LAB 5: FACTIONAL

实验目的

本次实验的目的在于通过手工实现来理解中断驱动的输入/输出如何中断正在运行的程序，执行中断服务历程，并且执行完返回到被中断的程序停止的地方，继续执行程序。实验中使用键盘作为中断正在运行的程序的输入设备。

实验原理

本次实验可以拆分成以下几个部分来实现：

1. 被中断的程序，它的功能是持续打印学号

   这一部分较为简单，直接将学号（后面需要加上一个空格，这样打印出的学号才会有间距）存储在内存中，然后使用一个循环来打印即可。过程中需要使用具有一个DELAY功能的循环，防止打印得太快。

2. 读取输入的字符，判断它是否为十进制数字，若为十进制数字还需判断它是否为8或9

   这一部分的逻辑是：首先打印输入的字符，因为最后不论字符是否为十进制数，都需要打印相关的信息，信息中需要打印字符。然后将输入的字符减去57（9的ASCII码，因为无法用立即数表示，所以需要保存在内存中），若结果大于0，则输入的字符不是十进制数，打印`is not a decimal digit.\n`（这也需要保存在内存中）；若结果小于0，则再将输入的字符加上9（相当于与0的ASCII码做比较），若结果小于等于0，则输入的字符不是ASCII码，同样打印`is not a decimal digit.\n`，否则输入的字符是十进制数，打印`is a decimal digit.\n`。

   然后还需要再打印一次输入的字符，然后再判断输入的字符是否大于等于8（再减去8判断结果是否非负即可），若是，则需要打印`! is too large for LC-3.\n`，否则打印`! =`，输出等式，然后调用`FACTIONAL`子程序计算n!并输出，最后再输出换行符`\n`即可。

3. 计算十进制数n的n!

   对于这一功能，有三种方法实现：

   - 因为需要计算的n只可能在0-7之间，所以可以直接将这8个结果存在内存里面，然后根据n来选择结果即可。
   - 利用公式，采用循环的方式计算
   - 利用公式，采用递归的方式计算

   其中第一种方法最为简单，但是我又觉得这样似乎太过取巧，考虑到本次实验中的目的就在于在中断处理中计算`n!`，并且讲义中要求`calculating the result`，所以我觉得计算过程应该也是本次实验的一个考察点，所以我最终分别使用了第一，二种方法来计算：

   使用两个寄存器来分别保存n和n!，这里我选择了`R2`和`R1`，然后让`R2`递减，若不为1，则将其与`R1`相乘，否则计算完成，返回。

乘法的实现如下：

使用两个寄存器`R3`和`R4`，分别保存`R2`和`R1`中的值，然后令`R3`递减，若`R3`不为0，则将`R4`加到`R1`上，否则结束循环。

4. 将计算的结果转换成ASCII码打印输出

因为本次实验中最多需要计算`7! = 5040`，所以只需要依次输出计算结果的千百十个位即可。

以千位为例，只需要将`R1`（我将计算结果存在`R1`中）不断减去1000（可以将-1000存在内存中）直至其小于0，在这过程中用一个初始化为0的寄存器`R0`在每次减去1000且`R1`仍非负时自加1。最终`R0`中存储的即为结算结果的千位。若其为0，则不需要打印；若其不为0，则将其加上48（0的ASCII码）后打印输出即可。

需要注意的是，对于最高位千位，如果为0可以直接不打印。但对于百位和十位，则需要判断其前面有无非零位，若有，则打印；若无，则不打印。这里可以使用一个寄存器初始化为0，当有一个非零位时，就将这个寄存器存储的值加1，这样就可以根据寄存器中的值来判断是否要打印0。而对于个位，即使它前面没有非零位，当其为0时，也需要打印，所以个位不需要判断，直接输出即可，并且由于计算出千百十位后，`R1`中存储的就是个位的数值，所以直接将其加上0的ASCII码放到`R0`中再`OUT`即可。

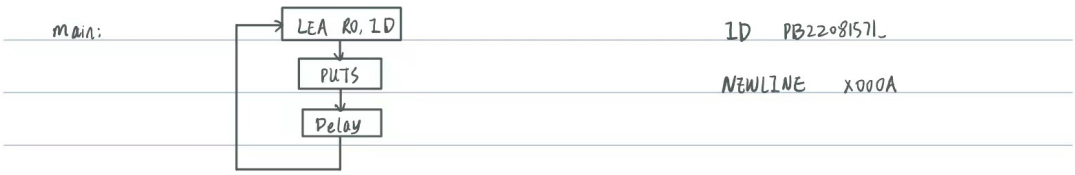
需要注意的是，如果选择直接将答案实现存储在内存中，就不需要这一步了，直接根据结果选择打印的字符串即可。

