字节数据或双字节数据块 symbol 中取一个双字节的数据存入寄存器 reg0，所取的双字节数据在数据块 symbol 中的位置由寄存器 G 的值决定。

21.int STOREB(void);

功能：将寄存器 reg0 的值存入字节数据或字节数据块 symbol 中的某个单元，存入单元的位置由寄存器 G 的值决定。用 C 的语法可将此指令的功能描述为：

```
symbol[G] = reg0
```

22.int STOREW(void);

功能：将寄存器 reg0 的值存入双字节数据或双字节数据块 symbol 中的某个单元，存入单元的位置由寄存器 G 的值决定。

23.int LOADI(void);

功能 将指令中的立即数 immediate 存入寄存器 reg0。立即数被当作 16 位有符号数，超出 16 位的高位部分被截掉。例如：

```
LOADI    B    65535
```

寄存器 B 的值为-1。

LOADI B 65537

寄存器 B 的值为 1。

24.int NOP(void);

功能：不执行任何操作，但耗用一个指令执行周期。

25.int IN(void);

功能：从输入端口(即键盘输入缓冲区)取一个字符数据，存入寄存器 reg0。

26.int OUT(void);

功能：将寄存器 reg0 的低字节作为字符数据输出到输出端口(即显示器)。

27.int ADD(void);

功能：将寄存器 reg1 的值加上 reg2 的值，结果存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

28.int ADDI(void);

功能：将寄存器 reg0 的值加上立即数 immediate，结果仍存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

29.int SUB(void);

功能：将寄存器 reg1 的值减去 reg2 的值，结果存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

30.int SUBI(void);

功能：将寄存器 reg0 的值减去立即数 immediate，结果仍存入寄存器 reg0。如果结

果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

31.int MUL(void);

功能：将寄存器 reg1 的值乘以 reg2 的值，结果存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

32.int DIV(void);

功能：将寄存器 reg1 的值除以 reg2 的值，结果存入寄存器 reg0，这里进行的是整数除运算。如果寄存器 reg2 的值为零，将发生除零错。

33.int AND(void);

功能：将寄存器 reg1 的值与 reg2 的值进行按位与运算，结果存入寄存器 reg0。

34.int OR(void);

功能：将寄存器 reg1 的值与 reg2 的值进行按位或运算，结果存入寄存器 reg0。

35.int NOR(void);

功能：将寄存器 reg1 的值与 reg2 的值进行按位异或(按位加)运算，结果存入寄存器 reg0。

36.int NOTB(void);

功能：将寄存器 reg1 的值按位取反后，结果存入寄存器 reg0。

37.int SAL(void);

功能：将寄存器 reg1 的值算术左移 reg2 位，结果存入寄存器 reg0。在进行算术左移时，低位空位用 0 填充。

38.int SAR(void);

功能：将寄存器 reg1 的值算术右移 reg2 位，结果存入寄存器 reg0。在进行算术右移时，高位空位用符号位填充。

39.int EQU(void);

功能：将两个寄存器 reg0 和 reg1 的值进行相等比较关系运算： $\text{reg0} == \text{reg1}$ ，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

40.int LT(void);

功能：将两个寄存器 reg0 和 reg1 的值进行小于关系运算： $\text{reg0} < \text{reg1}$ ，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

41.int LTE(void);

功能：将两个寄存器 reg0 和 reg1 的值进行小于等于关系运算： $\text{reg0} \leq \text{reg1}$ ，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

42.int NOTC(void);

功能：将程序状态字中的比较标志位 c 求反，即将逻辑真变为逻辑假，将逻辑假变为逻辑真。

五、试验结果

Add:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\zhao>e:

