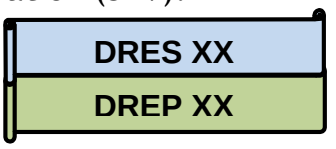
ROYAUME DU MAROC
MINISTERE DE L'EDUCATION NATIONALE
Académie de Casablanca
DÉLÉGATION DE MOHAMMEDIA
Lycée Technique Mohammedia

Matière :	Science de l'Ingénieur - **A.T.C** -	Pr.MAHBAB
Section :	Sciences et Technologies Électriques	Système n°5

❖ Sujet :

- Le sujet comporte au total 20 pages.
- Le sujet comporte 3 types de documents :

- Pages 01 à 04 : Socle du sujet comportant les situations d'évaluation (SEV).
- Pages 05 à 12 : Documents ressources portant la mention.
- Pages 13 à 20 : Documents réponses portant la mention.



20 pages

❖ 5 Fiches cours :

- Fiche cours n°21 « GRAFCET »
  - Partition d'un GRAFCET
  - Différents type d'action
  - Notion de synchronisation ou de coordination
- Fiche cours n°22 « EEPROM du 16 F 84 »
- Fiche cours n°23 « TIMER0 du 16 F 84 »
- Fiche cours n°24 « Les interruptions du 16 F 84 »
- Fiche cours n°25 « Programmation des PLD »

14 pages

❖ 2 TD:

- TD n°8 « Partition d'un GRAFCET et Notion de tâche »
- TD n°9 « Programmation des PLD »

12 pages

PARKING AUTOMATIQUE

Nombre de pages : 47

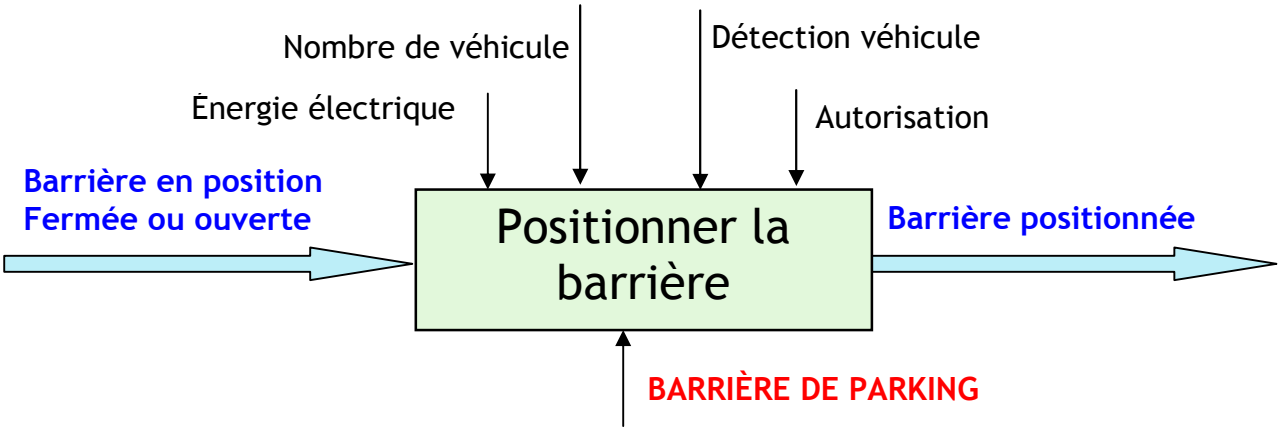
CLASSES : 2 STE.....

PARKING AUTOMATIQUE

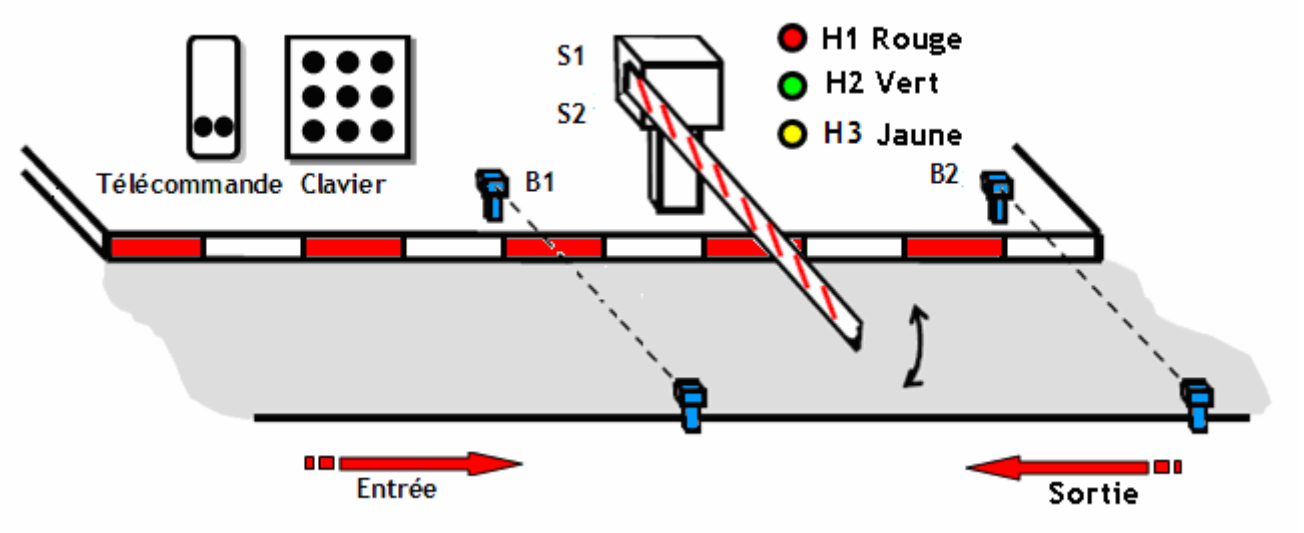
1- PRESENTATION DU SYSTEME :

On se propose d'étudier le fonctionnement d'une barrière de parking de stationnement de véhicule.
La barrière fonctionne en tenant compte de diverses informations suivantes :

- ✓ Présence d'une voiture devant la barrière (entrée ou sortie).
- ✓ Autorisation d'ouverture de la barrière (clavier ou télécommande) si entrée, sans autorisation si sortie.
- ✓ Nombre de véhicule dans le parking.



2- DESCRIPTION DU SYSTEME :



Le système est composé de :

- ❖ Une **barrière** ; commandée par un moteur asynchrone ;
- ❖ Deux cellules photoélectriques **B1** et **B2** associées à deux réflecteurs qui détectent les véhicules en entrée et en sortie du parking ;
- ❖ Deux capteurs à galet **S1** et **S2** détectant la position de la barrière (positions haute et basse) ;
- ❖ Un moteur asynchrone triphasé (**MAS**) associé à un réducteur mécanique ;
- ❖ Un **variateur de vitesse** pour moteur triphasé ;

- Une carte électronique à base du microcontrôleur **16 F 84**;
- Une **alimentation modulaire** régulée (intégrée dans le boîtier);
- Un **clavier** et une **télécommande** permettent de délivrer les consignes de fonctionnement (autorisation d'entrée des véhicules).
  - ✓ L'autorisation issue du clavier (**U1**) est délivrée si le code fourni est correct.
  - ✓ L'autorisation issue de la télécommande (**U2**) est délivrée lorsqu' on appuie sur le bouton gauche de la télécommande.
- Trois voyants permettent d'afficher des messages à destination des utilisateurs du système. Ces trois Voyants permettent de délivrer les messages suivants :
  - ✓ H_1 'Voyant rouge' : Accès non autorisé au parking.
  - ✓ H_2 'Voyant vert' : Accès autorisé au parking.
  - ✓ H_3 'Voyant jaune' : Parking complet.

3- FONCTIONNEMENT DÉSIRÉ :

3-1- Entrée d'un véhicule dans le parking :

Les étapes de fonctionnement sont les suivantes :

- Présence d'un véhicule devant l'entrée de la barrière détectée par le capteur **B1** ;
- Demande d'autorisation d'ouverture à l'aide de la **télécommande** ou du **clavier**.
- Si le parking n'est pas complet (**nombre** de véhicule dans le parking **< 50** 'voyant H3 éteint'), la barrière s'ouvre par fermeture du relais **KM1** et le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Ouverture complète de la barrière, détectée par l'action sur **S1**, le voyant H_2 ' Accès autorisé au parking' s'allume et le véhicule peut entrer dans le parking en franchissant la barrière.
- Entrée du véhicule ;
- Entrée complète du véhicule dans le parking détectée par le capteur **B2**.
- Le nombre de véhicule présent dans le parking est alors **incrémenté** de **1**.
- Fermeture de la barrière par fermeture du relais **KM2**, après une temporisation de **10 s** ;
- Pendant la fermeture de la barrière, le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Fermeture complète de la barrière, détectée par l'action sur **S2** ;

3-2- Sortie d'un véhicule du le parking :

Les étapes de fonctionnement sont les suivantes :

- Présence d'un véhicule devant la sortie de la barrière détectée par le capteur **B2** ;
- Ouverture de la barrière s'ouvre par fermeture du relais **KM1** et le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Ouverture complète de la barrière, détectée par l'action sur **S1**, le voyant H_2 ' Accès autorisé au parking' s'allume et le véhicule peut sortir du parking en franchissant la barrière.
- Sortie du véhicule ;
- Sortie complète du véhicule du parking détectée par le capteur **B1**.
- Le nombre de véhicule présent dans le parking est alors **décrémenté** de **1**.
- Fermeture de la barrière par fermeture du relais **KM2**, après une temporisation de **10 s** ;
- Pendant la fermeture de la barrière, le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Fermeture complète de la barrière, détectée par l'action sur **S2** ;

4- DESCRIPTION DE LA PARTIE COMMANDE :

La commande de la barrière est réalisée par une carte électronique à base d'un microcontrôleur de type PIC 16 F 84 dont les caractéristiques sont données au **DRES 01** page 13.

On suppose que le nombre de véhicules présents dans le parking est stocké dans une case mémoire de L'EEPROM (adresse 02h).

On dispose des 5 sous programmes suivants :

- **Lecture_EEPROM :** la donnée se trouvant dans l'EEPROM à l'adresse 02_H, sera enregistrée dans la RAM à l'adresse $0C_H$ (soit Cp).
- **Ecriture_EEPROM :** la donnée se trouvant dans la RAM à l'adresse $0C_H$ (soit Cp) sera enregistrée dans l'EEPROM (adresse 02_H).
- **TEMPO :** un sous programme de temporisation de 10 s.

  Ce sous programme de temporisation tempo, utilise le TIMER 0 pour compter un temps T1 de 10 s. Le TIMER0 utilise une horloge externe – fréquence 1 KHz - appliquée sur la broche TOCKI/ RA4, avec une division par 64 et active sur front descendant.
- **Ouvrir_Barrière :** ce sous programme permet d'ouvrir la barrière et d'allumer le voyant vert.
- **Fermer_Barrière :** ce sous programme permet de fermer la barrière.

On utilise ces sous programmes pour structurer le programme de fonctionnement du parking. Le nombre de véhicule présent dans le parking ; est stocké dans la ROM électrique du PIC 16 F 84 ; à fin d'éviter la perte de cette information ; lorsqu'il y a coupure du courant dans le parking.

SEV 1	GRAFCET	/ pts

RESSOURCES A EXPLOITER : **DRES 02** page 14,
'Description' et 'Fonctionnement' pages 01 et 02.

Tâche	Partition du GRAFCET	/ pts

Sur le document **DRES 03** page 15, on donne le GRAFCET global du parking du point de vue partie opérative.

1. Sur le document **DREP 01** page 05, compléter le GRAFCET du point de vue partie commande ; correspondant au GRAFCET global du document **DRES 03** page 15.
2. Sur le document **DREP 02** page 06, compléter le GRAFCET du point de vue partie commande codé PIC; correspondant au GRAFCET global du document **DRES 03** page 15.
3. On veut réécrire ce GRAFCET d'une manière simplifiée en considérant les séquences 'Entrée d'un véhicule' et 'Sortie d'un véhicule' comme des tâches (sous programme) intitulées Tâche 1 et Tâche 2.

En se referant au document **DRES 03** page 15, compléter le GRAFCET principal et les GRAFCETS tâche 1 et tâche 2, sur le document **DREP 03** page 07.

SEV 2 — PROGRAMME DE FONCTIONNEMENT — / pts

RESSOURCES A EXPLOITER : **DRES 01, DRES 02, DRES 06, DRES 07** page 13, 14, 18 et 19

Tâche 1 — Ouvrir et Fermer la barrière — / pts

1. Sur le document **DREP 04** page 08, compléter l'organigramme **Ouvrir_Barrière.**
2. Sur le document **DREP 04** page 08, compléter l'organigramme **Fermer_Barrière.**
3. Sur le document **DREP 04** page 08, compléter le sous programme **Ouvrir_Barrière.**
4. Sur le document **DREP 04** page 08, compléter le sous programme **Fermer_Barrière.**

Tâche 2 — EEPROM et TIMER — / pts

RESSOURCES A EXPLOITER : 'DESCRIPTION DE LA PARTIE COMMANDE' pages 03
DRES 04, DRES 05 page 16, 17

1. Sur le document **DREP 05** page 09, compléter le sous programme **Lecture_EEPROM.**
2. Sur le document **DREP 05** page 09, compléter le sous programme **Ecriture_EEPROM.**
3. Sur le document **DREP 05** page 09, compléter le sous programme **Tempo.**

Tâche 3 — Programme de fonctionnement — / pts

1. Compléter l'organigramme du document **DREP 06** page 10.
2. sur le document **DREP 07** page 11, Compléter le programme de configuration du PIC. Pour la configuration du TIMER0 utiliser le document **DRES 05** page 17 et pour la configuration des PORTA et B utiliser le document **DRES 06** page 18
3. Compléter le programme principal du parking, sur le document **DREP 07** page 11.

SEV 3 — AFFICHAGE DU NOMBRE DE VEHICULE — / pts

RESSOURCES A EXPLOITER : **DRES 08** page 20

Pour l'affichage du nombre de véhicule dans le parking, on utilise ; un décodeur Binaire/BCD ; 2 décodeurs BCD/7 segments et 2 afficheurs 7 segments. Voir document **DRES 08** page 20.

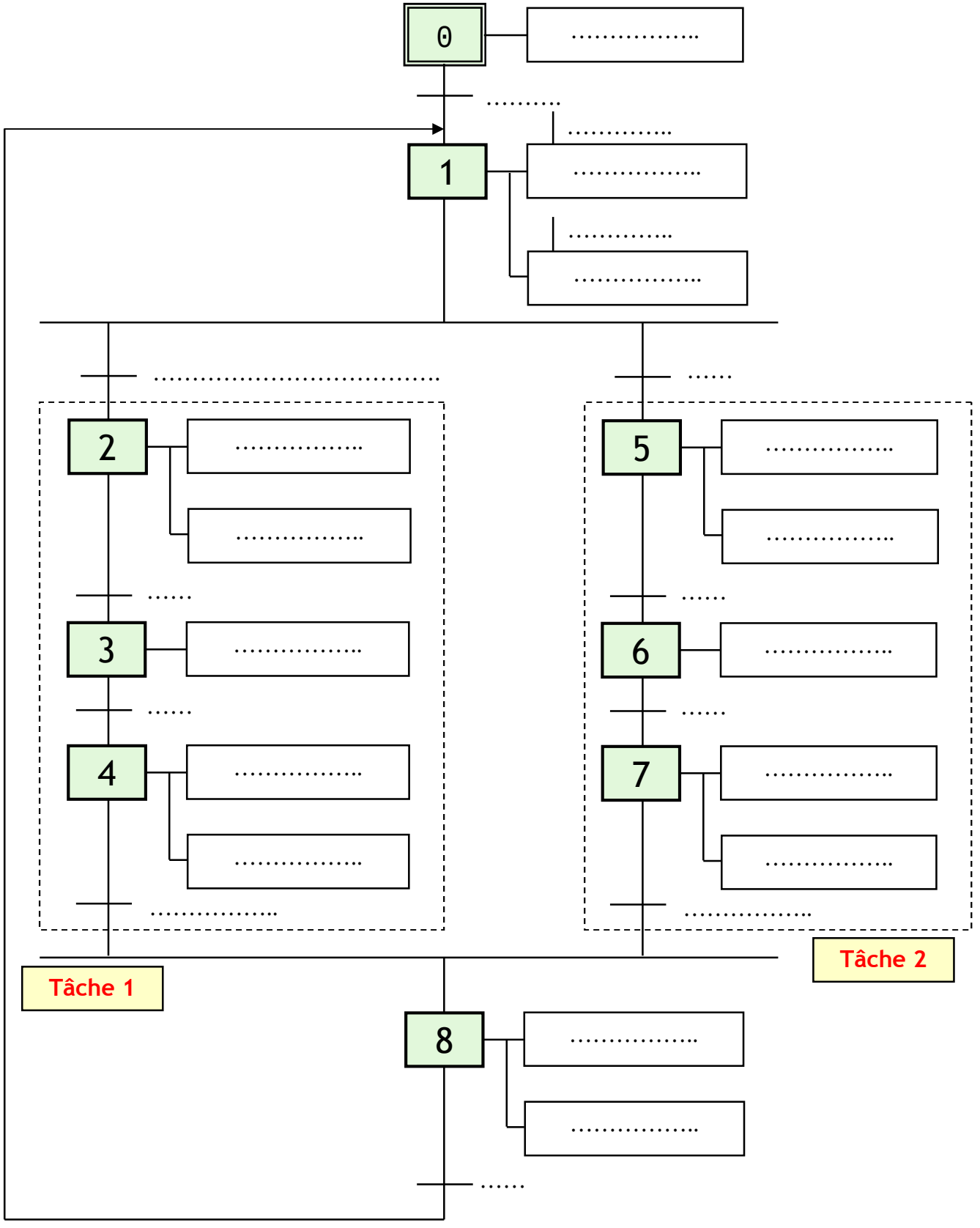
Tâche — Programmation du GAL 16V8 — / pts

Le décodeur BCD/7 segments est réalisé par un circuit logique programmable du type 16 V 8. Sur le document **DREP 08** page 12 ; compléter le programme ABEL du décodeur.