实验过程

流程图

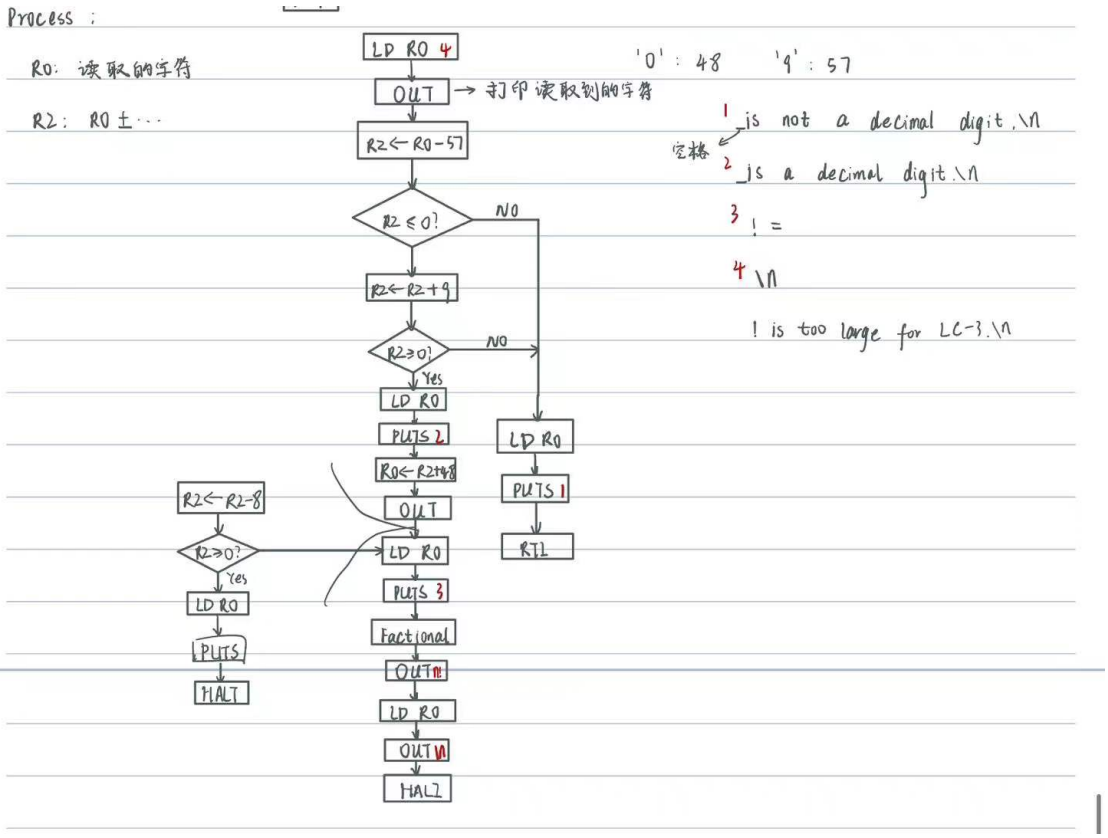
先依次画出上面所说的几个模块的流程图如下：

1. 打印学号部分：

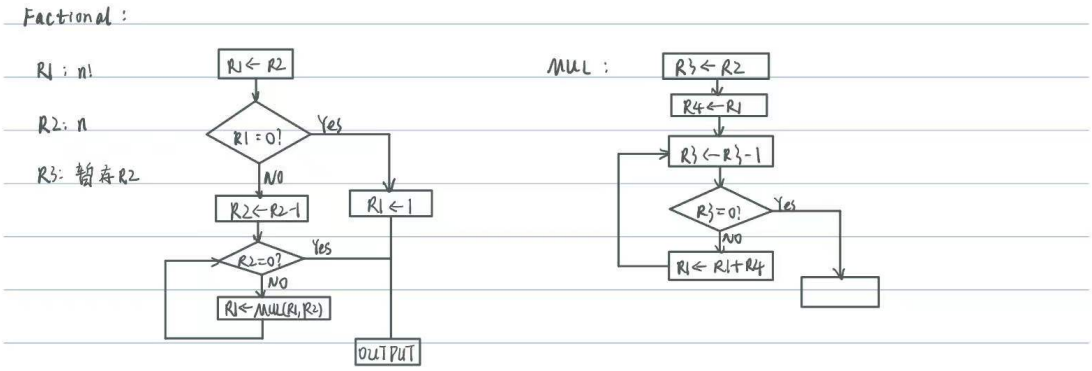


因为DELAY部分教程中已经给出了，所以流程图中我就没有再将其具体实现画出来。

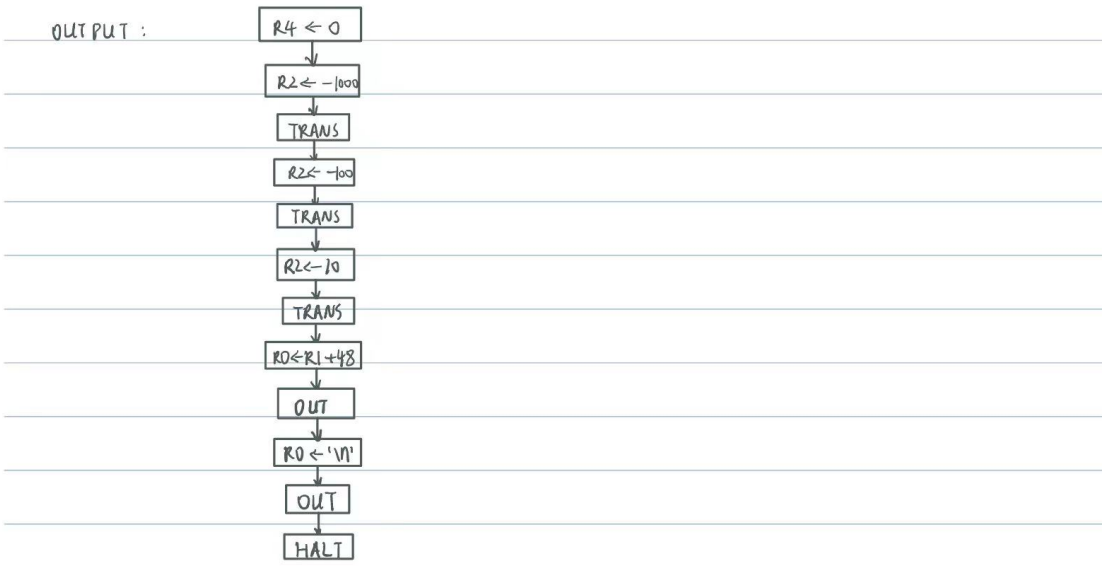
2. 判断部分：

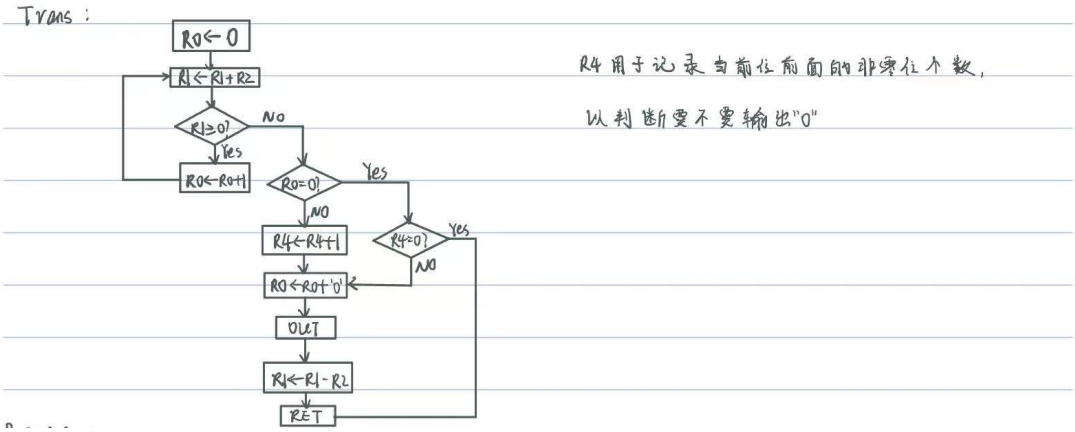

Process :
R0: 读取的字符
R2: R0 土···
LD R0 4
OUT
→ 打印读取到的字符
R2←R0-57
R2≤0?
NO
R2←R2+9
R2≥0?
NO
Yes
LD R0
PUTS 2
R0←R2+48
OUT
LD R0
PUTS 3
Factional
OUT
LD R0
OUT
HALT
LD R0
PUTS 1
RTI
R2←R2-8
R2≥0?
Yes
LD R0
PUTS
HALT
'0' : 48
'9' : 57
1 is not a decimal digit.\n
空格
2 is a decimal digit.\n
3 ! =
4 \n
! is too large for LC-3.\n

