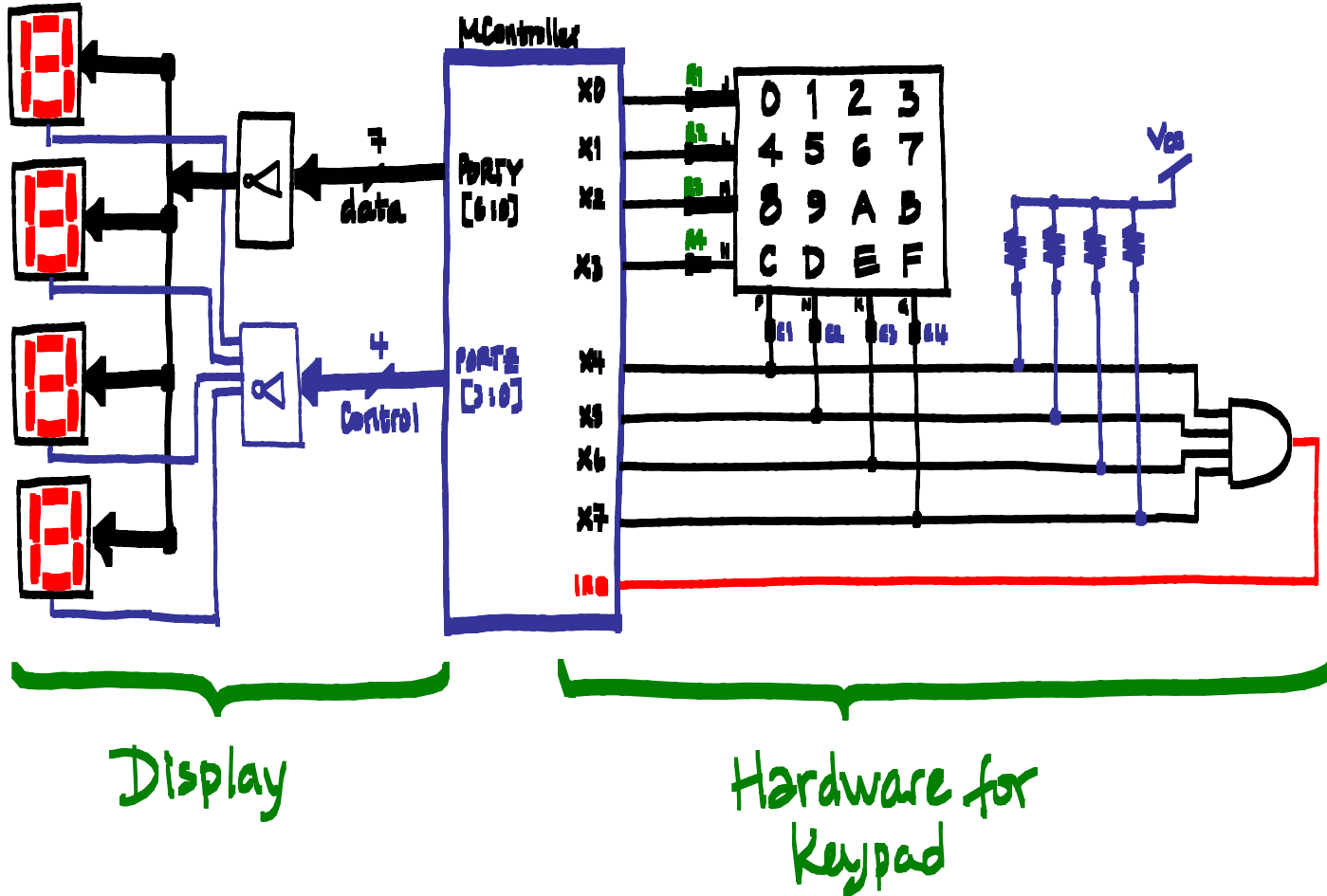


What's next ?