DREP 01

DOCUMENT A RENDRE

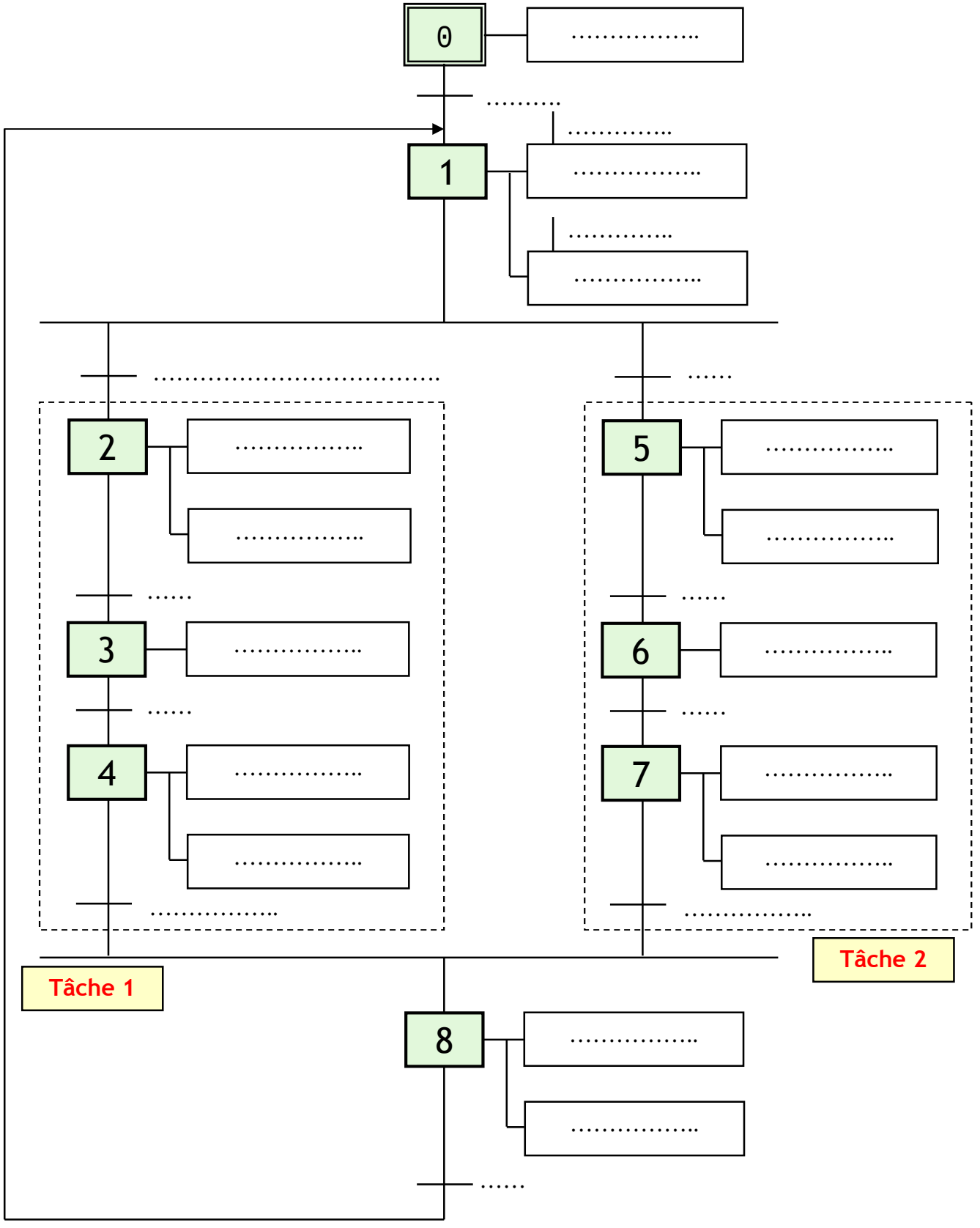
GRAFCET point de vue partie commande



DREP 02

DOCUMENT A RENDRE

GRAFCET point de vue partie commande codé PIC 16 F 84



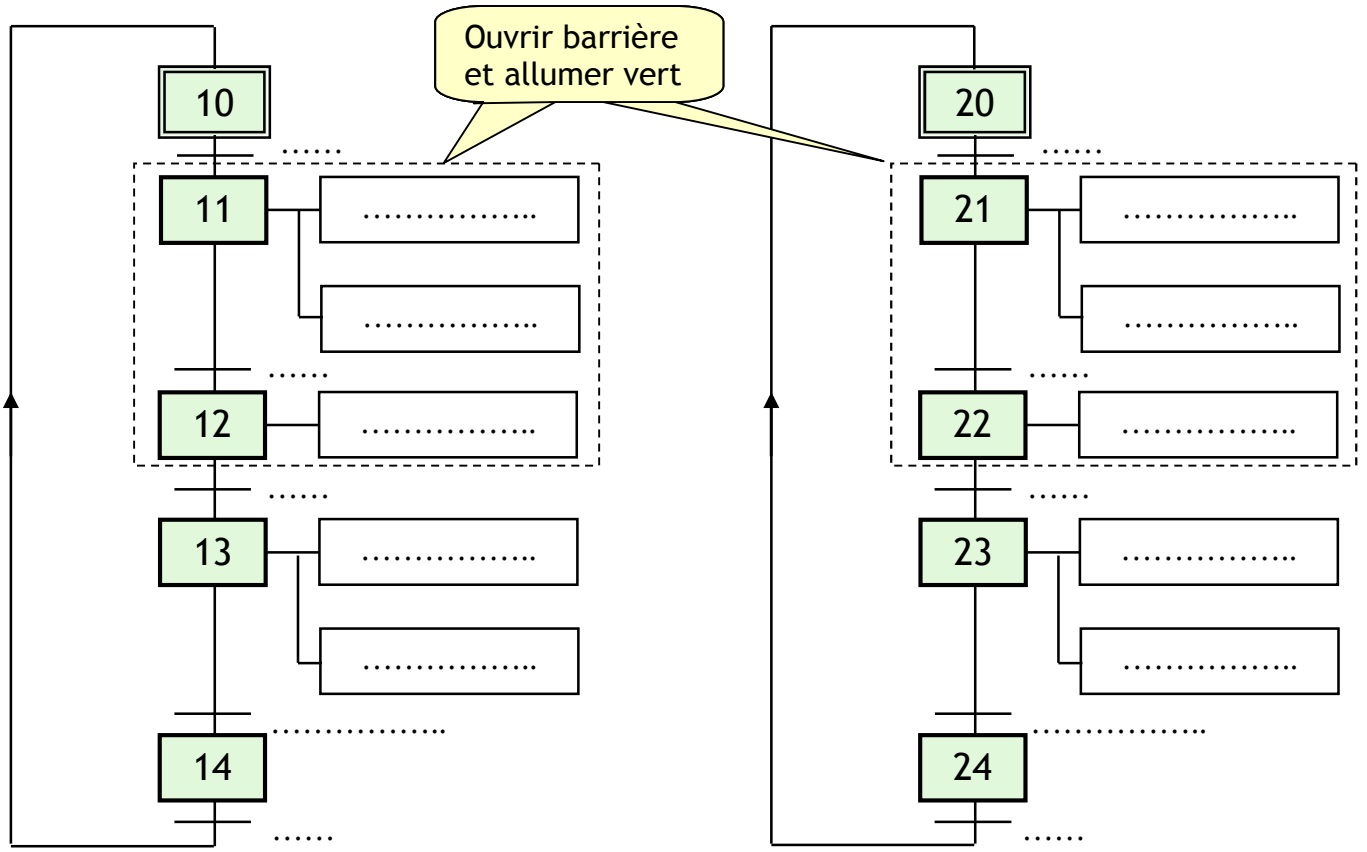
DREP 03

DOCUMENT A RENDRE

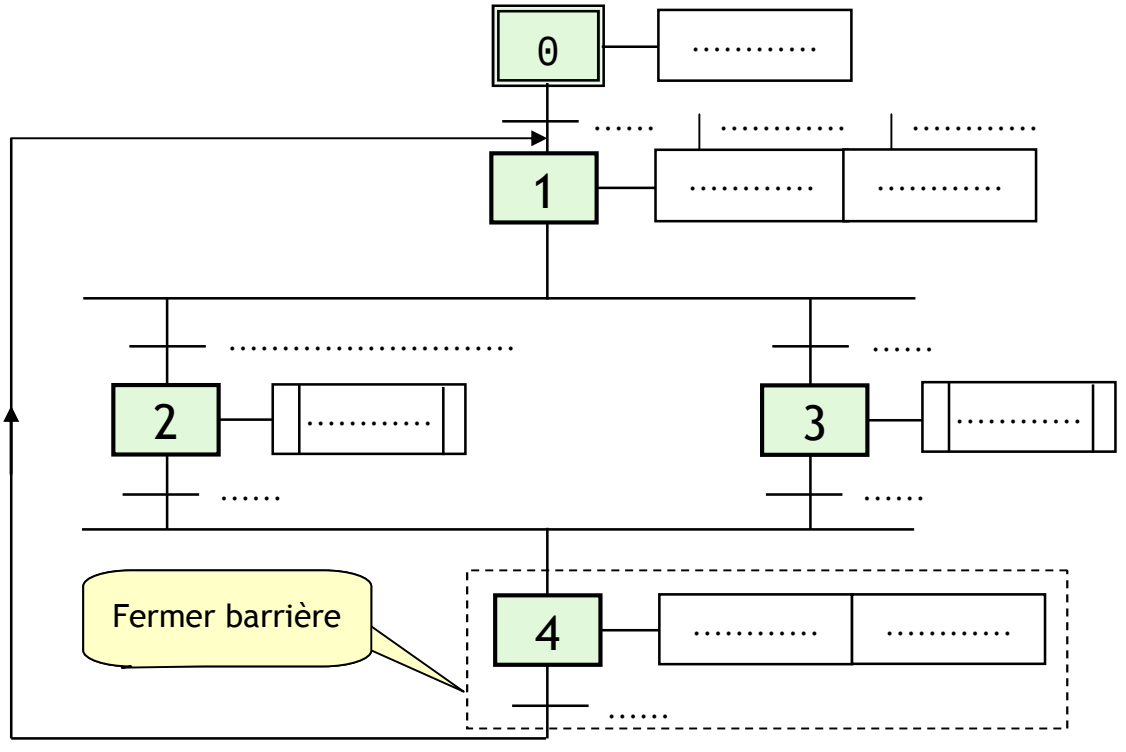
Partition du GRAFCET global (P.C)

GRAFCET Tâche 1

GRAFCET Tâche 2



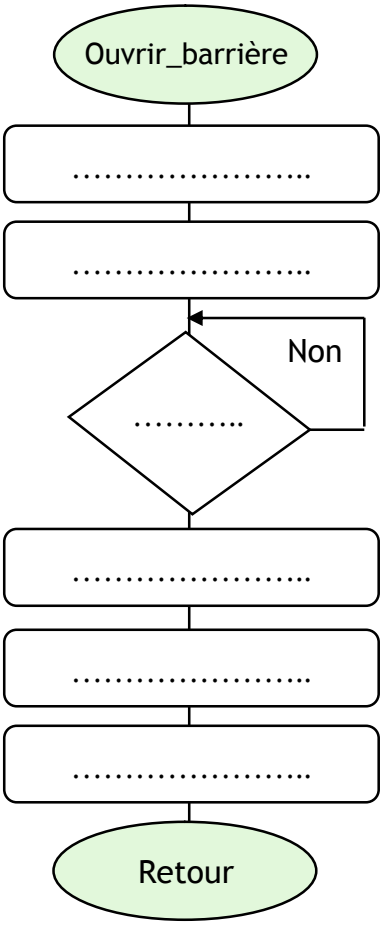
GRAFCET PRINCIPAL



DREP 04

DOCUMENT A RENDRE

Sous programme ouvrir_Barrière

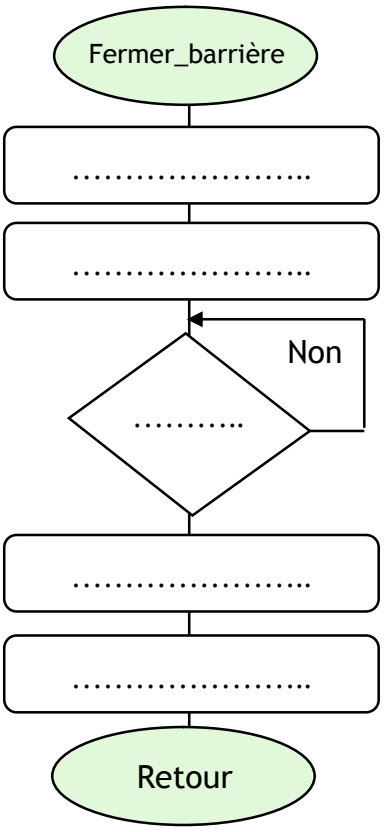


Organigramme

Programme

Ouvrir_barriere .....
.....
LAB5 .....
.....
.....
.....
.....
.....

Sous programme fermer_Barrière



Organigramme

Programme

Fermer_barriere .....
.....
LAB6 .....
.....
.....
.....
.....