E:\>cd c

E:\C>cd 课程设计

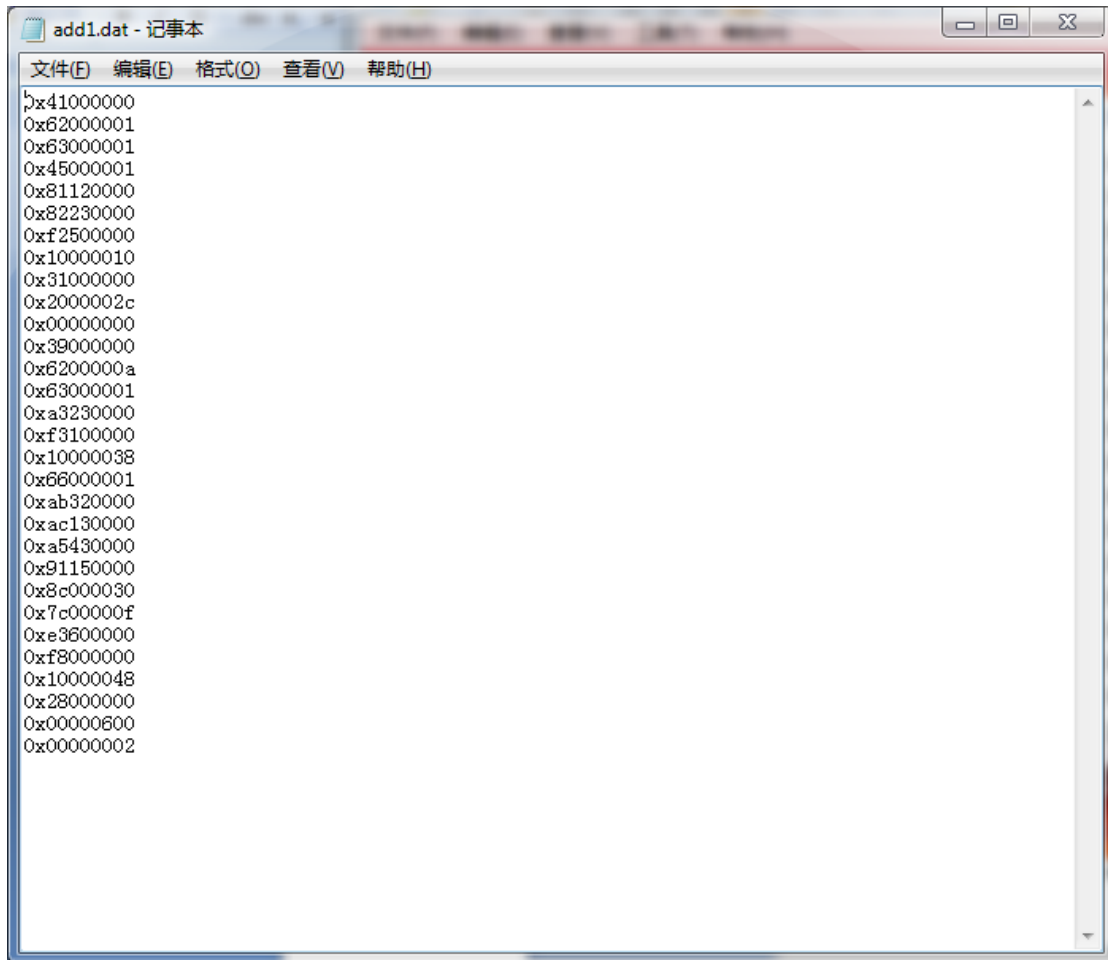
E:\C\课程设计>sas add1.txt add1.dat

E:\C\课程设计>ssim add1.dat
21
E:\C\课程设计>_
```

```
add1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

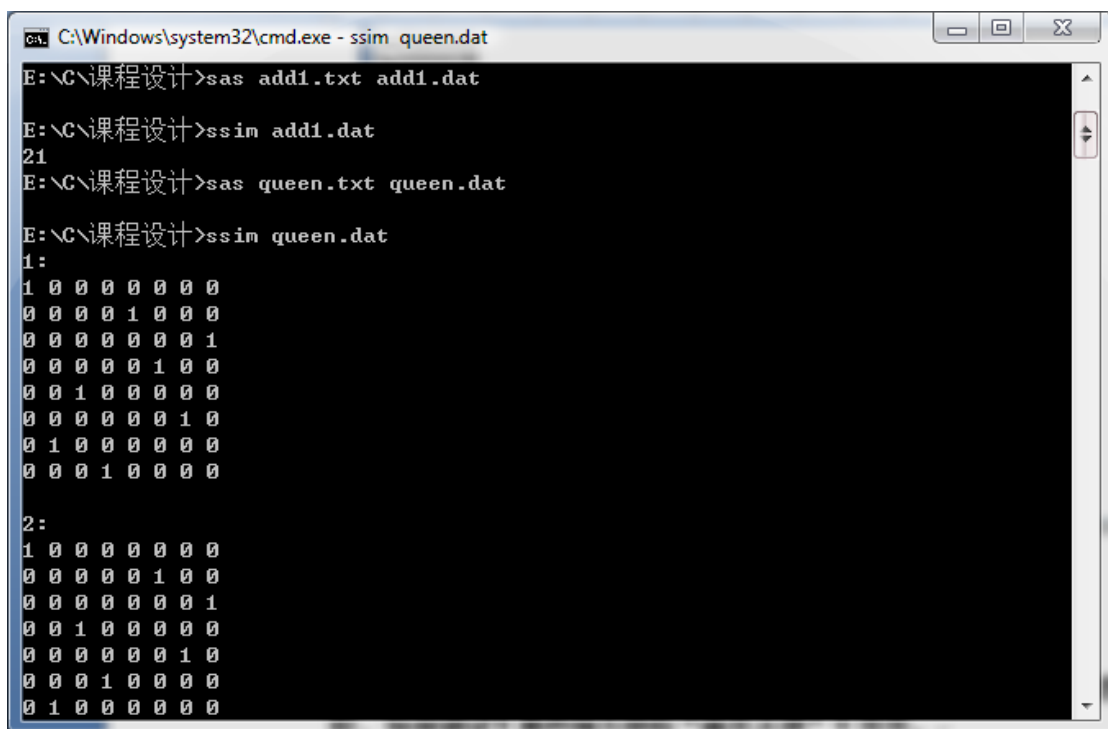
        BYTE sum = 0
        BYTE num = 6
        LOADB A sum          #和
        LOADI B 1            #当前数字
        LOADI C 1            #递增值
        LOADB E num
loop:    ADD A A B
        ADD B B C
        LTE B E
        CJMP loop
        PUSH A
        CALL prnt            # 调用子程序prnt
        HLT                  # 终止程序运行

        # 输出一个整数
prnt:    POP A
        LOADI B 10           #递减值
        LOADI C 1            #mask
loop1:   MUL C B C
        LTE C A              # c<=A
        CJMP loop1
        LOADI F 1            #退出值
loop2:   DIV C C B
        DIV D A C
        MUL E D C
        SUB A A E
        ADDI D 48
        OUT D 15
        EQU C F
        NOTC
        CJMP loop2
        RET
```



```
add1.dat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0x41000000
0x62000001
0x63000001
0x45000001
0x81120000
0x82230000
0xf2500000
0x10000010
0x31000000
0x2000002c
0x00000000
0x39000000
0x6200000a
0x63000001
0xa3230000
0xf3100000
0x10000038
0x66000001
0xab320000
0xac130000
0xa5430000
0x91150000
0x8c000030
0x7c00000f
0xe3600000
0xf8000000
0x10000048
0x28000000
0x00000600
0x00000002
```

Queen:



```
C:\Windows\system32\cmd.exe - ssim queen.dat
E:\C\课程设计>sas add1.txt add1.dat

E:\C\课程设计>ssim add1.dat
21
E:\C\课程设计>sas queen.txt queen.dat

E:\C\课程设计>ssim queen.dat
1:
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0

2:
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
```



```

C:\Windows\system32\cmd.exe
0 0 0 0 0 1 0 0

91:
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0

92:
0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0

```

```

queen.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

# 八皇后问题

WORD    cnt = 0          # 解计数器
BYTE    sltn[8] = {0,0,0,0,0,0,0,0} # 存放解的数组，元素值依次为各行上皇后的位置
BYTE    cell[64]         # 元素值表示对应单元位置受皇后攻击状况

# 将数组cell的元素值初始化为0
init:    LOADI    A      64          # 设置数组下标的上界为64
         LOADI    G      0          # 数组下标初始化为0
         STOREB   Z      cell       # 将寄存器Z的值存入cell[G]
         ADDI     G      1          # 下标增加1
         LT       G      A          # 关系运算 G < 64
         CJMP     init    # 比较结果为真则转至标号init，否则往下执行

         LOADI    B      0          # 将行号0存入寄存器B中
         PUSH     B          # 将B值压入堆栈
         CALL     dfs         # 调用子程序dfs

         HLT                     # 终止程序运行

dfs:     # 深度优先搜索算法，采用递归实现
         POP      B            # 从堆栈中取出行号值存入寄存器B
         LOADI    C      8      # 将行号的上界值8存入寄存器C，行号取值范围0-7
         LT       B      C      # 关系运算 B < C，比较行号是否越界
         CJMP     next    # 没有越界则转至标号next，否则往下执行
         CALL     prnt    # 行号越界表明得到了一个解，调用子程序prnt，输出解
         RET                     # 控制返回子程序被调用处

next:    LOADI    A      0          # 将0存入寄存器A，从B行的第0列开始测试
n1:      MUL      D      B      C    # 计算 D = B * C，计算第B行元素的起始下标
         ADD      G      A      D    # 计算 G = A + D，得到第B行第A列的元素下标值
         LOADB    D      cell       # 将cell[G]取出，存入寄存器D
         EQU      D      Z          # 关系运算 D == 0(Z)，为真表明第B行第A列没有受到皇
后攻击
         NOTC     # 将比较标志位的值反转
         CJMP     n2      # 此时比较标志为真表示D不等于0，转至标号n2，否则可
在此处放置皇后
         PUSH     A          # 将寄存器A的值压入堆栈

```



```
assembly.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

        IN A 0          #十以内加减法
        IN E 0
        IN B 0
        SUBI A 48
        SUBI B 48
        ADD A A B
        PUSH A
        CALL print      # 调用子程序print
        HLT             # 终止程序运行

print:   # 输出一个整数
        POP A
        LOADI B 10      #递减值
        LOADI C 1       #mask
loop1:   MUL C B C
        LTE C A         # c<=A
        CJMP loop1
        LOADI F 1       #退出值
loop2:   DIV C C B
        DIV D A C
        MUL E D C
        SUB A A E
        ADDI D 48
        OUT D 15
        EQU C F
        NOTC
        CJMP loop2
        RET
```