| Phase | Description | Outcome |
|-------|--------------------------------------------------------------------------------------------|---------------------------------------------------|
| ∅ | Intro to the problem | UNDERSTANDING. |
| 1 | Hardware discussion Which components How to connect them Anticipate for debugging | H1a Proven HW. |
| 2 | Software discussion Flow diagram/architecture variable definition CODING | H1b: flow diagram & Implementation strategy |
| 3 | Implementation / Lab demo | Working solution |
| 4 | Documentation | Design Notebook. |

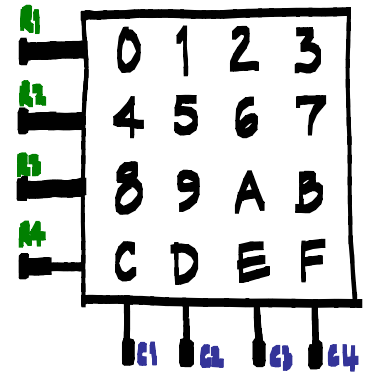
The main program

Flow diagram

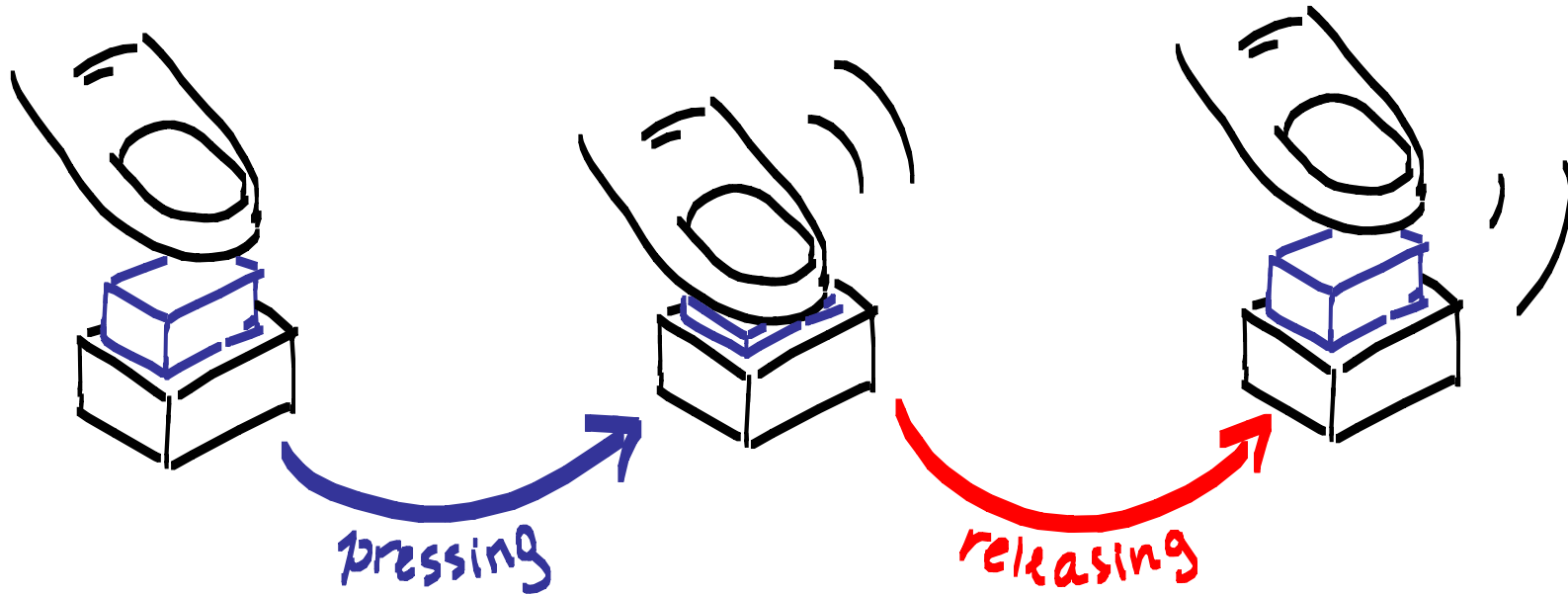


IRQ's INTERRUPT SERVICE ROUTINE (ISR) OPERATION

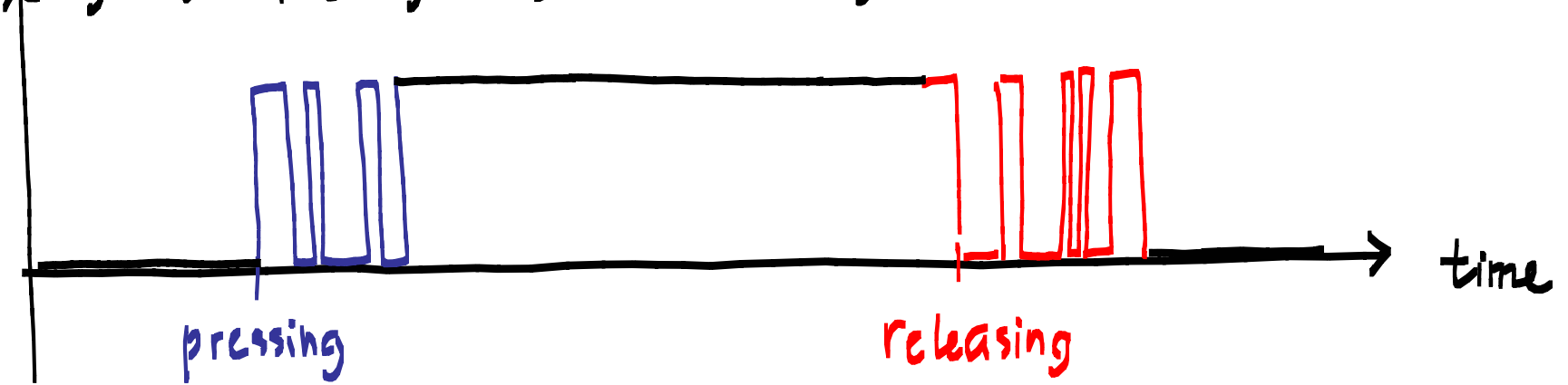
Once the interrupt has been requested and starts being serviced, the μC must decode the key that was pressed. If two keys were pressed at the same time, nothing should happen.



THE KEY BOUNCING PROBLEM



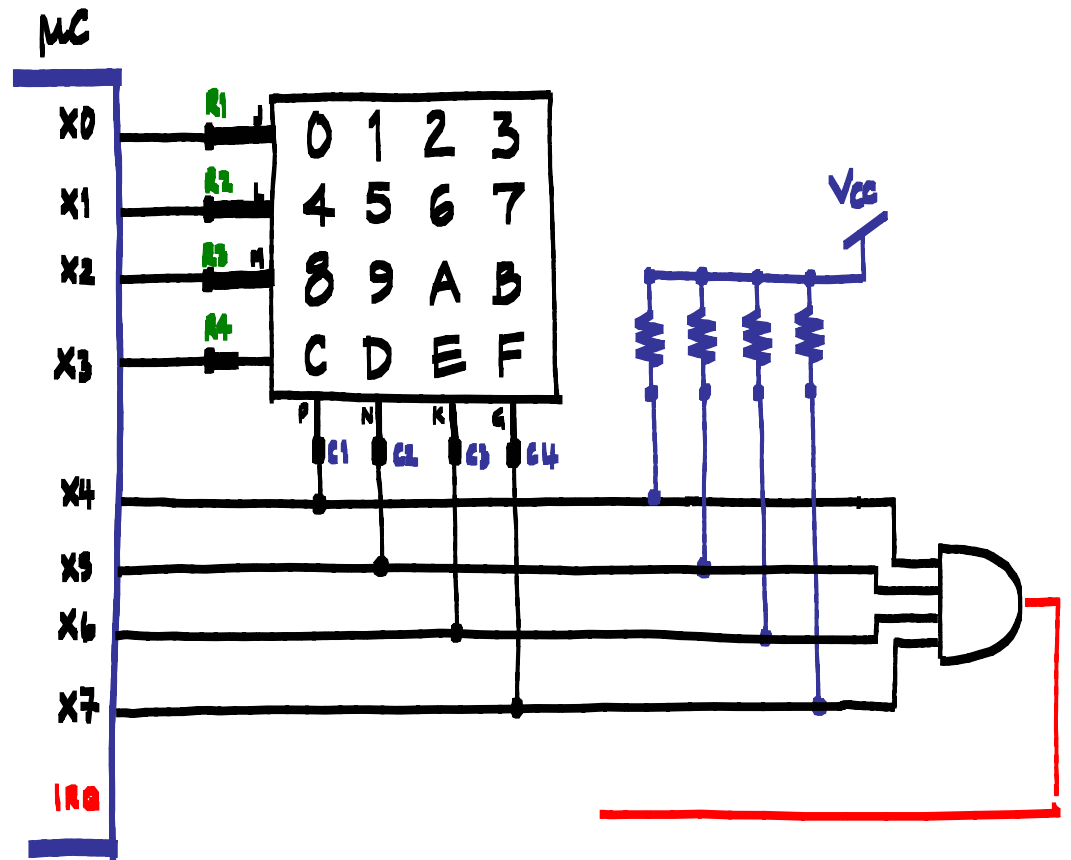
Signal transferred by the switch (assuming LOW when SWITCH IS OPEN)