DREP 05

DOCUMENT A RENDRE

Sous programme Lecture_EEPROM

```
Lecture_EEPROM  .....................   ;
                .....................   ; l'adresse à lire
                .....................   ; Bank 1
                .....................   ; lecture EPROM
                .....................   ; Bank 0
                .....................   ; EEDATA dans W
                .....................   ; W dans Cp
                RETURN                  ;
```

Sous programme Ecriture_EEPROM

```
Ecriture_EEPROM .....................   ;
                .....................   ; définition de l'adresse
                .....................   ;
                .....................   ; définition de la donnée
                .....................   ; Bank 1
                .....................   ; autorisation de l'écriture
                .....................   ;
                .....................   ; écriture de 0x55
                .....................   ;
                .....................   ; écriture de 0xAA
                .....................   ; écriture dans EEPROM
        LAB7    .....................   ;
                .....................   ; écriture terminée
                BCF     EECON, EEIF     ; remise à zéro du témoin EEIF
                BCF     STATUS, RP0     ; Bank 0
                RETURN                  ;
```

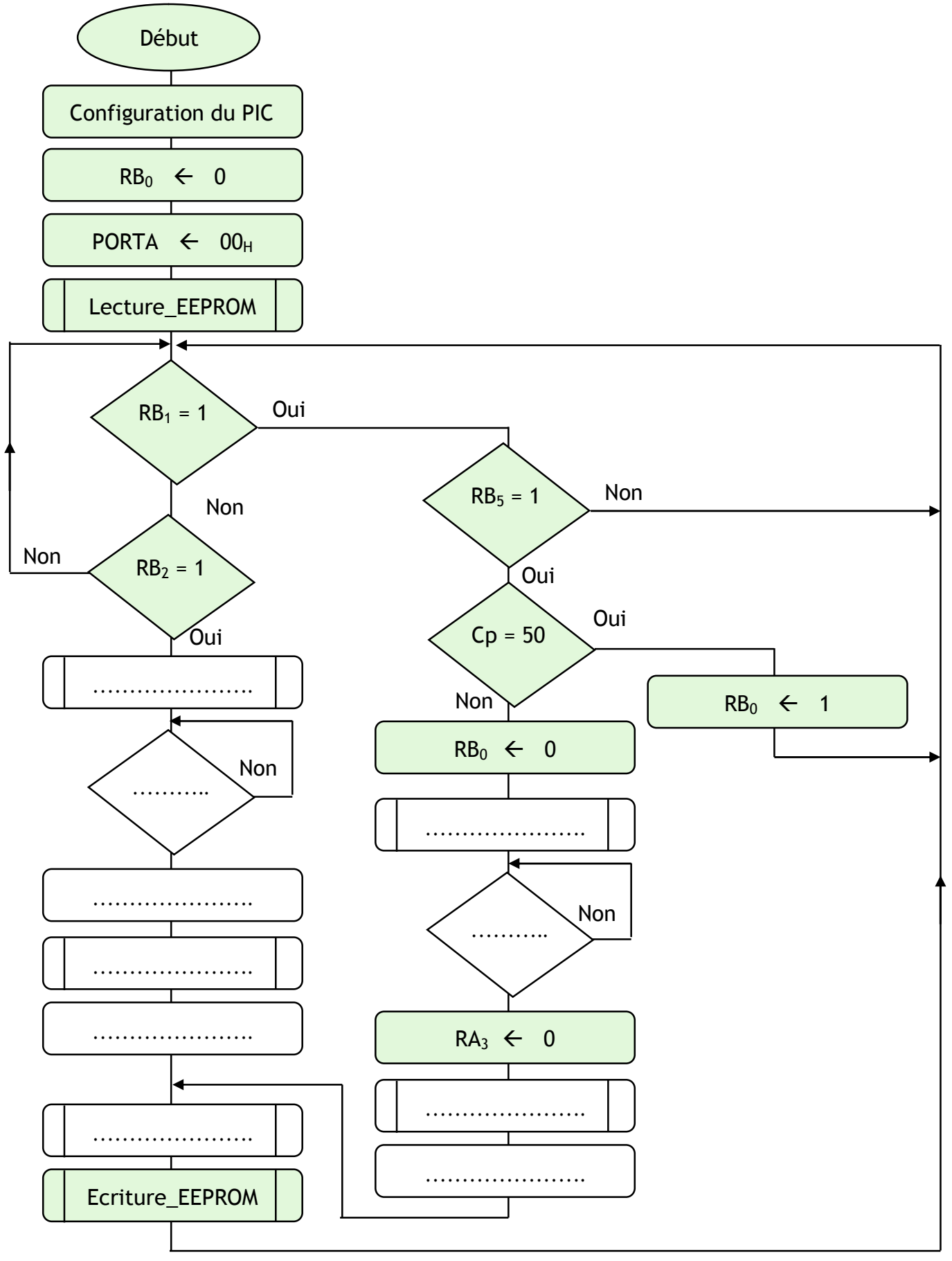
Sous programme TEMPO

```
        TEMPO   .....................   ; doit compter 156 impulsions
                .....................   ; écriture de 100
        LAB8    .....................   ; fin de comptage
                .....................   ;
                BCF     INTCON, TOIF    ; remise à zéro du témoin TOIF
                RETURN                  ;
```

DREP 06

DOCUMENT A RENDRE

Organigramme à compléter



DREP 07

DOCUMENT A RENDRE

Programme à compléter

Initialisation du PIC 16 F 84

```
                ..............................   ; accès à la BANK 1
                ..............................   ;
                ..............................   ; configuration du PORTA
                ..............................   ;
                ..............................   ; configuration du PORTB
                ..............................   ;
                ..............................   ; configuration du TIMER 0
                ..............................   ; accès à la BANK 0
```

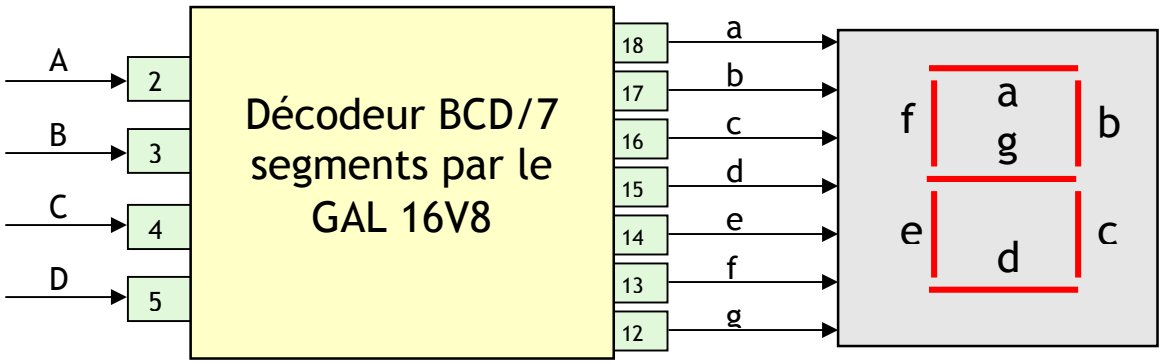
Programme principal

```
                CLRF    PORTA                    ; moteur et voyants au repos
                CALL    Lecture_EPROM            ; nombre de véhicule dans Cp
DEBUT           BTFSS   PORTB, 1                 ; test si entrée
                GOTO    SORTIE                   ;
                GOTO    ENTRÉE                   ;
SORTIE          ..............................   ; test si sortie
                ..............................   ;
                ..............................   ; ouvrir la barrière
LAB1            ..............................   ; test si véhicule est sortie
                ..............................   ;
                ..............................   ; éteindre vert
                ..............................   ; Temporisation T1
                ..............................   ; décrémenter compteur
                ..............................   ;
ENTREE          BTFSS   PORTB, 5                 ; test si autorisation
                GOTO    DEBUT                    ;
                MOWLW   D'50'                    ;
                SUBWF   Cp, W                    ; comparer Cp et 10
                BTFSS   STATUS, Z                ; saut si Cp = 10
                GOTO    LAB2                     ; parking disponible
                BSF     PORTB, 0                 ; allumer jaune
                GOTO    DEBUT                    ; parking complet
LAB2            ..............................   ; éteindre jaune
                ..............................   ; ouvrir la barrière
LAB3            ..............................   ; test si véhicule est entré
                ..............................   ;
                ..............................   ; éteindre vert
                ..............................   ; Temporisation T1
                ..............................   ; incrémenter compteur
LAB4            CALL    Fermer_barriere          ; fermer la barrière
                CALL    Ecriture_EPROM           ; enregistrer compteur
                GOTO    DEBUT                    ;
                END                              ; directive de fin de programme
```

DREP 08

DOCUMENT A RENDRE

Décodeur BCD/7 segments par le GAL 16V8

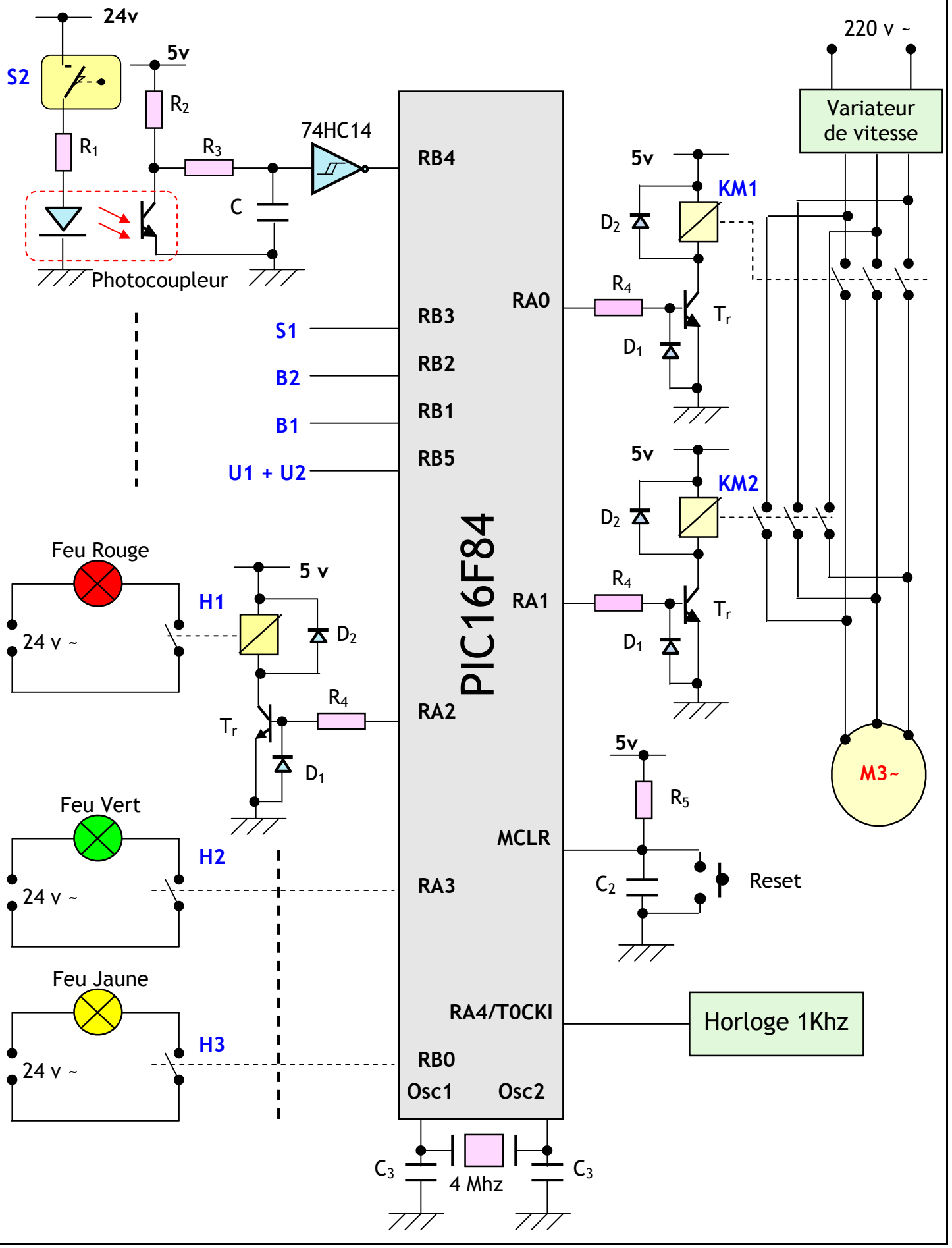


Programme 'ABEL'

MODULE BCD
TITLE 'Décodeur BCD 7 segments '
DECLARATIONS
..... // PLD à programmer
..... // Variables d'entrée
.....
..... // Variables sortie
..... // Définition du bus N
S = [a, b, c, d, e, f, g] ; // Définition du bus S
.....
When N = 0 THEN S = [1, 1, 1, 1, 1, 1, 0] ELSE
When N = 1 THEN S = [0, 1, 1, 0, 0, 0, 0] ELSE
When N = 2 THEN S = [1, 1, 0, 1, 1, 0, 1] ELSE
.....
.....
.....
.....
.....
.....
When N = 9 THEN S = [1, 1, 1, 1, 0, 1, 1];
TEST_VECTORS .....
0 -> [1, 1, 1, 1, 1, 1, 0] ;
1 -> [0, 1, 1, 0, 0, 0, 0] ;
.....
.....
.....
.....
.....
.....
.....
.....
END .....

DRES 01

Carte électronique à base du PIC16 F 84



DRES 02

Tableau d'affectation d'Entrées/Sorties

Mouvement	Actionneur	Préact.	Ordres	Sortie PIC
Ouvrir la barrière	Moteur M1	Relais KM1	KM1	RA0
Fermer la barrière	Moteur M1	Relais KM2	KM2	RA1

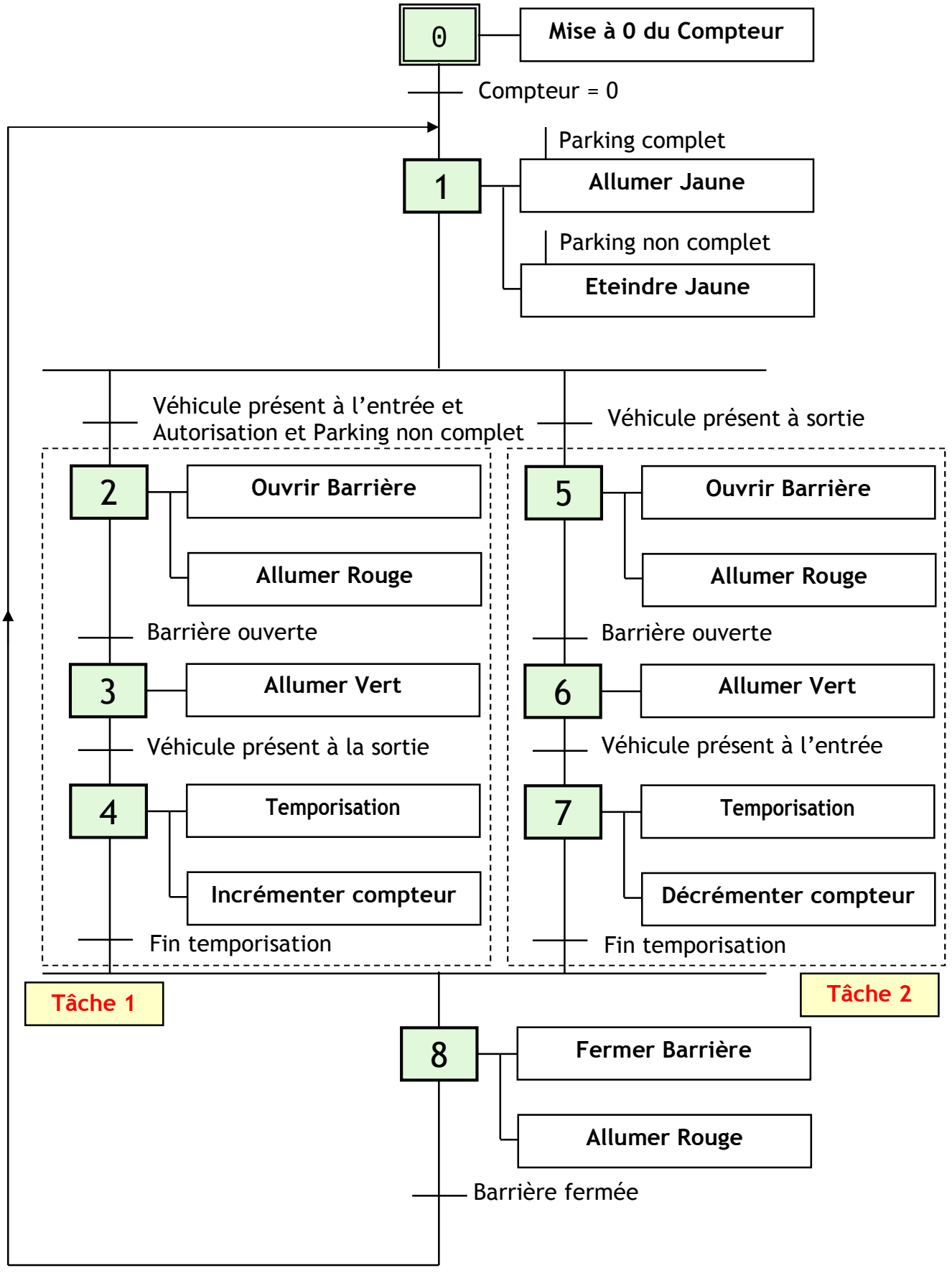
Message	Voyant	Préact.	Ordres	Sortie PIC
Accès non autorisé au parking	Feu Rouge	Relais H1	H1	RA2
Accès autorisé au parking	Feu Vert	Relais H2	H2	RA3
Parking complet	Feu Jaune	Relais H3	H3	RB0

Compte-rendu	Capteur	Mnem.	Entrée PIC
Présence véhicule à l'entrée	Détecteur photoélectrique reflex	B1	RB1
Présence véhicule à la sortie	Détecteur photoélectrique reflex	B2	RB2
Barrière ouverte	Détecteur mécanique à levier	S1	RB3
Barrière fermée	Détecteur mécanique à levier	S2	RB4

Consigne	Constituant	Mnem.	Entrée PIC
Code d'entée	Clavier	U1	RB5
Impulsion télécommande	Télécommande	U2	
Compteur	Case mémoire d'adresse $0C_H$	Cp	-
Temporisation 10 s	Sous programme Tempo	T1	-

DRES 03

GRAFCET point de vue partie opérative



DRES 04

EEPROM du 16 F 84

Structure du registre EECON1

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
x	x	x	EEIF	WRERR	WREN	WR	RD

Bit 0: **RD - ReaD EEPROM-**

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une lecture de l'EEPROM. Après le cycle de lecture, il est mis automatiquement à 0.

