5 janvier 2016 *Licence SPI AGE – Semestre 5*

Aix-Marseille Université

Contrôle des connaissances

UE3 – Automatisme et informatique industrielle

Seul le polycopié de cours est autorisé
Calculatrices autorisées.

Cet examen de 2 heures est composé de 2 parties :

- partie *'Informatique industrielle'* (12 points / approximativement 1h).
- partie *'Automatisme'* (8 points / approximativement 50mn),

Première partie

Informatique industrielle

Questions de cours (5 pts)

1. Le micro-contrôleur présenté *figure 1* est-il un microprocesseur 8 bits, 16 bits ou 32 bits ? Justifier votre réponse.

2. Identifier les éléments ci-dessous sur le schéma du micro-contrôleur présenté *figure 2 (page suivante)* :

- le compteur programme,
- le bus de données,
- l'unité arithmétique et logique,
- la mémoire programme,
- les ports entrées-sorties.

3. Expliquez le rôle du vecteur RESET dans la mémoire programme représentée *figure 1* ci-dessous.

4. Expliquez le rôle du vecteur d'INTERRUPTION placée à l'adresse 0x0008 dans la mémoire programme représentée *figure 1* ci-dessous.

5. Expliquez pourquoi il y a plusieurs vecteurs d'INTERRUPTION dans la mémoire programme représentée *figure 1* ci-dessous.