```
assembly.dat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0x71000000
0x75000000
0x72000000
0x99000030
0x9a000030
0x81120000
0x31000000
0x20000024
0x00000000
0x39000000
0x6200000a
0x63000001
0xa3230000
0xf3100000
0x10000030
0x66000001
0xab320000
0xac130000
0xa5430000
0x91150000
0x8c000030
0x7c00000f
0xe3600000
0xf8000000
0x10000040
0x28000000
0x00000000
```

六、实验体会

由课程设计想到的

大一上学期末时，卢老师给我们布置了课设的任务，对于初出茅庐的我们来说，无疑是个挑战。有所为，必有所悟，感觉体会还是多而深刻的。

整个汇编的任务分为两个部分：汇编程序和模拟器。第一部分难度相对较大。当时我甚至根本没有接触过汇编语言，只在开学时在班主任口中有所耳闻，后来百度俱乐部的学长也介绍了一些与汇编语言相关的知识。总的来说，汇编语言是有价值的。它的代码虽然繁琐而冗长，但是对我们深刻的了解计算机系统内部的运行原理是很有帮助的。其实这次课设也为我们以后学习汇编语言打下了基础。

在上学期最后一个星期中，我们去了几次机房，卢老师也不厌其烦的为我们讲解整个程序的思路，就差没直接帮我们敲代码了。就我个人而言，其实当时根本没弄明白这个程序到底有什么存在的意义。于是每天下午老师的讲课也变成了对牛弹琴。当时我是比较急躁的，因为之前编程从来没有毫无头绪的感觉。心里一慌，思路就更乱。恶性循环中，便开始干脆放下，投入复习。这是我这次课设的第一个障碍，也是最难克服的一个。其实，跨出第一步便是好的。后来我在寒假开始静下心来翻看书本。对阵下药的方法确实行之有效。我不懂汇编语言，就一遍又一遍的看 Richard Blum 编写的《汇编语言程序设计》，开始逐渐了解汇编语言的几大特点。借助书上的样例程序，我对于整个程序的架构已经可以做到心中有数了。其实我们这次课设就是将汇编源代码用 C 语言实现的过程。看这些代码让我变得更耐心，也更有信心了。

在寒假看了两天之后我开始做第一个 SAS，之前在学校学着书上做了一些基础工作，其实根本没弄懂，于是干脆从头开始做。然而，跟书上代码不同的是，

卢老师给我们的汇编源代码加上了标号和伪指令。而且提出了更多要求。我遇到的第二个麻烦是做指令助记符及操作码的哈希表,一开始我非常自信,觉得这个,肯定不需要翻书。可以自己把所有的配套的函数都写出来。结果花了半天时间后,一直报错的编译器让我不得不拿起书本开始模仿起来。结果发现自己的问题很多,如哈希函数的计算忘了除以它的函数指针数组下标的最大值、install 函数的指针赋值出现漏洞等。孰能生巧,果然如此。

Sas 中最大的麻烦来源于伪指令的处理。一开始我根本没有想过扫描两遍,以为整体读一遍就可以万事大吉,后来发现程序只读一遍太过繁琐。而且我也回忆起老师的讲解,需要读两遍指令。联系到我们所学的链表,我决定把标号和变量分别用两个链表储存起来,并用 typedef 来简化代码。但是,链表中信息的储存的选择又让我困扰了许久。在同学的帮助下,我大概知道了哪些是关于变量的有用的信息。读取变量时,我通过互联网搜索到了一些书写较为复杂的更为高级读入办法,但是省去了一些代码。其实%后面是可以加一些其他的東西的。储存标号则相对较为简单,只需要储存标号名和行数就可以了。保存的代码会记录偏移量的值 (excursion)。处理了伪指令,八种操作码对应的代码格式只需要照着书上进行一些位运算就可以完成。程序最后需要保存变量的信息。在整个储存变量的过程中,给变量赋值并存储是重要但并不容易的一步,这些都需要耐心与时间。此外,在读入指令时,由于临时数组中大小的限制(初始值设为 50)及 fgets 函数中对数组大小的限制使我在编译过程中出了不少苦头。一开始在调试时不断修改函数读入方式均没有解决问题,后来观察 txt 文件发现有些行长度早已超过 50,有时候简单的细节也会让编程停滞不前。故严密性是非常重要的。在最后调试 sas 的过程中,由于不小心改动了宏定义的标点符号,程序直接崩溃,显示

appcrash。我一开始在程序中设置各种输出记号用以观察执行，都没有发现到底出了什么问题。后来决定建工程调试，发现函数在计算 hash 值时停滞不前。于是开始改 hash 函数，问题仍然得不到解决。于是我决定观察程序变量的变化，发现在 install 部分的指令名输入有误，最后修改了宏定义。总之，在 sas 这一部分，对伪指令变量的处理让我收获很多，各种细节性的问题也层出不穷，解决它们的过程是充实的。

在第二个程序中，老师要求的是新增几个“段”来辅助 ssim 的执行，这点与书上不同。

PC 的指令计数器仍然沿用书本，SS、ES、DS、CS 分别表示堆栈段、状态段、数据段、代码段。在写此部分函数时，需要先开始配置好几个段的初始地址。这部分需要准确定位几个初始地址。这是第一个困扰。随后就是三十二个函数的编写问题了，由于书上提供了一些函数，课设任务书里也有许多详尽的解释，这一部分完成的比较顺利。但 call 函数的出现又让变成难度增加了一些，尽管可以接受。

这次课设中，我发现我最大的问题是不严谨，也可能是急于求成的结果。在出现各种 bug 的同时，我做的 debug 工作变得异常的频繁，这也让我重新认识到调试的重要性，并学会了许多调试技巧。其次，通过课设这样庞大的工程，我的耐心增加了不少，熟练度也得到了相应的提升。最后的收获应该是学会了自学，通过查阅资料和浏览网上的信息，我学到了很多课本之外的东西，其实，这也是大学生提高自己的主要渠道和必备技能。

在体会的最后，衷心地感谢卢萍老师在半年以来对我们的循循善诱，让我们在编程方面有了很大的提高。卢老师每次实验课必到，而且总是最后离开，这让大家都非常感动。老师，妇女节快乐！

参考文献:

1. 《c 语言实验与课程设计》 李开 卢萍 曹计昌 编著 科学出版社
2. 《c 语言程序设计》 李开 卢萍 曹计昌 编著 科学出版社
3. 《c 程序设计语言》 Brian W.Kernighan ,Dennis M.Ritchie 著 机械工业出版社
4. 《算法导论》 Thomas H.Cormen ,Charles E.Leiserson 等 著 机械工业出版社
5. 《数据结构》 Ellis Horowitz 等 著 机械工业出版社
6. 《汇编语言程序设计》 Richard Blum 著 机械工业出版社

程序清单:

源代码:

汇编程序:

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "ctype.h"

#define MAXLEN 200
#define HASHSIZE 101
#define TYPEAMOUNT 8
#define INSTR_SYM {"HLT", "JMP", "CJMP", "OJMP", "CALL",
"RET","PUSH",\
"POP", "LOADB", "LOADW", "STOREB", "STOREW",
"LOADI",\
"NOP", "IN", "OUT", "ADD", "ADDI", "SUB", \
"SUBI", "MUL", "DIV", "AND", "OR", "NOR", \
"NOTB", "SAL", "SAR", "EQU", "LT", "LTE", "NOTC" \
}

typedef struct Label{
    char name[10];
    int line_num;
    struct Label *next;
}label;

char *g_instr_name[] = INSTR_SYM;
char instr_format[33] = "12222133444451667575777778778881";
struct Instr_list {
    struct Instr_list *next;
    char *name;
    int op_num;
};

typedef struct Variable{
    char name[10];
```

```

        int size;           /*个数*/
        int excursion;      /*偏移量*/
        int value[200];     /*值*/
        int type;           /*类型*/
        struct Variable *next;
    }varia;

    struct Instr_list *hashtab[HASHSIZE];
    unsigned hash(char *);
    struct Instr_list *lookup( char *);
    char *strddup(char *);
    struct Instr_list *install(char *,int );

    int GetInstrCode(char *op_sym);           /*由指令
    助记符得指令操作码*/
    unsigned long TransToCode(char *, int , label *, varia *);           /*指令的译码*/
    int GetRegNum(char *, char *);           /*由寄存
    器名 (A-G, Z)得到寄存器编号*/
    int SaveVaria(char *, int , int , varia *, int );           /*保存变量*/
    void PrintVaria(FILE *, varia *);           /*在目标文件
    中输出变量*/
    int SaveLabel(char *, int , label *);           /*保存标号的信
    息在链表里*/

    int main(int argc, char **argv)
    {
        char a_line[MAXLEN];
        char op_sym[TYPEAMOUNT];
        int op_num , i;
        char *pcPos;
        int type;
        FILE *pfIn, *pfOut;
        label *head_label = (label *)malloc(sizeof(label));
        head_label->next = NULL;
        varia *head_varia = (varia *)malloc(sizeof(varia));
        head_varia->next = NULL;
        char label_temp[20];           /*存放标号名称*/
        char string[200];           /*存放定义变量 name 和 size 的字符串*/
        char varia_size_str[20];
        int varia_size_int;
        int excursion_num = 0;
        int line_num = 0;           /*行数*/

