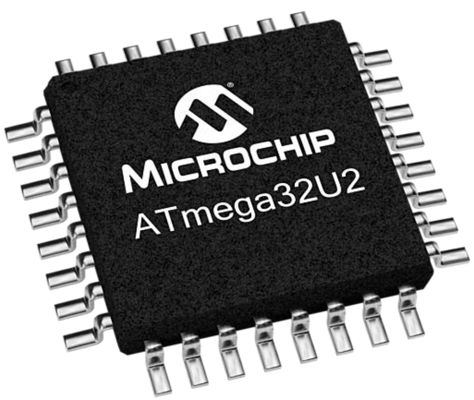
UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

ENCE260 Computer Systems: Embedded Systems

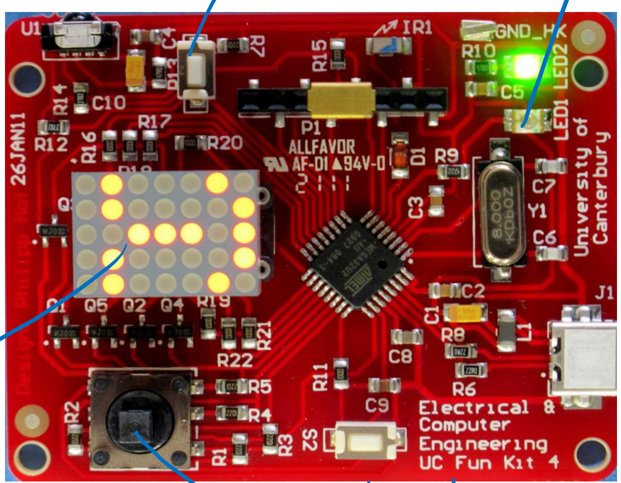
Lecture 2: Data Input/Output (I/O)