Bit 1: **WR -WRite EEPROM-**

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une écriture de l'EEPROM. Après le cycle d'écriture, il est mis automatiquement à 0.

Bit 2: **WREN - WRite ENABLE EEPROM -**

C'est un bit de confirmation d'écriture dans l'EEPROM. En effet, il ne suffit pas de définir un cycle d'écriture uniquement avec le bit WR. Il faut impérativement valider le bit WREN (WREN=1) pour autoriser une écriture.

Bit 3: **WRERR -EEPROM WRite ERROR flag-**

Ce drapeau indique qu'une erreur s'est produite lors d'un cycle d'écriture dans l'EEPROM.

WRERR=1 une opération d'écriture a échoué.

WRERR=0 le cycle d'écriture s'est déroulé normalement.

Bit 4: **EEIF -EEPROM Interrupt Flag-**

EEIF est un drapeau qui génère une interruption lorsqu'un cycle d'écriture s'est déroulé normalement. Il doit être mis à 0 lors de la routine d'interruption.

EEIF=1 l'opération s'est déroulé correctement.

EEIF=0 soit l'opération n'a pas commencé, soit n'est pas terminée.

Lecture d'une donnée

- Placer l'**adresse** de la donnée à lire dans **EEADR**.
- Mettre le bit **RD** de EECONN1 à **1**.
- Lire le contenu du registre **EEDATA.**

Ecriture d'une donnée

- Placer l'**adresse** de la donnée à écrire dans **EEADR**.
- Placer la **donnée** à écrire dans **EEDATA**.
- Mettre le bit **WREN** de EECONN1 à **1** pour autoriser l'écriture.
- Placer **0x55** dans **EECON2**.
- Placer **0xAA** dans **EECON2**.
- Mettre le bit **WR** de EECONN1 à **1**.
- Attendre que le bit **EEIF** soit à **1**.
- On peut utiliser l'interruption produite par **EEIF** en la validant par le bit **EEIE** de **INTCON**.
- N'oublier pas de remettre **EEIF** à **0**.

DRES 05

TIMER0 du 16 F 84

Structure du registre OPTION

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

BIT 6: **INTEDG -INTerrupt EDGe-**

INTEDG=1 alors la broche RBO/INT génère une interruption sur un front montant.

INTEDG=0 alors la broche RBO/INT génère une interruption sur un front descendant.

BIT 5: **TOCS -TMRO Clock Source-**

Il permet de sélectionner le mode de fonctionnement du Timer/Compteur.

TOCS=1 sélection de l'horloge externe (broche RA4) qui correspond au COMPTEUR.

TOCS=0 sélection de l'horloge interne et permet au module de travailler en mode TIMER.

BIT 4: **TOSE -TMRO Source Edge-**

Ce bit détermine sur quel front -montant ou descendant- l'entrée RA4 incrémentera le registre TMRO.

TOSE=1 Front descendant.

TOSE=0 Front montant.

Bit 3: **PSA -PreScaler Assignment-**

PSA=1 alors le Prescaler est associé avec le WDT.

PSA=0 alors le Prescaler est associé avec le TIMER.

Bit 0, 1, 2: **PS0, PS1, PS2 - Prescaler Select -**

Ces trois bits effectuent une division de la fréquence d'horloge du Prescaler.

PS2	PS1	PS0	RATIO TMRO	RATIO WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Structure du registre INTCON

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

BIT 2: **TOIF -TMRO Overflow Interrupt Flag-**.

Ce drapeau indique un dépassement du registre TMRO (passage de FF à 00).

TOIF=1 dépassement de TMRO.

TOIF=0 pas de dépassement.

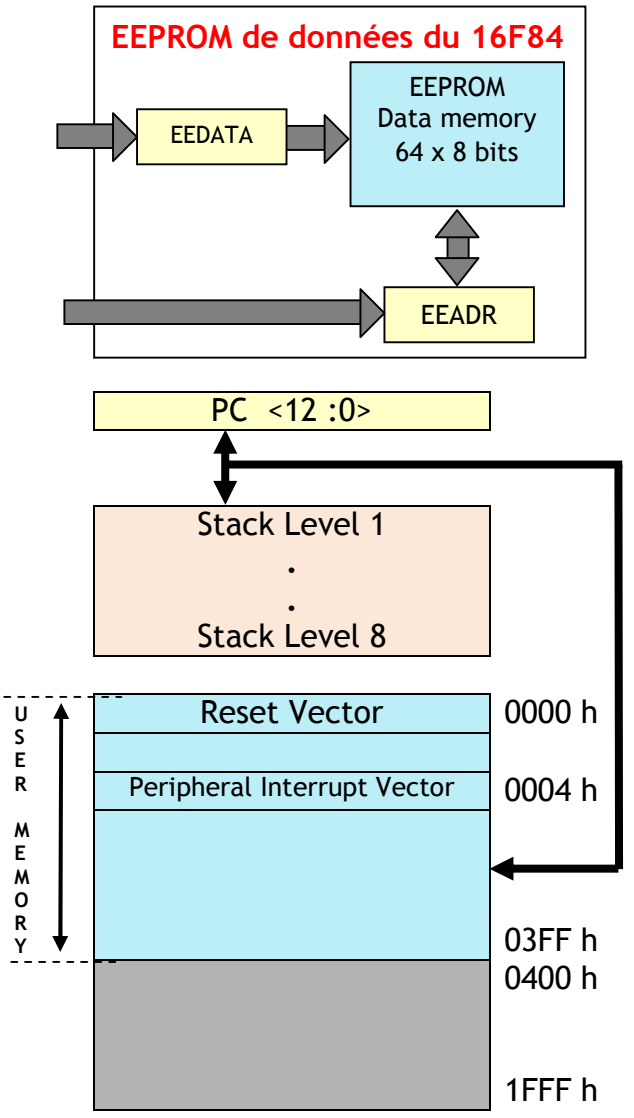
DRES 06

Plan mémoire du 16 F 84

Les mémoires du 16 F 84

	BANK 0	BANK 1	
00 h	INDF	INDF	80 h
01 h	TMR0	OPTION	81 h
02 h	PCL	PCL	82 h
03 h	STATUS	STATUS	83 h
04 h	FSR	FSR	84 h
05 h	PORTA	TRISA	85 h
06 h	PORTB	TRISB	86 h
07 h			87 h
08 h	EEDATA	EECON1	88 h
09 h	EEADR	EECON2	89 h
0A h	PCLATH	PCLATH	8A h
0B h	INTCON	INTCON	8B h
0C h	36 General Pourpose Regisers (SRAM)	Mapped (accesses) In Bank 0	8C h
2F h			AF h
30 h			B0 h
7F h			FF h
ADDR FILE			ADDR FILE

RAM du 16F84



EEPROM programme du 16F84

Configuration des PORTx, les registres PORTx et TRISx

Tous les ports sont pilotés par deux registres :

- ✓ Le registre de **PORTx**, si le **PORTx** ou certaines lignes de **PORTx** sont configurées en sortie, ce registre détermine l'état logique des sorties.
- ✓ Le registre **TRISx**, c'est le registre de direction. Il détermine si le **PORTx** ou certaines lignes de Port sont en entrée ou en sortie. L'écriture d'un **1** logique correspond à une **entrée** (**1** comme Input) et l'écriture d'un **0** logique correspond à une sortie (**0** comme Output).

Remarque :

Les registres **TRISx** appartiennent à la **BANQUE 1** des **SFR**. Lors de l'initialisation du **µC** il ne faut pas oublier de changer de bank mémoire pour les configurer.

Pour accéder aux banques mémoire, on utilise le bit RP0 (5ième bit du registre STATUS).

RP0 = 0 : Accès à la BANK 0.

RP0 = 1 : Accès à la BANK 1.

DRES 07

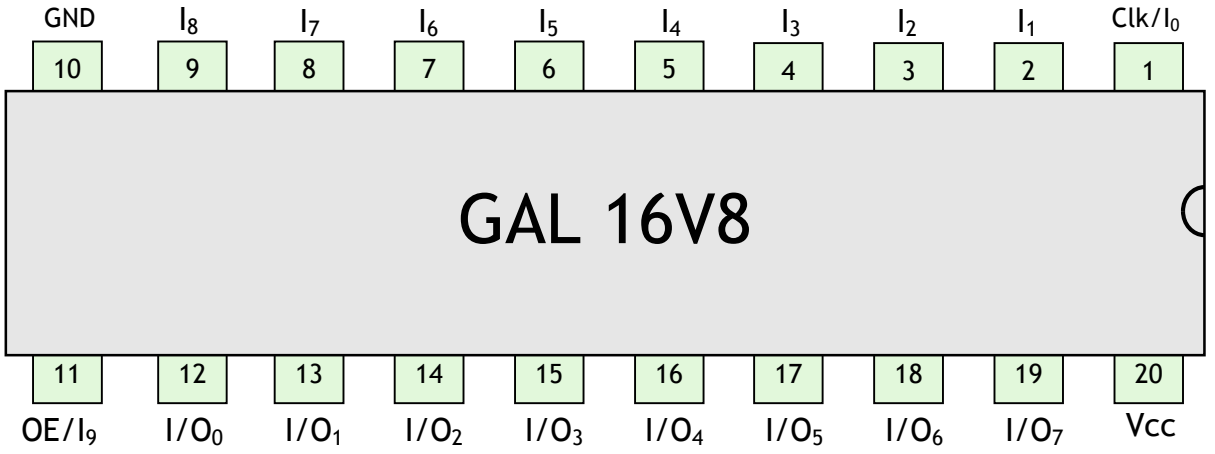
Jeux d'instructions du micro contrôleur16F84

INSTRUCTIONS OPERANT SUR REGISTRE		indicateurs	Cycles
ADDWF F,d	W+F →{W,F ? d}	C, DC, Z	1
ANDWF F,d	W and F → {W,F ? d}	Z	1
CLRF F	Clear F	Z	1
COMF F,d	Complémente F → {W,F ? d}	Z	1
DECF F,d	Décrémente F → {W,F ? d}	Z	1
DECFSZ F,d	Décrémente F → {W,F ? d} skip if 0		1(2)
INCF F,d	Incrémente F → {W,F ? d}	Z	1
INCFSZ F,d	Iincrémente F → {W,F ? d} skip if 0		1(2)
IORWF F,d	W or F → {W,F ? d}	Z	1
MOVF F,d	F → {W,F ? d}	Z	1
MOVWF F	W → F		1
RLF F,d	Rotation à gauche de F à travers C → {W,F? d}	C	1
RRF F,d	Rotation à droite de F à travers C → {W,F? d}	C	1
SUBWF F,d	F - W → {W,F ? d}	C, DC, Z	1
SWAPF F,d	Permute les 2 quartets de F → {W,F ? d}		1
XORWF F,d	W xor F → {W,F ? d}	Z	1
INSTRUCTIONS OPERANT SUR LES BITS		**indicateurs**	**Cycles**
BCF F,b	Mise à 0 du bit b de F	C, DC, Z	1
BSF F,b	Mise à 1 du bit b de F	Z	1
BTFSC F,b	Teste le bit b de F, sauter si 0	Z	1(2)
BTFSS F,b	Teste le bit b de F, sauter si 1	Z	1(2)
INSTRUCTIONS OPERANT SUR CONSTANTE		**indicateurs**	**Cycles**
ADDLW K	W + K → W	C, DC, Z	1
ANDLW K	W and K → W	Z	1
IORLW K	W or K → W	Z	1
MOVLW K	L → W	Z	1
SUBLW K	K - W → W	C, DC, Z	1
XORLW K	W xor K → W	Z	1
AUTRES INSTRUCTIONS		**indicateurs**	**Cycles**
CLRW	Clear W	Z	1
CLRWDT	Clear Watchdog timer	TO', PD'	1
CALL L	Branchement à un sous programme de label L		2
GOTO L	Branchement à la ligne de label L		2
NOP	Pas d'opération		1
RETURN	Retour d'un sous programme		2
RETFIE	Retour d'interruption		2
RETLW K	Retour d'un sous programme avec K dans W		2
SLEEP	Se met en mode standby	TO', PD'	1

DRES 08

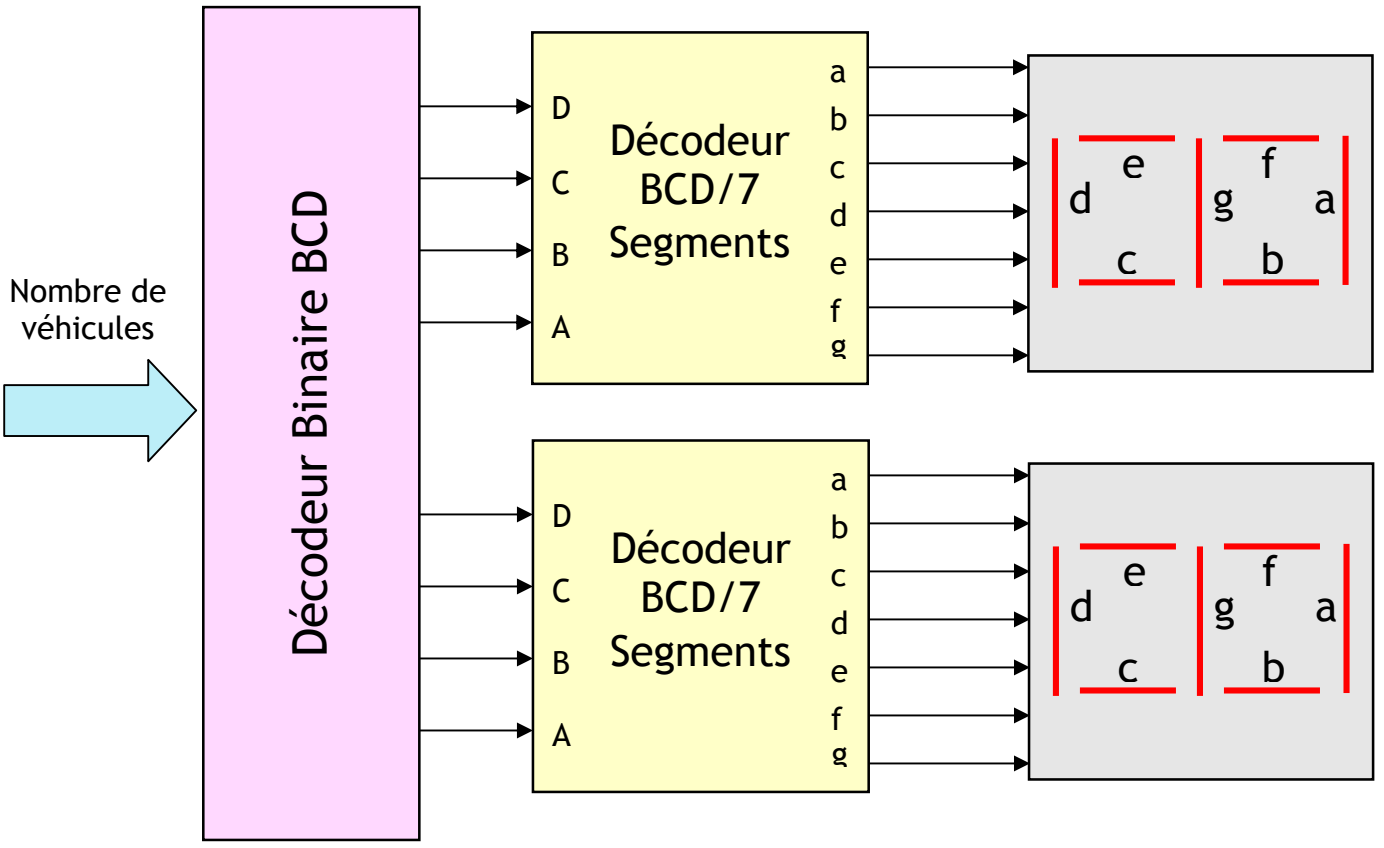
AFFICHAGE DU NOMBRE DE VÉHICULES

GAL 16 V 8



C'est un circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.
La lettre V veut dire que ce circuit est versatile, ce qui veut dire qu'il est possible par programmation de choisir entre une configuration de sortie combinatoire ou séquentielle.

