2 October 2015

INFO-0064 Embedded systems

Exercise session 2

Introduction to microcontrollers II

Kevin Hogan

kevin.hogan@ulg.ac.be

Planning for today

- Small recap
- I have my microcontroller, how do I make it work?
- Power supply
- Oscillator
- Programming
- Application exemple: Blinky

I have my μC, how to I make it works?

It won’t work on its own, it needs some external circuitry!

- We need to **power it up** correctly.
- We need to give it a **clock signal**.
- We need to be able to **program it**.

Power supply

- *What does my µC need?*

➔ Stable voltage, with low noise. Enough current with a protection circuit.

- *How can we obtain that?*

➔ Voltage regulators (LM2940, LM7805, etc…)

Power supply

Basic setup

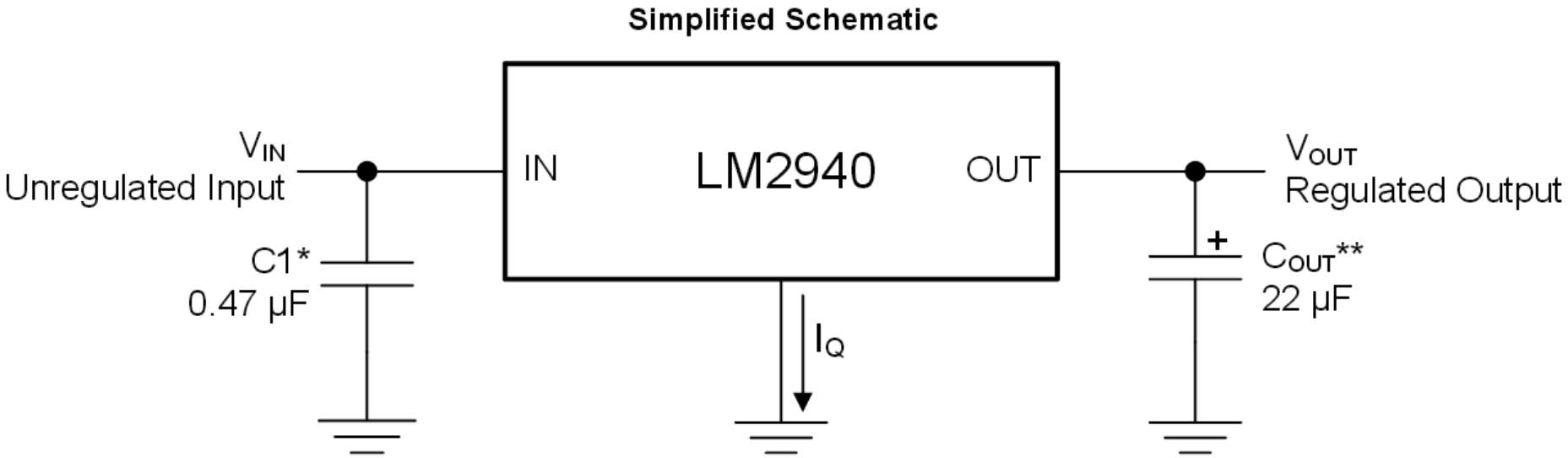


Figure 1 Typical application of the LM2940 (*from the datasheet*)

Power supply

- *In practice however, one needs to be careful…*

➔Never suppose that a power supply is perfect! It’s not!

- **Ripple** (for switching power supplies [cf. Power electronic])
- **Transients** (originating from switching devices (transistors, etc.) or power devices (motors, etc.))

➔You do not want to burn your microcontroller

We have to limit the maximal current and have some protection

Power supply

Improved setup

Tested and approved!

+ Filtering capacitors & inductors
+ Fuse (to limit the current)
+ Schottky diode