button

blue LED



kia ora koutou



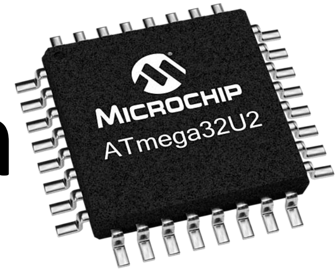
Richard Clare

13th September 2017

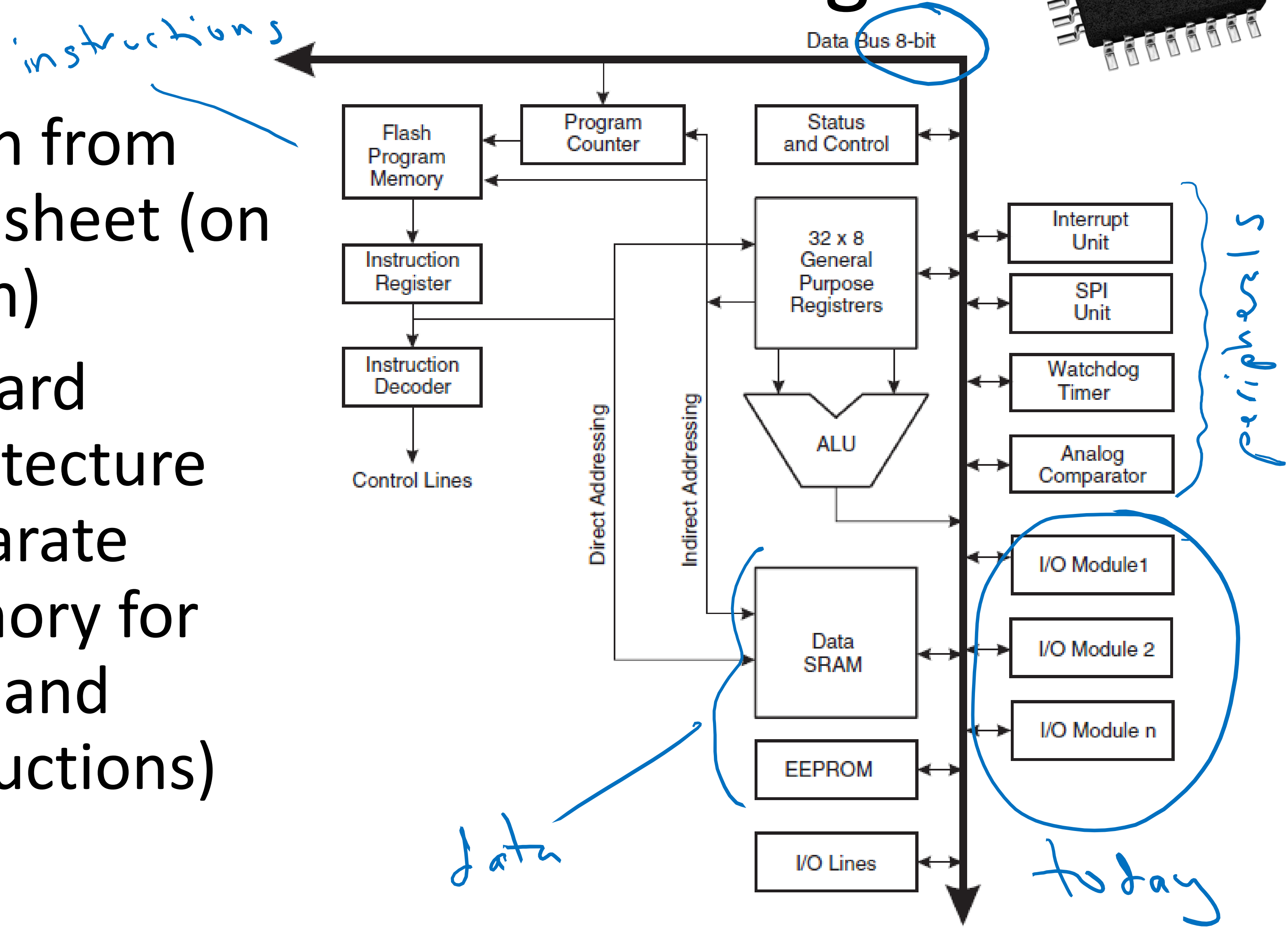
LED array

navigational switch

AVR Architecture Block Diagram



- Taken from data sheet (on Learn)
- Harvard architecture (separate memory for data and instructions)

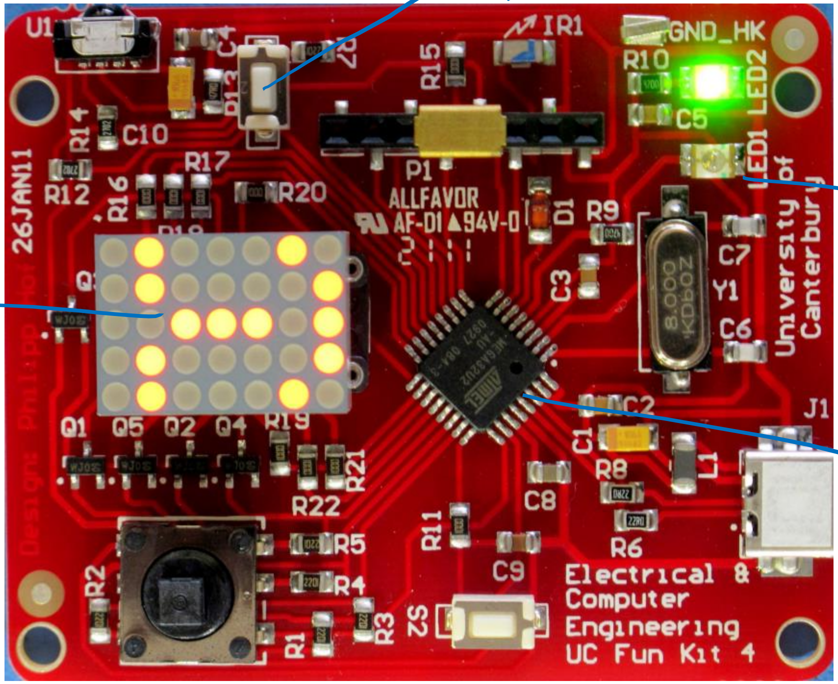


Programmable I/O

- All microcontrollers have programmable I/O (PIO) pins to interface with external hardware such as buttons and Light Emitting Diodes (LEDs)