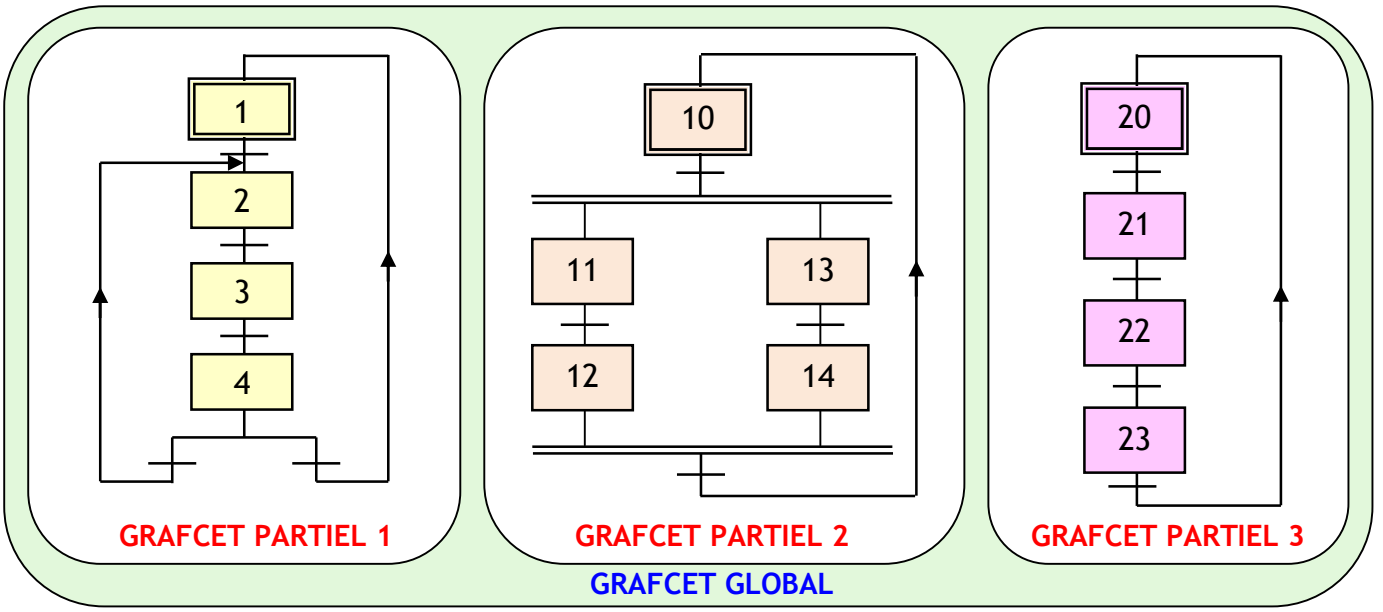
AFFICHAGE DU NOMBRE DE VÉHICULES



GRAFCET 'NOTION DE TÂCHE'

1- Notion de GRAFCET partiel :

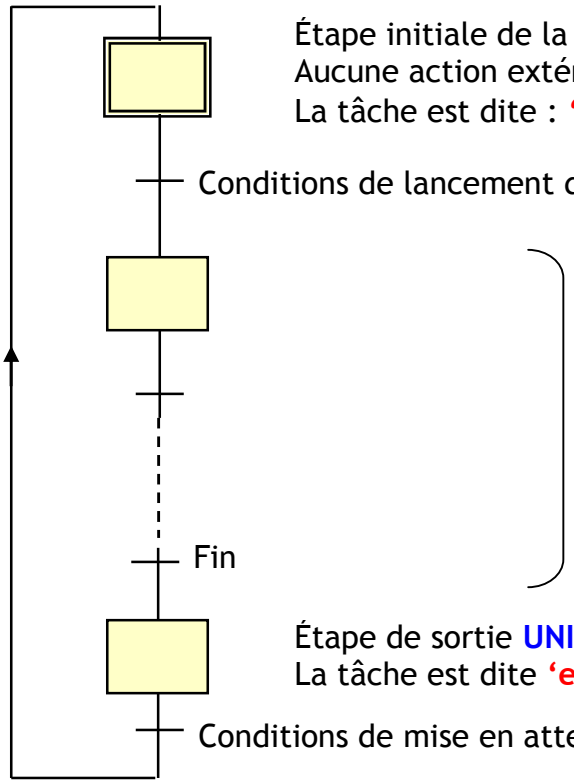
Constitué d'un ou plusieurs GRAFCETS connexes. Un GRAFCET partiel résulte d'une partition, selon des critères méthodologiques, du GRAFCET global décrivant le comportement de la partie séquentielle d'un système.



Le GRAFCET **global** est constitué des GRAFCETS partiels **G1**, **G2** et **G3**.

2- Synchronisation de GRAFCET :

2.1- Notion de GRAFCET de Tâche :



Étape initiale de la tâche.
Aucune action extérieure n'y est associée.
La tâche est dite : **'en attente d'exécution'**

Conditions de lancement de la tâche.

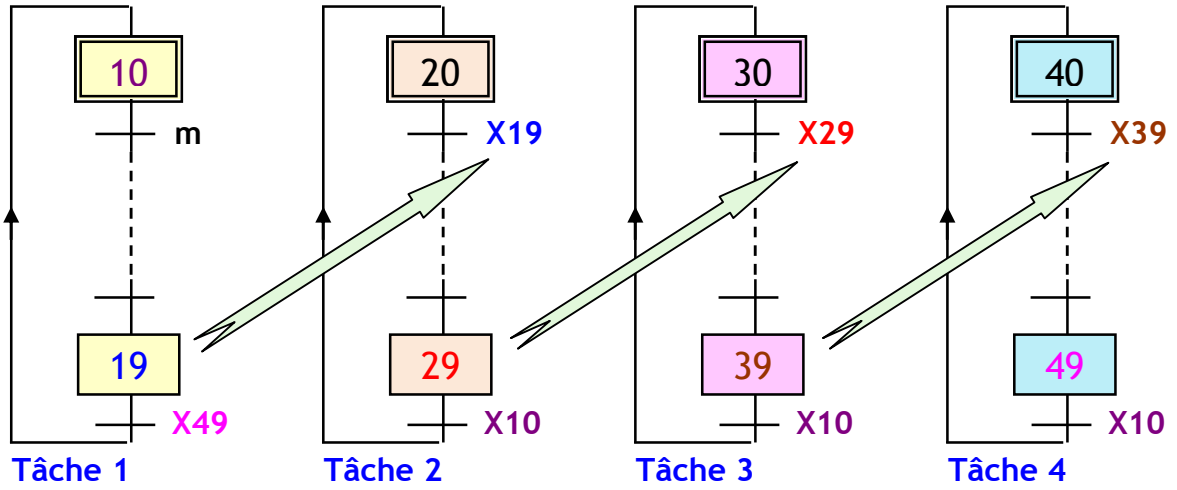
GRAFCET traduisant les cycle commandés par tâche.
La tâche est dite :
'en cours d'exécution'

Fin

Étape de sortie **UNIQUE**.
La tâche est dite **'exécutée'**

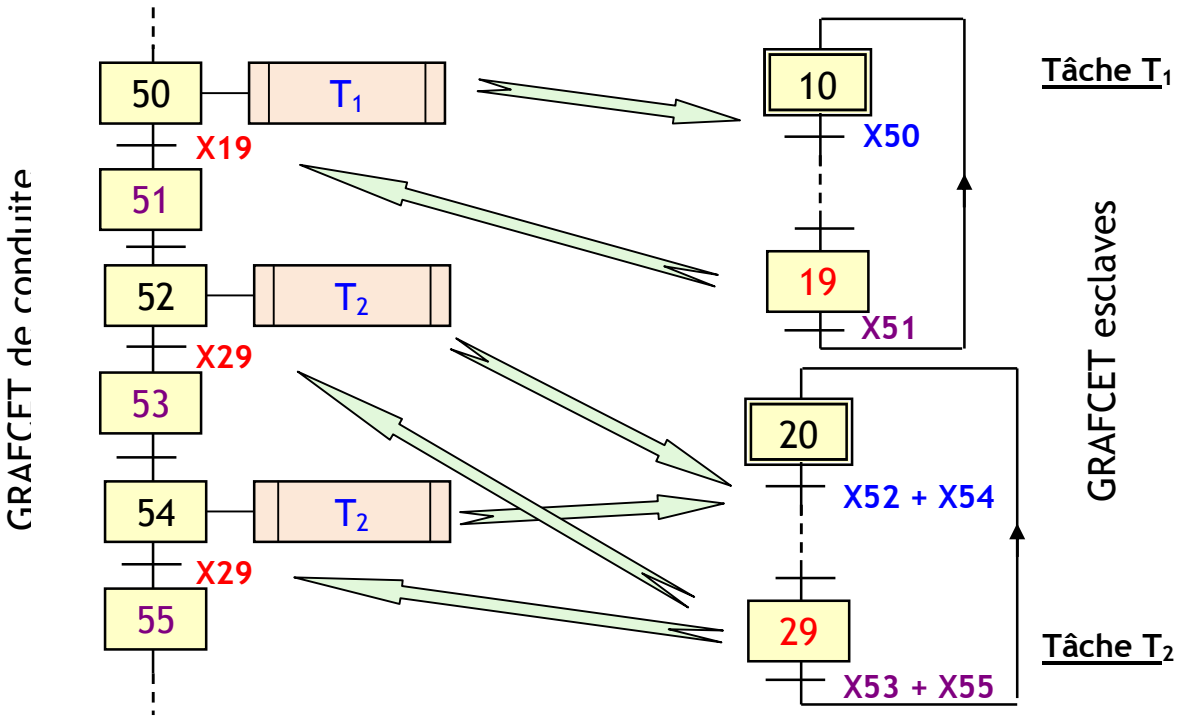
Conditions de mise en attente.

2.2- Coordination horizontale :



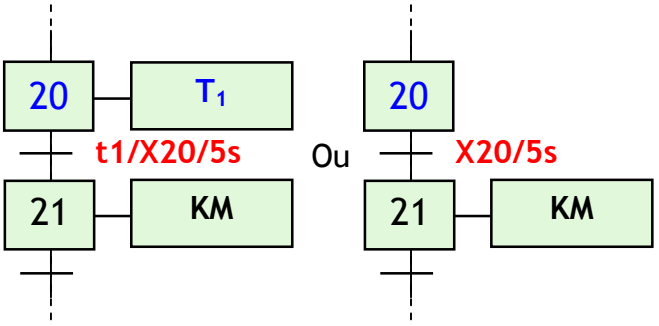
1 seule tâche à la fois

2.3- Coordination Verticale asynchrone :



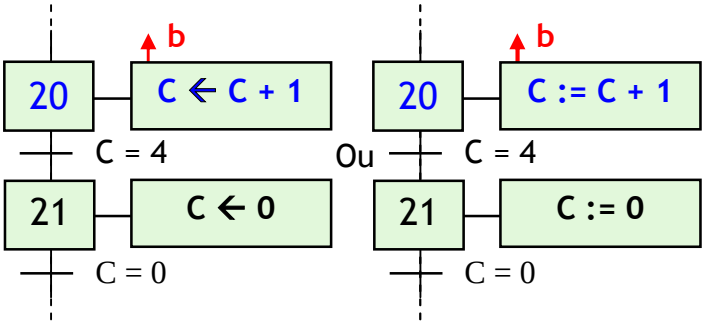
3- Différents types d'action :

3.1- Temporisations :



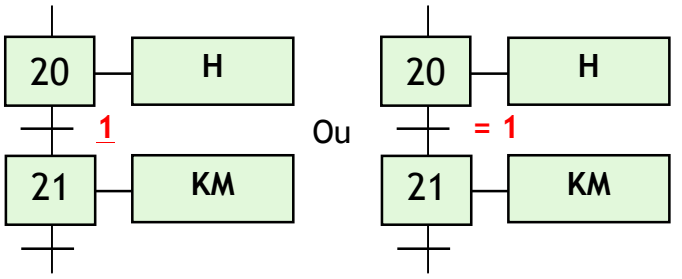
La transition 20 - 21 est franchie lorsque la temporisation, démarrée à l'étape 20 est écoulée, soit au bout de 5s

3.2- Comptage :



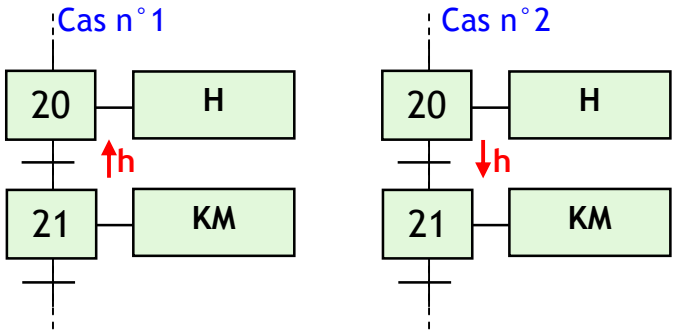
La transition 20 - 21 est franchie lorsque le contenu du compteur C est égal à 4.Le compteur est incrémenté sur front montant du signal b. Il est mis à zéro à l'étape 21.

3.3- Réceptivité toujours vraie :



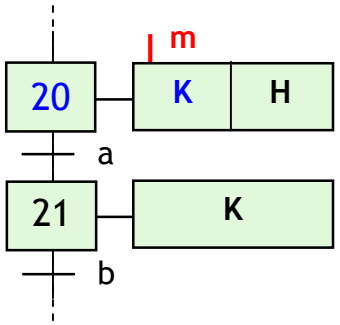
L'étape 21 est active, à l'activation de l'étape 20

3.4- Évènements (fronts) :

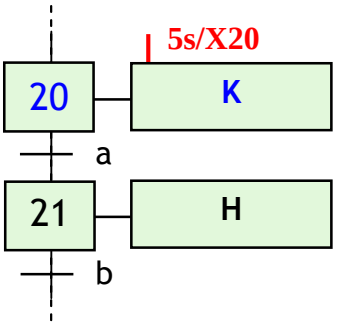


La transition 20 - 21 est franchie lors d'un front montant sur h (cas n°1), ou lors d'un front descendant sur h (cas n°2).

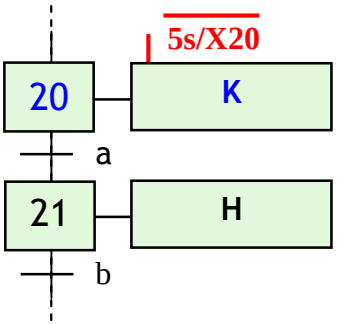
3.5- Action conditionnelle :



L'action K devient effective à l'étape 20, lorsque la condition m est vraie.
K = X20. m + X21

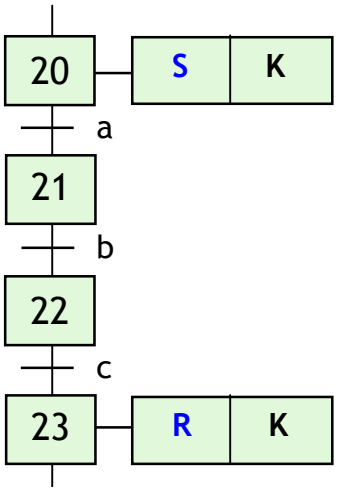


L'action K devient effective à l'étape 20, après 5 secondes

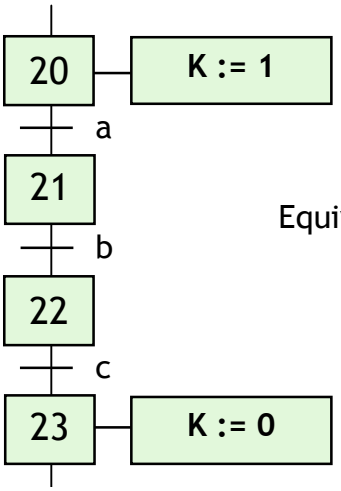


L'action K devient effective à l'étape 20, et dure 5 secondes

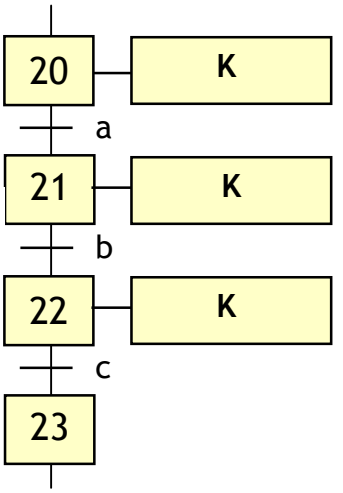
3.6- Action mémorisée :



OU



Equivalent à



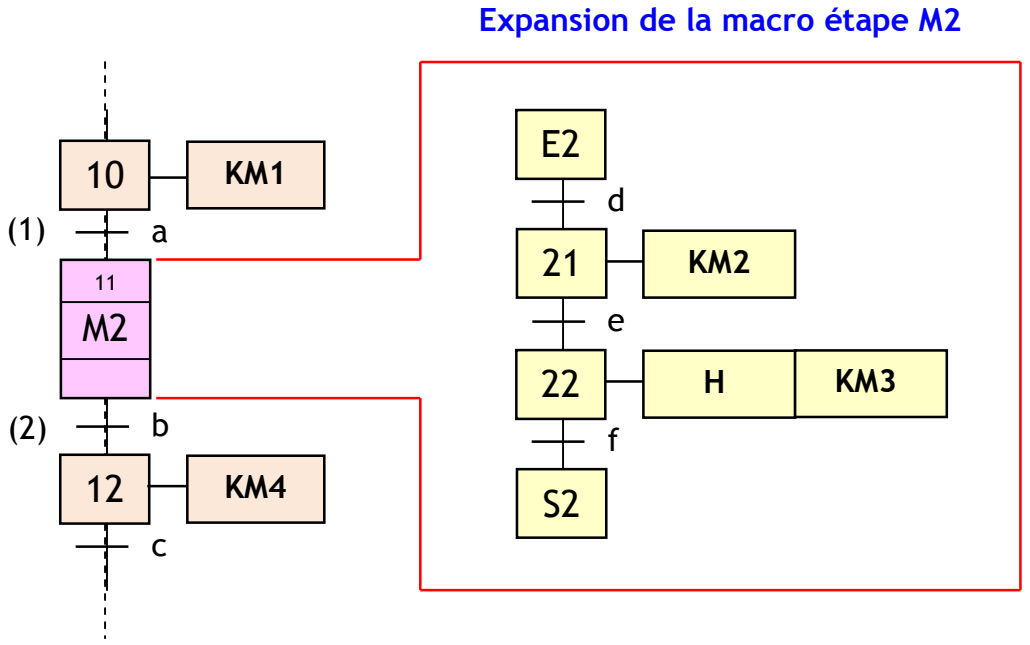
Mise à 1 de l'action par la lettre S (set)
Mise à 0 de l'action par la lettre R (reset)

L'action K est active aux étapes 20, 21 et 22.