6. Identifier les modules de TIMER. Expliquez à quoi sert un module TIMER.

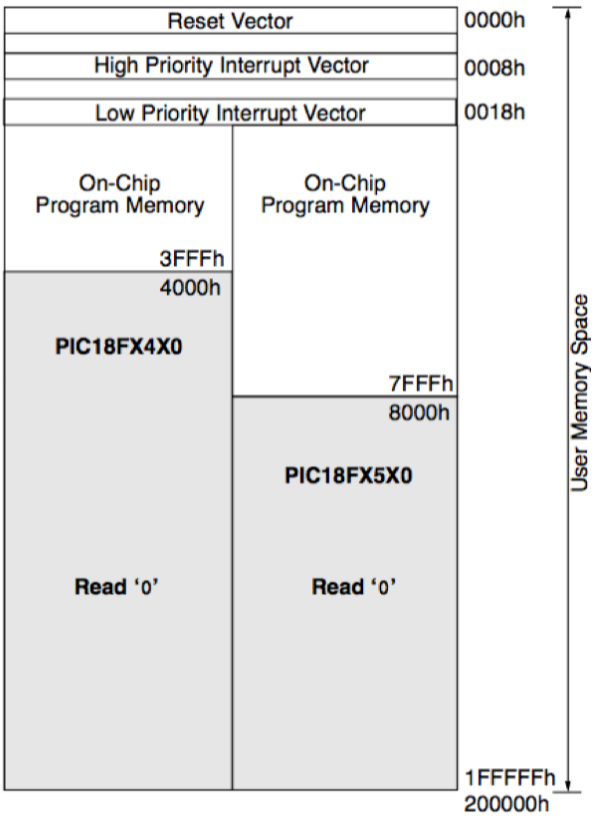
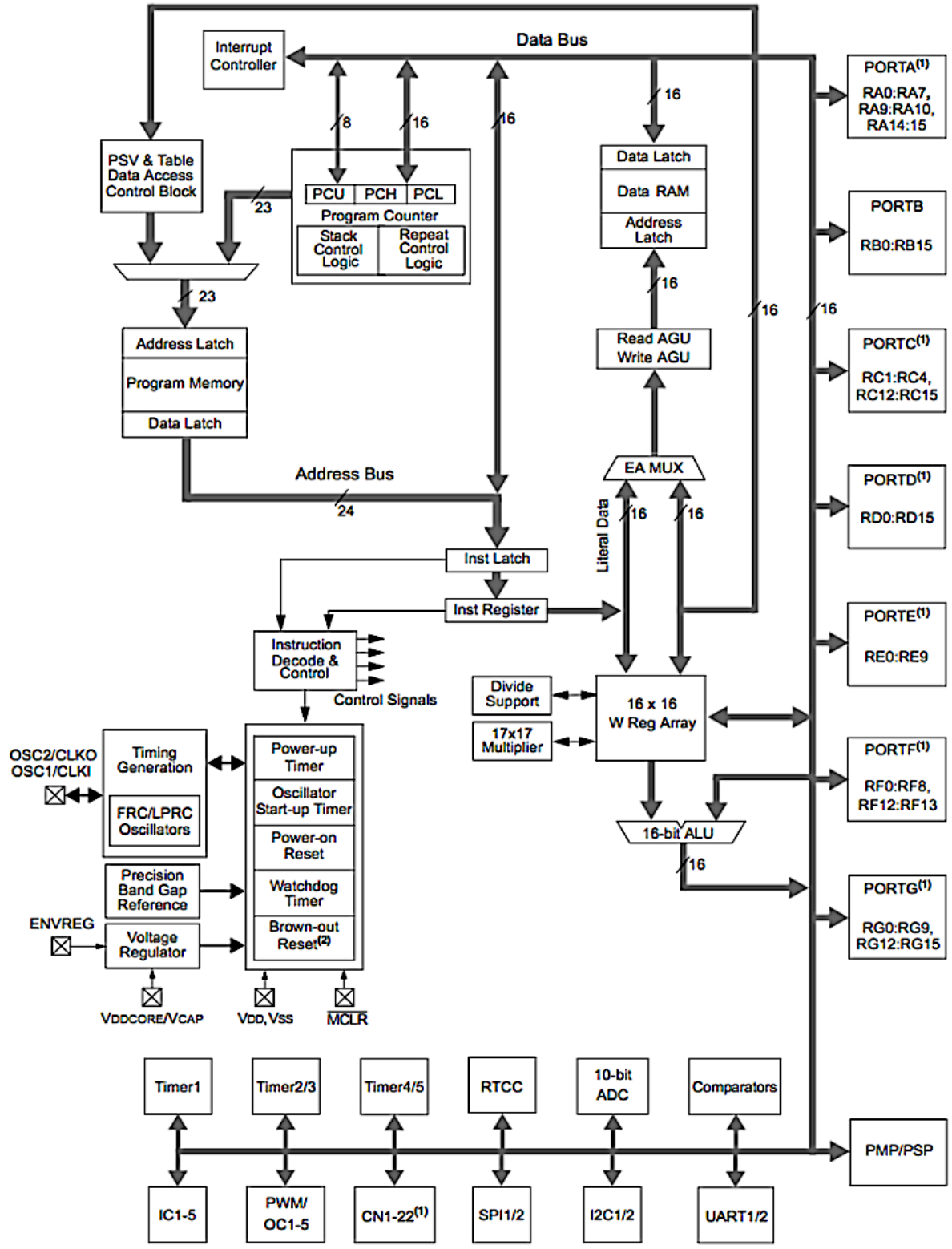


Figure 1 : organisation de la mémoire programme du PIC 18F4520.



Note 1: Not all pins or features are implemented on all device pinout configurations. See Table 1-2 for I/O port pin descriptions.
2: BOR functionality is provided when the on-board voltage regulator is enabled.