push button

7X5 LED array



blue LED

microcontroller

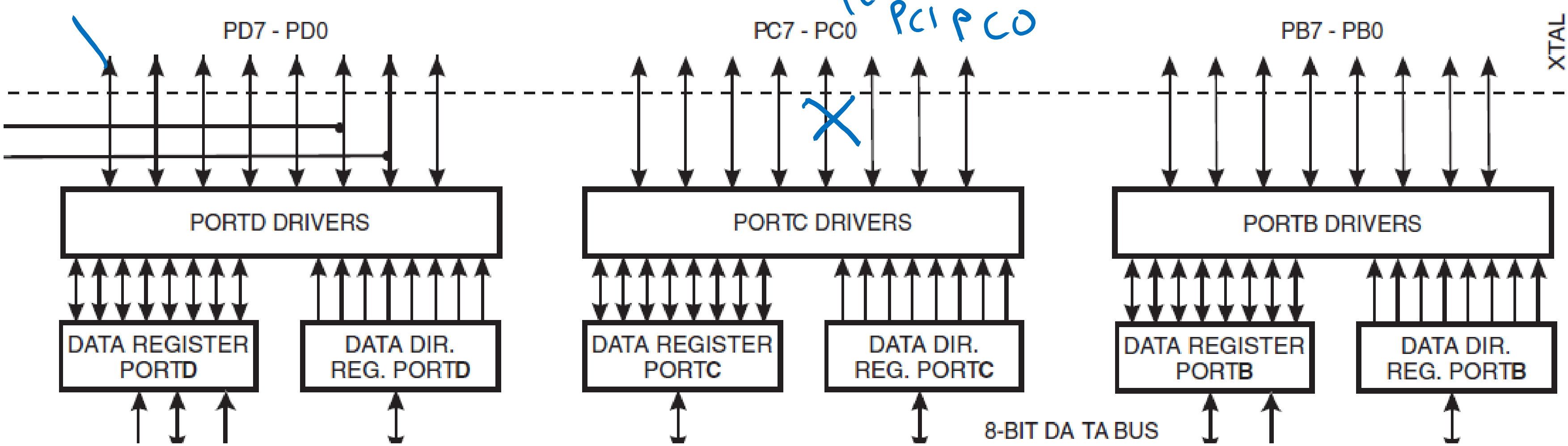
Data I/O on the ATMEGA32u2

- ATmega32u2 has 23 PIO pins split over three 8 bit ports: Port B, Port C, and Port D. PC3 is not used
- Each pin can be configured as an input (eg a button) or an output (eg an LED)

blue LED

button PD7

PC2 PC1 PC0



I/O Port Registers

- The I/O registers control the flow of input/output to/from the microcontroller.
- Each I/O Port has three 8-bit registers.
- For example, for Port x (B, C or D), these are:

DDRx - Sets the Data Direction for Port x. ie is this pin an input or an output? 0 = input 1 = output

PORTx – Sets the output state of each pin for Port x. ie set as high=1=5 Volts or low=0=0 Volts.

PINx – Gets the input state of each pin of Port x. ie is the input high=1=5 Volts or low=0=0 Volts?

Vcc supply voltage = 5V

PC2 PC1 PC0

Logic Levels

- Each PIO pin is connected to a logic buffer to determine whether it is logic high (Vcc=5V) or logic low (Ground = 0V)