4- Notion de Macro- étape :

Avec la notion de macro représentation, on se donne le moyen de reporter à plus tard ou sur une autre page la description détaillée de certaines séquences.

La macro-étape est la représentation unique d'un ensemble d'étapes et de transitions nommé expansion de macro-étape.



L'expansion de la macro-étape commence par une seule étape d'entrée et se termine par une seule étape de sortie, étapes qui représentent les seuls liens possibles avec le GRAFCET auquel elle appartient.

Le franchissement de la transition (1) active l'étape E2.

La transition (2) ne sera validée que lorsque l'étape S2 sera active.

Le franchissement de la transition (2) désactive l'étape S2.

EEPROM DU 16 F 84

1- Présentation :

Le PIC 16F84 possède une EEPROM de **64** octets pour y stocker les données. Contrairement à l'EEPROM de programme, elle n'est pas adressée par le PC, mais par un registre séparé appelé **EEADR** se trouvant à l'adresse **09**H dans le fichier des registres. Les données sont accessibles par le registre **EEDATA** d'adresse **08**H. 2 registres de contrôle sont associés à cette mémoire **EECON1** et **EECON2** d'adresse 88H et 89H. La durée d'écriture d'un octet est de l'ordre de 10ms.

2- Les registres :

- **Le registre EEDATA :**
  Ce registre permet de lire ou d'écrire une donnée dans la mémoire non volatile (EEPROM).
- **Le registre EEADR :**
  Ce registre contient l'adresse de la donnée se trouvant dans l'EEPROM.
- **Le registre EECON1 :**
  C'est un registre de contrôle qui permet d'exécuter une lecture ou une écriture dans l'EEPROM. Seuls les 5 bits de poids faible sont utilisés.
- **Le registre EECON2 :**
  Ce registre est exclusivement utilisé pour les séquences d'écritures dans l'EEPROM. Il n'a pas d'adresse physique et la lecture de ce registre retourne une valeur nulle.
  Une donnés ne peut être écrite qu'après avoir écrit successivement 0x55 et 0xAA dans EECON2.

3- Structure du registre EECON1 :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
x	x	x	EEIF	WRERR	WREN	WR	RD

Bit 0: **RD - ReaD EEPROM-**
Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une lecture de l'EEPROM. Après le cycle de lecture, il est mis automatiquement à 0.

Bit 1: **WR -WRite EEPROM-**
Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une écriture de l'EEPROM. Après le cycle d'écriture, il est mis automatiquement à 0.

Bit 2: **WREN - WRite ENABLE EEPROM -**
C'est un bit de confirmation d'écriture dans l'EEPROM. En effet, il ne suffit pas de définir un cycle d'écriture uniquement avec le bit WR. Il faut impérativement valider le bit WREN (WREN=1) pour autoriser une écriture.

Bit 3: **WRERR -EEPROM WRite ERROR flag-**
Ce drapeau indique qu'une erreur s'est produite lors d'un cycle d'écriture dans l'EEPROM.
WRERR=1 une opération d'écriture a échoué.
WRERR=0 le cycle d'écriture s'est déroulé normalement.

Bit 4: **EEIF -EEPROM Interrupt Flag-**
EEIF est un drapeau qui génère une interruption lorsqu'un cycle d'écriture s'est déroulé normalement. Il doit être mis à 0 lors de la routine d'interruption.
EEIF=1 l'opération s'est déroulé correctement.
EEIF=0 soit l'opération n'a pas commencé, soit n'est pas terminée.

4- Ecriture et lecture de l'EEPROM :

4.1- Lecture d'une donnée :

- Placer l'**adresse** de la donnée à lire dans **EEADR**.
- Mettre le bit **RD** de EECONN1 à **1**.
- Lire le contenu du registre **EEDATA.**

```
.....        ; Bank 0
.....        ;
.....        ; l'adresse à lire
.....        ; Bank 1
.....        ; lecture EPROM
.....        ; Bank 0
.....        ; W ← EEDATA
```

4.2- Ecriture d'une donnée :

- Placer l'**adresse** de la donnée à écrire dans **EEADR**.
- Placer la **donnée** à écrire dans **EEDATA.**
- Mettre le bit **WREN** de EECONN1 à **1** pour autoriser l'écriture.
- Placer **0x55** dans **EECON2**.
- Placer **0xAA** dans **EECON2**.
- Mettre le bit **WR** de EECONN1 à **1**.
- Attendre que le bit **EEIF** soit à **1**.
- On peut utiliser l'interruption produite par **EEIF** en la validant par le bit **EEIE** de **INTCON**.
- N'oublier pas de remettre **EEIF** à **0**.

```
        .....        ; Bank 0
        .....        ;
        .....        ; définition de l'adresse
        .....              ;
        .....        ; définition de la donnée
        .....        ; Bank 1
        .....        ; autorisation de l'écriture
        .....        ;
        .....        ; écriture de 0x55
        .....        ;
        .....        ; écriture de 0xAA
        .....        ; écriture dans EEPROM
Lab     .....        ;
        .....        ; écriture terminée
        .....        ; Bank 0
```

TIMER DU 16 F 84

1- Présentation :

Le PIC 16F84 dispose d'un TIMER, c'est un module programmable dont les fonctions principales sont :

- La génération de signaux périodiques (astable),
- La génération d'impulsions (monostable),
- Le comptage d'évènements (compteur),
- La génération de signaux PWM (modulation de largeur d'impulsions pour les MCC).

2- Les registres du TIMER0 :

❖ Le registre TMRO :

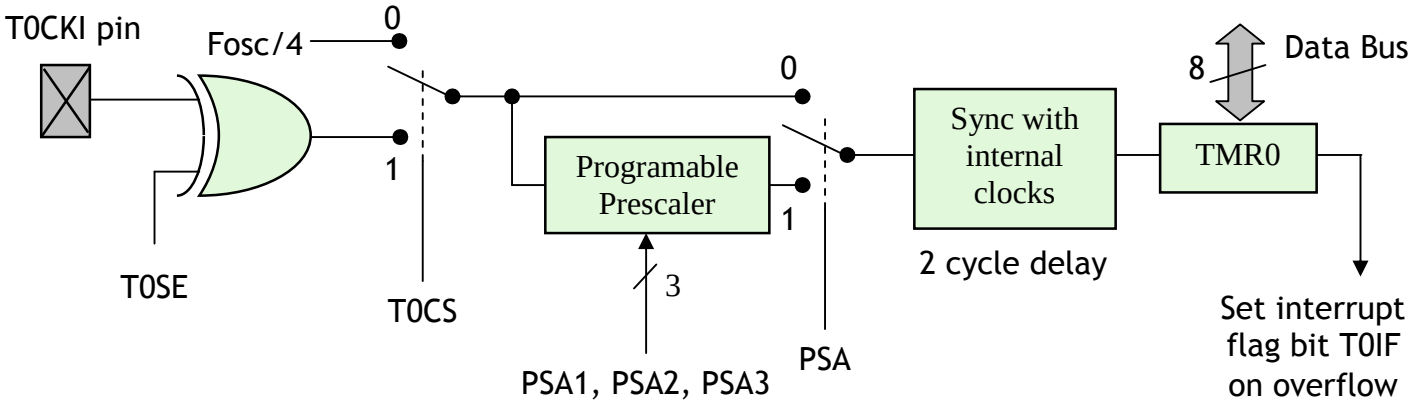
Ce registre de 8 bits s'incrémente de "**1**" a chaque impulsion de l'horloge **interne** (**Fosc/4**) ou par une horloge **externe** appliquée sur la broche **TOCKI/ RA4**. Il est associé au module Timer/Compteur. Ce registre se trouve à l'adresse 01H.

❖ Le registre OPTION :

Ce registre contiens les bits de contrôles du **PRESCALER**, de l'interruption externe **INT**, de la sélection **Timer/Compteur** et du "**tirage au plus**" du PORTB. Ce registre se trouve à l'adresse 81H.Ce registre se trouve à l'adresse 81H.

3- Fonctionnement :

3.1- Schéma simplifié :



3.2- Structure du registre OPTION :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

BIT 7: **RBPU -PORTB Pull-Up-.**

RBPU=1 Le "tirage au plus" interne du PORT B est désactivé.
RBPU=0 Le "tirage au plus" interne du PORT B est activé.

BIT 6: **INTEDG -INTerrupt EDGe-**

INTEDG=1 alors la broche RBO/INT génère une interruption sur un front montant.
INTEDG=0 alors la broche RBO/INT génère une interruption sur un front descendant.

BIT 5: **TOCS -TMRO Clock Source-**

Il permet de sélectionner le mode de fonctionnement du Timer/Compteur.

TOCS=1 sélection de l'horloge externe (broche RA4) qui correspond au COMPTEUR.

TOCS=0 sélection de l'horloge interne et permet au module de travailler en mode TIMER.

BIT 4: **TOSE -TMRO Source Edge-**

Ce bit détermine sur quel front -montant ou descendant- l'entrée RA4 incrémentera le registre TMRO.

TOSE=1 Front descendant.

TOSE=0 Front montant.

Bit 3: **PSA -PreScaler Assignment-**

PSA=1 alors le Prescaler est associé avec le WDT.

PSA=0 alors le Prescaler est associé avec le TIMER.

Bit 0, 1, 2: **PS0, PS1, PS2 - Prescaler Select -**

Ces trois bits effectuent une division de la fréquence d'horloge du Prescaler.

PS2	PS1	PS0	RATIO TMRO	RATIO WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Remarque :

Lorsque le TIMER0 déborde, le BIT TOIF du registre INTCON passe à 1; après utilisation, n'oublier pas de remettre TOIF à zéro.

4- Le WATCHDOG :

C'est un compteur incrémenté en permanence – Free running – par l'horloge interne. Ce compteur 8 bits, quand il arrive à **FF** – WDT Time Out – est capable de réinitialiser le microcontrôleur.

La durée d'un cycle de comptage est réglable grâce au Prescaler partagé avec le Timer 0.

- **Caractéristiques:**
  - ✓ Désactivation au démarrage ;
  - ✓ Remis à 0 sur appel de CLRWTD ;
  - ✓ Redémarre le PIC placé en mode SLEEP: Signalé par bit TO de statut
- **Utilisation**
  - ✓ Détection boucle dans programme (possible sur faute électrique ou du programmeur)
  - ✓ Réveil après SLEEP.

Interruptions du 16 F 84

1- Interruptions du PIC 16 F 84 :

Le PIC 16 F 84 possède 4 sources d'interruption :

- Changement d'état du PORTB (RB4 à RB7)
- Front montant ou descendant sur la broche RB0/INT
- Dépassement du registre TMRO (passage de FF à 00)
- Déroulement normal d'un cycle d'écriture dans L'EEPROM.

Ces interruptions sont validées :

- Globalement par le bit GIE du registre INTCON.
- Localement par les bits EEIE, RTIE, INTE et RBIE du registre INTCON.

4 drapeaux pouvant provoquer l'interruption correspondante :

- EEIF du registre EECON1.
- T0IF, INTF, RBIF du registre INTCON.

2- Service d'une interruption :

Le service d'une interruption comporte les actions suivantes :

- Le drapeau correspondant passe a 1.
- PC est empilé puis affecté de l'adresse 004.
- Le masque GIE est automatiquement mis a 0 pour interdire d'autres interruptions.
- Fin (instruction RETIE)
- PC est dépilé
- GIE est remis a 1

Le programme doit:

- Identifier la source de l'interruption en consultant les drapeaux.
- Remettre à 0 le drapeau qui a provoqué l'interruption.
- Sauvegarder éventuellement certains registres.

3- Le registre INTCON:

Le registre INTCON contient tous les bits de validation de chaque source d'interruption ainsi que leur drapeau (Flag). Les drapeaux doivent être mise à 0 après l'interruption.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Bit 0: **RBIF -RB port change Interrupt Flag-**

C'est un drapeau d'interruption qui indique un changement d'état du PORT B (RB4 à RB7).

RBIF=1 une broche (RB4 à RB7) à changé d'état.

RBIF=0 pas de changement d'état.

Bit 1: **INTF -INT Interrupt Flag-**

Drapeau d'interruption de l'entrée RB0.

RBIF=1 une interruption est apparue.

RBIF=0 pas d'interruption.

Bit 2: **TOIF -TMRO Overflow Interrupt Flag-**

Ce drapeau indique un dépassement du registre TMRO (passage de FF à 00).

TOIF=1 dépassement de TMRO.

TOIF=0 pas de dépassement.

Bit 3: **RBIE -RB Interrupt Enable-**

Valide ou non de l'interruption généré par le changement d'état du PORT B (RB4 à RB7).

RBIE=1 Autorise l'interruption.

RBIE=0 Les changement d'état du PORTB (RB4 à RB7) ne généront pas d'interruption.

Bit 4: **INTE -INT Interrupt Enable-**

Valide ou non de l'interruption généré par la broche RB0/INT

INTE=1 Valide l'interruption INT.

INTE=0 Pas d'interruption provenant de INT.

BIT 5: **TOIE -TMRO Overflow Interrupt Enable-**

Autorise ou non l'interruption provoqué par le dépassement du registre TMRO (passage de FF à 00)

TOIE=1 Valide l'interruption.

TOIE=0 Pas d'interruption provenant du registre TMRO.

BIT 6: **EEIE -EE write Interrupt Enable-**

Autorise ou non une interruption lorsqu'un cycle d'écriture dans L'EEPROM s'est déroulé normalement.

EEIE=1 Génère une interruption lorsqu'une cycle d'écriture c'est déroulé normalement.

EEIE=0 Pas d'interruption.

BIT 7: **GIE -Global Interrupt Enable-**

Autorise ou non toutes les interruptions.

GIE=1 Toutes les interruptions sont prises en compte par le microcontrôleur.

GIE=0 Aucune interruption ne sera validée.