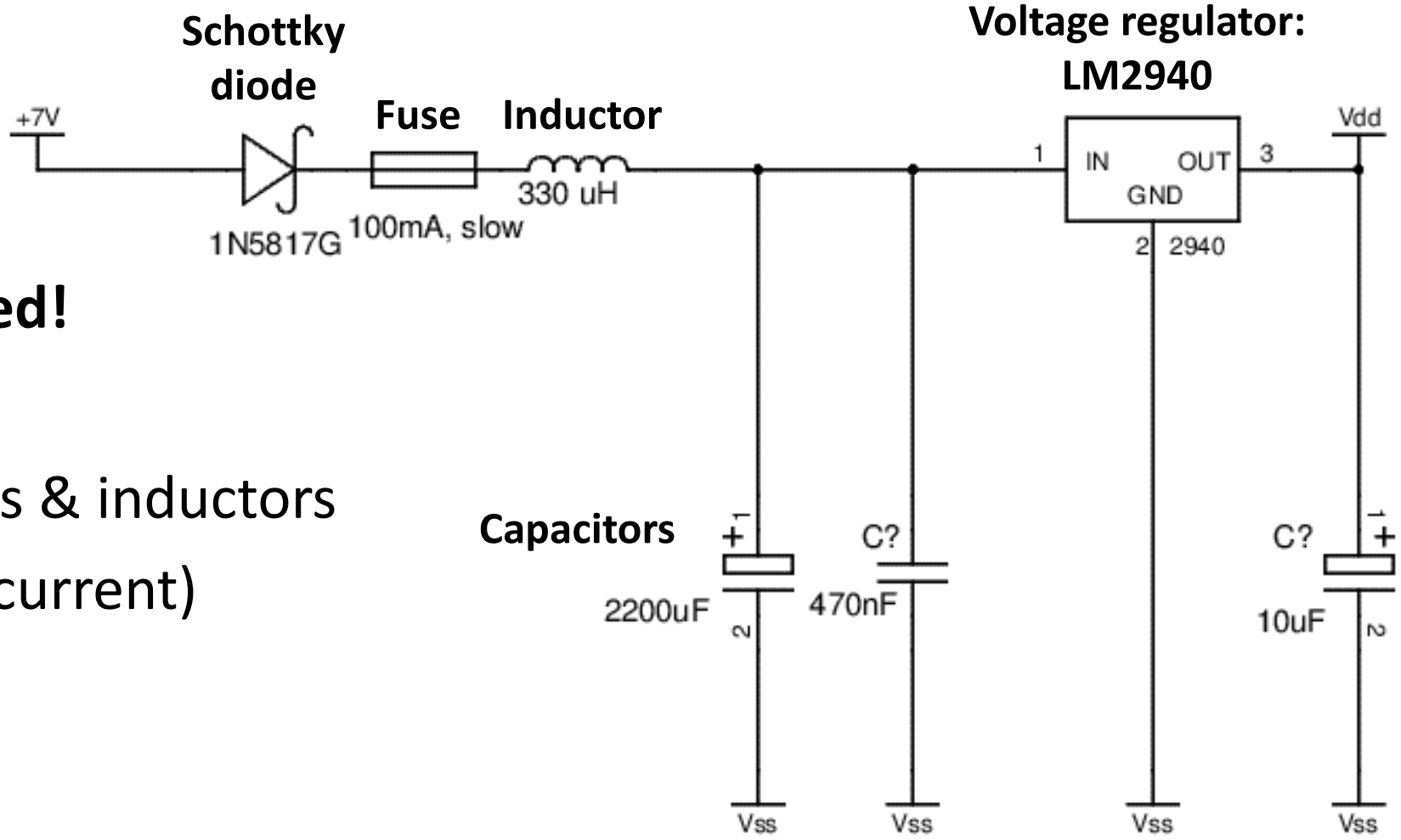


Figure 2 Improved setup for the power supply based on the LM2940

Power supply

Connect **ALL** the μC power pins!

Add decoupling capacitors

- Help stabilizing the voltage locally
- Have to be placed as close as possible to the power input pins