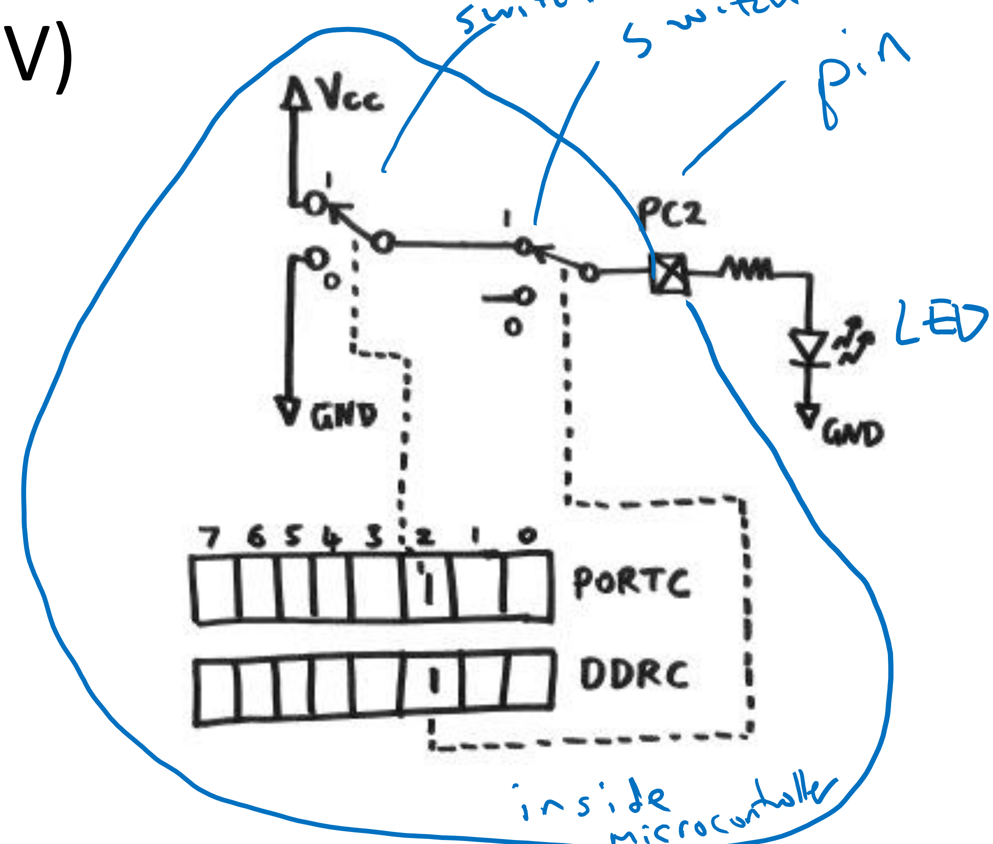
DDRC bit #2 = 1

⇒ output

PORTC C bit #2 = 1

⇒ we set the output to high = 5V

logic 1



Bitwise Operators

- In C, we use the following bitwise (manipulating bit by bit) operators:
  - ~ Bitwise complement (not) $\sim(0101) = 1010$
  - | Bitwise or $0101 \mid 1010 = 1111$
  - & Bitwise and $0101 \& 1010 = 0000$
  - ^ Bitwise exclusive or (xor) $0101 \wedge 1011 = 1110$
  - << Bitwise left shift $0001 << 2 = 0100$
  - >> Bitwise right shift

Bit manipulation

- The bitwise shift operators in C are useful for moving bits into registers.
- Typically we shift a 1 or 0 into an 8-bit I/O register while leaving the other bits unaltered.

eg 1 << 2

1 in 8 bit binary 0000 0001

1 << 2 0000 0100

= 4 in decimal

PC2 PC1 PC0

Bit operations

- For PC2 (Port C 3rd least significant bit):
- Setting a bit (ensures that it is 1):

```
PORTC = PORTC | (1<<2);
PORTC |= (1<<2);
```

- Clearing a bit (ensures that it is 0):

```
PORTC = PORTC & ~(1<<2);
PORTC &= ~(1<<2);
```

- Toggling a bit (changes bit status ie 0 ->1, 1->0):

```
PORTC = PORTC ^ (1<<2);
PORTC ^= (1<<2);
```

- Test a bit (return 1 if bit is set, else 0):

```
result = (PORTC & (1<<2));
```

Bit Masks

- We can manipulate multiple bits in a byte simultaneously. eg to set bits PC2 and PC3:

```
PORTC |= (1<<2) | (1<<3);
```

or

```
PORTC |= 12;
```

12 = 00001100

or

```
PORTC |= 0x0C;
```

hexadecimal

BIT macro

- We can also define a marco to make life simpler:

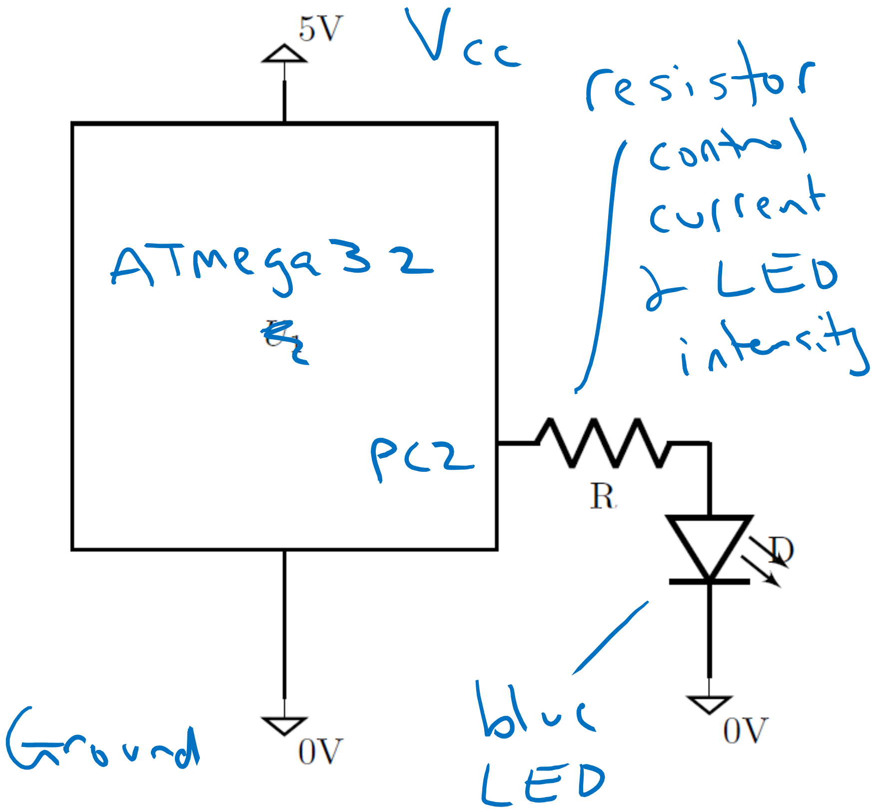
  #define BIT(X) (1 << (X))

- The macro called BIT has a single argument X. The proprocessor substitutes eg BIT(2) with (1<<(2))
- We can now set PC2 and PC3 using this macro

PORTC |= BIT(2) | BIT(3);

Writing to an I/O Port

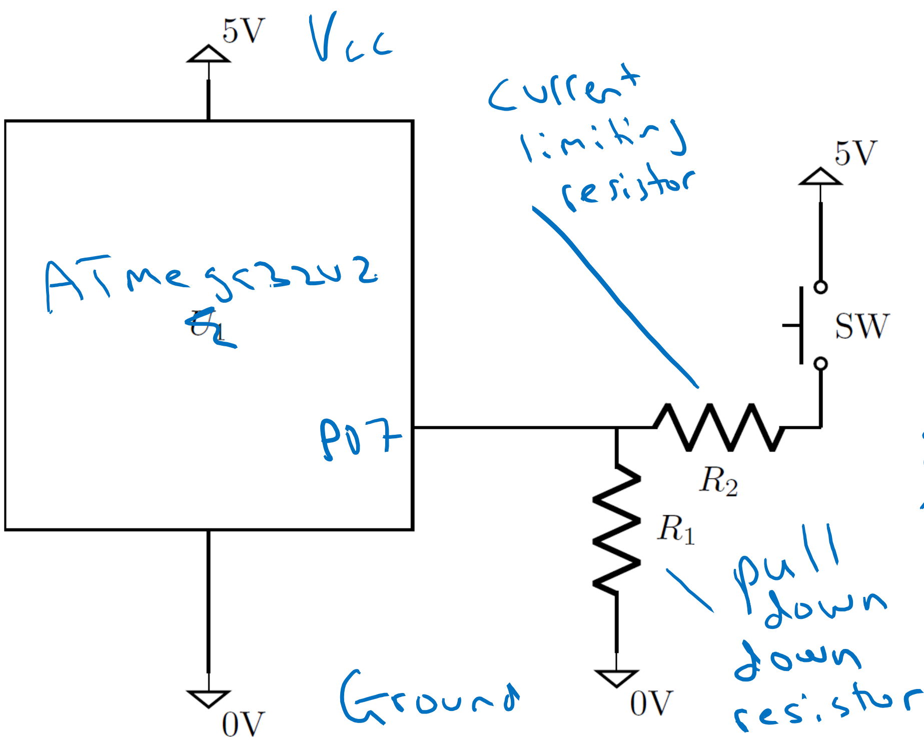
- For UCFK4, to drive the blue LED we need to write to bit #2 of Port C (PC2).