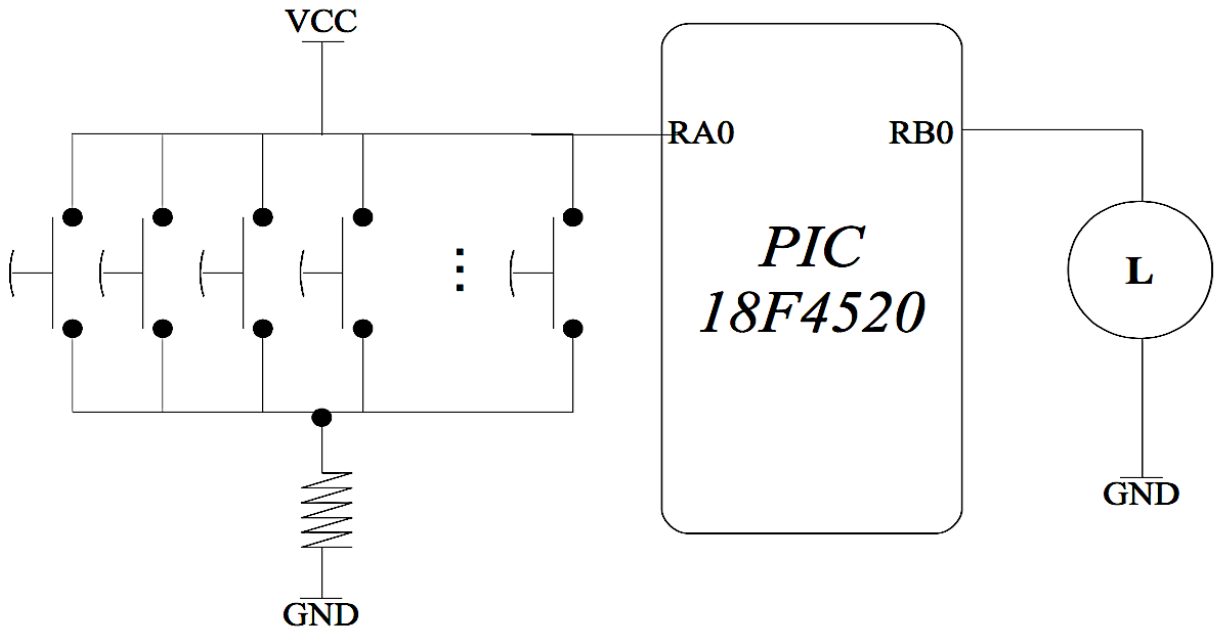
Figure 2 : Schéma de principe du PIC 24FJ128.

Pensez à rendre la figure 2 complétée avec votre copie.

5

Dépannage d'un programme pour PIC18F4520 (7 points)

Vous devez intervenir pour dépanner un système d'éclairage. Ce système d'éclairage est piloté par un microcontrôleur d'après un câblage qui est représenté ci-dessous.



Vous notez en particulier que les boutons poussoirs sont connectés sur la broche RA0 du PORTA et que la lampe d'éclairage est connectée sur la broche RB0 du PORTB. Un autre point important est que les boutons poussoirs sont câblés en logique dite « inverse », c'est à dire que l'action d'un bouton poussoir génère un état bas sur RA0.

Pour dépanner le programme, le seul élément dont vous disposez est le programme source (en langague C) qui a été écrit par le concepteur. Ce programme est donné ci-dessous.

```
//=============================================================================
// Filename: telerupteur.c
// Date:  2016/01/06
//=============================================================================
// Ce programme en langage C effectue la fonction télérupteur, c.a.d. elle permet de complémenter l'état
// d'une lampe connectée sur la broche RB0 par une action sur la broche RA0 du microcontroleur.
//=============================================================================
#include <p18f4520.h>
#define BROCHE_IN          RA4
#define BROCHE_OUT         RB0

// Programme principal
// ------------------------------------------------------------
void main(){

    // Configuration des ports E/S
    // -----------------------------------------------------
    ADCON1                  = 0x0F ;     // On force les broches A/N en mode numérique
    TRISAbits.BROCHE_IN     = 0;         // Basculement de BROCHE_IN en entrée
    TRISBbits.BROCHE_OUT    = 1;         // Basculement de BROCHE_OUT en sortie
    PORTBbits.BROCHE_OUT    = 0 ;        // On éteint la lampe connectée sur BROCHE_OUT

    // Boucle infinie...
    // -----------------------------------------------------
    while(1){
```

```
        // On bloque tant que BROCHE_IN n'est pas actionné…
        while (BROCHE_IN != 0);

        // On complémente l'état de BROCHE_OUT …
        PORTBbits.BROCHE_OUT = PORTBbits.BROCHE_OUT ^ 1 ;

        // On bloque tant que BROCHE_IN n'est pas actionné…
        while (BROCHE_IN !=0) ;
    }
}
```

? **1.** Donnez l'algorigramme du programme qui correspond au programme principal ci-dessus.

? **2.** Après compilation, on constate que le programme ne fonctionne pas. Il y a donc un ou plusieurs bugs dans le programme ci-dessus. Identifiez trois bugs dans le programme ci-dessus et corrigez les.

Pour économiser l'énergie, on souhaite faire évoluer le programme de manière à éviter que la lampe reste allumée indéfiniment. On va donc incorporer dans l'algorigramme une *temporisation* de manière à ce que la lampe s'éteigne au bout de T secondes.

