

Aix-Marseille Université

## Contrôle des connaissances

### UE3 – Automatismes et informatique industrielle

**Seul le polycopié de cours est autorisé  
Calculatrices autorisées.**

**Cet examen de 2 heures est composé de 2 parties :**

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

## **Première partie**

*Informatique industrielle*

## Questions de cours (4 pts)

### Attention

*Vous êtes notés sur la clarté de vos réponses. Pour avoir le maximum de points, elles devront être justes, courtes, et précises. Utilisez vos brouillons avant de rédiger sur vos copies d'examen.*

1. Identifier les éléments ci-dessous sur le schéma du micro-contrôleur présenté *figure 1* :

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

2. Le micro-contrôleur présenté *figure 1* est-il de structure *Von-Neumann* ou *Harvard* (justifiez votre choix)?

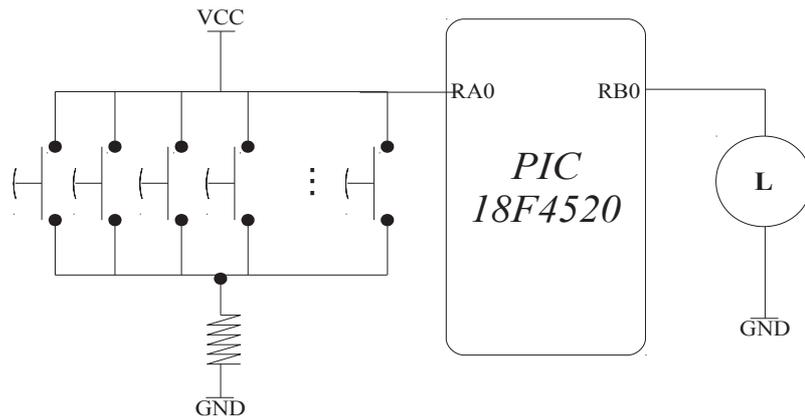
3. Le micro-contrôleur présenté *figure 1* est-il un microprocesseur 8 bits, 16 bits ou 32 bits ?

4. Identifier les modules de TIMER et de Conversion Analogique-Digital. Expliquez à quoi servent ces modules.

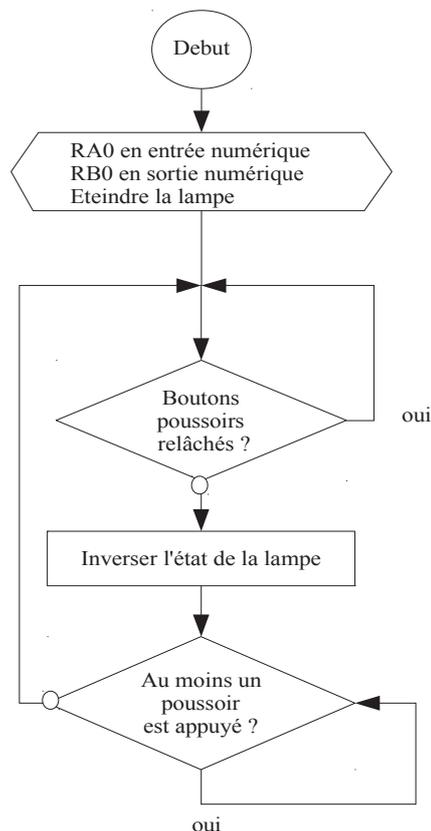


## Gestion d'un télerrupteur par PIC18F4520 (8 points)

On souhaite gérer l'éclairage d'une cage d'escalier par un dispositif de type « télerrupteur ». Un télerrupteur est un dispositif permettant de commander l'éclairage d'une lampe avec des boutons poussoirs selon le principe suivant : tout appui sur un des poussoirs inverse l'état de la lampe (0: la lampe est éteinte, 1: la lampe est allumée). On souhaite réaliser cette fonction à partir d'un microcontrôleur. Les poussoirs sont connectés sur la broche RA0 du PORTA et la lampe sur la broche RB0 du PORTB comme reproduit ci-dessous.



On notera que les boutons poussoirs sont câblés en logique inversée, c'est à dire que l'action d'un bouton poussoir génère un état bas sur RA0. On donne ci-dessous l'algorithme du programme permettant de réaliser la fonction souhaitée.