3. 计算部分


Factional :
R1 : n!
R2: n
R3: 暂存R2
R1←R2
R1=0?
Yes
NO
R2←R2-1
R1←1
R2=0?
Yes
NO
R1←MUL(R1,R2)
OUTPUT
MUL :
R3←R2
R4←R1
R3←R3-1
R3=0?
Yes
NO
R1←R1+R4

4. 打印部分：


OUTPUT :
R4←0
R2←-1000
TRANS
R2←-100
TRANS
R2←-10
TRANS
R0←R1+48
OUT
R0←'\n'
OUT
HALT



其中trans的功能为计算相应的位并决定是否要打印，若要打印，则加上0的ASCII码后再`OUT`即可。

需要注意的是，对于个位，因为一定需要打印，并且计算到最后`R1`中存储的就是个位，所以最后直接将其加上0的ASCII码存到`R0`中打印即可。

5. 因为根据结果来选择打印的部分就只需要一个一个比较，选择需要打印的字符串的地址即可，逻辑上很简单，所以我就没有再画流程图。

汇编代码

根据流程图可以写出汇编代码如下（在写代码对流程图中的部分过程做了优化，例如设置了几个Label方便跳转，省去重复的过程等）：

```
            .ORIG   x800
            ; (1) Initialize interrupt vector table.
            LD      R0, VEC
            LD      R1, ISR
            STR     R1, R0, #0

            ; (2) Set bit 14 of KBSR.
            LDI     R0, KBSR
            LD      R1, MASK
            NOT     R1, R1
            AND     R0, R0, R1
            NOT     R1, R1
            ADD     R0, R0, R1
            STI     R0, KBSR

            ; (3) Set up system stack to enter user space.
            LD      R0, PSR
            ADD     R6, R6, #-1
            STR     R0, R6, #0
            LD      R0, PC
            ADD     R6, R6, #-1
            STR     R0, R6, #0
            ; Enter user space.
            RTI

VEC         .FILL   x0180
ISR         .FILL   x1000
KBSR        .FILL   xFE00
```

```
MASK        .FILL   x4000
PSR         .FILL   x8002
PC          .FILL   x3000
            .END



            .ORIG   x3000
            ; *** Begin user program code here ***
            ;循环打印学号
LOOP        LEA     R0, ID                          ;获得字符串首地址
            PUTS                                    ;打印学号+空格
            LD      R1, COUNT                       ;获得COUNT，用于执行DELAY功
能
REP         ADD     R1, R1, #-1
            BRp     REP
            BRnzp   LOOP                            ;经过DDELAY后继续打印学号+空
格


COUNT       .FILL       #25000                      ;用于设置延迟时间的长短
ID          .STRINGZ    "PB22081571 "               ;student id(后面需要有空格
来分隔打印的学号)

            ; *** End user program code here ***
            .END



            .ORIG   x3FFF
            ; *** Begin factorial data here ***
FACT_N      .FILL   xFFFF
            ; *** End factorial data here ***
            .END



            .ORIG   x1000
            ; *** Begin interrupt service routine code here ***

            LD      R0, NEWLINE
            OUT                                     ;打印换行符
            GETC                                    ;读取输入的字符到R0中
            OUT                                     ;打印读取到的字符
            LD      R3, ASCIININEN
            ADD     R2, R0, R3                      ;判断输入字符的ASCII码是否小
于'9'
            BRp     ERROR                           ;若大于'9'，则不是十进制数
            ADD     R2, R2, #9                      ;判断输入的字符是否大于等
于'0'
            BRn     ERROR                           ;若小于0，也不是十进制数
            LEA     R0, RIGHTN                      ;否则是十进制数，打印信息
            PUTS
            LD      R3, SAVEn                       ;获得n的存储地址x3FFF放在R3
中
```

```
            STR     R2, R3, #0                          ;存储读取到的n于x3FFF中
            LD      R3, ASCIIZERO                       ;获得0的ASCII码，由于无法用
立即数表示，故存在内存中
            ADD     R0, R2, R3                          ;恢复R0
            OUT                                         ;打印R0
            ADD     R0, R2, #-8                         ;判断输入的数字是否大于等于8
            BRzp    LARGE
            ;BR      FACTIONAL2                          ;如果使用将结果存在内存中的
方式，此时可直接跳转
            LEA     R0, EQUAL                           ;打印等式
            PUTS
            BRnzp   FACTIONAL                           ;计算n!


LARGE       LEA     R0, LARGEN
            PUTS                                        ;打印信息
            BRnzp   OVER

ERROR       LEA     R0, ERRORN
            PUTS                                        ;打印信息

RETURNI     RTI
            ; *** End interrupt service routine code here ***


;
; 计算n!
; 其实这里也可以直接将相应的结果存在内存中，再根据n的值取出
; 但是考虑到本次实验应该需要考察这方面的处理，所以这里采用计算的方式获得结果
;
FACTIONAL   AND     R1, R1, #0
            ADD     R1, R1, R2
            BRz     ZERO0                               ;0!单独处理