Pour mettre en œuvre cette fonctionnalité, on va utiliser la fonction TIMER0 incluse dans le micro-contrôleur 18F4520.

La description du module TIMER0 (extraite du datasheet du 18F4520) est donnée en annexe I. En particulier, la partie « 11.4 **Timer0 Interrupt** » décrit comment une *interruption* peut être générée par le TIMER0 quand celui-ci subit un débordement de format (c'est à dire, quand le compteur « déborde »).

A la lecture de l'annexe I, on comprend que le TIMER0 est un « gros compteur » qui s'incrémente par des pas temporels Δ_T (en unité de temps)

$$\Delta_T = T_{\mathbf{cyc}} \times \mathbf{Prescaler}$$

où $T_{\mathbf{cyc}}$ est la durée du cycle d'horloge et $\mathbf{Prescaler}$ est une *valeur entière* qui peut prendre les valeurs suivantes

$$\mathbf{Prescaler} = 2^n \quad \mathbf{with} \quad n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$$

Par ailleurs, le temps *T* (en unité de temps) séparant le départ du compteur et son débordement s'écrit

$$(N_{\mathbf{max}} - N_{\mathbf{start}}) \times \Delta_T = T$$

avec $N_{\mathbf{start}}$ la *valeur de départ* du TIMER0 et $N_{\mathbf{max}}$ *la valeur max* du TIMER0.

3. **Après avoir lu l'annexe I, donnez les deux valeurs possibles de N_{max} ? Déduisez-en les deux valeurs maximums possibles pour T (exprimées en seconde).**

Note : pour ce calcul, on notera que qu'il faut 4 période d'horloge pour obtenir un temps de cycle, et que la période d'horloge vaut 0.25 micro-secondes.

4. **On se fixe Prescaler = 256. Donnez les valeurs des paramètres, N_{max} et N_{start} pour obtenir *exactement* T=16 secondes.**

5. **Donnez la valeur du registre de configuration du TIMER0 (registre** TMR0CON**, voir Annexe I) pour que les valeurs *N*max et Prescaler soient celles que vous souhaitez.**

6. **Donnez la valeur du registre de configuration des interruptions (registre** INTCON**, voir Annexe II) pour qu'une interruption soit générée lors du débordement de format du TIMER0.**

7. **Donnez les algorigrammes (programme principal et programme d'interruption) pour obtenir la fonctionnalité souhaitée.**

ANNEXE 1

PIC18F2420/2520/4420/4520

11.1 Timer0 Operation

Timer0 can operate as either a timer or a counter; the mode is selected with the T0CS bit (T0CON<5>). In Timer mode (T0CS = 0), the module increments on every clock by default unless a different prescaler value is selected (see **Section 11.3 "Prescaler"**). If the TMR0 register is written to, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

The Counter mode is selected by setting the T0CS bit (= 1). In this mode, Timer0 increments either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (T0CON<4>); clearing this bit selects the rising edge. Restrictions on the external clock input are discussed below.

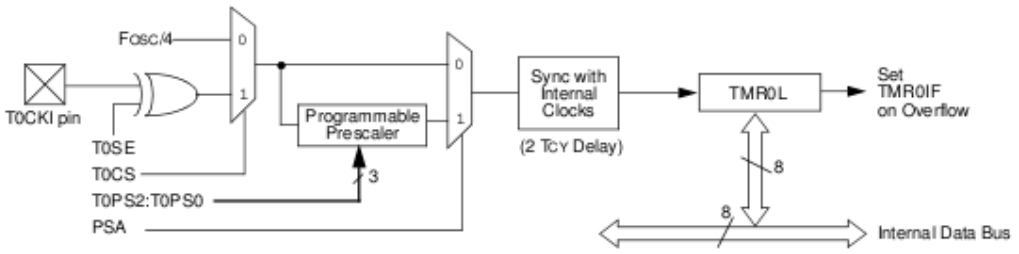
An external clock source can be used to drive Timer0; however, it must meet certain requirements to ensure that the external clock can be synchronized with the internal phase clock (TOSC). There is a delay between synchronization and the onset of incrementing the timer/counter.

