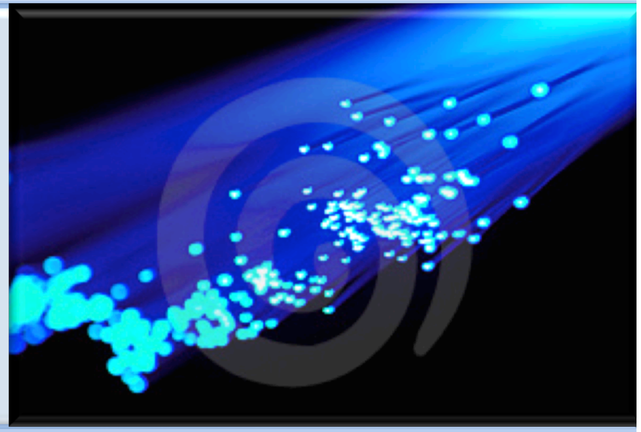
S34PH412 :

Systèmes Microprogrammés & Robotique

Architecture des Ordinateurs

Cours 7



Université de la Réunion

Année 2015/2016

Alicalapa F.

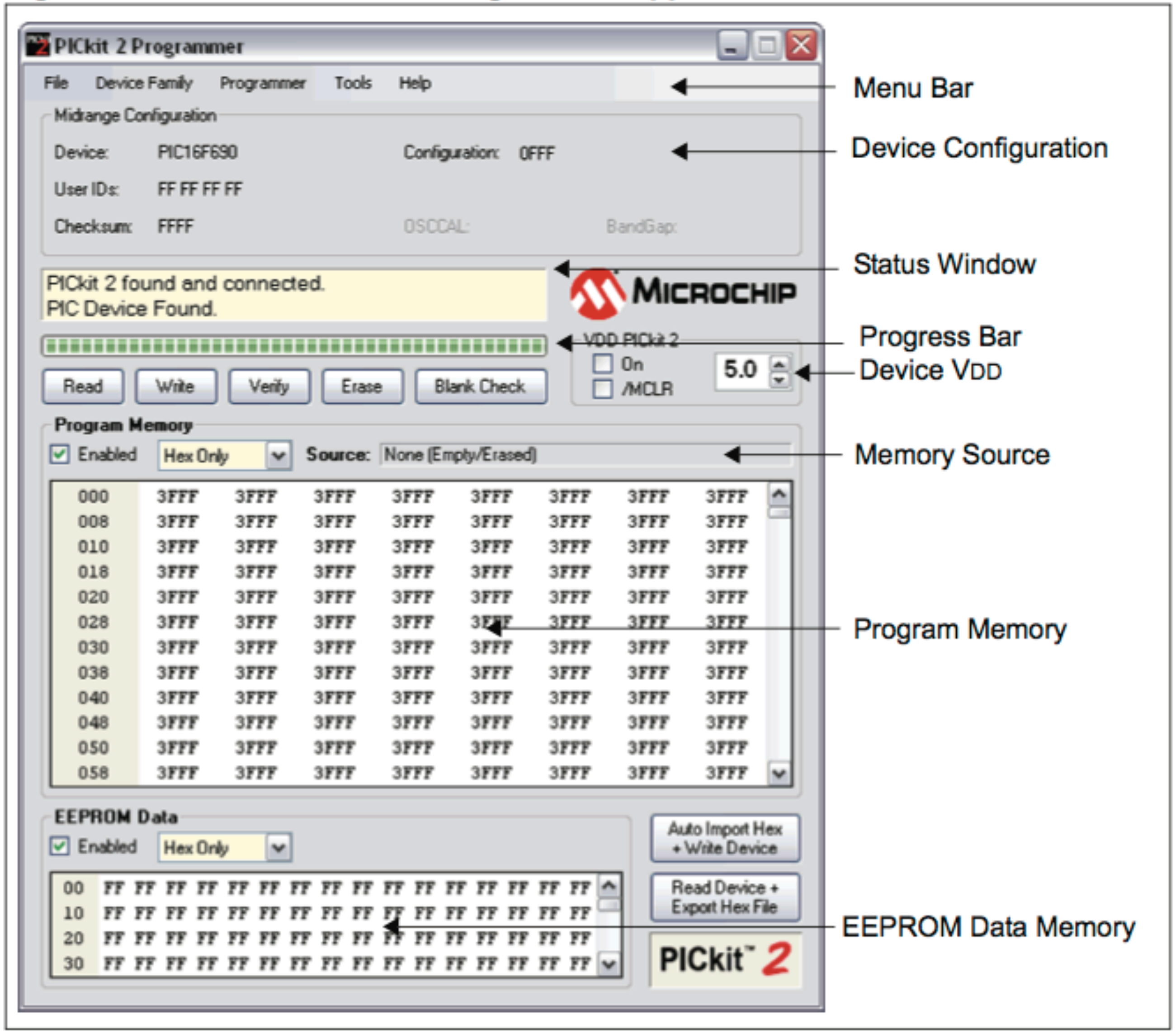
UNIVERSITÉ DE LA RÉUNION

S'ouvrir aux mondes

UFR Sciences et Technologies UNIVERSITÉ DE LA RÉUNION

- **PicKit 2 PROGRAMMER**: interface

Figure 1-3: PICkit™ 2 Programmer Application

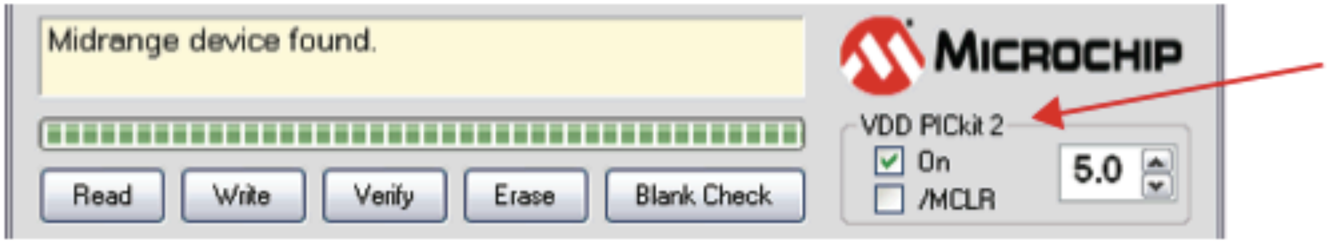


- **PicKit 2** : VDD

The PICkit 2 VDD may be turned on and off by clicking the checkbox “On”. The voltage may be set in the box on the right either by typing it directly or using the up/down arrows to adjust it a tenth of a volt at a time. The maximum and minimum allowed voltages will vary depending on the target device.

If the “On” checkbox is unchecked, PICkit 2 will automatically turn on the VDD at the set voltage during any requested programming operation.

FIGURE 1-4: PICKIT™ 2 SUPPLIED VDD



If the target device has its own power supply, then the PICkit 2 will display the detected VDD voltage in the box on the right, which will be grayed out to prevent being changed. The checkbox text changes to “check”, and clicking on the checkbox will update the detected VDD voltage value. If *Target VDD>Auto-Detect* is selected, clicking on the checkbox will revert the VDD mode back to PICkit 2 supplied VDD if a target power supply is no longer detected.