LOOP1       ADD     R2, R2, #-1                         ;计算R2!
            BRz     OUTPUT
            AND     R3, R3, #0
            ADD     R3, R3, R2                          ;R3存储R2的值
            AND     R4, R4, #0
            ADD     R4, R4, R1                          ;R4用于保存R1的值，因为在计算
时R1会改变
LOOP2       ADD     R3, R3, #-1                         ;计算R4*R2,结果存于R1中
            BRz     LOOP1
            ADD     R1, R1, R4
            BRnzp   LOOP2
ZERO0       ADD     R1, R1, #1                          ;若R1 = 0，则直接将其加1再输
出即可

            ;计算完成后打印结果，因为n! <= 7! = 5040，所以只需依次打印结果的千百个十位即可
OUTPUT      AND     R4, R4, #0                          ;R4用于标记当前位前面有没有非
零位，以判断要不要打印0
            LD      R2, THOUSANDN                       ;获得-1000
            JSR     TRANS                               ;计算字符的千位，转换成ASCII
码并输出
            LD      R2, HUNDREDN                        ;获得-100
```

```
            JSR     TRANS                           ;计算字符的百位，转换成ASCII
码并输出
            LD      R2, TENN                        ;获得-10
            JSR     TRANS                           ;计算字符的十位，转换成ASCII
码并输出
;因为这时R1为个位，不管是否为0，是第几个0，都需要打印，将其加上0的ASCII码放到R0中打印即可
            LD      R3, ASCIIZERO
            ADD     R0, R1, R3
            OUT
            ;JSR     OUT1
            LD      R0, NEWLINE                     ;最后打印换行符
            OUT

OVER        HALT



;将数字转化为ASCII码并输出
TRANS       AND     R0, R0, #0
LOOP3       ADD     R1, R1, R2                      ;R1 <- R1 + R2
            BRn     JUDGE1                          ;当R1为负时，此时R0中存储的即
为所求位的数值
            ADD     R0, R0, #1
            BRnzp   LOOP3
JUDGE1      ADD     R0, R0, #0                      ;设置条件码与R0相关
            BRnz    JUDGE2                          ;如果R0等于0则需要判断要不要
打印
            ADD     R4, R4, #1
OUT1        LD      R3, ASCIIZERO                   ;获得0的ASCII码
            ADD     R0, R0, R3                      ;将R0转换成对应的ASCII码以输
出
            OUT
            BRnzp   RESTORE                         ;因为前面将R1减到了负值，即多
加了一个R2，所以需要再减去R2

JUDGE2      ADD     R4, R4, #0                      ;设置条件码
            BRp     OUT1                            ;如果R4不为0，则说明当前位前
面有非零位，需要打印0

RESTORE     NOT     R2, R2
            ADD     R2, R2, #1                      ;将R2取反加1获得相反数
            ADD     R1, R1, R2                      ;前面将R1减到了负值，故还需要
再加上-R2
            RET



FACTIONAL2  ADD     R2, R2, #0
            BRp     ONE!
            LEA     R0, ZERO
            BR      OUTRES
ONE!        ADD     R2, R2, #-1
            BRp     TWO!
            LEA     R0, ONE
            BR      OUTRES
```

```
TWO!        ADD     R2, R2, #-1
            BRp     THREE!
            LEA     R0, TWO
            BR      OUTRES
THREE!      ADD     R2, R2, #-1
            BRp     FOUR!
            LEA     R0, THREE
            BR      OUTRES
FOUR!       ADD     R2, R2, #-1
            BRp     FIVE!
            LEA     R0, FOUR
            BR      OUTRES
FIVE!       ADD     R2, R2, #-1
            BRp     SIX!
            LEA     R0, FIVE
            BR      OUTRES
SIX!        ADD     R2, R2, #-1
            BRp     SEVEN!
            LEA     R0, SIX
            BR      OUTRES
SEVEN!      LEA     R0, SEVEN


OUTRES      PUTS
            BRnzp   OVER



NEWLINE     .FILL       x000A
ERRORN      .STRINGZ    " is not a decimal digit.\n"    ;输入的字符不是十进制数
RIGHTN      .STRINGZ    " is a decimal digit.\n"        ;输入的字符是十进制数
LARGEN      .STRINGZ    "! is too large for LC-3.\n"    ;
EQUAL       .STRINGZ    "! = "                          ;打印等式的中间部分
THOUSANDN   .FILL       #-1000
HUNDREDN    .FILL       #-100
TENN        .FILL       #-10
ASCIININEN  .FILL       #-57                            ;9的ASCII码的负值
ASCIIZERO   .FILL       #48                             ;0的ASCII码
SAVEn       .FILL       x3FFF
ZERO        .STRINGZ    "! = 1\n"
ONE         .STRINGZ    "! = 1\n"
TWO         .STRINGZ    "! = 2\n"
THREE       .STRINGZ    "! = 6\n"
FOUR        .STRINGZ    "! = 24\n"
FIVE        .STRINGZ    "! = 120\n"
SIX         .STRINGZ    "! = 720\n"
SEVEN       .STRINGZ    "! = 5040\n"

            .END
```