11.2 Timer0 Reads and Writes in 16-Bit Mode

TMR0H is not the actual high byte of Timer0 in 16-bit mode; it is actually a buffered version of the real high byte of Timer0 which is not directly readable nor writable (refer to Figure 11-2). TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides the ability to read all 16 bits of Timer0 without having to verify that the read of the high and low byte were valid, due to a rollover between successive reads of the high and low byte.

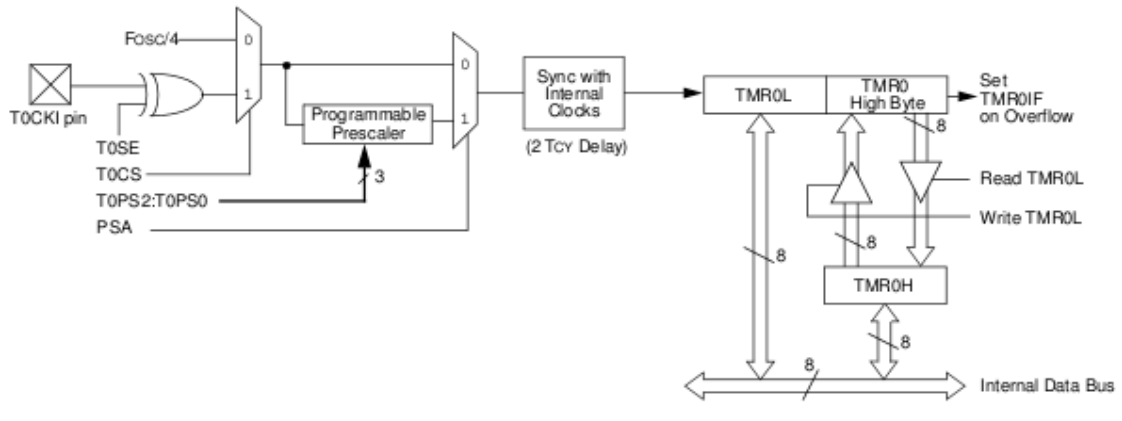
Similarly, a write to the high byte of Timer0 must also take place through the TMR0H Buffer register. The high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16 bits of Timer0 to be updated at once.

FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)



Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

FIGURE 11-2: TIMER0 BLOCK DIAGRAM (16-BIT MODE)



Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

11.3 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not directly readable or writable; its value is set by the PSA and T0PS2:T0PS0 bits (T0CON<3:0>) which determine the prescaler assignment and prescale ratio.

Clearing the PSA bit assigns the prescaler to the Timer0 module. When it is assigned, prescale values from 1:2 through 1:256 in power-of-2 increments are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., `CLRF TMR0`, `MOVWF TMR0`, `BSF TMR0`, etc.) clear the prescaler count.

Note:	Writing to TMR0 when the prescaler is assigned to Timer0 will clear the prescaler count but will not change the prescaler assignment.

11.3.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control and can be changed "on-the-fly" during program execution.

11.4 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or from FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF flag bit. The interrupt can be masked by clearing the TMR0IE bit (INTCON<5>). Before re-enabling the interrupt, the TMR0IF bit must be cleared in software by the Interrupt Service Routine.

Since Timer0 is shut down in Sleep mode, the TMR0 interrupt cannot awaken the processor from Sleep.

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register, Low Byte								50
TMR0H	Timer0 Register, High Byte								50
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	50
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	52

Legend: Shaded cells are not used by Timer0.

Note 1: PORTA<7:6> and their direction bits are individually configured as port pins based on various primary oscillator modes. When disabled, these bits read as '0'.

ANNEXE 2

9.0 INTERRUPTS

The PIC18F2420/2520/4420/4520 devices have multiple interrupt sources and an interrupt priority feature that allows most interrupt sources to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 0008h and the low priority interrupt vector is at 0018h. High priority interrupt events will interrupt any low priority interrupts that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

In general, interrupt sources have three bits to control their operation. They are:

- **Flag bit** to indicate that an interrupt event occurred
- **Enable bit** that allows program execution to branch to the interrupt vector address when the flag bit is set
- **Priority bit** to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>) enables all interrupts that have the priority bit set (high priority). Setting the GIEL bit (INTCON<6>) enables all interrupts that have the priority bit cleared (low priority). When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 0008h or 0018h, depending on the priority bit setting. Individual interrupts can be disabled through their corresponding enable bits.

