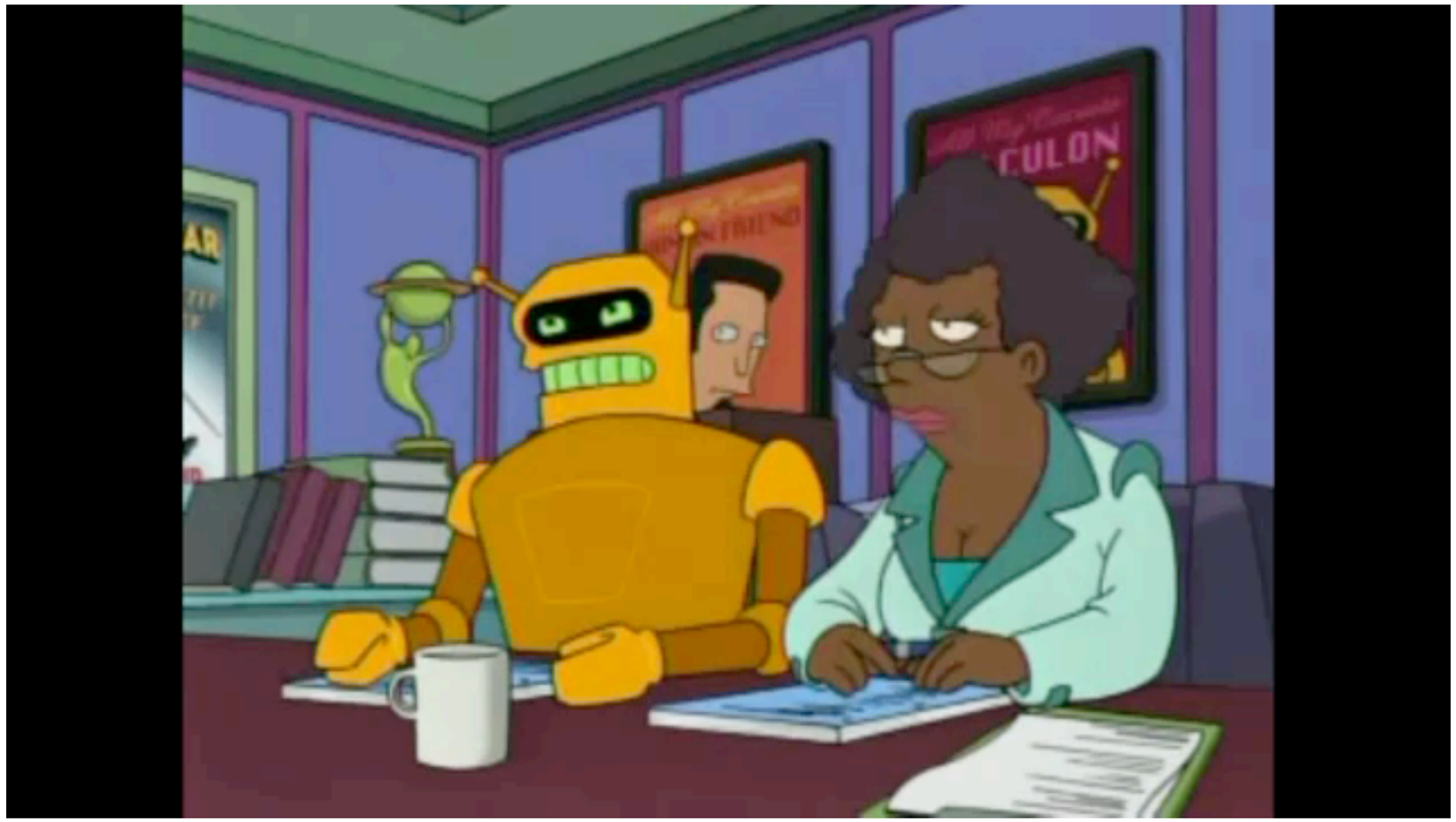
Jimi Hendrix

Ana Vidovic

response to exam,
homework, or class?



