```
/*
A demonstration program for GCBASIC.
-----------------------------------------------------------------
-------------------------
Mechanical switches play an important and extensive
role in practically every computer, microprocessor and
microcontroller application.

Mechanical switches are inexpensive, simple and
reliable. However, switches can be very noisy
electrically.
The apparent noise is caused by the closing and opening
action that seldom results in a  clean electrical
transition.
The connection makes and breaks several, perhaps even
hundreds, of times before the final switch state
settles.
The problem is known as switch `bounce`. Some of the
intermittent activity is due to the switch contacts
actually bouncing off each other.
Imagine slapping two billiard balls together. The hard
non-resilient material doesn’t absorb the kinetic
energy of motion.
Instead, the energy dissipates over time and friction
in the bouncing action against the forces push the
billiard balls together.
Hard metal switch contacts react in much the same way.
Also, switch contacts are not perfectly smooth. As the
contacts move against each other, the imperfections and
impurities on the surfaces cause the electrical
connection to be interrupted.
The result is switch `bounce`.

The consequences of uncorrected switch bounce can range
from being just annoying to catastrophic.
For example, imagine advancing the TV channel, but
instead of getting the next channel, the selection
```

skips one or two.

This is a situation a designer should strive to avoid.

Switch bounce has been a problem even before the earliest computers. The classic solution involved filtering, such as through a resistor-capacitor circuit, or through re-set-
table shift registers (see Figure 3-4 and Figure 3-5, PDF 40001296b.pdf).
These methods are still effective but they involve additional cost in material, installation and board real estate.

Debouncing in software eliminates these additional costs. This is the purpose of this demonstration.

****

The demonstration uss an interrupt that happens when the SWITCH is pushed. You can test the switch state in the ISR, and, determine the correct action.

The ISR sets a variable `Pushed` = 1. And, the main code examines this variable.

@author   EvanV
@license   GPL
@version   1.00
@date   2024-08-17
****
*/

```
#chip 16F887
#option explicit


/*
           ------------PORTA---------------
    Bit#:  -7---6---5---4---3---2---1---0---
    IO:    ----------------------------AN0--
    IO:    ---------------------------------

           ------------PORTB----------------
    Bit#:  -7---6---5---4---3---2---1---0---
    IO:    ----------------------------SW---
    IO:    ---------------------------------

           ------------PORTC----------------
    Bit#:  -7---6---5---4---3---2---1---0---
    IO:    ---------------------------------
    IO:    ---------------------------------

           ------------PORTD----------------
    Bit#:  -7---6---5---4---3---2---1---0---
    IO:   -DS8-DS7-DS6-DS5-DS4-DS3-DS2-DS1--
    IO:    ---------------------------------

*/

DIR PORTD OUT
PORTD.7 = 1

DIR PORTB.0 In
#define SWITCH PORTB.0

Dim ADCValue as Byte
Dim Pushed as Bit

On Interrupt ExtInt0 Call ISR
```

```
Do

    //Wait for the switch to change state - the
variable Pushed is set in the Sub ISR, so, we wait here
until the button is pressed
    If Pushed = 1 Then

        //We get here only when Pushed = 1 (set in the
ISR), so, we have to set Pushed to 0 to clear the
switch event
        Pushed = 0

        // Ensure the Carry bit is clear
        Set C OFF

        //Rotate the port to the right, shift the bits
of the port to the right
        ROTATE PORTD RIGHT

        //Did the rotate set the carry bit? If, yes,
set the bit to 1
        IF C = 1 Then PORTD.7 = 1
    End If

Loop

End


Sub ISR
        Pushed = 1
        Do
            // wait 5 ms
            wait 5 ms

        // Check the SWITCH is still pressed.  Loop,
```

```
else if released the ISR will exit.
        Loop While SWITCH = 1


End Sub
```