➔100 nF capacitors

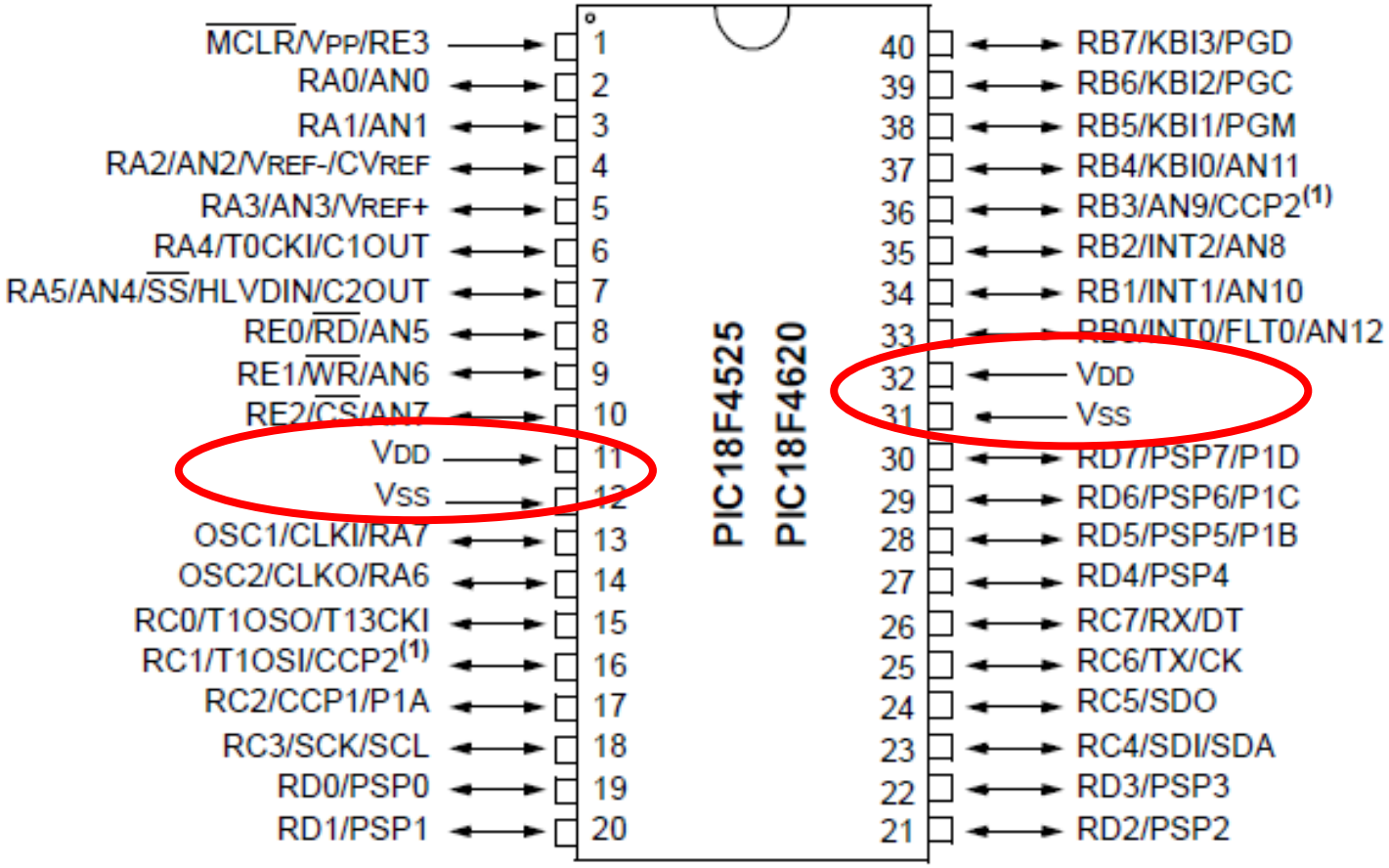


Figure 3 Microchip PIC18F4620 pinout

Oscillator

It is the heartbeat of the microcontroller!

You have several available clocks:

- Quartz
- Internal oscillator
- External clock
- Etc.

➔You set the source of the clock in your code

Oscillator

It is the heartbeat of the microcontroller!

C1 and C2?

➔ PIC18F4620 datasheet

For a 20 MHz crystal,

➔ **C1 = C2 = 33pF**

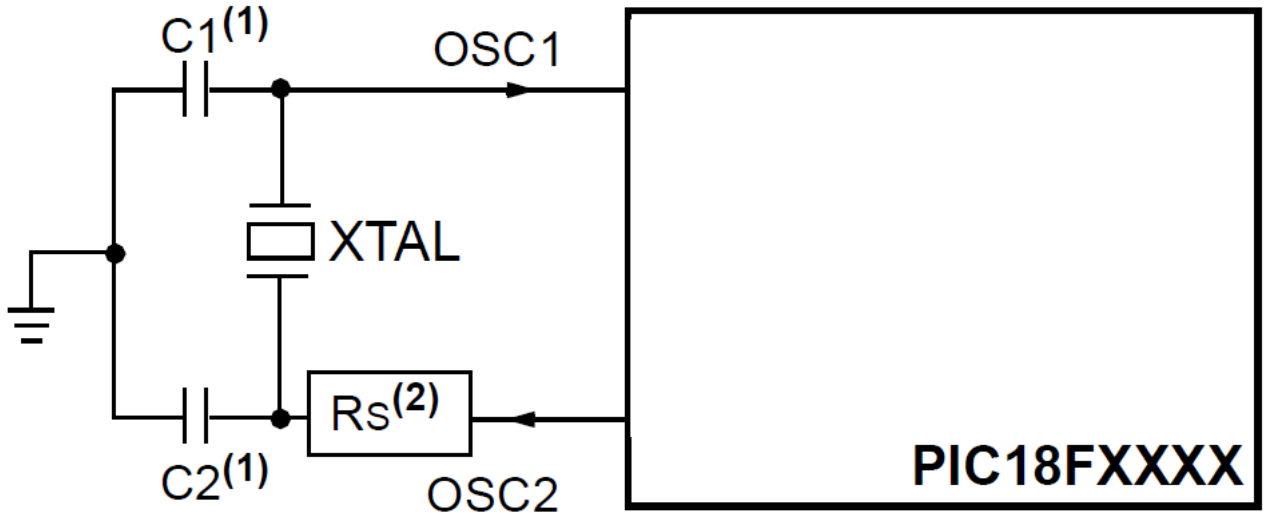
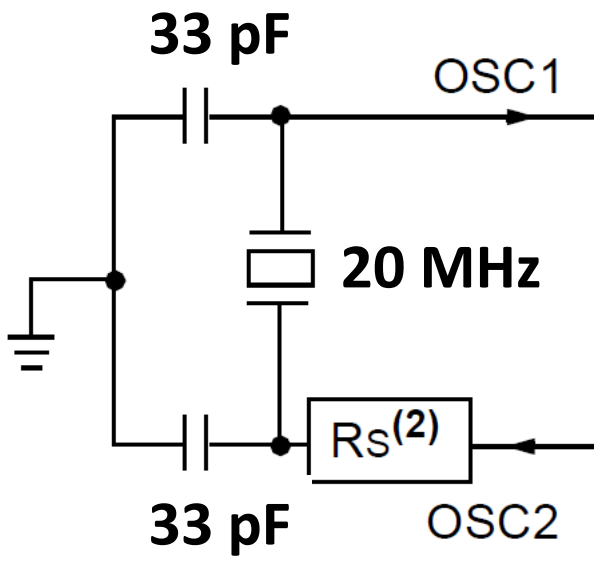


Figure 4 Crystal operation for PIC18F (cf. datasheet)