1. a natural feeling...that should abate...
2. evaluations
3. class next week

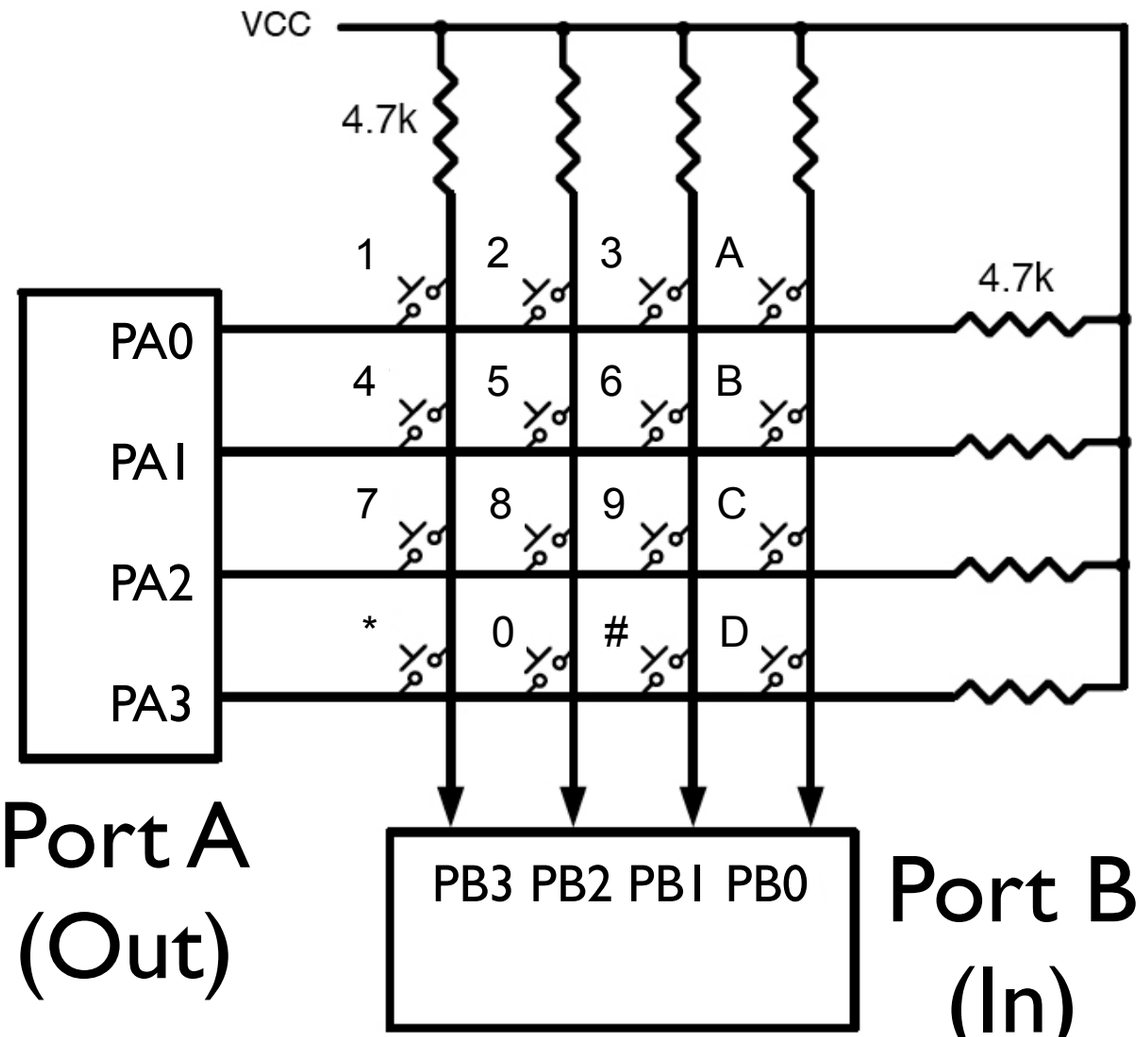
Keypads & Seven Segment Displays

ECE 3710

A travel agent told I could spend 7 nights in HAWAII no days just nights.