```



```

int count;                                /*建立助记符操作码哈希表*/
for(count = 0; count < 32; count++)
{
    install(g_instr_name[count],count);
}

int n;
if(argc<3)                                /*检查命令行参数数目*/
{
    printf("ERROR: no enough  command line arguments\n");
    return 0;
}
if((pfIn = fopen(argv[1], "r")) == NULL)/*打开源代码文件*/
{
    printf("ERROR: cannot open file %s for reading!\n", argv[1]);
    return 0;
}
if((pfOut = fopen(argv[2], "w")) == NULL)/*打开目标代码文件*/
{
    printf("ERROR: cannot open file %s for writing!\n", argv[2]);
    return 0;
}
//printf("ok");
while(!feof(pfIn))
{
    fgets(a_line, 200, pfIn);  /*从源文件取一行命令*/
    if((pcPos = strchr(a_line, '#')) != NULL)
    {
        *pcPos = '\0';      /*去掉注释*/
    }
    n = sscanf(a_line, "%s", op_sym); /*取指令助记符*/
    if(n<1)                      /*空行和注释行的处理*/
    {
        continue;
    }
    if((pcPos = strchr(a_line, ':')) != NULL)
    {
        sscanf(a_line,"%%[^:]",label_temp);
        SaveLabel(label_temp, line_num, head_label);
    }
}

```

```

else if(strstr(a_line,"WORD") != NULL)
{
    sscanf(a_line,"%*s %s",string);
    type = 2;
    if((pcPos = strchr(string, '[')) != NULL)
    {
        sscanf(string, "%*[^[][%[^]", varia_size_str);      /*保存变量*/
        varia_size_int = atoi(varia_size_str);
    }
    else
    {
        varia_size_int = 1;
    }
    SaveVaria(a_line, excursion_num, varia_size_int, head_varia, type);
    excursion_num += 2*varia_size_int;
    line_num--;
}
else if(strstr(a_line,"BYTE") != NULL)
{
    sscanf(a_line,"%*s %s",string);
    type = 1;
    if((pcPos = strchr(string, '[')) != NULL)
    {
        sscanf(string, "%*[^[][%[^]", varia_size_str);
        varia_size_int = atoi(varia_size_str);
    }
    else
    {
        varia_size_int = 1;
    }
    SaveVaria(a_line, excursion_num, varia_size_int, head_varia, type);
    excursion_num += varia_size_int;
    line_num--;
}
line_num++;
}
fclose(pfIn);
pfIn=fopen(argv[1],"r");
while(!feof(pfIn))
{
    fgets(a_line, 200, pfIn);
    if((pcPos = strchr(a_line,'#'))!=NULL)
    {
        *pcPos='\0';

```

```

    }
    if((pcPos = strstr(a_line,"WORD"))!=NULL)
    {
        continue;
    }
    else if((pcPos = strstr(a_line,"BYTE"))!=NULL)
    {
        continue;
    }
    else if((pcPos = strchr(a_line, ':')) != NULL)
    {
        for(i = 0; a_line[i] != ':'; i++)
        {
            a_line[i]=' ';           /*label 处理*/
        }
        a_line[i]=' ';
    }
    n=sscanf(a_line,"%s",op_sym);
    if(n<1){
        continue;
    }
    op_num = GetInstrCode(op_sym);
    if(op_num > 31)
    {
        printf("ERROR: %s is a invalid instruction! \n", a_line);
        exit(-1);
    }
    fprintf(pfOut, "0x%08lx\n", TransToCode(a_line, op_num, head_label,
head_varia));
}
PrintVaria(pfOut, head_varia);
fclose(pfIn);
fclose(pfOut);
return 0;
}

/*由指令助记符得到指令操作码*/
int GetInstrCode( char *op_sym)
{
    struct Instr_list *np;
    if((np=lookup(op_sym))!=NULL)
        return np->op_num;
    return 0;
}

```

```

unsigned long TransToCode(char *instr_line, int instr_num, label *head1, varia *head2)
{
    unsigned long op_code;
    unsigned long arg1, arg2, arg3;
    unsigned long instr_code = 0ul;
    char op_sym[8], reg0[8], reg1[8], reg2[8];
    unsigned long addr;
    int immed, port;
    char string[20];
    label *p1;
    varia *p2;
    int n;

    switch(instr_format[instr_num])    /*根据指令格式，分别进行译码*/
    {
        case '1':
        {
            op_code = instr_num;
            instr_code = op_code << 27;
            break;
        }
        case '2':
        {
            n = sscanf(instr_line, "%s %s", op_sym, string);

            if(n<2)
            {
                printf("ERROR:bad instruction format!\n");
                exit(-1);
            }
            for(p1 = head1->next; p1 != NULL; p1 = p1->next)
            {

                if(strcmp(p1->name,string)==0)
                {

                    break;
                }
            }
            if(p1 == NULL)
            {
                printf("ERROR:%s wrong instruction line!",instr_line);
                exit(-1);
            }
        }
    }
}

```

```

    }
    addr = (unsigned long)((p1->line_num)*4);
/*地址为行数乘四*/
    op_code = GetInstrCode(op_sym);
    instr_code = (op_code << 27) | (addr & 0x00ffffff);
    break;
}
case '3':
{
    n = sscanf(instr_line, "%s %s", op_sym, reg0);
    if(n < 2)
    {
        printf("ERROR:bad instruction format!\n");
        exit(-1);
    }
    op_code = GetInstrCode(op_sym);

    arg1 = GetRegNum(instr_line, reg0);
    instr_code = (op_code << 27) | (arg1 << 24);
    break;
}
case '4':
{
    n = sscanf(instr_line, "%s %s %s", op_sym, reg0, string);
    if(n < 3)
    {
        printf("ERROR:bad instruction format!\n");
        exit(-1);
    }
    for(p2 = head2->next; p2 != NULL; p2 = p2->next)
    {
        if(strcmp(p2->name, string) == 0)
        {
            break;
        }
    }
    if(p2 == NULL)
    {
        printf("ERROR:%s, wrong instruction line!", instr_line);
        exit(-1);
    }
    addr = (unsigned long)(p2->excursion);
    op_code = GetInstrCode(op_sym);
    arg1 = GetRegNum(instr_line, reg0);