Oscillator



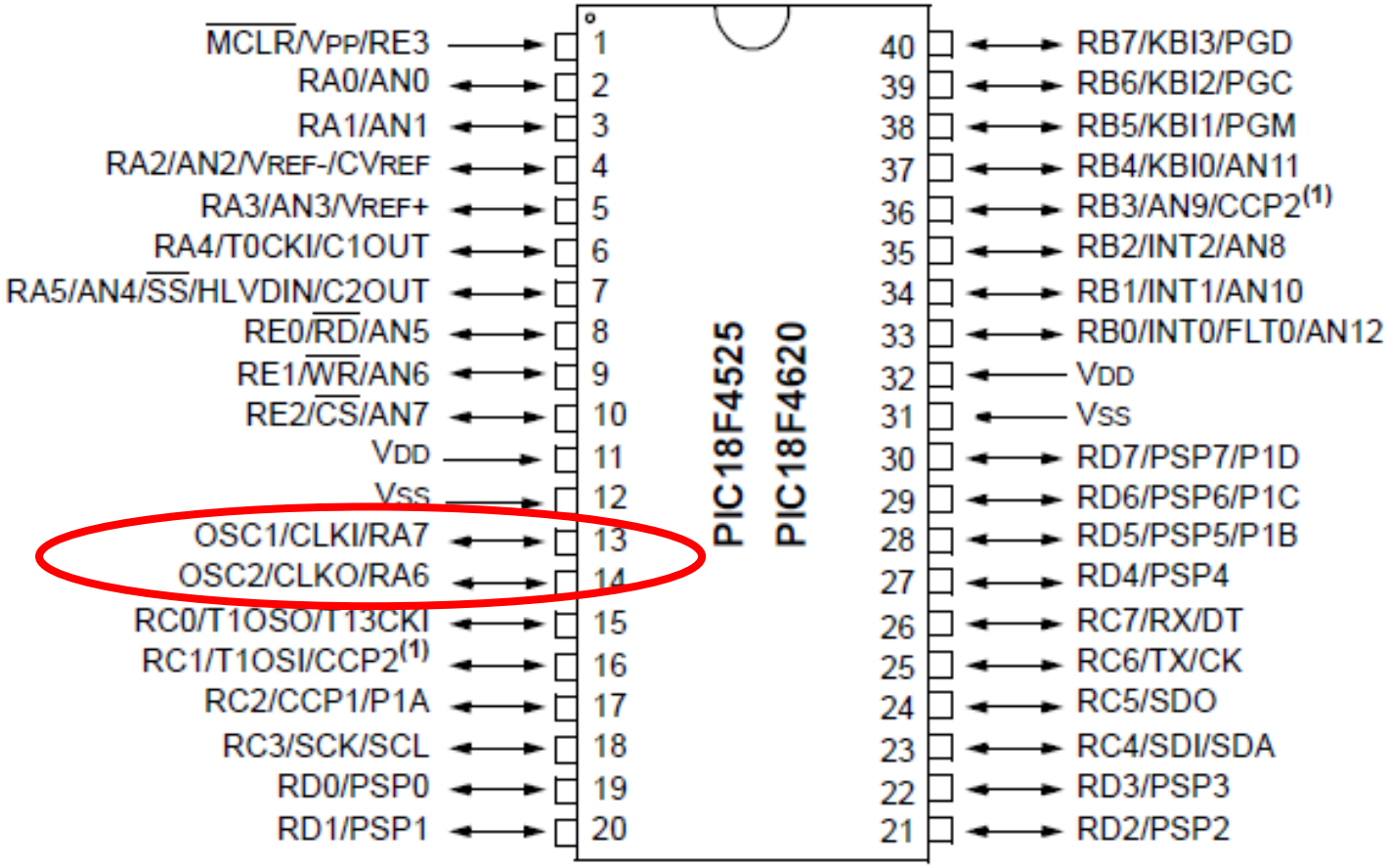


Figure 3 Microchip PIC18F4620 pinout

Programming

Ok! My µC is up and running, now let's program it! What do I need?

➔A programmer! ICD3 or PICkit 3

It writes your compiled code in the program memory of the μC.

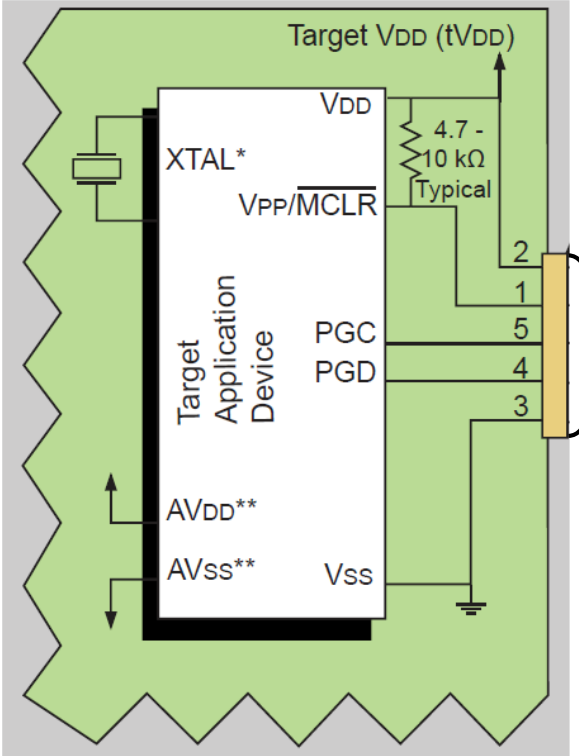
ICD 3
(available at the R100 laboratory)