Les éléments de base d’un ordinateur

- **PicKit 2** : interface

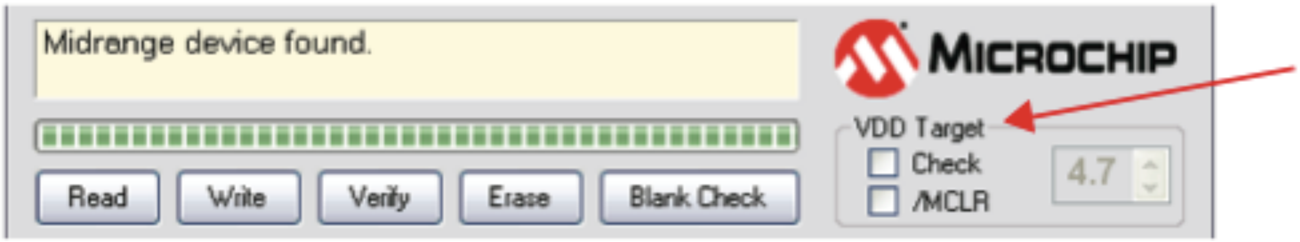
Target VDD Source

- Auto-Detect – The PICkit 2 will automatically detect whether the target device has its own power supply or needs to be powered by the programmer on each operation.
- Force PICkit 2 – The PICkit 2 will always attempt to supply VDD to the target device.
- Force Target – The PICkit 2 will always assume the target has its own power supply.

- **PicKit 2** : /MCLR

The "/MCLR" checkbox shown in Figure 1-4 and Figure 1-5 has the same functionality as the menu selection *Programmer>Hold Device in Reset*. When the box is checked the target device will be held in Reset. When unchecked, the target circuit is allowed to pull MCLR up to V_{DD} to release the device from Reset. This function can be used to prevent a device from executing code before and after programming.

Note:	If the target device allows the $\overline{\text{MCLR}}$ pin to be configured as an input port, and it is configured as such, PICkit 2 will not be able to hold the device in Reset.



Hold Device in Reset – When checked, the $\overline{\text{MCLR}}$ (V_{PP}) pin is held low (asserted). When unchecked, the pin is released (tri-stated), allowing an external pull-up to bring the device out of Reset.

- **PicKit 2** : **syntaxe ASM** : utilisation des DIRECTIVES

Table 4.1 Some common MPASM Assembler directives

Assembler directive	Summary of action
list	Implement a listing option*
#include	Include additional source file
org	Set program origin
equ	Define an assembly constant; this allows us to assign a value to a label
end	End program block

```
;DELAY.ASM

;EQUATES SECTION

TMR0        EQU   1      ;TMR0 is FILE 1.
PORTA       EQU   5      ;PORTA is FILE 5.
PORTB       EQU   6      ;PORTB is FILE 6.
STATUS      EQU   3      ;STATUS is FILE3.
TRISA       EQU   85H    ;TRISA (the PORTA I/O selection)
TRISB       EQU   86H    ;TRISB (the PORTB I/O selection)
OPTION_R    EQU   81H    ;the OPTION register is file 81H
ZEROBIT     EQU   2      ;ZEROBIT is Bit 2.
COUNT       EQU   0CH    ;USER RAM LOCATION.
;*********************************************************
LIST        P=16F84      ;We are using the 16F84.
ORG         0            ;0 is the start address.
GOTO        START        ;goto start!
;*********************************************************
;Configuration Bits

__CONFIG H'3FF0'         ;selects LP oscillator, WDT off, PUT on,
                         ;Code Protection disabled.
```

Opération Équivalente à include 16F690.h

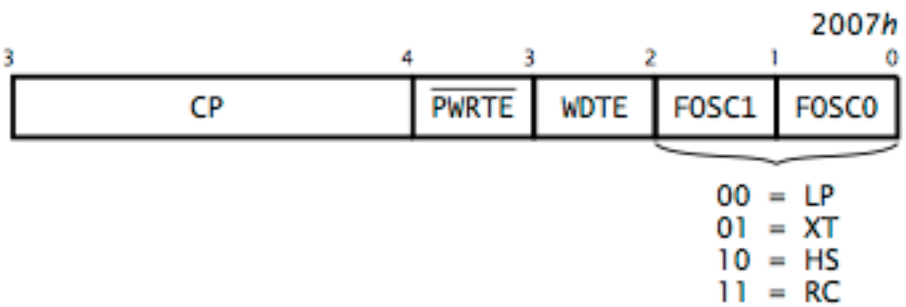
Stockage code compilé à partir adresse 0h dans l'espace mémoire