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PICmicro® mid-range devices. In Compatibility mode, the interrupt priority bits for each source have no effect. INTCON<6> is the PEIE bit, which enables/disables all peripheral interrupt sources. INTCON<7> is the GIE bit, which enables/disables all interrupt sources. All interrupts branch to address 0008h in Compatibility mode.

When an interrupt is responded to, the global interrupt enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt. Low priority interrupts are not processed while high priority interrupts are in progress.

The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (0008h or 0018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit or the GIE bit.

Note:	Do not use the `MOVFF` instruction to modify any of the interrupt control registers while **any** interrupt is enabled. Doing so may cause erratic microcontroller behavior.

9.1 INTCON Registers

The INTCON registers are readable and writable registers, which contain various enable, priority and flag bits.

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 9-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

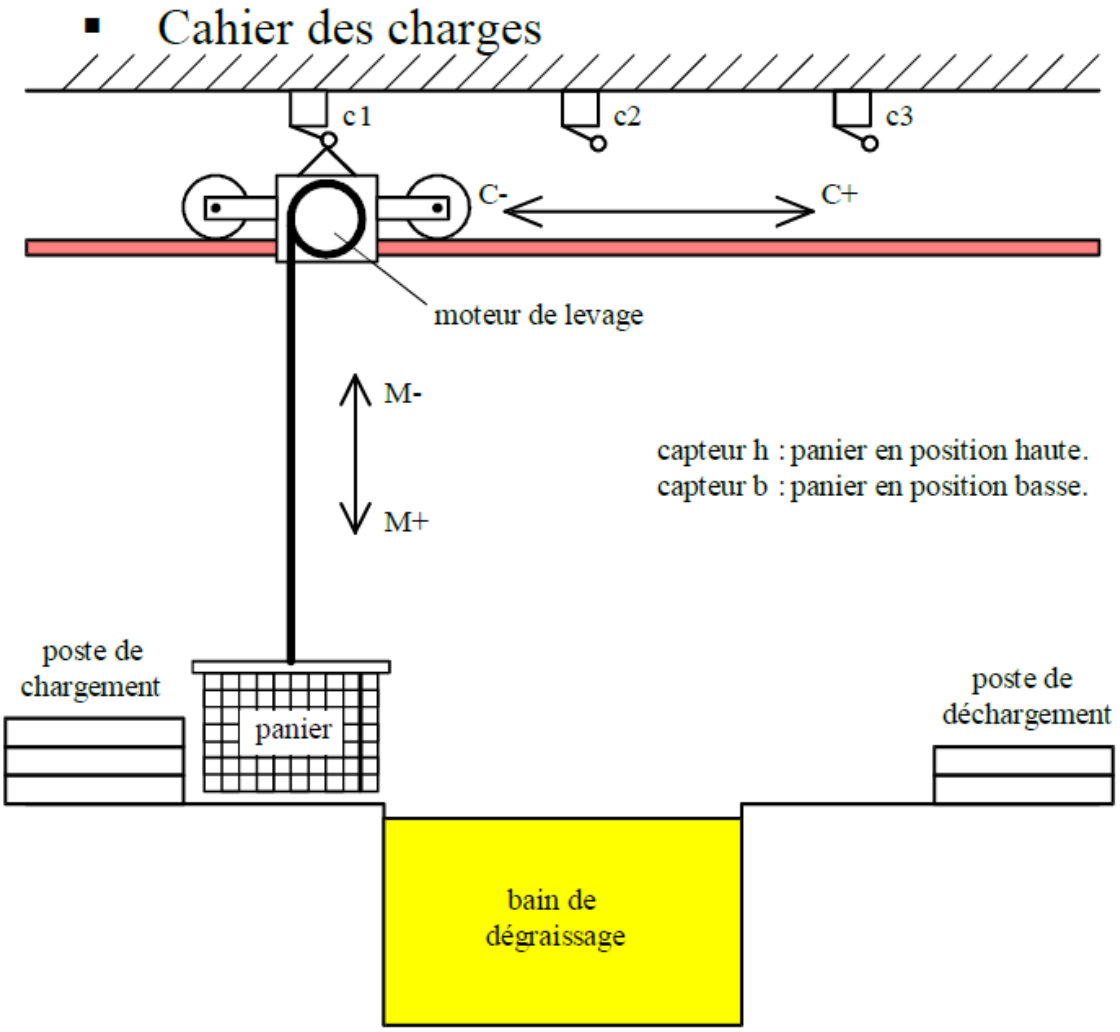
Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Seconde partie