- Rodney Dangerfield

finding a key press

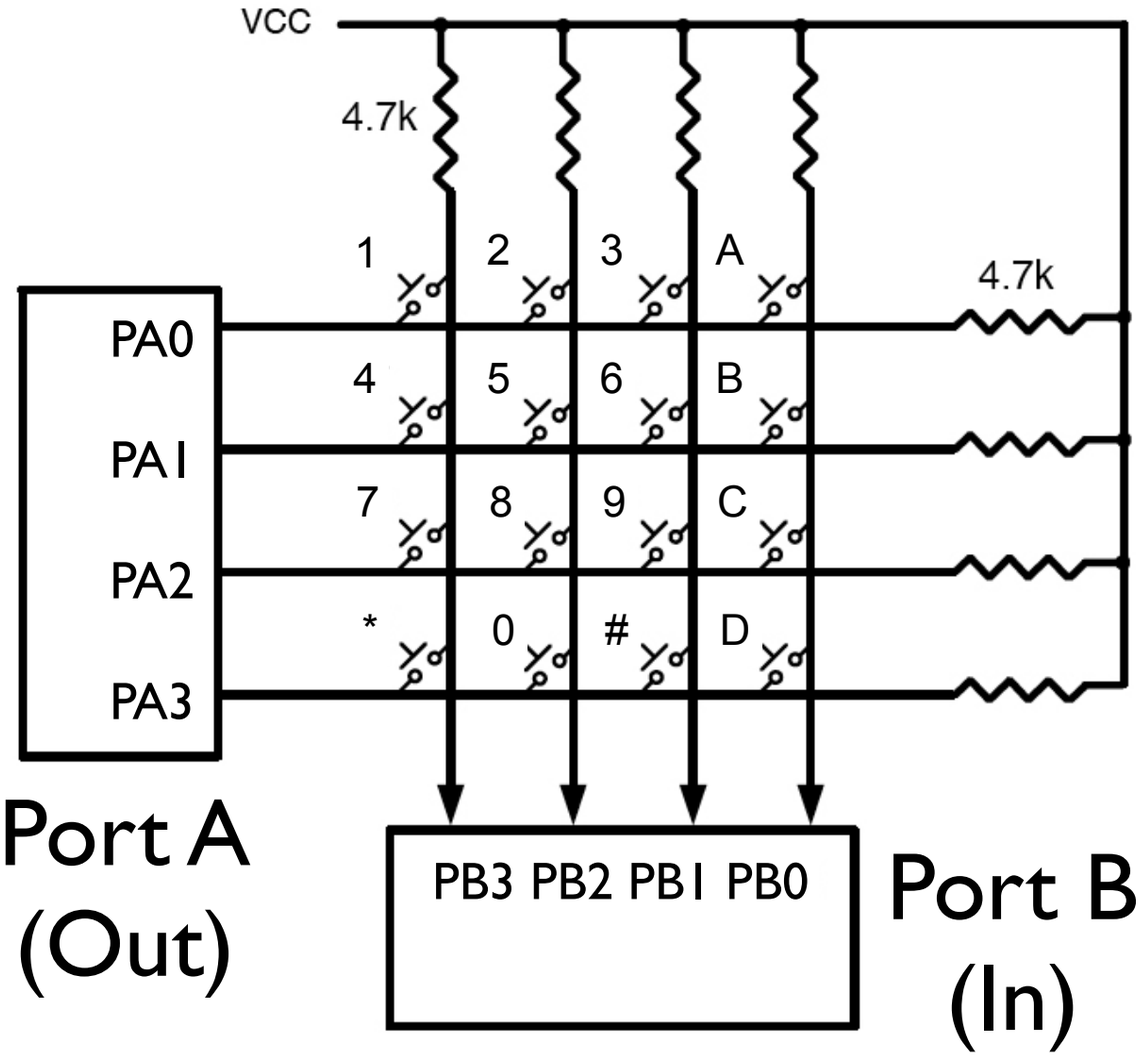



```
void keyScan()
{
//PA[3:0]=1;
//PB[3:0]=1;
 for(i=0;i<4;i++)
   {
     PA[i]=0;
for(j=0;j<4;j++)
  if(PB[j]==0)
    keyPress(i,j);

PA[i]=1;
   }
}
```

wouldn't this be nice

how to know if key has been pressed?

(method one)