Début à 0h

- **PicKit 2** : **syntaxe ASM** : utilisation des DIRECTIVES

Config word

```
PORTB       EQU    6       ;PORTB is FILE 6.
STATUS      EQU    3       ;STATUS is FILE3.
TRISA       EQU    85H     ;TRISA (the PORTA I/O selection)
TRISB       EQU    86H     ;TRISB (the PORTB I/O selection)
OPTION_R    EQU    81H     ;the OPTION register is file 81H
ZEROBIT     EQU    2       ;ZEROBIT is Bit 2.
COUNT       EQU    0CH     ;USER RAM LOCATION.
;***********************************************************
LIST        P=16F84        ;We are using the 16F84.
ORG         0              ;0 is the start address.
GOTO        START          ;goto start!
;***********************************************************
;Configuration Bits

__CONFIG H'3FF0'         ;selects LP oscillator, WDT off, PUT on,
                         ;Code Protection disabled.
```



Oscillator in XT mode
Bits 1:0 = 01

Watchdog timer off
Bit 2 = 0

Power-up timer on
Bit 3 = 0

No code protection
Bits 13:4 = 1111111111

Then the directive

```
__config b'11111111110001'    ; or 3FF1h
```

- **PicKit 2** : **syntaxe ASM**

INSTRUCTION FORMATS

Most instructions follow one of three formats: Byte oriented instructions, Bit oriented instructions and Literal instructions.

Byte instructions contain 7-bit data address, a destination bit, and 6-bit op code. The data address plus the RP0 and RP1 bits create a 9-bit data memory address for one operand. The other operand is the Working register (called W or Wreg). After the instruction executes, the destination bit (d) specifies whether the result will be stored in W or back in the original file register. For example:

```
ADDWF  data,f
```

adds the contents of `Wreg` and `data`, with the result going back into `data`.

Bit instructions operate on a specific bit within a file register. They contain 7 bits of data address, 3-bit number and the remaining 4 bits are op code. These instructions may set or clear a specific bit within a file register. They may also be used to test a specific bit within a file register. For example:

```
BSF    STATUS,RP0
```

set the RP0 bit in the Status register.

Literal instructions contain the data operand within the instruction. The Wreg becomes the other operand. `Calls` and `GOTO`'s use 11 bits as a literal address.

```
MOVLW'A'
```

Moves the ASCII value of 'A' (0x41) into Wreg.

- **PicKit 2** : exemple de programme **delay_1ms – instruction BTFSS**

```
Delay_1ms     macro
LOOP          local

              movlw  d'250'    ; Count from 250d
LOOP          addlw  -1        ; Decrement
              btfss  STATUS,Z  ; to zero
               goto  LOOP

              endm
```

Label LOCAL

Mots clé pour créer macro-fonction ASM réutilisable

BTFSS f,b

(Bit test, skip if set)

skip if f(b) = 1

f: (00 à 4F); d: (0 à 7)

Teste le bit b du registre f:

- **Si f(b)=1, on saute l'instruction qui suit pour exécuter celle qui vient après**
- **Si f(b)=0, on exécute l'instruction qui suit**

Exemple:

- btfsc Casemem,b
- goto Bita0 ; exécuté si b=0
- xxxxx ; exécuté si b=1

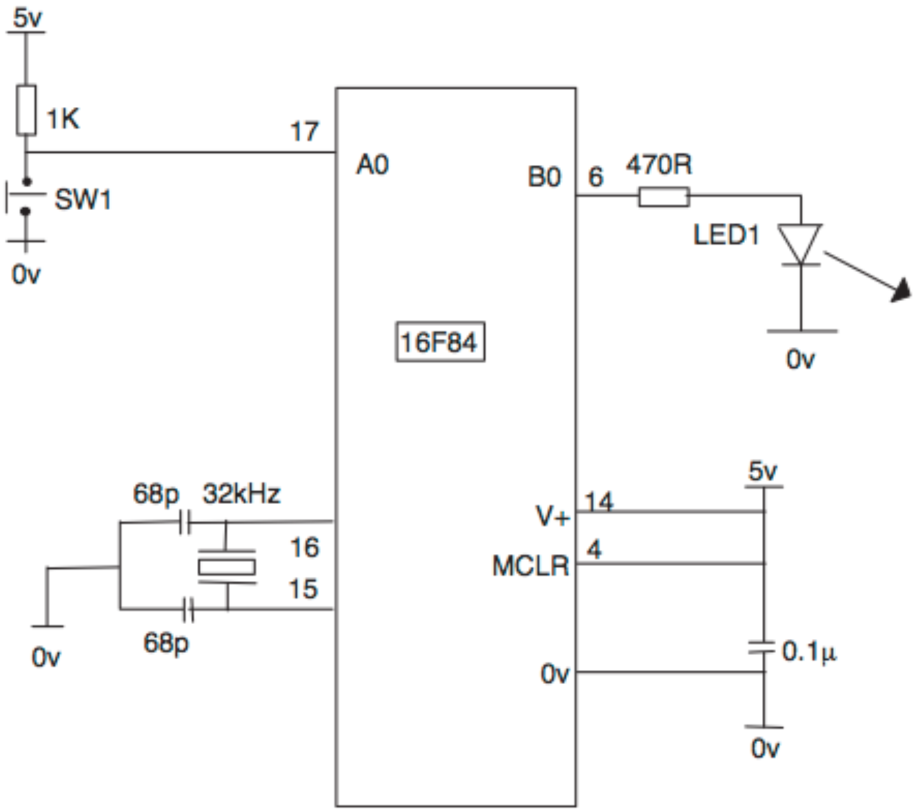
- **PicKit 2** : exemple de programme **delay_1ms – instruction BTFSS**

```
Delay_1ms     macro
LOOP          local

              movlw   d'250'    ; Count from 250d
LOOP          addlw   -1        ; Decrement
              btfss   STATUS,Z  ; to zero
               goto   LOOP

              endm
```

- **Comment obtenir des delais plus grand ??? Test ?? Lecture ??**

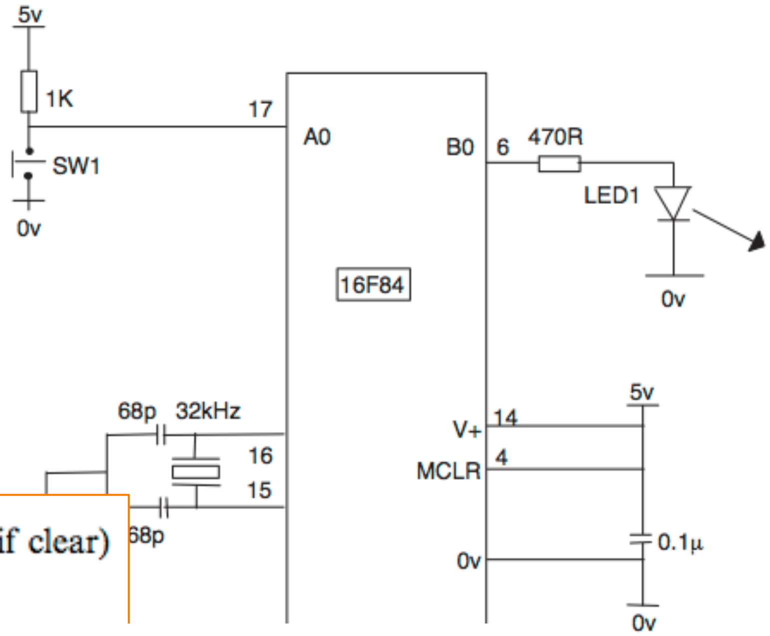
- **CAS d'étude 1 : objectifs**

1. Wait for SW1 to close.
2. Turn on LED1.
3. Wait for SW1 to open.
4. Turn off LED1.
5. Repeat.



- **CAS d'étude 1**

1. Wait for SW1 to close.
2. Turn on LED1.
3. Wait for SW1 to open.
4. Turn off LED1.
5. Repeat.



Le bout de code répond-il à 1 point du CdC ?
Lequel ?

```
BEGIN   BTFSC   PORTA,0 (test bit 0 in file PORTA skip if clear)
        GOTO    BEGIN
        BSF     PORTB,0
```