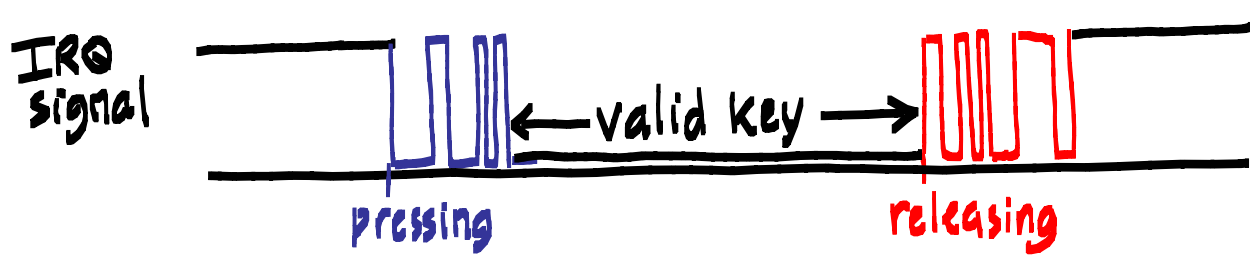
How do we cope with the BOUNCING problem?

(1) Hardware solution

(2) Software solution



Filtering key bouncing via SOFTWARE



Assuming falling edge IR signal

cheap

WHAT

HOW

WHY NOT

elegant

```

// Event counter using the IRQ pin
#include <hidef.h>                /* for EnableInterrupts macro */
#include "derivative.h"          /* include peripheral declarations */
#include "M68DEMO908GB60.h"

unsigned char counter=0;
unsigned char store=0;

// This is the IRQ interrupt servicing routine
void interrupt VectorNumber_Virq IRQ_isr()
{
    IRQSC_IRQACK = 1;             // acknowledge IRQ interrupt (clear IRQF)
    counter++;                   // increment counter everytime IRQ pin
                                // detects a active edge
}

void main(void)
{
    PTADD = 0;                   //set port A for input (switches)
    PTAPE = 0xf0;                //set the Pull-Up-Enable for port A
    PTFDD = 0x0f;                //set port F for output (LEDs 1 through 4)
    PTFD = 0xff;                 // Turn all LEDs OFF

    IRQSC = bIRQPE | bIRQIE;     // IRQ pin enabled, IRQ interrupt enabled
                                // default falling edge detection
    // IRQSC = bIRQPE | bIRQIE | bIRQEDG // IRQ pin enabled, inter enable, rising edge

    EnableInterrupts;           // enable interrupts (CCR:I = 0)
    IRQSC_IRQIE = 1;            // enable IRQ interrupt

    while (1)
    {
        __RESET_WATCHDOG(); /* feeds the dog */

        store = ~(counter);     // invert counter, LEDs 1-4 contain bits 0-3 of counter
        PTFD = store;           // PTFD = negated counter (all bits inverted)
    }
}