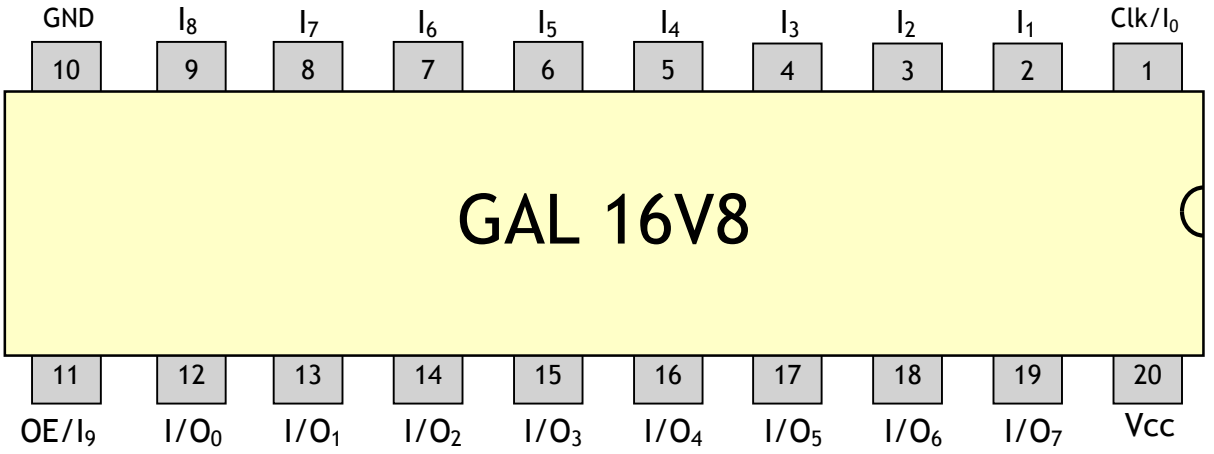
Programmation des PLD

1- Introduction :

La programmation d'une fonction logique dans un PLD doit passer par les étapes suivantes :

- Définition de la fonction par des équations logiques et ou des logigrammes ;
- Choix du PLD (PAL, GAL, MACH…) doit être choisit selon les critères suivants :
  - ✓ Coût du système
  - ✓ Matériel de programmation
  - ✓ Outil de développement (programme)
  - ✓ Nombre d'entrées et de sorties nécessaire pour réaliser la fonction
  - ✓ Type de la fonction et degrés de complexité de sa structure interne…
- L'écriture du programme de description en utilisant différentes solutions (équations, tables de vérité, diagrammes d'état)
- Test du programme par une éventuelle simulation.
- Transfert du programme au composant PLD.

2- Exemple de PLD :



C'est circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.

3- Initiation au langage ABEL :

3.1- Structure :

Section d'en-tête

Section des déclarations

Section de description

Section de test

Déclaration de fin de fichier

```
MODULE logique1
TITLE 'alarme de phares'
DECLARATIONS
        C, L, P pin 18, 19, 20; // Variables d'entrée
        S1, S2 pin 16, 17 istype 'com'; // Variables de sortie
EQUATIONS
        S1 = !C & L & P;
        S2 = P;
TEST_VECTORS
        ([C, L, P ] -> [S1, S2]);
        [ 0, 0, 0 ] -> [.X.,.X.];
        [ 0, 1, 1 ] -> [.X.,.X.];
END logique1
```

Un programme en langage ABEL se compose de 5 sections principales : l'entête, les déclarations, la description, le test et la fin ;

3.2- Entête et fin de fichier :

L'entête est reconnue par le mot clé **MODULE** suivi du nom du fichier ; on peut aussi ajouter au programme un titre précédé du mot **TITLE** ; la section du fin est identifiée par le mot clé **END** suivi du nom de fichier qui a été attribué au MODULE .
Dans tout le programme on peut ajouter des commentaires précédés du symbole **//**.

3.3- Partie Déclarations :

Elle commence par le mot clé **DECLARATIONS**. Dans cette section, on indiquera :

- La spécification du PLD à programmer.
- L'identification des broches utilisées -noms et numéros-
- La spécification du type des sorties.

La directive **device** permet de spécifier le type du PLD à programmer.

Exemple : Decodeur device 'P22V10; //il s'agit du GAL 22V10

La directive **pin** permet d'affecter un numéro de broche à une variable -entrée ou sortie-.

Exemple : C, L, P pin 18, 19, 20 ;

Le mot **istype**, suivi d'un argument définit le type de la variable de sortie :

- **Istype 'com, buffer'** indique que la sortie est **combinatoire** active sur niveau **haut**.
- **Istype 'com, invert'** indique que la sortie est **combinatoire** active sur niveau **bas**.
- **Istype 'reg, buffer '** indique que la sortie est **séquentielle** - registre - active sur niveau **haut**.
- **Istype 'reg, invert '** indique que la sortie est **séquentielle** - registre - active sur niveau **bas**.

Exemple : S1, S2 pin 16, 17 istype 'com, buffer';

On peut aussi trouver dans cette section les déclarations de groupement de variables -bus-.

Exemple : N = [E3, E2, E1, E0] ;
S = [Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0] ;

- ✓ Le mot N est constitué des bits E3, E2, E1 et E0.
- ✓ E3 constitue le MSB (bit de poids fort) et E0 le LSB (bit de poids faible).
- ✓ Le mot S est constitué des bits Q7, Q6, Q5, Q4, Q3, Q2, Q1 et Q0.
- ✓ Q7 constitue le MSB et Q0 le LSB.

Le langage ABEL permettra de traiter ces variables individuellement ou le nombre résultant de leur association. Ceci permettra de simplifier considérablement l'écriture de certaines équations.

3.4- Partie test :

Est commencé par le mot clé **TEST_VECTORS** ; l'utilisateur vérifie avec des exemples de valeurs données aux entrées les états des sorties et s'assure du bon fonctionnement du programme ;
Exemple :

```
TEST_VECTORS      ([a, b, c, d]  =>  [s1, s2])// Variables d'entrée et variables de sortie
      ([0, 1, 0, 1]  =>  [1, 1]) ; // Les sorties sont à 1 pour la combinaison 0101 des entrées.
      ([1, 0, 1, 0]  =>  [0, 0]) ;  // Les sorties sont à zéro pour la combinaison 1010 des entrées.
```

3.5- Partie de description logique:

C'est dans cette partie qui est décrite la fonction à programmer, la description peut se faire de plusieurs moyens (équations, table de vérité, diagramme d'état)

A- Description par équations :

Elle est annoncée par le mot clé **EQUATIONS**, elle décrit les relations liant les sorties aux entrées ; ces relations peuvent être des fonctions logiques, arithmétiques ou relationnelles :
Le langage ABEL met à la disposition de l'utilisateur un jeu d'opérateurs complet en voici les détails :

Opérateurs logiques	
!	Complément
#	OU
&	ET
$	OU exclusif
=	Affectation combinatoire
:=	Affectation séquentielle

Opérateurs arithmétiques	
+	Addition
-	Soustraction
*	Multiplication
/	Division entière
%	Reste de la division entière
<<	Décalage à gauche
>>	Décalage à droite

Opérateurs relationnels	
==	Test d'égalité
!=	Test différent
<	Test inférieur
>	Test supérieur
<=	Test inférieur ou égal
>=	Test supérieur ou égal

Pour travailler avec les nombres, ABEL précise la base du système de numération adoptée en utilisant les symboles suivants :

- ✓ Nombre binaire : ^b exemple : ^b1001
- ✓ Nombre octal : ^o exemple : ^o 45
- ✓ Nombre hexadécimal : ^h exemple : ^h FA
- ✓ Nombre décimal : ^d ou rien exemple : ^ 95

Exemple 1 :

```
EQUATIONS
    S1 = a &! b & ! c # ! a & b & c;
    S2 = ! (a # b & c);
    L = A1==B1 ;
```

Exemple 2 :

```
EQUATIONS
    S= A>B; // S = 1 si A > B ;
    I = A<B; // I = 1 si A < B ;
    E= A=B; // S =1 si A = B ;
```

Exemple 3 :

Utilisation de **When** …. **Then** ….

```
EQUATIONS
    When A > B then S = 1;
    When A = = B then I = 1;
    When A < B then E = 1;
```

Exemple 4 :

```
DECLARATIONS
    A3,A2,A1,A0 pin 2,3,4,5;
    S3,S2,S1,S0 pin 8,9,10,11 istype 'com';
    S=[S3,S2,S1,S0]; // déclaration du groupe S
    A=[A3,A2,A1,A0]; // et du groupe A
EQUATIONS
    A= S<<1 ;
 //décalage à gauche de 1 bit
 //A3=S2  A2=S1 A1=S0  A0=0
    S= A & [0, 1, 0,0] ;
 // masque (test du bit A2) : S= [0, A2, 0,0]
```

Exemple 5 :

Utilisation de **When** …. **Then** …. **Else** ….

```
EQUATIONS
    When A > B then S = 1 ELSE
        When A = = B then I = 1 ELSE
            E = 1;
```

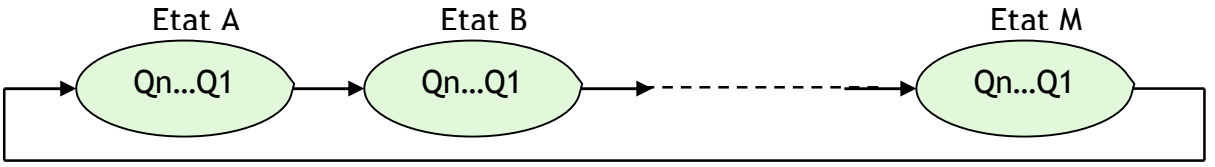
B- Description par table de vérité :

Elle est annoncée par le mot clé **Truth_Table** suivi des nombres de variables d'entrée et de sortie, on établit juste après la table de vérité en respectant les dispositions des variables d'entrée et de sortie comme le montre l'exemple suivant :

```
Truth_table ([A, B] -> [S1, S2, S3])
            [0, 0] -> [0, 1, 0];
            [0, 1] -> [0, 0, 1];
            [1, 0] -> [1, 0, 0];
            [1, 1] -> [1, 1, 1];
```

C- Description par diagramme d'état :

Le diagramme d'état est souvent avantagé dans le cas des machines séquentielles dont le fonctionnement est décrit par l'enchaînement séquentiel des états de sortie.



Instruction **GOTO :** (aller à)

Utilisée pour les transition d'état inconditionnelles. **GOTO** peut être utilisée si l'état présent n'offre qu'un seul état suivant.

Format du diagramme :

```
QSTATE      =      [Qn …Q1] ;
      A     =      [Qn …Q1] ;
      B     =      [Qn …Q1] ;
      ..
      M     =      [Qn …Q1] ;

STATE_DIAGRAM  QSTATE
      State A:     GOTO B;
      State B:     When Y = = 0 Then C Else A;
      State C:     GOTO D;
      State D:     When Y = = 0 Then A Else C;
```

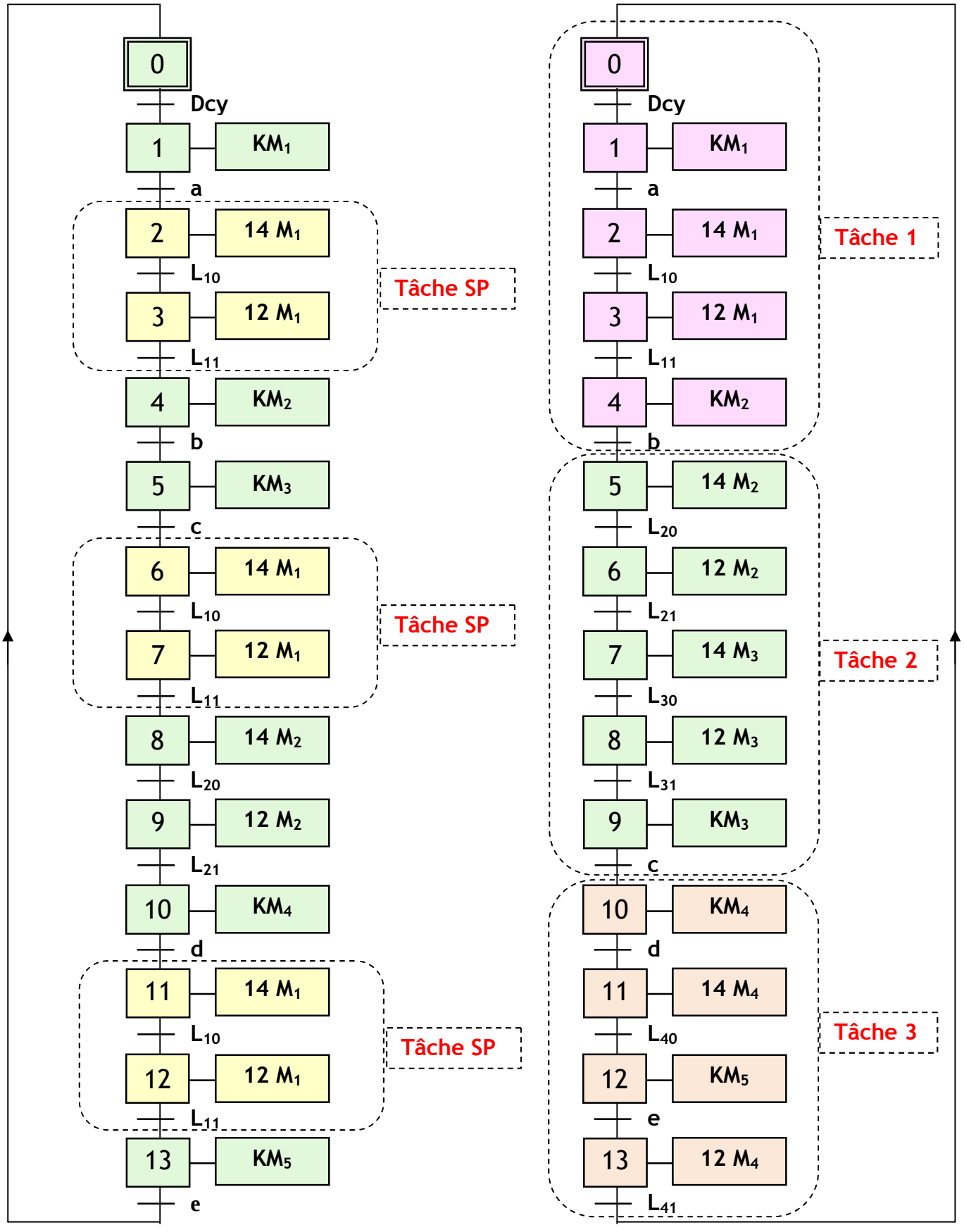
3.6- Constantes spéciales et extensions :

Constantes	
.C.	entrée d'horloge bas-haut-bas (monostable)
.K.	entrée d'horloge haut-bas-haut (monostable)
.U.	front montant d'une entrée horloge
.D.	front descendant d'une entrée horloge
.X.	valeur indéterminée à calculer
.Z.	valeur trois état
.P.	préchargement dans un registre

extensions	
.clk	entrée d'horloge pour bascule
.AR	reset asynchrone de la bascule
.AP	preset asynchrone de la bascule

PARTITION D'UN GRAFCET

Soient les deux GRAFCETS ci-dessous :

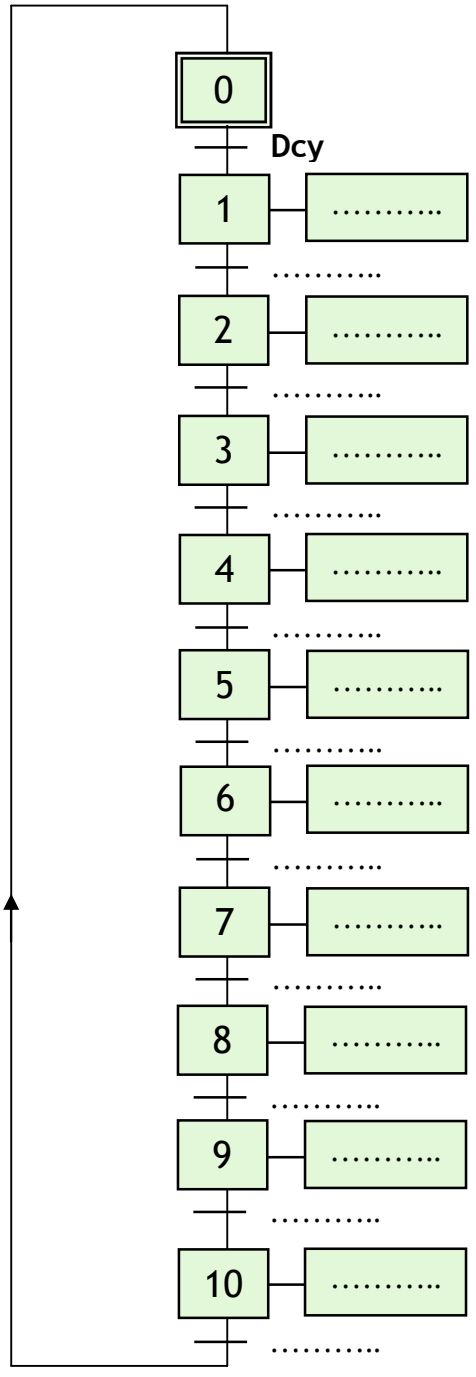


GRAFCET n°1 GRAFCET n°2

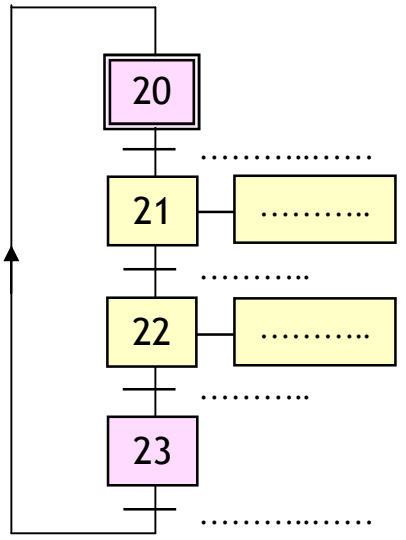
Coordination Verticale asynchrone :

GRAFCET n°1 :

Le GRAFCET n°1 contient une séquence répétitive. On veut réécrire ce GRAFCET d'une manière simplifiée en considérant la séquence répétitive comme une tâche (sous programme) intitulée SP. Compléter le GRAFCET **principal** et le GRAFCET de tâche **SP**.