BTFSC f,b

(Bit test, skip if clear)

skip if f(b) = 0

f: (00 à 4F); d: (0 à 7)

Teste le bit b du registre f:

- **Si f(b)=0, on saute l'instruction qui suit pour exécuter celle qui vient après**
- **Si f(b)=1, on exécute l'instruction qui suit**

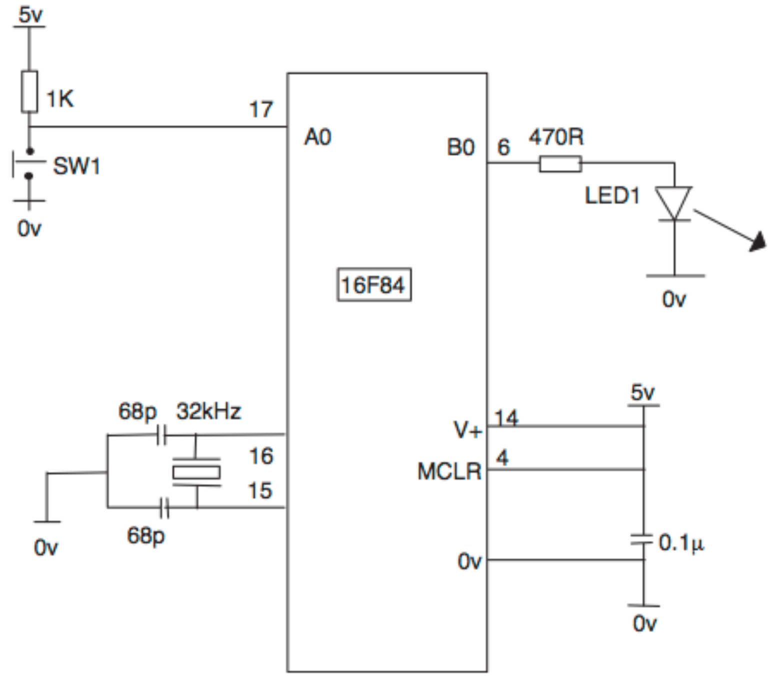
Exemple:

- btfsc Casemem,b
- goto Bita1 ; exécuté si b=1
- xxxxx ; exécuté si b=0

Nombre de cycles d'horloge:1 si b=1 ou 2 si b=0

- **CAS d'étude 1**

1. Wait for SW1 to close.
2. Turn on LED1.
3. Wait for SW1 to open.
4. Turn off LED1.
5. Repeat.



Le bout de code répond-il à 1 point du CdC ?
Lequel ?

```
SWOFF   BTFSS   PORTA,0
        GOTO    SWOFF
        BCF     PORTB,0
        GOTO    BEGIN
```

BTFSS f,b

(Bit test, skip if set)

skip if f(b) = 1

f: (00 à 4F); d: (0 à 7)

Teste le bit b du registre f:

- **Si f(b)=1, on saute l'instruction qui suit pour exécuter celle qui vient après**
- **Si f(b)=0, on exécute l'instruction qui suit**

Exemple:

- btfsc Casemem,b
- goto Bita0 ; exécuté si b=0
- xxxxx ; exécuté si b=1

- code ASM : comment comprendre ce bout de code ?? Réflexes à mettre en place ?

```
clrwdt                ; Clears postscaler and wdt
bsf     STATUS,RP0    ; Change-over to Bank0
movlw   b'11110001'   ; External clock on low-going edge
movwf   OPTION_REG    ; 1:4 Timer0 prescaler
bcf     STATUS,RP0    ; Back to Bank1
```