PICkit 3

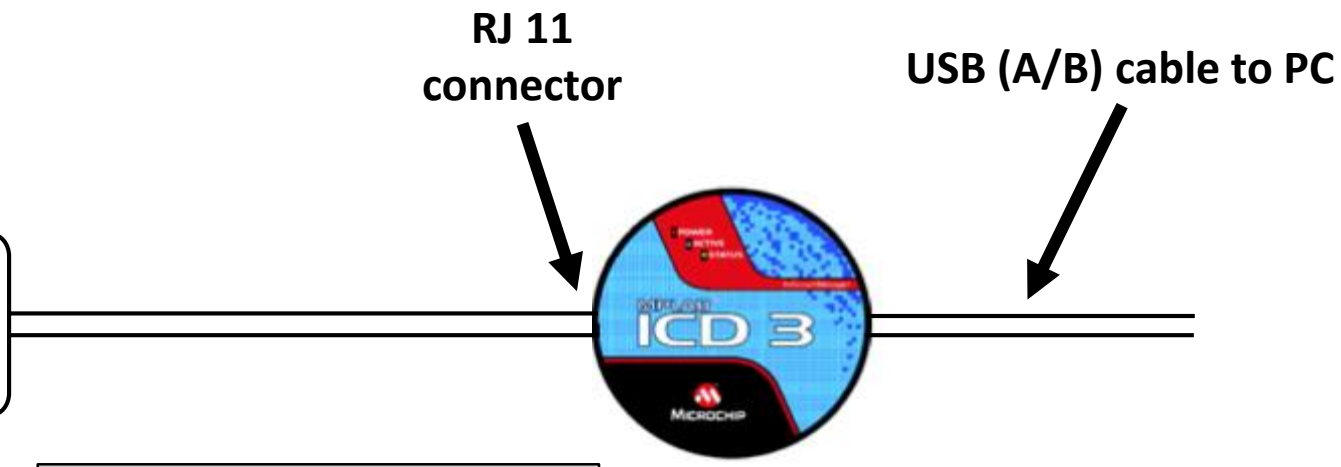
Programming

How do I connect it to my µC?

➔ [Datasheet](#) of the ICD3

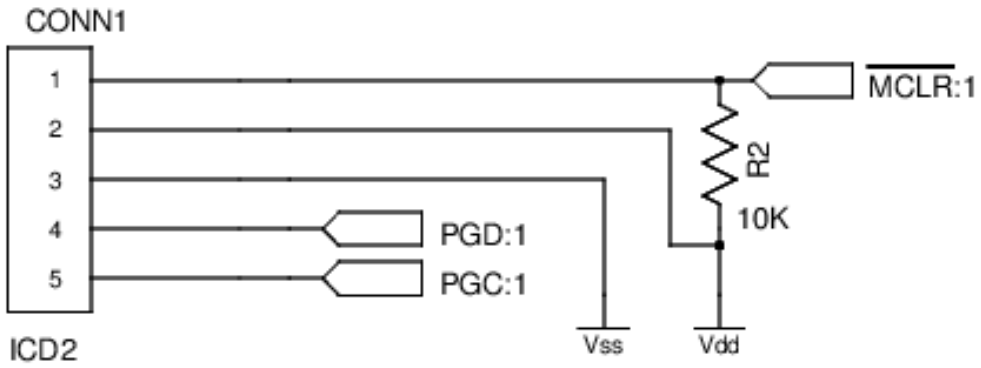


1	$\overline{\text{MCLR}}$/VPP
2	VDD
3	Ground
4	PGD (ICSPDAT)
5	PGC (ICSPCLK)