0. both ports = 0xF

1. `PA=0x0`

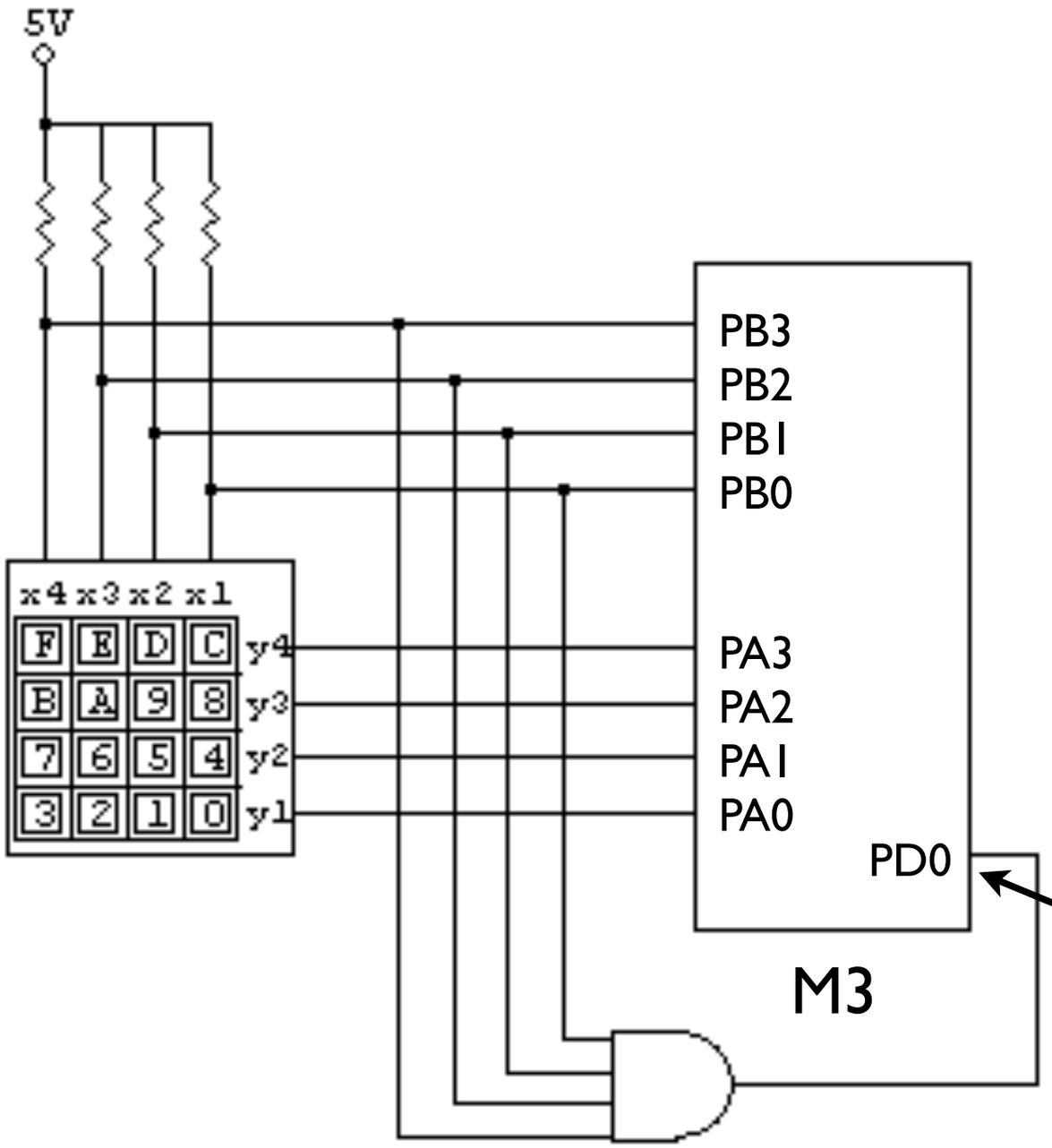
2. if `PB < 0xF`

then a key is pressed

still have to figure out which one, though

how to know if key has been pressed?

(method two)



0. both ports = 0xF

1. `PA&=0b11110000`

2. interrupt if a key is pressed

(any of PB[3:0]=0 then AND outputs 0)

configured for external interrupt

still have to figure out which one, though

Q: display key press using ISR and 7SD (C)

Q: display key press using ISR and 7SD (C)