**?** 1. Expliquez le rôle des deux tests booléens qui apparaissent dans cet algorithme.

On donne ci-dessous un « programme à trou » en langage C pour réaliser l'algorithme précédent.

**?** 2. Recopiez le « programme à trou » ci-dessous sur votre copie en n'oubliant pas de compléter les parties manquantes (parties grisées).

```

=====
; Filename: telerupteur.c
; Company: Universite Paul Cezanne
; Revision: 1.00
; Date: 2014/01/06
;
; Ce programme en langage C effectue la fonction télérupteur, c.a.d. elle permet de compléter l'état
; d'une lampe connectée sur la broche RB0 par une action sur la broche RA0 du microcontrôleur.
=====
#include <p18f4520.h>

#define MASQUE_TRISA 0b11111111
#define MASQUE_TRISB 0b11111110
#define MASQUE_PORTB 0b00000001

void main(){

    // _____
    ADCON1          = 0x0F ;                // Passage des broches A/N en numérique

    PORTA          = 0x00;
    TRISA          = TRISA | MASQUE_PORTA;  // _____
    PORTB          = 0x00;
    TRISB          = TRISB | MASQUE_PORTB;  // _____

    PORTBbits.RB0  = 0 ;                  // _____

    // _____
    while(1){

        // _____
        while (RA0!= 0);

        // _____
        PORTB = PORTB ^ _____;

        // _____
        while (_____);

    }
}

```

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'algorithme 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 microcontrôleur 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  $\Delta_T$  (en unité de temps)

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

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

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

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

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

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

- ? 3. **Après avoir lu l'annexe I, donnez les deux valeurs possibles de  $N_{\text{max}}$  ? 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,  $N_{\text{max}}$  et  $N_{\text{start}}$  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  $N_{\text{max}}$  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 algorithgrammes (algorithgramme du programme principal + algorithgramme du programme d'interruption) pour obtenir la fonctionnalité souhaitée.**

*Note : Pour l'algorithgramme principale, vous pourrez modifier l'algorithgramme donné page 6. Vous penserez notamment à modifier la partie *initialisation* de cet algorithgramme pour tenir compte du TIMER0 et de la gestion des interruptions pour ce TIMER0.*

- ? 8. **Complétez le programme à trou ci-dessous qui implémente les deux algorithgrammes précédents.**

```
//=====
// Filename: telerupteur2.c
// Company: Universite Paul Cezanne
// Revision: 1.00
// Date: 2014/01/06
```