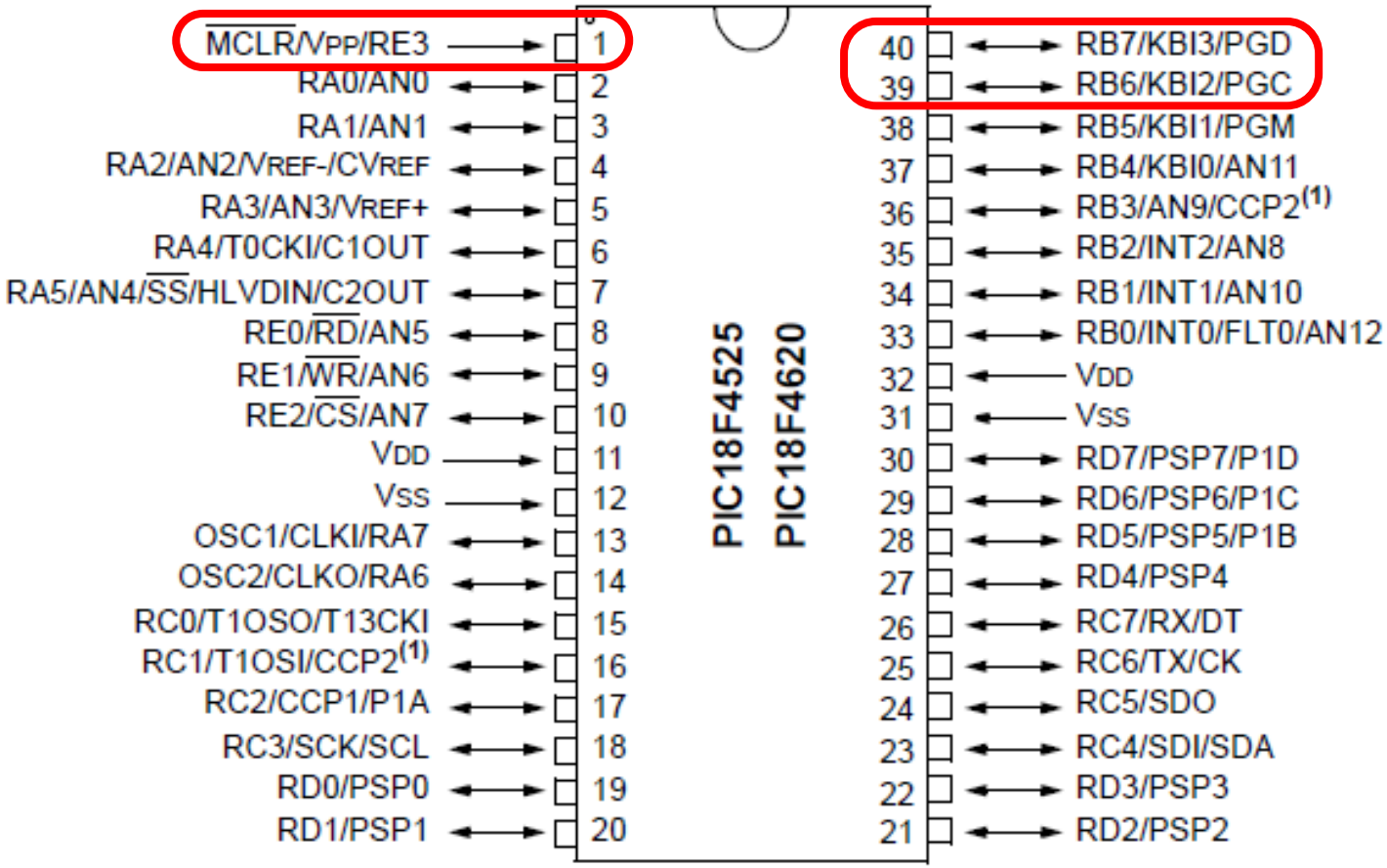


I can lend you those two cables!

Programming



1	$\overline{MCLR}$/VPP
2	VDD
3	Ground
4	PGD (ICSPDAT)
5	PGC (ICSPCLK)



Application example: Blinky

Our µC is ready to be used! Let's make something!

➔ The *hello world!* of electronics: the blinking LED ("blinky")

<u>Goal</u>: Make a LED blink at a given frequency (1 Hz)

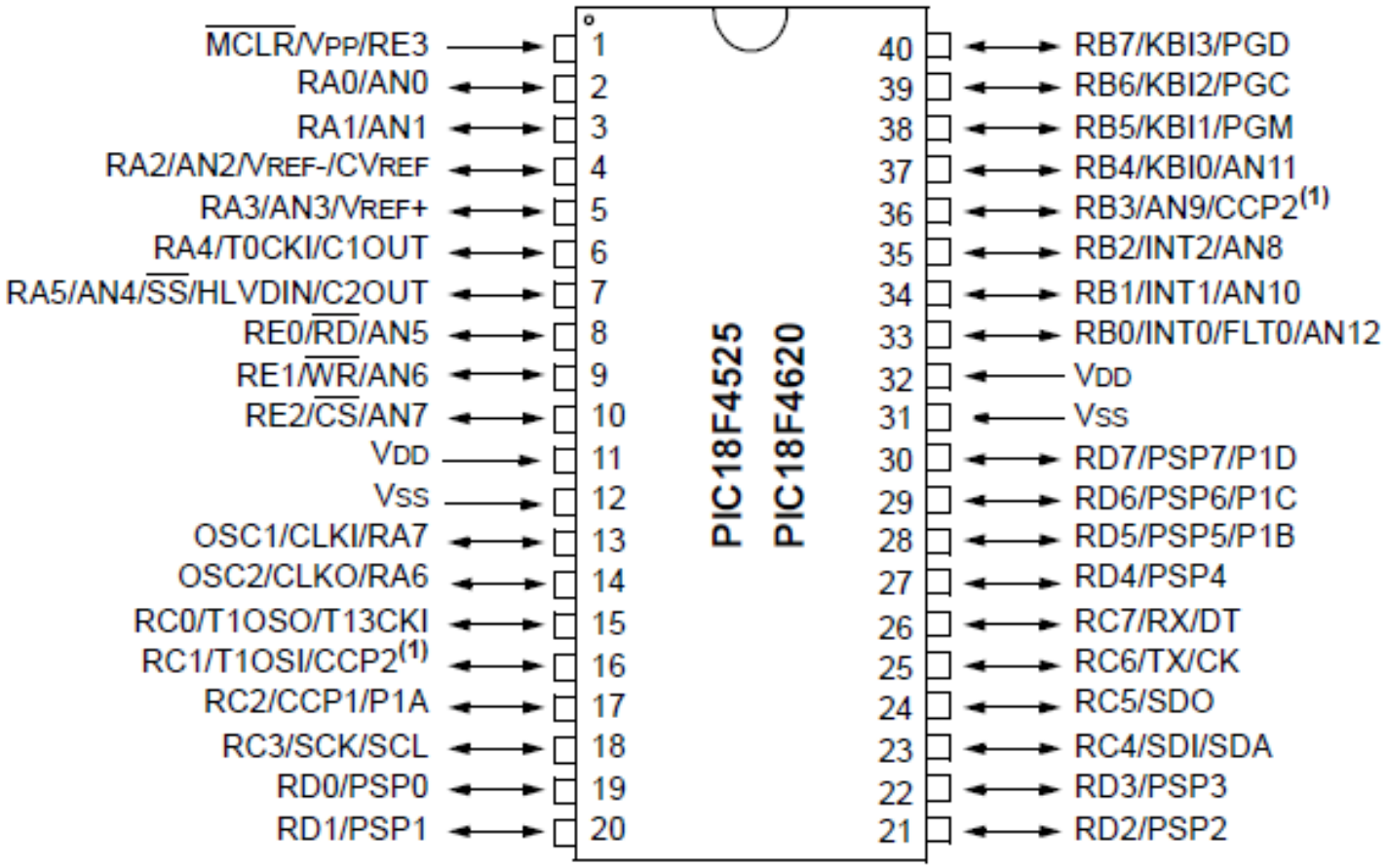
- How can we make the LED blink? ➔ GPIO
- How to get the timing we want? ➔ Timer & interrupts

General Purpose Input/Output (GPIO)

- 5 ports (PORT{A,B,C,D,E})
- PORT{A,B,C,D} = 8 pins
- PORTE = 4 pins
- A pin can be either:
  - a logical input/output or
  - used by the peripheral hardware (i.e. analogical input for the ADC)

Output current limitations!