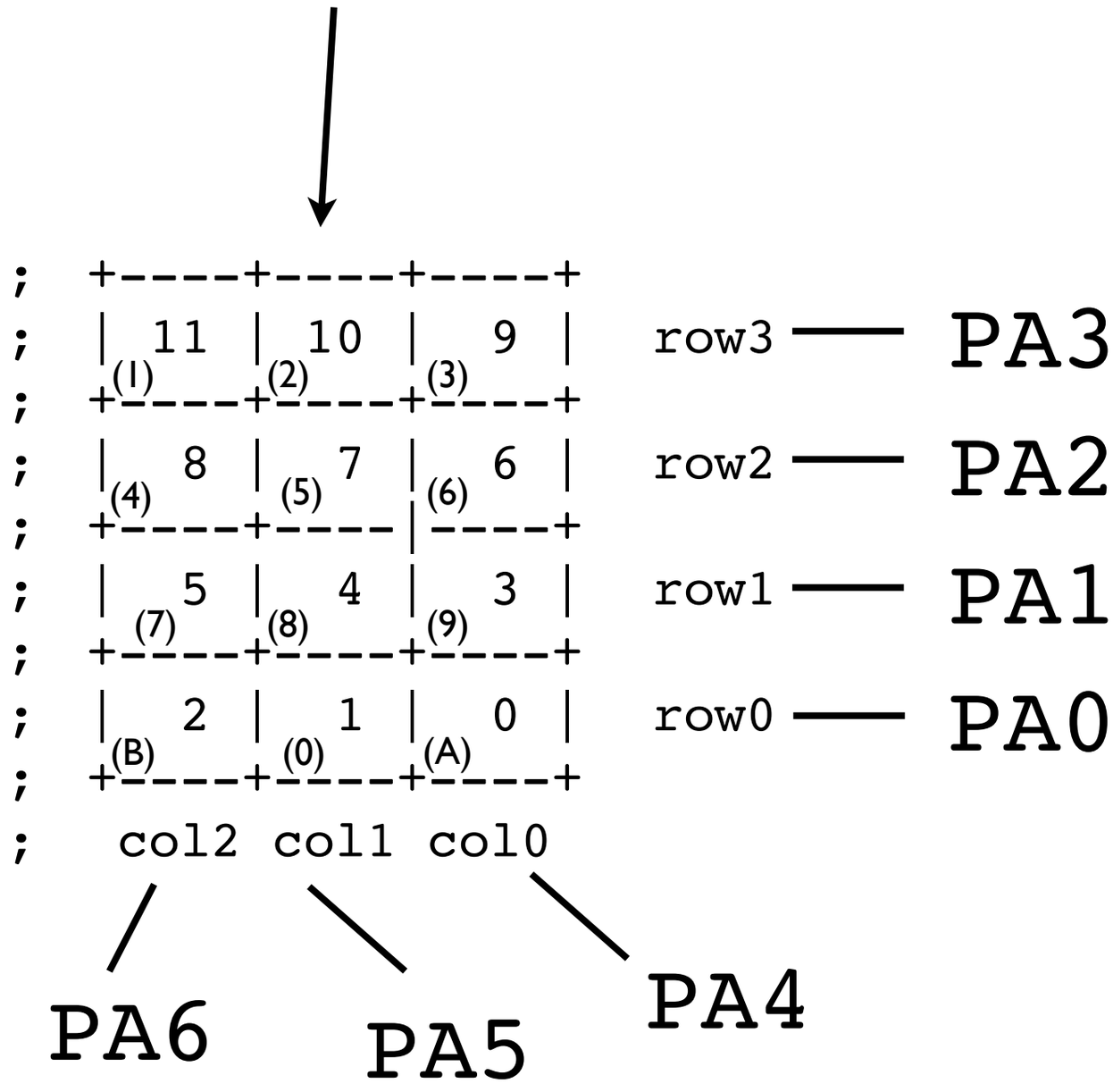
A:

1. keyScan in C
2. ISR

ex: finding a key press (C)

4x3 generic keypad

(numbers refer to key position)



to find key:

use array with bit combination for each key

key zero

(port looks like this)

```
keys={0b11101110,
      0b11011110,
                 ...,
      0b10110111};
```

ex: finding a key press (C)

```
// port a pins
unsigned char KEYPAD __attribute__((at(0x400043FC)));
// port b pins
unsigned char SSD __attribute__((at(0x400053FC)));

//what port looks like when each key is pressed (assume MSB is always one)
unsigned char KEYS[12] = {0xEE,0xDE,0xBE,0xED,0xDD,0xBD,0xEB,0xDB,0xBB,0xE7,0xD7,0xB7};
//what does port look like when key's row is grounded: needed to scan keypad
unsigned char RKEYS[12] = {0xFE,0xFE,0xFE,0xFD,0xFD,0xFD,0xFB,0xFB,0xFB,0xF7,0xF7,0xF7};
//bit combinations to display key on seven segment display: A,0,b,9,8,7,6,5,4,3,2,1
unsigned char SSDCH[12] = {0x88,0xC0,0x83,0x98,0x80,0xF8,0x82,0x92,0x99,0xB0,0xA4,0xF9};

int main(void)
{
    PABInit();

  while(1)
    keyScan();
}
```

procedure:

1. ground row (RKEYS)
2. see if port matches key (KEYS)

ex: finding a key press (C)

```
// port a pins
unsigned char KEYPAD __attribute__((at(0x400043FC)));
// port b pins
unsigned char SSD __attribute__((at(0x400053FC)));

//what port looks like when each key is pressed (assume MSB is always one)
unsigned char KEYS[12] = {0xEE,0xDE,0xBE,0xED,0xDD,0xBD,0xEB,0xDB,0xBB,0xE7,0xD7,0xB7};
//what does port look like when key's row is grounded: needed to scan keypad
unsigned char RKEYS[12] = {0xFE,0xFE,0xFE,0xFD,0xFD,0xFD,0xFB,0xFB,0xFB,0xF7,0xF7,0xF7};
//bit combinations to display key on seven segment display: A,0,b,9,8,7,6,5,4,3,2,1
unsigned char SSDCH[12] = {0x88,0xC0,0x83,0x98,0x80,0xF8,0x82,0x92,0x99,0xB0,0xA4,0xF9};

void keyScan()
{
    unsigned char i;

    for(i=0;i<12;i++)
    {
        KEYPAD = RKEYS[i]; //ground row

        if(KEYPAD == KEYS[i]) //see if we've found the key
        {
            SSD = SSDCH[i]; //output char to SSD
            KEYPAD = 0xFF; //done scanning, return
            break;
        }
    }
}
```

Q: level or edge triggering

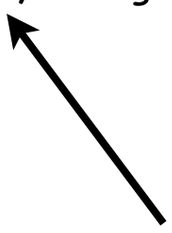
Q: level or edge triggering

A: edge

(if level then have to wait for key to be released)

display key press using ISR and 7SD (C)

```
// port a pins
unsigned char KEYPAD __attribute__((at(0x400043FC)));
// port b pins
volatile unsigned char SSD __attribute__((at(0x400053FC)));

//what port looks like when each key is pressed (assume MSB is always one)
unsigned char KEYS[12] = {0xEE,0xDE,0xBE,0xED,0xDD,0xBD,0xEB,0xDB,0xBB,0xE7,0xD7,0xB7};
//what does port look like when key's row is grounded: needed to scan keypad
unsigned char RKEYS[12] = {0xFE,0xFE,0xFE,0xFD,0xFD,0xFD,0xFB,0xFB,0xFB,0xF7,0xF7,0xF7};
//bit combinations to display key on seven segment display: A,0,b,9,8,7,6,5,4,3,2,1
unsigned char SSDCH[12] = {0x88,0xC0,0x83,0x98,0x80,0xF8,0x82,0x92,0x99,0xB0,0xA4,0xF9};

// get key and output it to seven segment display
void keyScan();
// interrupt when a key is pressed: call keyScan from there to find key
void GPIOPortD_Handler(void)

int main(void)
{
  PABDInit();

  KEYPAD = 0xF0; //ground all rows and wait for interrupt

  while(1);
}
```



remember, to scan for any key being pressed

```
void GPIOPortD_Handler(void)
{
    // acknowledge interrupt
    PD[0x41C] = 1;

    // bring row pins high then low will trigger another interrupt; disable
    PD[0x410] = 0;

    // find the key
    keyScan();

    //get ready for next key press
    KEYPAD = 0xF0;

    //re-enable interrupts
    PD[0x410] = 1;
}
```

important

```
void keyScan()
{
    unsigned char i;

    for(i=0;i<12;i++)
    {
        KEYPAD = RKEYS[i]; //ground row

        if(KEYPAD == KEYS[i]) //see if we've found the key
        {
            SSD = SSDCH[i]; //output char to SSD
            break;
        }
    }
}
```

don’t reset port here

Inline Assembly

ECE 3710

```
int main(void)
{
   int i;
   int cnt = 0;

   for(i=0;i<255;i++)
   {
      cnt++;
   }
}
```

Q: how to create an infinite loop?