```

(3) CONFIGURE THE IRQ Status Control Register (IRQSC)

IRQ Pin Enable
 1 = IRQ pin function is enabled.
 0 = IRQ pin function is disabled.

IRQ Interrupt Enable
 1 = Hardware interrupt requested whenever IRQF = 1.
 0 = Hardware interrupt requests from IRQF disabled (use polling)

IRQ Flag
 1 = IRQ event detected.
 0 = No IRQ request.

IRQ Mode
 1: EDGE & LEVEL
 0: EDGE ONLY

FALLING or RISING EDGE
 1 = IRQ is rising edge or rising edge/high-level sensitive.
 0 = IRQ is falling edge or falling edge/low-level sensitive.

IRQSC

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|--------|--------|---|--------|-------|--------|--------|-------|--------|
| Read: | 0 | 0 | IRQEDG | IRQPE | IRQF | 0 | IRQIE | IRQMOD |
| Write: | [Grey] | | | | [Grey] | IRQACK | | |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[Grey] = Unimplemented or Reserved

IRQ Acknowledge
 (write 1 to clear IRQF)

```

// DEMO9S08QG8 IRQ-controlled Led flasher
#include <hidef.h>                /* for EnableInterrupts macro */
#include "derivative.h"          /* include peripheral declarations */
#include "M68DEMO908GB60.h"

unsigned char blinking=0;

// This is the IRQ interrupt servicing routine
void interrupt VectorNumber_Virq IRQ_isr()
{
    IRQSC_IRQACK = 1;           // acknowledge IRQ interrupt (clear IRQF)
                                // IRQ interrupt is disabled below to avoid bouncing noise:
    IRQSC_IRQIE = 0;           // disable IRQ interrupt
    blinking = !blinking;      // change blinking state
}

void main(void)
{
    unsigned int temp;
    PTADD = 0;                 //set port A for input (switches)
    PTAPE = 0xf0;              //set the Pull-Up-Enable for port A
    PTFDD = 0x0f;              //set port F for output (LEDs 1 through 4)

    PTFD = 0xff;              // ALL LEDs OFF

    IRQSC = bIRQPE | bIRQIE;    // IRQ pin enabled, IRQ interrupt enabled
                                // default falling edge detection
    // IRQSC = bIRQPE | bIRQIE | bIRQEDG // IRQ pin enabled, inter enable, rising edge

    EnableInterrupts;          // enable interrupts (CCR:I = 0)
    while (1)
    {
        __RESET_WATCHDOG(); /* feeds the dog */
        if (blinking)
        {
            LED4 = ON; //PTFD_PTFD3 = 0           // LED4 = on
            for (temp=50000; temp; temp--);     // wait for a while
            LED4 = OFF; //PTFD_PTFD3 = 1         // LED4 = off
            for (temp=50000; temp; temp--);     // wait for a while
        }
        IRQSC_IRQIE = 1;          // enable IRQ interrupt
    }
}

```

In line Assembly for "C"

Listing 11.1 strlen() definition

```
int strlen (char *str)
/** The 'str' character array is passed on the stack. strlen returns
length of 'str'.
This procedure assumes len(str) is smaller than 256! */
{
  __asm {
    LDHX str ; load pointer
    CLRA ; init counter
    BRA test ; go to test
  loop:
    AIX #1 ; increment pointer
    INCA ; increment counter
  test:
    TST 0,X ; not end of string?
    BNE loop ; next char
    CLRX ; return value in X:A(see later)
  };
  /* C statements could follow here */
}
```