```

```

        instr_code = (op_code << 27) | (arg1 << 24) | (addr & 0x00ffffff);
        break;
    }
    case '5':
    {
        n = sscanf(instr_line, "%s %s %i", op_sym, reg0, &immed);
        if(n < 3)
        {
            printf("ERROR:bad instruction format!\n");
            exit(-1);
        }
        op_code = GetInstrCode(op_sym);
        arg1 = GetRegNum(instr_line, reg0);
        instr_code = (op_code << 27) | (arg1 << 24) | (immed & 0x0000ffff);
        break;
    }
    case '6':
    {
        n = sscanf(instr_line, "%s %s %i", op_sym, reg0, &port);
        if(n < 3)
        {
            printf("ERROR:bad instruction format!\n");
            exit(-1);
        }
        op_code = GetInstrCode(op_sym);
        arg1 = GetRegNum(instr_line, reg0);
        instr_code = (op_code << 27) | (arg1 << 24) | (port & 0x0000ffff);
        break;
    }
    case '7':
    {
        n = sscanf(instr_line, "%s%s%s%s", op_sym, reg0, reg1, reg2);
        if(n < 4)
        {
            printf("ERROR:bad instruction format!\n");
            exit(-1);
        }
        op_code = GetInstrCode(op_sym);
        arg1 = GetRegNum(instr_line, reg0);
        arg2 = GetRegNum(instr_line, reg1);
        arg3 = GetRegNum(instr_line, reg2);
        instr_code = (op_code << 27) | (arg1 << 24) | (arg2 << 20) | (arg3 << 16);
        break;
    }
}

```

```

        case '8':
        {
            n = sscanf(instr_line, "%s%s%s", op_sym, reg0, reg1);
            if(n < 3)
            {
                printf("ERROR:bad instruction format!\n");
                exit(-1);
            }
            op_code = GetInstrCode(op_sym);
            arg1 = GetRegNum(instr_line, reg0);
            arg2 = GetRegNum(instr_line, reg1);
            instr_code = (op_code << 27) | (arg1 << 24) | (arg2 << 20);
            break;
        }
    }
    return instr_code;          /*返回目标代码*/
}
/*由寄存器名（A-G,Z）得到寄存器编号*/
int GetRegNum(char *instr_line, char *reg_name)
{
    int reg_num;
    if(tolower(*reg_name) == 'z')
    {
        reg_num = 0;
    }
    else if((tolower(*reg_name) >= 'a') && (tolower(*reg_name) <= 'g'))
    {
        reg_num = tolower(*reg_name) - 'a' + 1;
    }
    else
    {
        printf("ERROR:bad register name in %s!\n", instr_line);
        exit(-1);
    }
    return reg_num;
}

int SaveLabel(char *label_name, int line_num, label *head)
{
    label *np;
    for(np = head; np->next != NULL; np = np->next)
        ;
    label *p = (label *)malloc(sizeof(label));
    np->next = p;
}

```

```

    p->line_num = line_num;
    strcpy(p->name, label_name);
    p->next = NULL;
    return 0;
}

int SaveVaria(char *a_line, int excursion, int size, varia *head, int type)
{
    varia *p1;
    short value1;
    char *p = NULL;
    char value_str[10];
    int i, k;
    char variname_temp[10];
    char string[200];
    int assign;
    sscanf(a_line, "%*s %s", string);
    sscanf(string, "%[^"]", variname_temp);

    for(p1 = head; p1->next != NULL; p1 = p1->next)
        ;

    varia *pnew = (varia *)malloc(sizeof(varia));
    p1->next = pnew;
    pnew->excursion = excursion;
    pnew->size = size;
    pnew->type = type;
    strcpy(pnew->name, variname_temp);
    pnew->next = NULL;

    for(i = 0; i < size; i++)
    {
        pnew->value[i] = 0;
    }

    if(p = strchr(a_line, '=')) /*当只存一个值时*/
    {
        if(size == 1)
        {
            sscanf(a_line, "%*[^]=%d", &value1);
            pnew->value[0] = value1;
        }
        else
        {

```



```

    for(; p != NULL; p++)
    {
        if(*p == '{')
        {
            ++p;
            assign = 1;
            k = 0;
            while(*p != '}')
            {
                for(i = 0; *p != ',' && *p != '}' ; p++)
                {
                    value_str[i++] = *p;
                }
                value_str[i] = '\0';
                pnew->value[k++] = atoi(value_str);
                if(*p == ',')
                {
                    ++p;
                }
            }
        }
        else if(*p == '"')
        {
            assign = 1;
            ++p;
            k = 0;
            for(i = 0; *p != '"'; p++)
            {
                pnew->value[k++] = *p;
            }
            pnew->value[k] = '\0';
        }
        if(assign)
            break;
    }
}

return 0;
}

char *strddup(char *s)
{
    char *p;
    p = (char *) malloc(strlen(s)+1);

```

```

if (p != NULL)
    strcpy(p, s);
return p;

}

unsigned hash(char *s)
{
    unsigned hashval;

    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}

struct Instr_list *lookup( char *s)
{
    struct Instr_list *np;

    for (np = hashtable[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;          /* found */
    return NULL;                /* not found */
}

struct Instr_list *install(char *name, int number)
{
    struct Instr_list *np;
    unsigned hashval;
    if ((np = lookup(name)) == NULL) { /* not found */

        np = (struct Instr_list *) malloc(sizeof(struct Instr_list));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtable[hashval];
        hashtable[hashval] = np;
        np->op_num=number;
    } else /* already there */
        free((void *) np->op_num);

    return np;
}

```

```

void PrintVaria(FILE *pfOut, varia *head)
{
    varia *p;
    int i;
    long k = 0;
    unsigned long addr = 0ul;
    for(p = head->next; p != NULL; p = p->next)
    {
        for(i = 0; i < p->size; i++)
        {
            if((k % 4) == 0)
            {
                if((p->type) == 1)
                {
                    addr += p->value[i];
                    k += 1;
                }
                else if((p->type) == 2)
                {
                    addr += (p->value[i])<<8;
                    k += 2;
                }
            }
            else if((k % 4) == 1)
            {
                if((p->type) == 1)
                {
                    addr += (p->value[i])<<8;
                    k += 1;
                }
                else if((p->type) == 2)
                {
                    addr += (p->value[i])<<16;
                    k += 2;
                }
            }
            else if((k % 4) == 2)
            {
                if((p->type) == 1)
                {
                    addr += (p->value[i])<<16;
                    k += 1;
                }
                else if((p->type) == 2)
            }
        }
    }
}