1. Configure PC2 as an output:
   DDRC |= (1<<2);
2. Set PC2 high (5 Volts)
   PORTC |= (1<<2);
3. Leave other bits in PORTC and DDRC the same

Reading from an I/O Port (active high)

- For UCFK4, to read whether a button has been pushed we need to check bit #7 of PIND register (PD7).



1. Configure PD7 as an input:

```
DDRD &= ~(1<<7);
```

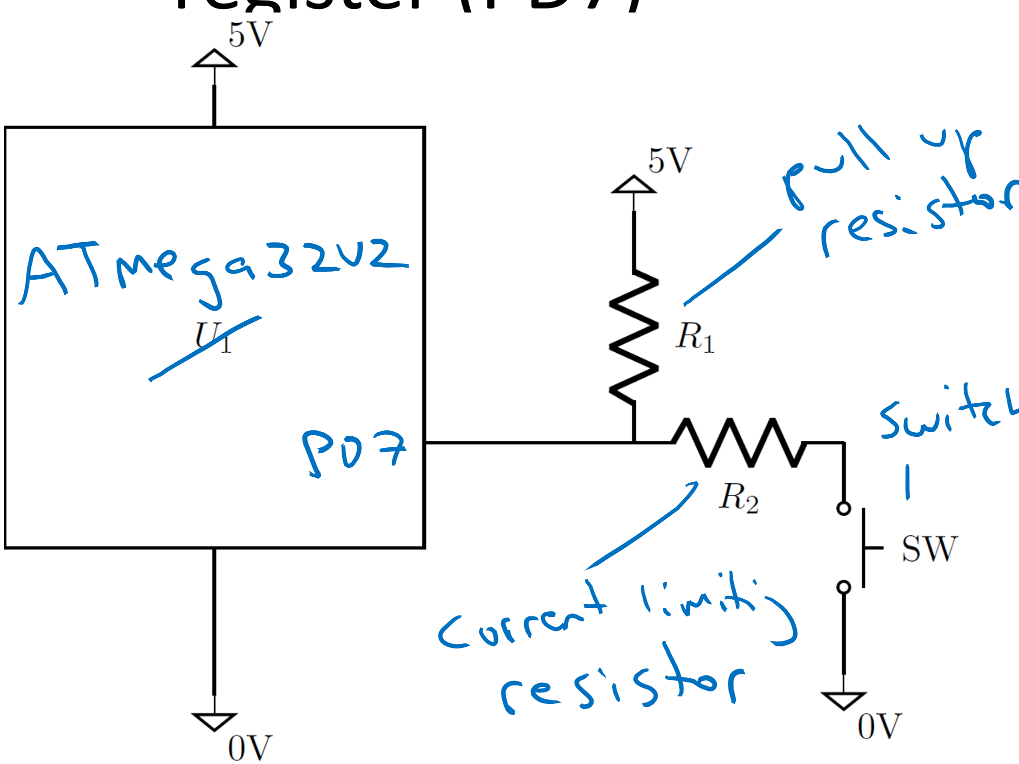
2. Test state of PD7

```
if ((PIND & (1<<7)))
{
/* switch has been pushed */
}
```

3. Leave other bits in PIND and DDRD the same

Reading from an I/O Port (active low)

- For UCFK4, to read whether a button has been pushed we need to check bit #7 of PIND register (PD7)



1. Configure PD7 as an input:

```
DDRD &= ~(1<<7);
```

2. Test state of PD7

```
if ((PIND & (1<<7))==0)
{ /* switch has been pushed */
}
```

3. Leave other bits in PIND and DDRD the same

Software implementation

PC2

```c
#include "button.h"
#include "led.h"
#include "system.h"

int main (void)
{
    system_init ();

    led_init ();
    button_init ();

    while (1)
    {
        if (button_pressed_p ())
        {
            led_on ();
        }
        else
        {
            led_off ();
        }
    }
}
```

```c
/** Initialise LED1.  */
void led_init (void)
{
    DDRC |= (1 << 2);
}
```

Initialising the data direction registers only needs to be done once

Next Lecture:

Clocks and Timers (Monday 18th September)