```
TITLE Getkey.asm
include        console.mac     ; Book's Macros for I/O
include        io.mac          ; My Macros for I/O

DIM   EQU   4                  ; Number of COLUMNS
CR    EQU   0dh                ;ASCII code for carriage return
LF    EQU   0ah                ;ASCII code for line feed

DOSSEG
.MODEL SMALL

.STACK 100h

DATA

CODES   DB 01h,02h,03h,0Fh,04h,05h,06h,0Eh,07h,08h,09h,0D
h,0Ah,00,0Bh,0Ch
; scan codes for the keys on keypad.

.CODE

EXTRN   NEWLINE:NEAR, DECOUT:NEAR, EXIT:NEAR,
LEDCNT:NEAR, DELAY:NEAR

Start:
      mov     ax, @DATA
      mov     ds, ax

      call    INIT
      mov     ax, 0
      push    ax
      call    LEDCNT
```

```
TOP:
      mov    cx, 4
NROW:
      push   cx
      call   CHKCOL                 ; ax = CHKCOL(cx)
      cmp    al, 0FFh
      jz     DLABEL
      push   ax
      call   RVRT                   ; ax = RVRT(ax)
      dec    cx
      push   cx
      push   ax
      call   WHICH                  ; ax = WHICH(cx, ax)
      push   ax
      call   LEDCNT                 ; LEDCNT(ax)
      call   DELAY
      jmp    TOP
DLABEL:
      loop   NROW
      jmp    TOP

      call   EXIT                   ; EXIT()
```

How does the Keypad work ?

```
        mov     dx, 302H
        mov     al, 00001110B       ; enable row 0
        out     dx, al              ; by outputting al to port
        in      al, dx              ; get input from port
; check input
        cmp     al, 11101110B       ; key in row 0
                                    ; col 0 was pressed?
        je      key1                ; key 1

        cmp     al, 11011110B       ; key 2?
        je      key2

        cmp     al, 10111110B       ; key 3?
        je      key3

        cmp     al, 01111110B       ; key F?
        je      keyf
. . . . . .

key1:
        mov     ax, 1
        push    ax
        call    LEDCNT              ; LEDCNT(ax)
```

More Recursion:

Writing recursive programs means thinking recursively.

That is:

There must be a way of breaking up a problem into a smaller version of the same problem.

Factorial example:

fact(n) = n * fact(n - 1)

here fact(n - 1) is the smaller problem

There must also be a way to stop the recursion. In other words, as you express larger problems in terms of ever smaller problems, you must stop at some point.

Factorial example:

fact(1) = 1

So when you express a larger problem in terms of 1! you know the answer to 1! and therefore stop the recursion.

```
int fact(int n)
{
        if (n == 1)
          return 1;
        else
          return (n * fact(n - 1));
}
```

Look at the addfirst example:

recursion:
addfirst(n) = n + addfirst(n - 1)

stopping criterio:
addfirst(1) = 1

```
int addfirst(int n)
{
        if (n == 1)
          return 1;
        else
          return (n + fact(n - 1));
}
```

Think about the INC function

**Restricting yourself to positive numbers
can you define addition in terms of inc**

let us write INC as the 1+ function in C

add(n,m) = 1+ add(n, m - 1)

add(n,0) = n

```
int add(int n, int m)
{
        if(m == 0)
          return n;
        else
          return (1+ add(n, m - 1));
}
```

subtraction can be done the same way with dec

With addition and subtraction possible,

multiplication and division are also possible.

So in principle:

**If your microprocessor only had the inc/dec instructions
you could write s/w to implement:**

add, sub, mul, and div

Carrying this thought a bit further,

Is there a minimum set of functions so that any microprocessor (Computer) can be made equivalent to others ?

Back to recursion:

String length:

You were asked to write a macro to find string length.

Can you write a recursive procedure to do this?

string = firstchar|restofstring

strlength(s) = 1+ strlength(restofstring)

strlength(s) = 1 if restofstring is null