# 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)

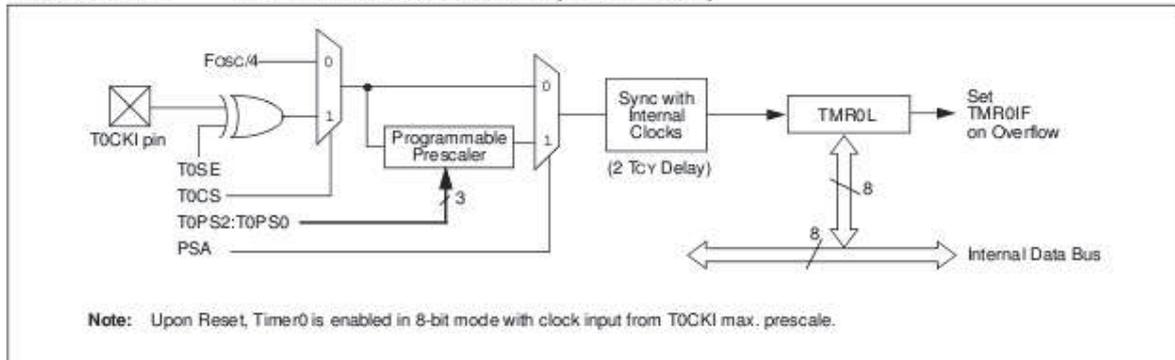
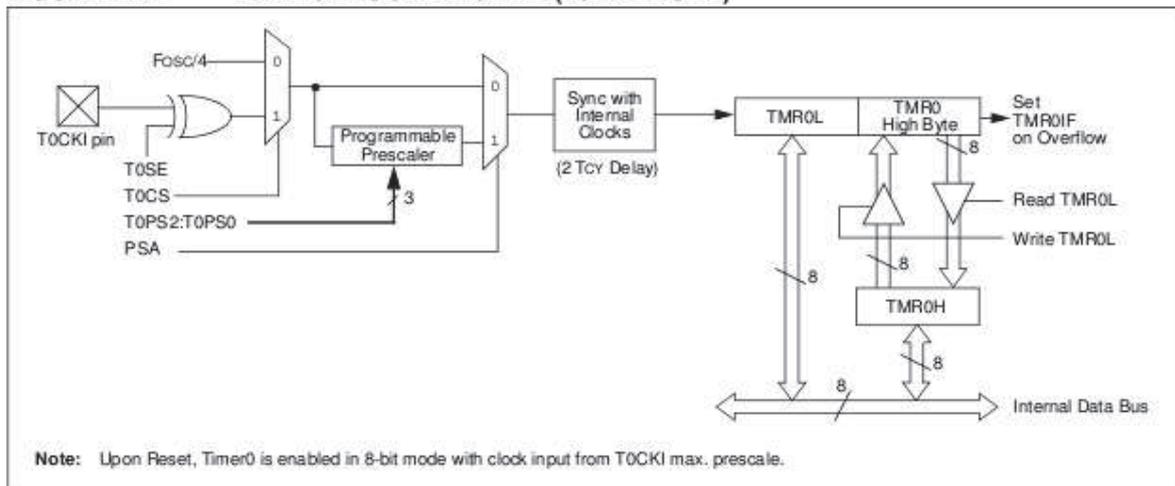


FIGURE 11-2: TIMER0 BLOCK DIAGRAM (16-BIT MODE)



# PIC18F2420/2520/4420/4520

## 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 <sup>(1)</sup>         | RA6 <sup>(1)</sup> | 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'.

## PIC18F2420/2520/4420/4520

### 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.

## PIC18F2420/2520/4420/4520

### 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 | <b>GIE/GIEH:</b> Global Interrupt Enable bit<br><u>When IPEN = 0:</u><br>1 = Enables all unmasked interrupts<br>0 = Disables all interrupts<br><u>When IPEN = 1:</u><br>1 = Enables all high priority interrupts<br>0 = Disables all interrupts                                                                                          |           |        |        |       |        |        |       |
| bit 6 | <b>PEIE/GIEL:</b> Peripheral Interrupt Enable bit<br><u>When IPEN = 0:</u><br>1 = Enables all unmasked peripheral interrupts<br>0 = Disables all peripheral interrupts<br><u>When IPEN = 1:</u><br>1 = Enables all low priority peripheral interrupts<br>0 = Disables all low priority peripheral interrupts                             |           |        |        |       |        |        |       |
| bit 5 | <b>TMR0IE:</b> TMR0 Overflow Interrupt Enable bit<br>1 = Enables the TMR0 overflow interrupt<br>0 = Disables the TMR0 overflow interrupt                                                                                                                                                                                                 |           |        |        |       |        |        |       |
| bit 4 | <b>INT0IE:</b> INT0 External Interrupt Enable bit<br>1 = Enables the INT0 external interrupt<br>0 = Disables the INT0 external interrupt                                                                                                                                                                                                 |           |        |        |       |        |        |       |
| bit 3 | <b>RBIE:</b> RB Port Change Interrupt Enable bit<br>1 = Enables the RB port change interrupt<br>0 = Disables the RB port change interrupt                                                                                                                                                                                                |           |        |        |       |        |        |       |
| bit 2 | <b>TMR0IF:</b> TMR0 Overflow Interrupt Flag bit<br>1 = TMR0 register has overflowed (must be cleared in software)<br>0 = TMR0 register did not overflow                                                                                                                                                                                  |           |        |        |       |        |        |       |
| bit 1 | <b>INT0IF:</b> INT0 External Interrupt Flag bit<br>1 = The INT0 external interrupt occurred (must be cleared in software)<br>0 = The INT0 external interrupt did not occur                                                                                                                                                               |           |        |        |       |        |        |       |
| bit 0 | <b>RBIF:</b> RB Port Change Interrupt Flag bit<br>1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)<br>0 = None of the RB7:RB4 pins have changed state<br><b>Note:</b> 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*





## I. Maquette Domotique

Considérez le programme FBD suivant (figure 1) :

Ce programme, tel qu'il est, est sensé allumer la lampe de la boîte 'éclairage' de la maquette de domotique, et est associé à l'automate dont le schéma de câblage est présenté en figure 2. Le programme doit aussi allumer la lampe de la partie 'chauffage'.