(by inserting assembly into loop)

```
    24:          int cnt = 0;
    25:
0x000001C8 2200      MOVS     r2,#0x00
    26:          for(i=0;i<1;i++)
    27:          {
0x000001CA 2100      MOVS     r1,#0x00
0x000001CC E001      B        0x000001D2
    28:              cnt++;
    29:          }
0x000001CE 1C52      ADDS     r2,r2,#1
0x000001D0 1C49      ADDS     r1,r1,#1
0x000001D2 2901      CMP      r1,#0xFF
0x000001D4 DBFB      BLT      0x000001CE
    30: }
0x000001D6 2000      MOVS     r0,#0x00
0x000001D8 4770      BX       lr
```

i stored in R1

infinite loop:

```
int main(void)
{
    int i;
    int cnt = 0;

    for(i=0;i<255;i++)
    {
        cnt++;
        __asm
        {
            subs r1,r1,#1
        }
    }
}
```

compiles to

```
    24:          int cnt = 0;
    25:
0x000001C8 2300      MOVS     r3,#0x00
    26:          for(i=0;i<255;i++)
    27:          {
0x000001CA 2100      MOVS     r1,#0x00
0x000001CC E002      B        0x000001D4
    28:              cnt++;
    29:              __asm
    30:              {
0x000001CE 1C5B      ADDS     r3,r3,#1
    31:                subs r1,r1,#1
    32:              }
    33:          }
0x000001D0 1E52      SUBS     r2,r2,#1
0x000001D2 1C49      ADDS     r1,r1,#1
0x000001D4 29FF      CMP      r1,#0xFF
0x000001D6 DBFA      BLT      0x000001CE
    34: }
0x000001D8 2000      MOVS     r0,#0x00
0x000001DA 4770      BX       lr
```

‘inline’ assembly

specified R1
compiler uses R2?

we can’t even make an infinite loop?

infinite loop:

```
int main(void)
{
    int i;
    int cnt = 0;

    for(i=0;i<255;i++)
    {
        cnt++;
        __asm
        {
            subs r1,r1,#1
        }
    }
}
```

compiles to

```
    24:          int cnt = 0;
    25:
0x000001C8 2300      MOVS     r3,#0x00
    26:          for(i=0;i<255;i++)
    27:          {
0x000001CA 2100      MOVS     r1,#0x00
0x000001CC E002      B        0x000001D4
    28:              cnt++;
    29:              __asm
    30:              {
0x000001CE 1C5B      ADDS     r3,r3,#1
    31:                subs r1,r1,#1
    32:              }
    33:          }
0x000001D0 1E52      SUBS     r2,r2,#1
0x000001D2 1C49      ADDS     r1,r1,#1
0x000001D4 29FF      CMP      r1,#0xFF
0x000001D6 DBFA      BLT      0x000001CE
    34: }
0x000001D8 2000      MOVS     r0,#0x00
0x000001DA 4770      BX       lr
```

‘inline’ assembly

specified R1
compiler uses R2?

compiler throws warning:

```
main.c(31): warning:  #1267-D: Implicit physical register R1 should be
                                                    defined as a variable
```

Inline assembly code for the compiler always specifies virtual registers. The compiler chooses the physical registers to be used for each instruction during code generation, and enables the compiler to fully optimize the assembly code and surrounding C or C++ code.

can't accidentally impede program
(not all compilers support this)

protection: compiler creates var for inline assembly unless you reference previously defined var

infinite loop (2):

```
int main(void)
{
    int i;
    int cnt = 0;

    for(i=0;i<255;i++)
    {
        cnt++;
        __asm
        {
            subs i,i,#1
        }
    }
}
```

compiles to

```
    24:          int cnt = 0;
    25:
0x000001C8 2200      MOVS     r2,#0x00
    26:          for(i=0;i<255;i++)
    27:          {
0x000001CA 2100      MOVS     r1,#0x00
0x000001CC E002      B        0x000001D4
    28:              cnt++;
    29:              __asm
    30:              {
0x000001CE 1C52      ADDS     r2,r2,#1
    31:                subs i,i,#
    32:              }
    33:          }
0x000001D0 1E49      SUBS     r1,r1,#1
0x000001D2 1C49      ADDS     r1,r1,#1
0x000001D4 29FF      CMP      r1,#0xFF
0x000001D6 DBFA      BLT      0x000001CE
    34: }
0x000001D8 2000      MOVS     r0,#0x00
0x000001DA 4770      BX       lr
```

'inline' assembly (ref vars not registers)

can access *vars* and manipulate in asm

Watchdog Timer(s)

ECE 3710

what if this happens
to your uC:

need someway to bring it back...