```

```

        {
            addr += (p->value[i])<<24;
            k += 2;
            fprintf(pfOut, "0x%08lx\n", addr);
            addr = 0ul;
        }
    }
    else if((k % 4) == 3)
    {
        if((p->type) == 1)
        {
            addr += (p->value[i])<<24;
            k += 1;
            fprintf(pfOut, "0x%08lx\n", addr);
            addr = 0ul;
        }
        else if((p->type) == 2)
        {
            addr += ((p->value[i])&0x00ff)<<24;
            k += 2;
            fprintf(pfOut, "0x%08lx\n", addr);
            addr = (p->value[i])>>8;
        }
    }
}

}
if((k % 4) != 0)
{
    fprintf(pfOut, "0x%08lx\n", addr);
}
fprintf(pfOut, "0x%08lx\n", k);
return ;
}

```

模拟器:

```

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>

```

```

/*定义宏来模拟指令的解码*/
#define REG0 ((IR >> 24)& 0x07)
#define REG1 ((IR >> 20)& 0x0F)
#define REG2 ((IR >> 16)& 0x0F)
#define IMMEDIATE (IR & 0xFFFF)
#define ADDRESS (IR & 0xFFFFF)
#define PORT (IR & 0xFF)
#define OPCODE ((IR >> 27)& 0x1F)

typedef struct _PROG_STATE_WORD
{
    unsigned short overflow_flg:1;
    unsigned short compare_flg:1;
    unsigned short reserve:14;
}PROG_STATE_WORD;

typedef struct _ES
{
    short reg[7];
    unsigned short overflow_flg;
    unsigned short compare_flg;
    unsigned short reserve;
    unsigned long *address;
}es;

unsigned char *MEM;    /*用动态存储区模拟内存,大小由命令行参数确定*/
unsigned long *PC;     /*指令计数器, 用来存放下条指令的内存地址*/
short GR[8];          /*通用寄存器的模拟*/
PROG_STATE_WORD PSW;
unsigned long IR;
unsigned long *CS;
unsigned char *DS;
short *SS;
es *ES;

int HLT(void);
int JMP(void);
int CJMP(void);
int OJMP(void);
int CALL(void);
int RET(void);

```

```

int PUSH(void);
int POP(void);
int LOADB(void);
int LOADW(void);
int STOREB(void);
int STOREW(void);
int LOADI(void);
int NOP(void);
int IN(void);
int OUT(void);
int ADD(void);
int ADDI(void);
int SUB(void);
int SUBI(void);
int MUL(void);
int DIV(void);
int AND(void);
int OR(void);
int NOR(void);
int NOTB(void);
int SAL(void);
int SAR(void);
int EQU(void);
int LT(void);
int LTE(void);
int NOTC(void);

int main(int argc,char **argv)
{
    unsigned long instruction;
    unsigned long mem_size = 0x1000000;
    /*函数指针数组，用于指令对应函数的调用*/
    int ( *ops[])(void)={HLT,JMP,CJMP,OJMP,CALL,RET,PUSH,POP,LOADB,\
                          LOADW,STOREB,STOREW,LOADI,NOP,IN,OUT,ADD,\
                          ADDI,SUB,SUBI,MUL,DIV,AND,OR,NOR,NOTB,SAL,\
                          SAR,EQU,LT,LTE,NOTC};

    FILE *pfln;
    int ret=1;
    long length;

    if(argc<2){
        printf("ERROR:no enough command line arguments!\n");
        exit(-1);
    }
}

```

```

/*向系统申请动态存储区，模拟内存*/
if((MEM = (unsigned char *)malloc(mem_size)) == NULL)
{
    printf("ERROR:fail to allocate memory!\n");
    exit(-1);
}
PC = (unsigned long *)MEM;    /*使指令计数器指向模拟内存的顶端*/
if((pfIn = fopen(argv[1], "r")) == NULL)
{
    printf("ERROR:cannot open the file %s for reading!\n", argv[1]);
    exit(-1);
}
CS = PC;
while(!feof(pfIn))
{
    fscanf(pfIn,"%li",&instruction);
    memcpy(PC,&instruction,sizeof(instruction));
    PC++;
}
ES = MEM - sizeof(es) + mem_size;
SS = (short *)PC;
PC--;
length = *PC;
if(length%4 == 0)
{
    DS = (unsigned char *) (PC - length/4);
}
else
{
    DS = (unsigned char *) (PC - (length/4 + 1));
}
DS = DS-3;
fclose(pfIn);
PC = (unsigned long *)MEM;    /*使 PC 指向模拟内存 MEM 顶端的第一条指令*/
while(ret)                  /*模拟处理器执行指令*/
{
    IR = *PC;                /*取指: 将 PC 指示的指令加载到指令寄存器 IR */
    PC++;                    /*PC 指向下一条执行指令*/
    ret = (*ops[OPCODE])();  /*解码并执行指令*/
}
free(MEM);
return 0;
}

```

```

int HLT(void)
{
    return 0;
}

int JMP(void)
{
    PC=(unsigned long *) (MEM+ADDRESS);
    return 1;
}

int CJMP(void)
{
    if(PSW.compare_flg){
        PC=(unsigned long *) (MEM+ADDRESS);
    }
    return 1;
}

int OJMP(void)
{
    if(PSW.overflow_flg){
        PC=(unsigned long *) (MEM+ADDRESS);
    }
    return 1;
}

int CALL(void)
{
    int i;
    for(i = 0; i < 7; i++)
    {
        ES->reg[i] = GR[i+1];
    }
    ES->compare_flg = PSW.compare_flg;
    ES->overflow_flg = PSW.overflow_flg;
    ES->address = PC;
    ES--;
    PC=(unsigned long *) (MEM + ADDRESS);
    return 1;
}

int RET(void)
{

```



```

    int i;
    ES++;
    for(i = 0; i<7; i++){
        GR[i+1] = ES->reg[i];
    }
    PSW.compare_flg = ES->compare_flg;
    PSW.overflow_flg = ES->overflow_flg;
    PC = ES->address;
    return 1;
}

int PUSH(void)
{
    *SS = GR[REG0];
    SS++;
    return 1;
}

int POP(void)
{
    if(REG0==0){
        printf("ERROR!\n");
        exit(-1);
    }
    SS--;
    GR[REG0] = *SS;
    return 1;
}

int LOADB(void)
{
    GR[REG0]=(short)(*(DS + ADDRESS + GR[7] - 1));
    return 1;
}

int LOADW(void)
{
    GR[REG0]=(short)((*(DS+ADDRESS+GR[7]-1)<<8)+*(DS+ADDRESS+GR[7]));
    return 1;
}

int STOREB(void)
{
    *(DS + ADDRESS + GR[7] - 1) = GR[REG0];

```

```

        return 1;
    }

int STOREW(void)
{
    *(DS + ADDRESS + GR[7] - 1) = ((GR[REG0]&0xff00) >> 8);
    *(DS + ADDRESS + GR[7]) = GR[REG0]&0x00ff;
    return 1;
}

int LOADI(void)
{
    GR[REG0]=(short)(IMMEDIATE);
    return 1;
}

int NOP(void)
{
    return 1;
}

int IN(void)
{
    read(0,(char *)(GR+REG0),1);
    return 1;
}

int OUT(void)
{
    write(1,(char *)(GR+REG0),1);
    return 1;
}

int ADD(void)
{
    GR[REG0]=GR[REG1]+GR[REG2];
    if(GR[REG2]>0){
        if(GR[REG0]<GR[REG1]){
            PSW.overflow_flg=1;
        }
        else{
            PSW.overflow_flg=0;
        }
    }
}