Automatisme

Etude d'un système de traitement de pièces (3 points)



Fonctionnement

Un chariot se déplace sur un rail et permet en se positionnant au-dessus d'une cuve de nettoyer des pièces contenues dans un panier en les trempant dans un bac de dégraissage.

Cycle détaillé

Lorsque le chariot est en haut à gauche et que l'on appuie sur le bouton de départ cycle (dcy), le chariot va au-dessus du bac de dégraissage.

Le panier descend alors dans ce bac où on le laisse pendant 30s.

Après cette attente, la panier remonte.

Après cela, le chariot va jusqu'à l'extrême droite où il sera déchargé.

Quand le déchargement est terminé, le sstème revient dans sa position de départ.

Remarque

Le chargement et le déchargement du panier s’effectuent manuellement. Le contrôle du fait que le panier est déchargé sera validé par un bouton poussoir noté d.

1. Faire un inventaire des capteurs et actionneurs de ce système
2. Etablir le grafcet de niveau 1 de ce système
3. Ecrire en LADDER les actions associées à la descente et la remontée du panier.

15

Programme Crouzet (3 points)

On considère le schéma en Figure 1. C'est un programme permettant de faire fonctionner un automate programmable industriel.

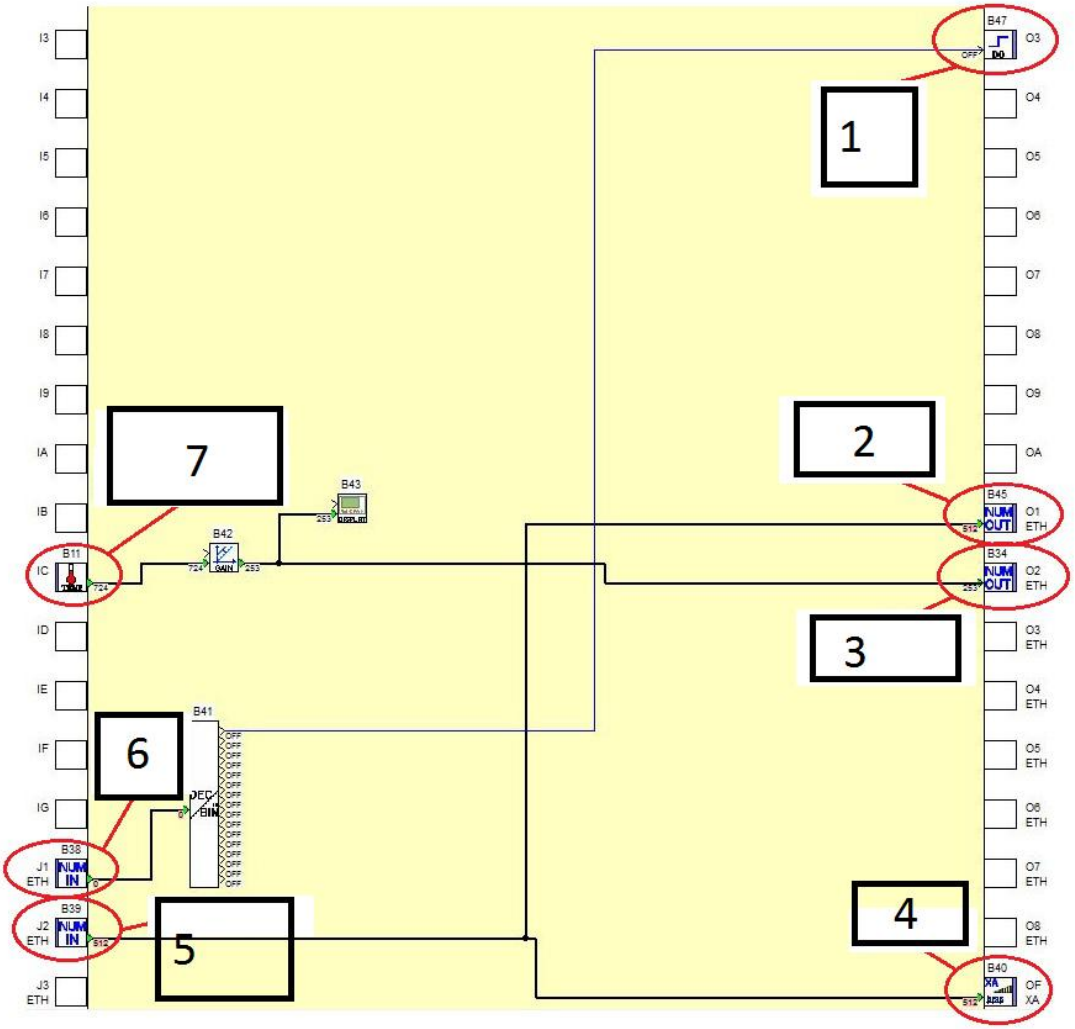


Figure 1: schéma de programmation associé à l'automate programmable industriel Crouzet