Figure 1: A typical watchdog setup

watchdog timer:

automatic reset after timer expiration(s)

why?

recover from faults:

1. time-critical systems
2. inaccessible systems

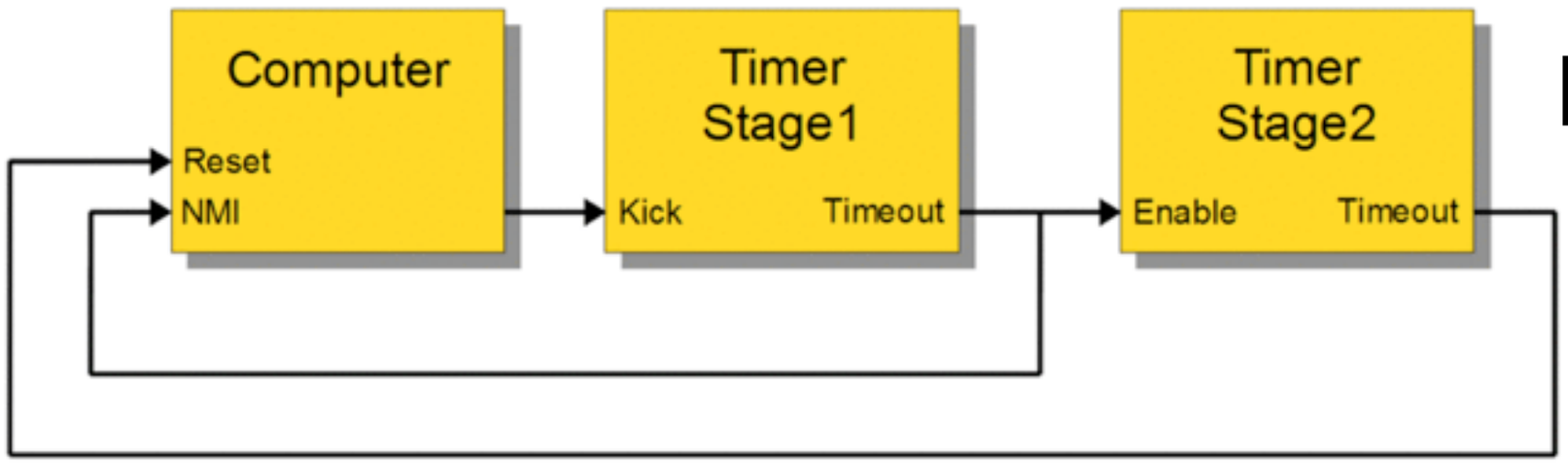
http://www.cs.princeton.edu/courses/archive/fall11/cos109/mars.rover.pdf

TI LM3S1968 watchdog timer

features:

1. 32-bit countdown timer
2. first expiration: interrupt
   (auto reload)
3. second expiration: reset
4. once enabled, only reset can disable
5. register locking
   (make it difficult to change)

guard against programming errors



TI LM3S1968 watchdog timer

configuration:

0. enable peripheral

1. set initial value

`WDTLOAD`, p373

2. expiration triggers reset or not

`WDTCTL`, p375

how we start timer →

3. enable interrupt

`WDTCTL`, p375

4. lock rest of watchdog registers

`WDTLOCK`, p380

5. enable watchdog interrupt in NVIC

to prevent changing reload value, e.g. ↑

locks `WDTCTL` ←

cannot unlock

(until reset)

ack of interrupt resets timer

TI LM3S1968 watchdog timer

```
M3CP EQU 0xE000E000
SYSCTL EQU 0x400FE000
WDT EQU 0x40000000

WDT_Init
  ; 0. enable clock
  ldr R1,=SYSCTL
  mov R0,#0x8 ;0x8=0b1000
  str R0,[R1,#0x100]
  ; 1. set initial watchdog value
  ldr R1,=WDT
  mov R0,#0xFF
  str R0,[R1,#0x0]
  ; 2/3. enable reset and interrupt (enables watchdog)
  mov R0,#0x3 ;0x3=0b11
  str R0,[R1,#0x8]
  ; 4. lock watch dog registers
  mov R0,#1
  str R0,[R1,#0xC00]
  ; 5. enable interrupts from wdt (bit 18)
  ldr R1,=M3CP
  mov R0,#0x4 ;0x4=0b100
  str R0,[R1,#0x102]
  bx LR
```

resets watchdog timer

```
WDT_Handler
  ; ack interrupt
  ldr R1,=WDT
  mov R0,#1
  str R0,[R1,#0xC]
  bx LR
```

a common sentiment
before the final exam:

this too will abate...

if you're at 'polly-wolly-doodle' and you feel like I'm asking you to play Hendrix:

Stevie Ray Vaughan

this is what I know you can do