```

```

else if(GR[REG2]<0){
    if(GR[REG0]>GR[REG1]){
        PSW.overflow_flg=1;
    }
    else{
        PSW.overflow_flg=0;
    }
}
else{
    PSW.overflow_flg=0;
}
return 1;
}

```

```

int ADDI(void)
{
    int n;
    n=GR[REG0]+IMMEDIATE;
    if(IMMEDIATE > 0){
        if(n < GR[REG0]){
            PSW.overflow_flg=1;
        }
        else{
            PSW.overflow_flg=0;
        }
    }
    else if(IMMEDIATE < 0){
        if(n > GR[REG0]){
            PSW.overflow_flg=1;
        }
        else{
            PSW.overflow_flg=0;
        }
    }
    else{
        PSW.overflow_flg=0;
    }
    GR[REG0] = n;
    return 1;
}

```

```

int SUB(void)
{
    GR[REG0]=GR[REG1]-GR[REG2];

```

```

if(GR[REG2]>0){
    if(GR[REG0]>GR[REG1]){
        PSW.overflow_flg=1;
    }
    else{
        PSW.overflow_flg=0;
    }
}
else if(GR[REG2]<0){
    if(GR[REG0]<GR[REG1]){
        PSW.overflow_flg=1;
    }
    else{
        PSW.overflow_flg=0;
    }
}
else{
    PSW.overflow_flg=0;
}
return 1;
}

```

```

int SUBI(void)
{
    int n;
    n = GR[REG0] - IMMEDIATE;
    if(IMMEDIATE < 0){
        if(n < GR[REG0]){
            PSW.overflow_flg=1;
        }
        else{
            PSW.overflow_flg=0;
        }
    }
    else if(IMMEDIATE > 0){
        if(n > GR[REG0]){
            PSW.overflow_flg=1;
        }
        else{
            PSW.overflow_flg=0;
        }
    }
    else{
        PSW.overflow_flg=0;
    }
}

```

```

    }
    GR[REG0] = n;
    return 1;
}

int MUL(void)
{
    GR[REG0]=GR[REG1]*GR[REG2];
    if(abs(GR[REG2])>1){
        if(abs(GR[REG0])>abs(GR[REG1])){
            PSW.overflow_flg=0;
        }
        else{
            PSW.overflow_flg=1;
        }
    }
    else{
        PSW.overflow_flg=0;
    }
    return 1;
}

int DIV(void)
{
    if(GR[REG2] == 0)
    {
        printf("0 cannot be divided\n");
        exit(-1);
    }
    GR[REG0]=GR[REG1]/GR[REG2];
    return 1;
}

int AND(void)
{
    GR[REG0]=GR[REG1]&GR[REG2];
    return 1;
}

int OR(void)
{
    GR[REG0]=GR[REG1]|GR[REG2];
    return 1;
}

```

```

int NOR(void)
{
    GR[REG0]=GR[REG1]^GR[REG2];
    return 1;
}

int NOTB(void)
{
    GR[REG0]=~GR[REG1];
    return 1;
}

int SAL(void)
{
    GR[REG0]=GR[REG1]<<GR[REG2];
    return 1;
}

int SAR(void)
{
    GR[REG0]=GR[REG1]>>GR[REG2];
    return 1;
}

int EQU(void)
{
    PSW.compare_flg=(GR[REG0]==GR[REG1]);
    return 1;
}

int LT(void)
{
    PSW.compare_flg=(GR[REG0]<GR[REG1]);
    return 1;
}

int LTE(void)
{
    PSW.compare_flg=(GR[REG0]<=GR[REG1]);
    return 1;
}

int NOTC(void)

```

```

{
    PSW.compare_flg=!PSW.compare_flg;
    return 1;
}

Add1:
    BYTE sum = 0
        BYTE num = 6
        LOADB A sum      #和
    LOADI B 1      #当前数字
    LOADI C 1      #递增值
    LOADB E num
loop:    ADD A A B
    ADD B B C
    LTE B E
    CJMP loop
    PUSH A
    CALL prnt      # 调用子程序 prnt
    HLT            # 终止程序运行

        # 输出一个整数
prnt:    POP A
    LOADI B 10      #递缩值
    LOADI C 1      #mask
loop1:    MUL C B C
    LTE C A          # c<=A
    CJMP loop1
    LOADI F 1      #退出值
loop2:    DIV C C B
    DIV D A C
    MUL E D C
    SUB A A E
    ADDI D 48
    OUT D 15
    EQU C F
    NOTC
    CJMP loop2
    RET

Queen:
    # 八皇后问题

```

```

WORD    cnt = 0                # 解计数器
BYTE    sltn[8] = {0,0,0,0,0,0,0,0} # 存放解的数组，元素值依次为各行上皇后的位置
BYTE    cell[64]                # 元素值表示对应单元位置受皇后攻击状况

# 将数组 cell 的元素值初始化为 0
LOADI   A      64                # 设置数组下标的上界为 64
LOADI   G      0                # 数组下标初始化为 0
init:   STOREB  Z      cell        # 将寄存器 Z 的值存入 cell[G]
        ADDI   G      1                # 下标增加 1
        LT     G      A                # 关系运算 G < 64
        CJMP   init                    # 比较结果为真则转至标号 init，否则往下执行

        LOADI  B      0                # 将行号 0 存入寄存器 B 中
        PUSH   B                        # 将 B 值压入堆栈
        CALL   dfs                    # 调用子程序 dfs

        HLT                                # 终止程序运行

# 深度优先搜索算法，采用递归实现
dfs:    POP     B                        # 从堆栈中取出行号值存入寄存器 B
        LOADI  C      8                # 将行号的上界值 8 存入寄存器 C，行号取值范围 0-7
        LT     B      C                # 关系运算 B < C，比较行号是否越界
        CJMP   next                    # 没有越界则转至标号 next，否则往下执行
        CALL   prnt                    # 行号越界表明得到了一个解，调用子程序 prnt，输出
解
        RET                                # 控制返回子程序被调用处

next:   LOADI  A      0                # 将 0 存入寄存器 A，从 B 行的第 0 列开始测试
n1:     MUL    D      B      C        # 计算 D = B * C，计算第 B 行元素的起始下标
        ADD    G      A      D        # 计算 G = A + D，得到第 B 行第 A 列的元素下标值
        LOADB  D      cell            # 将 cell[G]取出，存入寄存器 D
        EQU    D      Z                # 关系运算 D == 0(Z)，为真表明第 B 行第 A 列没有受
到皇后攻击
        NOTC                                # 将比较标志位的值反转
        CJMP   n2                    # 此时比较标志为真表示 D 不等于 0，转至标号 n2，否
则可在此处放置皇后
        PUSH   A                        # 将寄存器 A 的值压入堆栈
        PUSH   B                        # 将寄存器 B 的值压入堆栈
        CALL   tag                    # 调用子程序 tag 在 B 行 A 列放置皇，并在皇后所能攻
击到的位置上做标记
        ADD    D      B      Z        # 将寄存器 B 的值存入寄存器 D D = B + Z
        ADDI   D      1                # 将寄存器 D 中的值增加 1 D = D + 1，得到下一
行行号