对于以下这段代码：

```
            ;BR       FACTIONAL2                          ;如果使用将结果存在内存中的
方式，此时可直接跳转
            LEA       R0, EQUAL                           ;打印等式
            PUTS
            BRnzp     FACTIONAL                           ;计算n!
```

现在是采用第二种方法来计算，如果将第一行的注释去掉，将后面三行注释掉，就是采用第一种方法来打印结果。

实验中遇到的问题

1. 使用 PUTS 打印字符串时，起初我像往常从内存中取一个数一样使用了 LD 命令获取字符串，但是这样却无法打印，原因在于 LD 指令只能获得一个数，也就是获得了字符串的第一个字符，而 PUTS 命令需要的是字符串的首地址，所以这里需要用的是 LEA 命令。
2. 在输出结果时起初我选择了直接输出，忽略了打印字符需要用ASCII码，后来增加了 TRANS 这一子程序来实现这一功能。

实验结果

以下为部分运行结果截图（第一种和第二种方法运行的结果都一样）：

输入3:

```
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
3 is a decimal digit.
3! = 6


--- Halting the LC-3 ---
```

输入7:

```
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571
7 is a decimal digit.
7! = 5040
```

依次输入8和9：

```
PB22081571
8 is a decimal digit.
8! is too large for LC-3.


--- Halting the LC-3 ---

PB22081571 PB22081571 PB22081571 PB22081571
PB22081571
9 is a decimal digit.
9! is too large for LC-3.
```

依次输入a,b,4

```
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571
a is not a decimal digit.
PB22081571 PB22081571 PB22081571
b is not a decimal digit.
PB22081571 PB22081571 PB22081571
4 is a decimal digit.
4! = 24
```

依次输入h，5：

```
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571 PB22081571
PB22081571 PB22081571 PB22081571
h is not a decimal digit.
PB22081571 PB22081571 PB22081571
5 is a decimal digit.
5! = 120
```

输入0，1:

```
PB22081571 PB22081571 PB22081571 PB22081571
0 is a decimal digit.
0! = 1


--- Halting the LC-3 ---

PB22081571 PB22081571 PB22081571 PB22081571
1 is a decimal digit.
1! = 1
```

以上计算结果均正确且输出格式符合要求，故结果正确!