- **PicKit 2** : exemple de code ASM

```
Loop
    BSF     PORTC,0         ;turn on LED C0
    BCF     PORTC,0         ;turn off LED C0
    GOTO    Loop            ;do it again
```

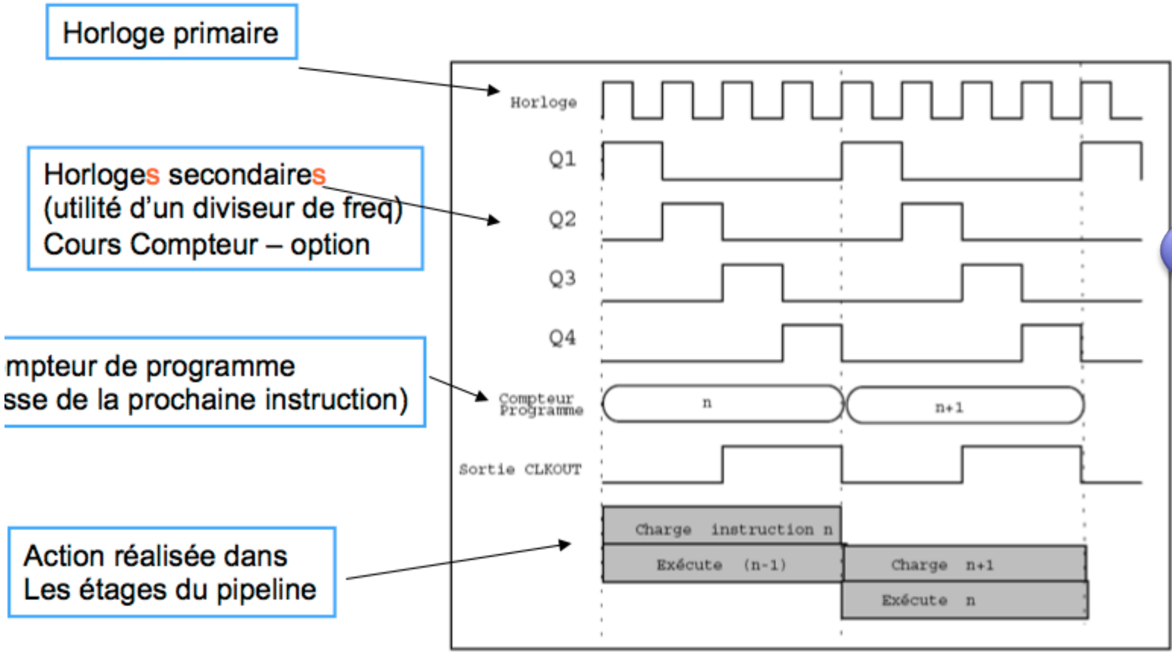
- Ou est l'erreur ?

- **PicKit 2** : exemple de code ASM

```
Loop
    BSF     PORTC,0         ;turn on LED C0
    BCF     PORTC,0         ;turn off LED C0
    GOTO    Loop            ;do it again
```

- Ou est l'erreur ?

Note: Counting cycles – Relating clock speed to instruction speed. The processor requires 4 clocks to execute an instruction. Since the internal oscillator as used in these lessons runs at 4 MHz, the instruction rate is 1 MHz.



Ancien cours

- ADC **langage C** – PIC: exemple de code

```
#include "16F877A.h"
#device ADC=8                                         //8-bit conversion

#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_D1)        //LCD output

void main() //*****************************************************
{
  int vin0;                                           // Input variable

  setup_adc(ADC_CLOCK_INTERNAL);                      // ADC clock
  setup_adc_ports(ALL_ANALOG);                        // Input combination
  set_adc_channel(0);                                 // Select RA0

  for(;;)
  {    delay_ms(500);
       vin0=read_adc();                               //Get input byte
       vin0=(vin0/32)+0x30;                           //Convert to ASCII

       putc(254); putc(1); delay_ms(10);              // Clear screen
       printf("Input="); putc(vin0);                  // Display input
  }
}
```