```


	PUSH	D			# 将下一行行号 D 压入堆栈
	CALL	dfs			# 递归调用, 在下一行合适的位置上放置皇后
	PUSH	A			# 将列号 A 压入堆栈
	PUSH	B			# 将行号 B 压入堆栈
	CALL	tag			# 再次调用 tag, 抹去标记
n2:	ADDI	A	1		# 列号增加 1
	LT	A	C		# 关系运算 列号 < 8
	CJMP	n1			# 为真则转至标号 n1, 继续测试新一列
	RET				# 否则返回子程序被调用处
	# 输出解				
prnt:	LOADI	G	0		# 下标置为 0
	LOADW	C	cnt		# 将解计数器的值加载到寄存器 C 中
	ADDI	C	1		# 计数器的值增加 1
	STOREW	C	cnt		# 存入计数器
	LOADI	E	1		# 将 1 存入寄存器 E, 用作按位与运算的屏蔽码 0x1
	AND	D	C	E	# 按位与运算, 将 C 的最低位取出, 存入寄存器 D 中
	PUSH	D			# D 中值为 1, 表明解的个数为奇数, 为 0 则是偶数,
	先将 D 入栈				
	LOADI	D	10		# D = 10, 十进制进位值
loop1:	DIV	E	C	D	# E = C / D, E 中为 C 除以 10 的商, 整数除, 切掉 C 的个位数字
	MUL	F	D	E	# F = D * E
	SUB	F	C	F	# F = C - F, 得到 C 除以 10 的余数
	PUSH	F			# 将余数 F 入栈
	ADDI	G	1		# G 用来为余数压栈次数进行计数
	ADD	C	E	Z	# 将商 E 存入 C
	LT	Z	C		# 关系运算 Z < C
	CJMP	loop1			# 为真表示商大于 0, 转至 loop1, 继续求余数
loop2:	POP	C			# 余数出栈, 存入 C
	ADDI	C	48		# C = C + 48, 将数字转为数字字符
	OUT	C	15		# 输出 C 中的数字字符
	SUBI	G	1		# 余数计数器减 1
	LT	Z	G		# 关系运算符 Z < G
	CJMP	loop2			# 为真表明入栈的余数还未取完, 转 loop2 继续取余数
	并转换为数字字符输出				
	LOADI	C	58		# 将字符 ':' 存入寄存器 C
	OUT	C	15		# 输出字符 ':'
	LOADI	C	10		# 将换行符 '\n' 存入寄存器 C
	OUT	C	15		# 输出换行符
	# 依次输出八行上皇后的位置				

```

        LOADI    A      8      # 将数组下标上限 8 存入寄存器 A
        LOADI    G      0      # 将下标置为 0
loop3:  LOADB    C      sltn    # C = sltn[G], 每个元素的值表示元素下标对应的行上皇
后放置的位置
        PUSH     C          # C 入栈, C 值表示皇后在第 G 行上的位置, 即列号
        CALL     aline      # 调用子程序 aline 输出第 G 行的摆子情况
        ADDI     G      1      # G = G + 1, 下标增加 1
        LT       G      A      # 关系运算 G < A, 判断下标是否越界
        CJMP     loop3      # 为真, 则转到标号 loop3 输出下一行, 否则继续执行
        LOADI    C      10     # 将换行符 '\n' 存入 C
        OUT      C      15     # 输出换行符

        POP      D          # 解个数的最低位出栈, 存入 D
        LT       Z      D      # 关系运算 0(Z) < D
        CJMP     loop4      # 为真, 表明解的个数是奇数, 转至 loop4, 否则执行
下一行
        IN       D      0      # 等待从键盘输入一个字符。这样实现了每输出两个解
程序停顿一下的效果

loop4:  RET              # 返回

        # 输出棋盘一行的摆子情况
aline:  POP      B          # 皇后所在的列号出栈到 B 中
        LOADI    A      8      # A = 8
        LOADI    D      0      # D = 0, 列号初始化为 0
        LOADI    E      32     # E = 32, 空格字符
a1:     LOADI    C      49     # C = 49, 数字字符 1
        EQU      B      D      # 关系运算 B == D
        CJMP     a2          # 为真, 表明 D 列上放置皇后, 转至 a2, 否则执行下
行
        SUBI     C      1      # C = C - 1, 得到数字字符 0
a2:     OUT      C      15     # 将 C 中的数字字符输出
        OUT      E      15     # 输出一个空格字符
        ADDI     D      1      # D = D + 1, 列号增加 1
        LT       D      A      # 关系运算 D < A
        CJMP     a1          # 为真, 表明此行还没有全部输出, 转至 a1 继续输出
        LOADI    C      10     # 将换行符 '\n' 存入 C
        OUT      C      15     # 输出一个换行符
        RET              # 返回

        # 在第 B 行 A 列上放置皇后, 在后面各行皇后能够攻击到的单元位置上做标记
tag:    POP      B          # 行号出栈到 B
        POP      A          # 列号出栈到 A
        ADD      G      B      Z      # G = B( + Z), 即将行号存入 G, 作为数组下标

```

	STOREB	A	sltn	# 将列号存入解数组元素 sltn[G], 表示第 B 行 A 列放置
皇后	LOADI	C	8	# C = 8
	LOADI	D	0	# D = 0
t1:	ADDI	D	1	# D = D + 1
	ADD	E	B D	# E = B + D, B 行后的第 D 行行号存入 E
	LT	E	C	# 关系运算 E < C, 行号小于 8
	CJMP	t2		# 为真则转到标号 t2
	RET			# 否则返回
				# 在 E 行 A 列单元上用行号 B 做标记
t2:	PUSH	A		# A 入栈
	PUSH	B		# B 入栈
	PUSH	E		# 需做标记的行 E 入栈
	CALL	mark		# 调用子程序 mark, 在第 E 行上做标记
				# 在 E 行 A-D 列单元上用行号 B 做标记
	SUB	F	A D	# F = A - D
	LT	F	Z	# 关系运算 F < 0
	CJMP	t3		# 列号小于 0, 转至 t3
	PUSH	F		
	PUSH	B		
	PUSH	E		
	CALL	mark		
				# 在 E 行 A+D 列单元上用行号 B 做标记
t3:	ADD	F	A D	# F = A + D
	LTE	C	F	# 关系运算 8 <= F
	CJMP	t1		
	PUSH	F		
	PUSH	B		
	PUSH	E		
	CALL	mark		
	JMP	t1		
				# 在 C 行 A 列单元上用 B 做标记
mark:	POP	C		
	POP	B		
	POP	A		
	LOADI	D	8	# D = 8
	MUL	E	C D	# E = C * D
	ADD	G	A E	# G = A + E, G 为 C 行 A 列单元数组元素下标
	LOADB	E	cell	# E = cell[G]
	LOADI	F	1	# F = 1

SAL	B	F	B	# B = F << B
NOR	E	E	B	# E = E ^ B
STOREB	E	cell		# 将 E 存入 cell[G]
RET				# 返回

Assembly:

```

    IN A 0      #十以内加减法
    IN E 0
    IN B 0
    SUBI A 48
    SUBI B 48
    ADD A A B
    PUSH A
    CALL prnt   # 调用子程序 prnt
    HLT         # 终止程序运行

                # 输出一个整数
prnt:  POP A
        LOADI B 10      #递减值
        LOADI C 1      #mask
loop1:  MUL C B C
        LTE C A          # c<=A
        CJMP loop1
        LOADI F 1      #退出值
loop2:  DIV C C B
        DIV D A C
        MUL E D C
        SUB A A E
        ADDI D 48
        OUT D 15
        EQU C F
        NOTC
        CJMP loop2
        RET

```