Question 1: De quel type de programme s'agit-il ? Ladder, ou FBD ? Quelle est la signification de l'acronyme FBD ?

Question 2: Ladder et FBD sont des langages de type graphique, citez un autre type de langage de programmation, avec un exemple.

Question 3: Associez chaque élément du schéma, de 1 à 7, avec (entre autres) les termes suivants:

-ON/OFF gradateur

-Mesure température

-Valeur PWM

...

16

Analyse de données de domotique (2 points)

Ce programme a permis de réaliser une montée en température dans une boîte munie d'une ampoule chauffante. La figure 2 montre les valeurs mesurées en rouge, et un modèle pour cette montée en température en bleu.

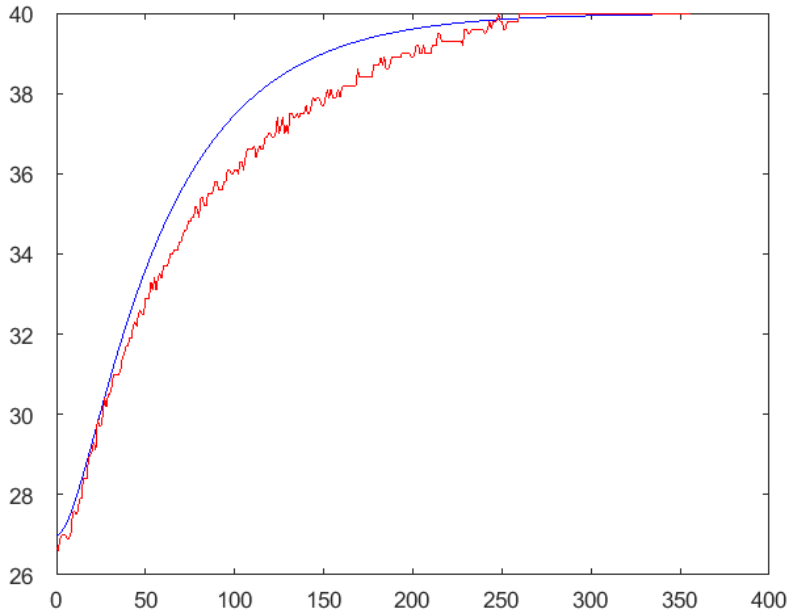


Figure 2: montée en température

La mesure de température est effectuée par l'entrée IC de l'automate qui est codée sur 10 bits; et le domaine de fonctionnement du capteur est constitué par l'intervalle [-10:40] en degrés Celsius.

On note T la température en degrés Celsius.

Question 1: donnez la relation entre T et IC.

Question 2: L'expérience correspondant à la figure 2 a été réalisée avec une commande PWM à 500. Une t_{500} =400 secondes environ ont été nécessaires pour passer de 27 à 40 degrés Celsius. Pouvez-vous donner une valeur approximative de t_{1000} le temps nécessaire pour passer de 27 à 40 degrés Celsius si la commande PWM avait été de 1000 ?