- **25 mA per pin**
- **200 mA for all ports**



General Purpose Input/Output (GPIO)

Do you remember the Special Function Registers?

- Each port has 3 Special Function Registers for its operation:

  TRISx ➔ specifies the data direction (input/output)

  LATx ➔ specifies the output value

  PORTx ➔ reads the level on the pin

Each bit of those registers corresponds to one pin.

- TRIS must be configure correctly even if a pin is used by a peripheral.

General Purpose Input/Output (GPIO)

Code to set the pin RB0 as an output high:

```
bcf    TRISB , 0     ; clear bit 0 of TRISB
bsf    LATB , 0      ; set bit 0 of LATB
```

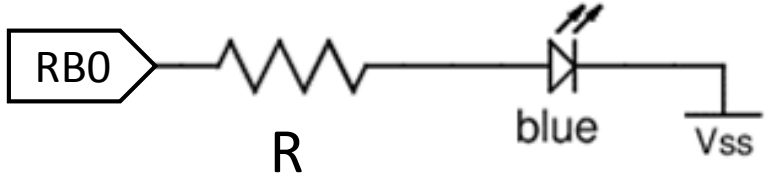
➔ RB0 = output

➔ RB0 = 5 V

General Purpose Input/Output (GPIO)

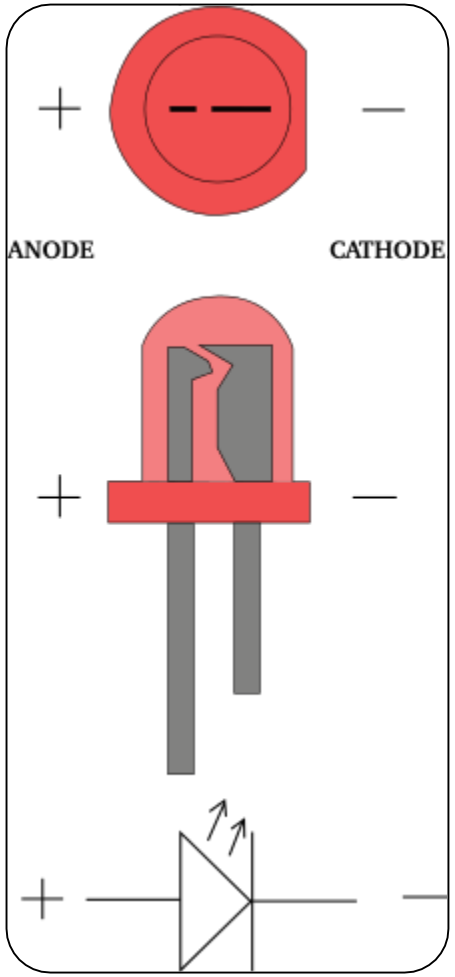
Let's connect the LED:

- It is *current* driven (Vdd can be large, it doesn't matter as long as the right current flows through the LED). You simply need a *resistor* to limit the current.
- Typically, you need 10 – 20 mA (< 25 mA, perfect!).