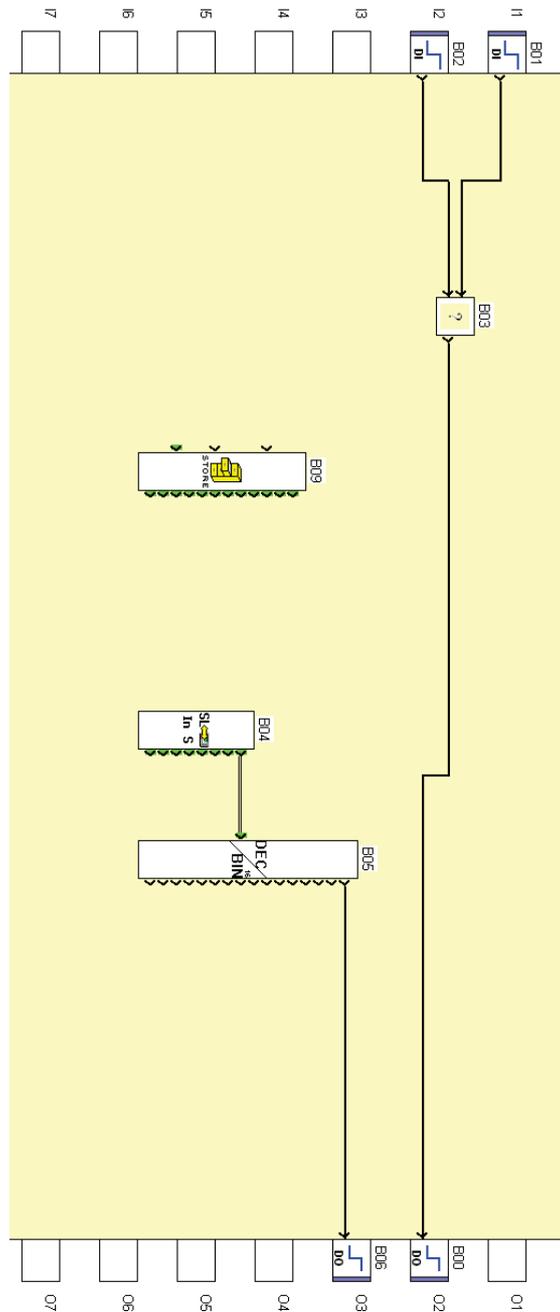


Figure 1

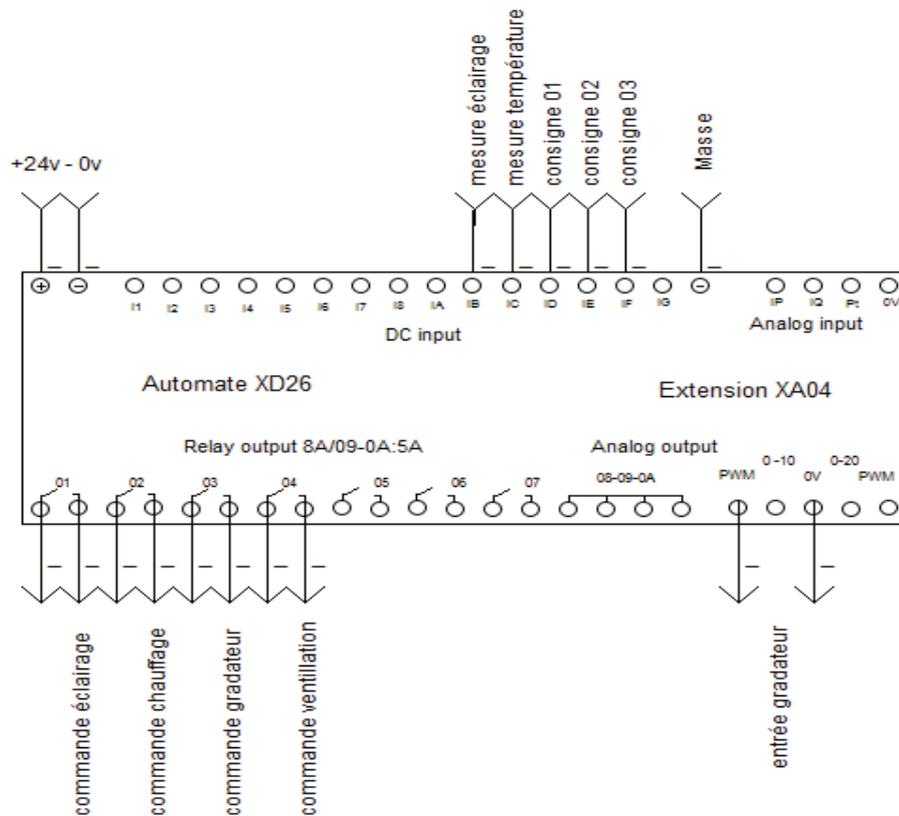


Figure 2

On s'intéresse tout d'abord à la boîte 'éclairage' de la maquette.

1. Le programme de la figure 1 est-il de type 'instruction list' ou 'functional block diagram' ? Il existe une erreur dans le programme. Quelle est-elle ?

On s'intéresse maintenant à la boîte 'chauffage' de la maquette.

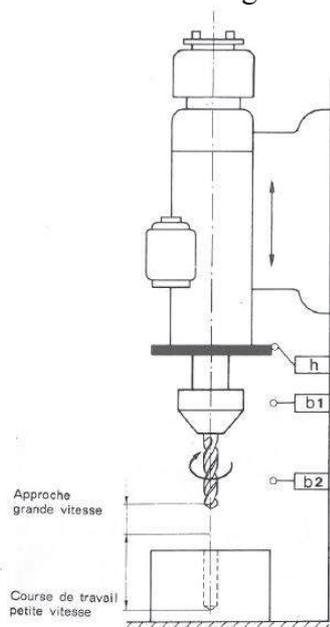
L'élément SL 'IN S' de la figure 1 est une entrée liaison série RS 232. On suppose qu'elle est connectée à l'automate par une extension qui n'est pas représentée sur la figure 2. Une autre extension, la 'XN05', est aussi présente, mais non représentée sur la figure 2.

2. D'où viennent à votre avis les informations qui sont en sortie de l'élément SL 'IN S' ? De la partie opérative du système 'maquette' ? de la partie commande ? De l'ordinateur ?
3. Remarquez que l'élément SL est connecté à l'output O3. Que souhaite-t-on faire par le biais de cet élément ?
4. Pour assurer une cohérence dans le programme, et au vu de la réponse à la question 3, quel est l'élément à placer en sortie OF XA du programme ?
5. Quel est le nom de la prise présente sur l'extension XN05 qui permet de faire l'enregistrement ?
6. Quel est le nom du protocole utilisé lors du transfert de données de l'automate vers l'ordinateur ?

## II. Commande d'une perceuse

Une perceuse se compose d'un bâti fixe et d'une console mobile par rapport au bâti. Les pièces à percer sont approvisionnées et fixées manuellement sur un montage solidaire du bâti.

La broche tourne en permanence. L'opérateur ayant fixé la pièce donne alors l'information de départ cycle (dcy). Après une approche à grande vitesse, le perçage s'effectue à petite vitesse. Dès le perçage terminé, la broche remonte en grande vitesse jusqu'à la position haute.



On désignera les actions à mener par :

Descendre en grande vitesse (DGV)  
Remonter en grande vitesse (RGV)  
Descendre en petite vitesse (DPV)  
Rotation de la broche (RB)

Les informations délivrées par la partie opérative sont :

Départ cycle (dcy)  
Position haute (h)  
Position basse (b2)  
Approche terminée (b1)

Etablir le GRAFCET de niveau 1 de la commande de cette perceuse.