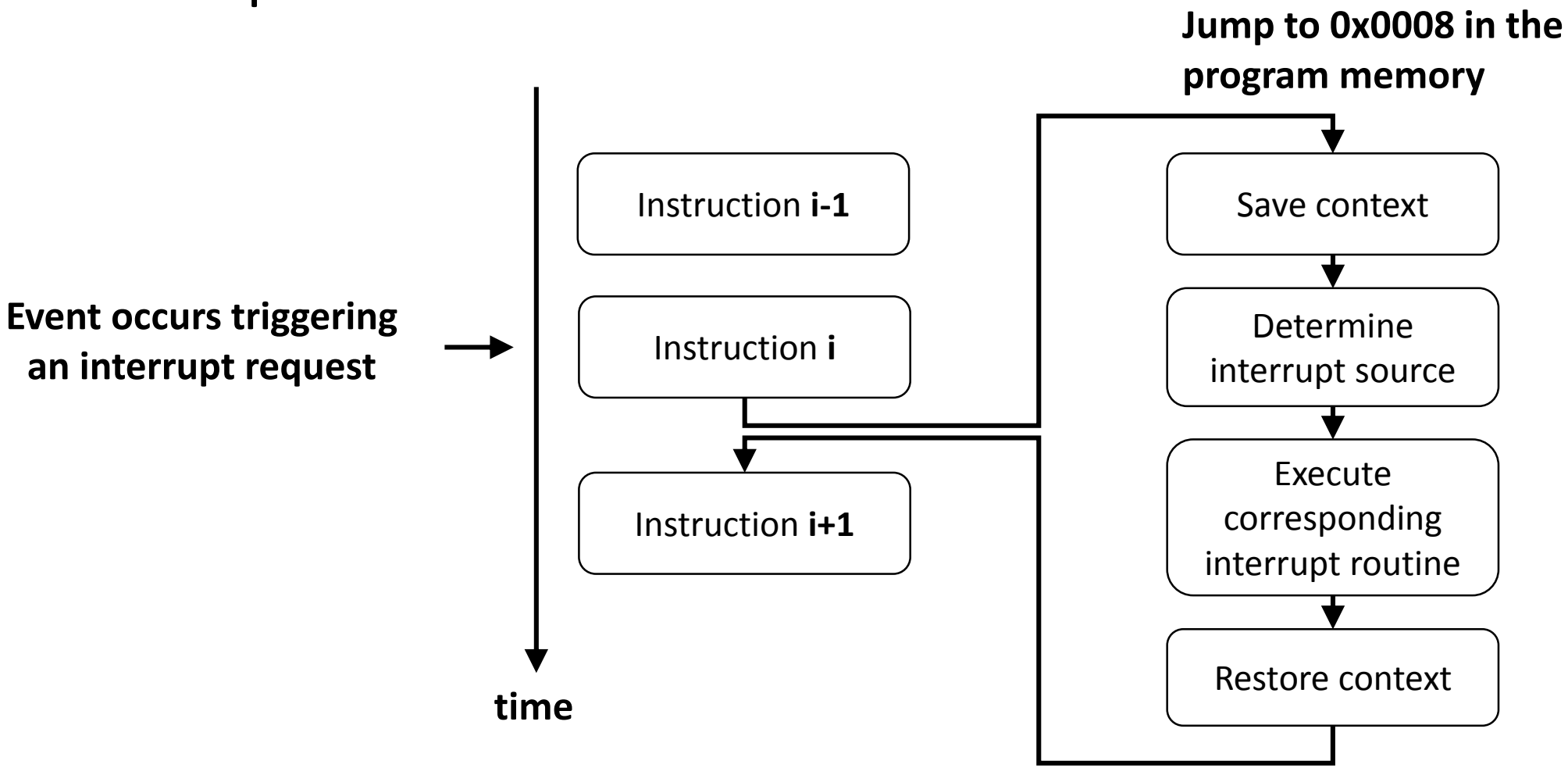


So if Vdd = 5 V and we want I = 15 mA

➔ R = Vdd / I ≈ 330 Ohm



Interrupts


Jump to 0x0008 in the program memory
Event occurs triggering an interrupt request
time
Instruction i-1
Instruction i
Instruction i+1
Save context
Determine interrupt source
Execute corresponding interrupt routine
Restore context

Interrupts

All interrupt sources are enabled by a certain bit in a SFR (the bit GIE in the INTCON register)

Many interrupt sources = Timers, input pin, UART, ADC, etc.

Interrupt sources have two bits (within SFR) to control their operation:

- **Flag bit**: indicating that interrupt event occurred
- **Enable bit**: allowing the program to jump to the interrupt vector when the flag bit is set

If a flag bit is set (as well as the corresponding enable bit), the μC jumps automatically to the interrupt vector.

Interrupts

When using interrupts, remember:

- Clear the interrupt flag before enabling the interrupt
- Check if the interrupt is enable before checking its interrupt flag
- Once in the interrupt code, first thing to do: **clear the interrupt flag** (it is not done automatically!)

Interrupts

Configure the interrupt for timer0

```
bcf   INTCON , TMR0IF   ; clear timer0 interrupt flag
bsf   INTCON , TMR0IE   ; enable interrupt for Timer0
bsf   INTCON , GIE      ; enable all interrupts
```

Configure the interrupt vector

```
org   0x08                                  ; jump to the
goto  high_priority_interrupt_vector        ; interrupt
                                            ; routine
```

Interrupts

Interrupt routine: find the interrupt source

```
high_priority_interrupt_vector:

test_int_Timer_0:
   btfss    INTCON , TMR0IE    ; Is Timer0 Interrupt
                               ; Enable bit set?
   goto     other_test_int     ; skip if yes
   btfsc    INTCON , TMR0IF    ; Is Timer0 Interrupt Flag
                               ; bit set?
   call     interrupt_Timer_0 ; skip if no
other_test_int:

end_of_test_int:
   retfie 1
```

Timer

- The PIC18F4620 has four timers (8-bit or 16-bit)
- A timer can increment every *cycle* or every 2,4,8, …, 256 *cycles* with the use of a prescaler
- Interrupt flag = overflow bit

- The register T0CON controls all aspects of the timer's operation

Timer

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Timer

Code to configure the Timer0 as 16-bit timer on the instruction cycle clock without prescaler:

```
; Timer0 ON - 16 bits mode - NO prescaling
movlw   b'10001000'
movwf   T0CON
```

Timers: exemple

Can we configure Timer0 to have an interrupt every second?

- Timer0 16bit ➔ it counts up to 2^{16} = 65536 cycles

F_{osc} = 20MHz ➔ 5 MIPS

$t_{cycle} = 1 / (5*10^6) = 0{,}2\ \mu s$

- An interrupt occurs every 65536*0,2µs = **13,1072 ms** Too fast!

➔ We will count 76 interrupts to obtain a second

Interrupts

Interrupt routine

```
interrupt_Timer_0:
      bcf    INTCON , TMR0IF  ; Clear the Interrupt Flag!
      decfsz counter , f      ; if counter = 0 then the next
                              ; instruction is skipped else
                              ; counter--
      return
      movlw  0x01
      xorwf  LATB , f         ; RB0 = !RB0
      movlw  76
      movwf  counter          ; reset the value of the counter

      return
```

Application exemple: blinky