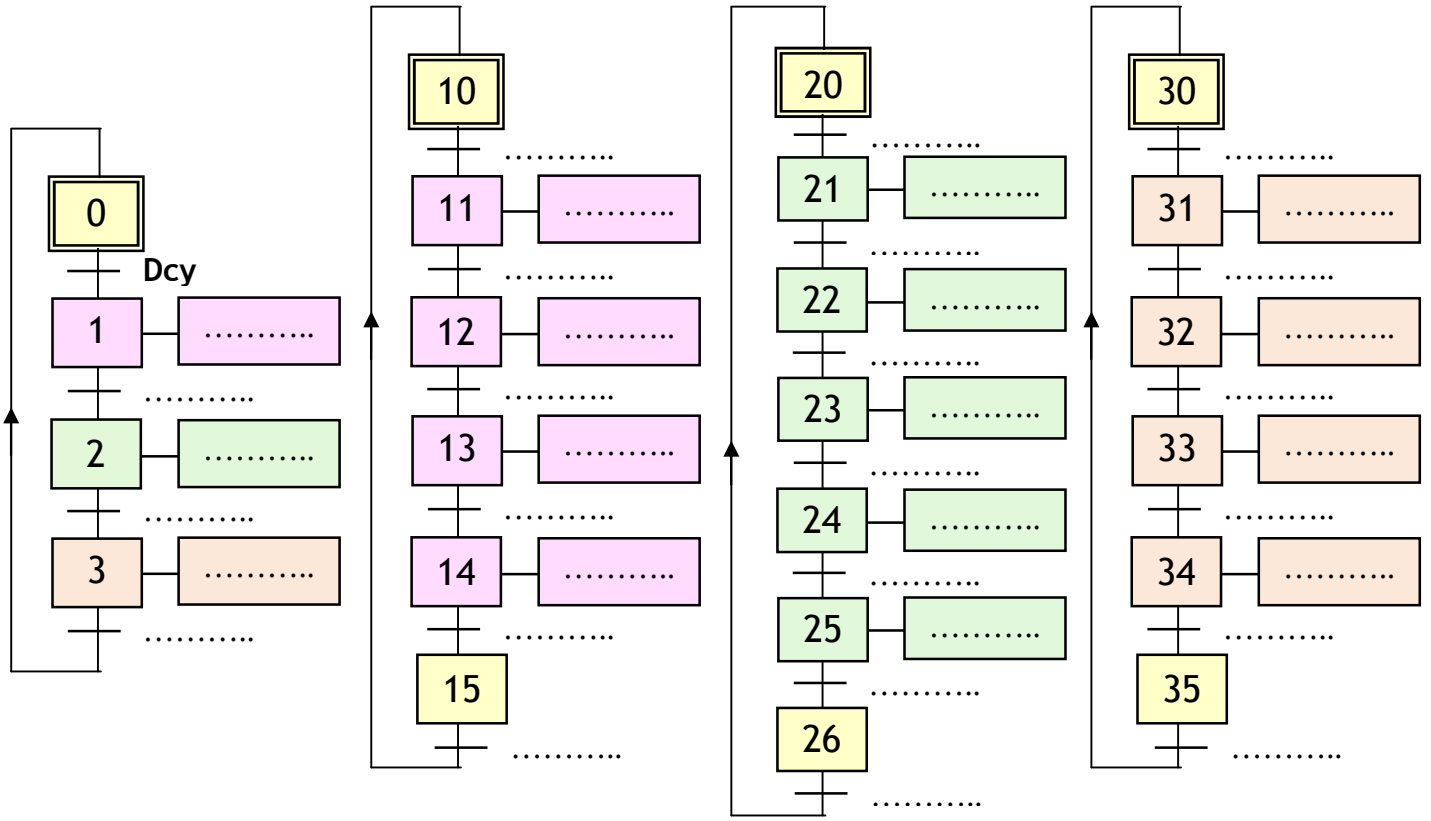
GRAFCET principal



GRAFCET de tâche SP

GRAFCET n°2 :

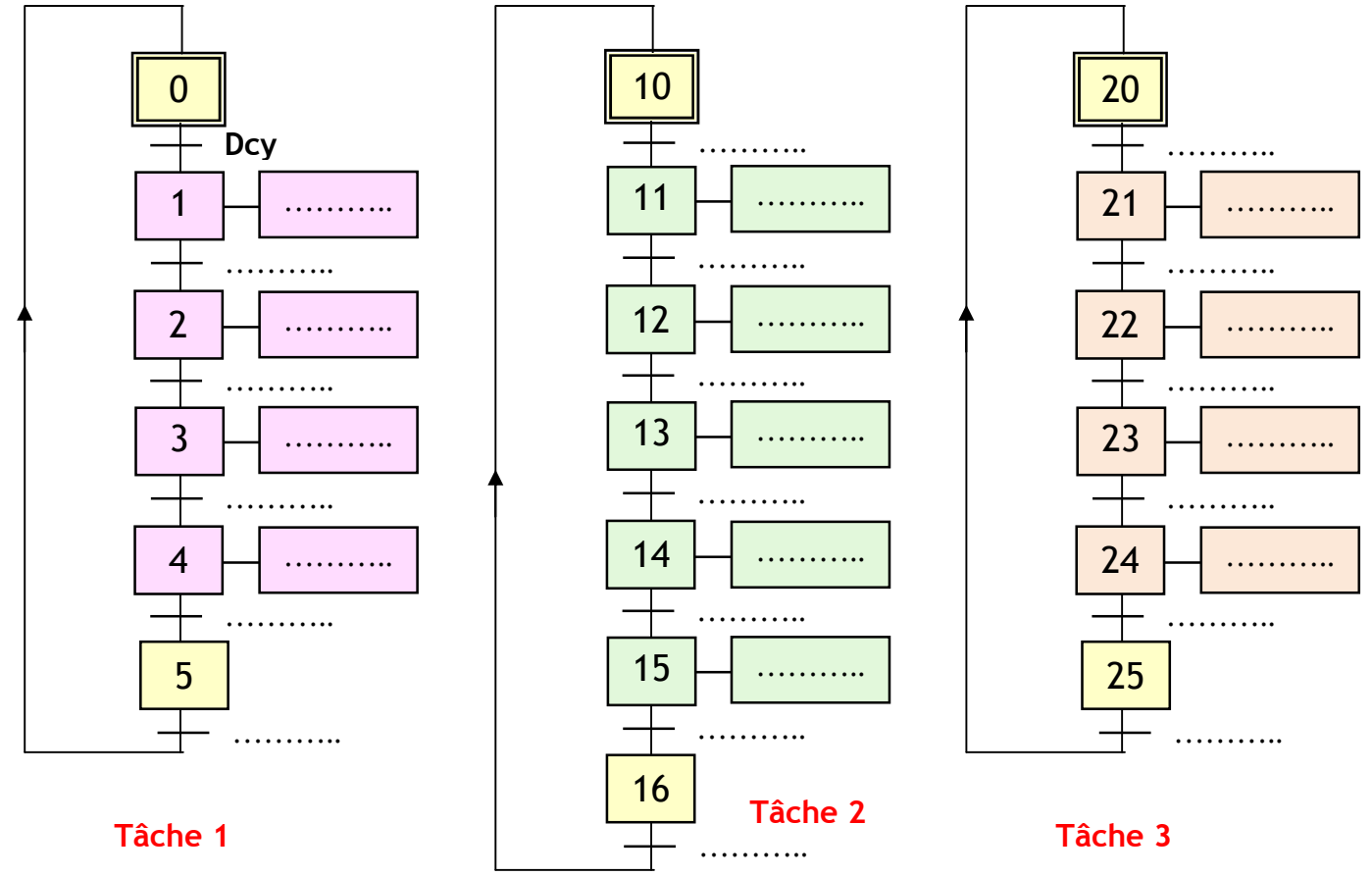
On veut réécrire le GRAFCET n°2 d'une manière simplifiée. Compléter le GRAFCET **principal** et les GRAFCETS de tâche **1, 2 et 3**.



GRAFCET principal **Tâche 1** **Tâche 2** **Tâche 3**

Coordination Horizontale :

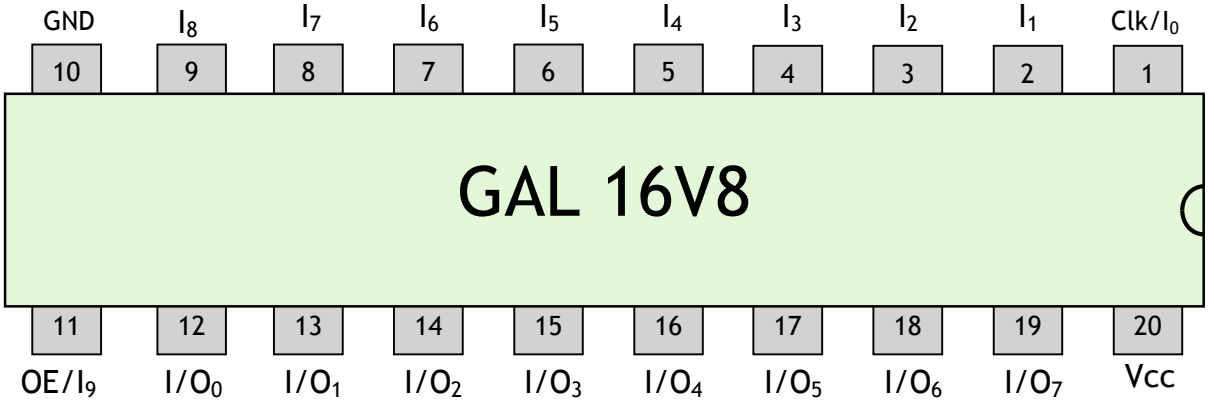
On veut réécrire ce GRAFCET n°2 d'une manière simplifiée. Compléter les GRAFCETS **Tâche 1, 2 et 3**.



Tâche 1 **Tâche 2** **Tâche 3**

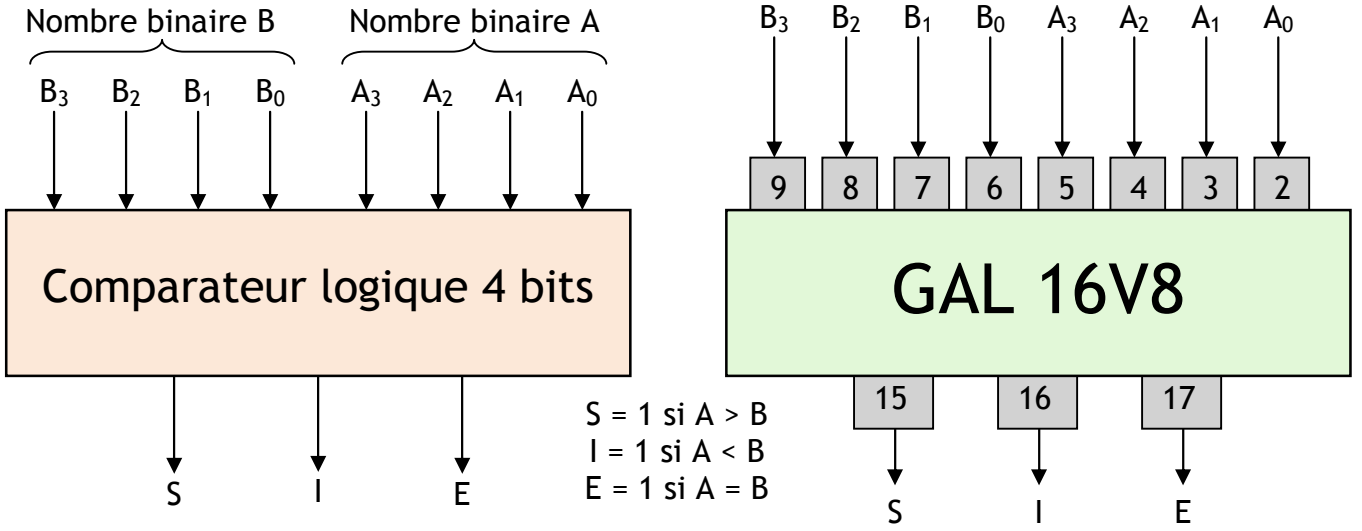
PROGRAMMATION DES PLD

GAL à programmer :



C'est circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmables.

1- Comparateur binaire 4 bits :



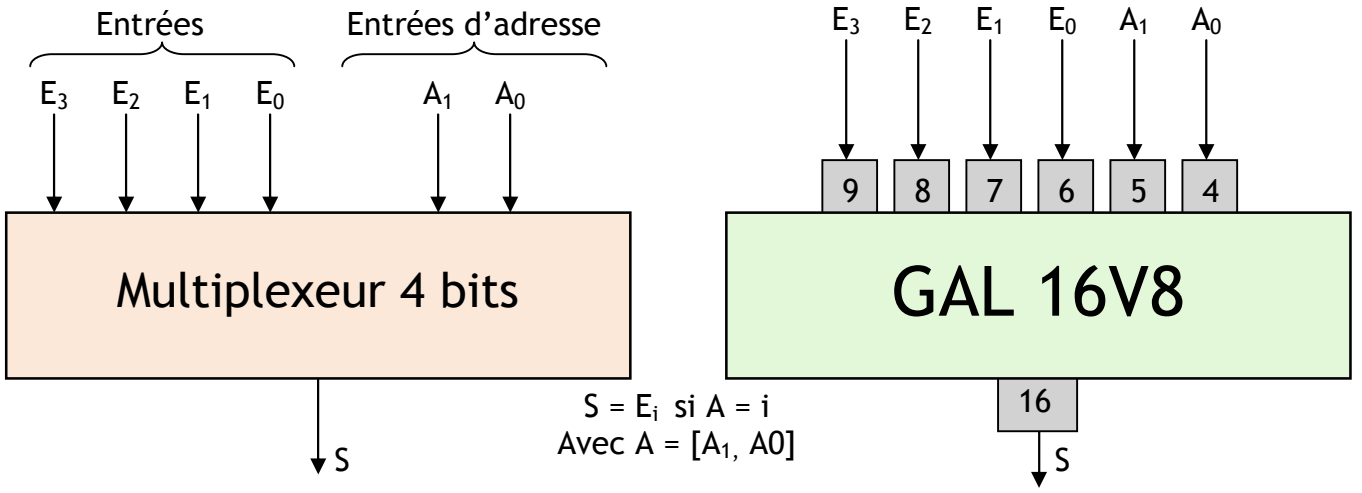
Description par équations :

MODULE comparateur
TITLE 'comparateur logique 4 bits'
DECLARATIONS
Comparateur **device** 'P16V8'; // PLD à programmer
..... // Variables d'entrée
..... // Variables d'entrée
..... // Définition du bus A
..... // Définition du bus B
..... // Variables de sortie
EQUATIONS
.....
.....
.....
TEST_VECTORS ([A, B] -> [S, E, I]);
.....
.....
.....
END comparateur

Utilisation de **When …. Then ….**

EQUATIONS
.....
.....
.....

2- Multiplexeur 4 bits :



Description par équations :

A- Solution 1 :

```
.....
.....
DECLARATIONS
        .....                    // PLD à programmer
        .....                    // Variables d'entrée
        A1, A0 pin 5, 4;         // Variables d'entrée
        .....                    // Variables de sortie
EQUATIONS
        .....
TEST_VECTORS  .....
                [0, 0, 0, 0, 0, 0] -> .....
                [0, 0, 0, 1, 0, 0] -> .....
                [0, 0, 0, 0, 0, 1] -> .....
                [0, 0, 1, 0, 0, 1] -> .....
                [0, 0, 0, 0, 1, 0] -> .....
                [0, 1, 0, 0, 1, 0] -> .....
                [0, 0, 0, 0, 1, 1] -> .....
                [1, 0, 0, 0, 1, 1] -> .....
END Multiplexeur.
```

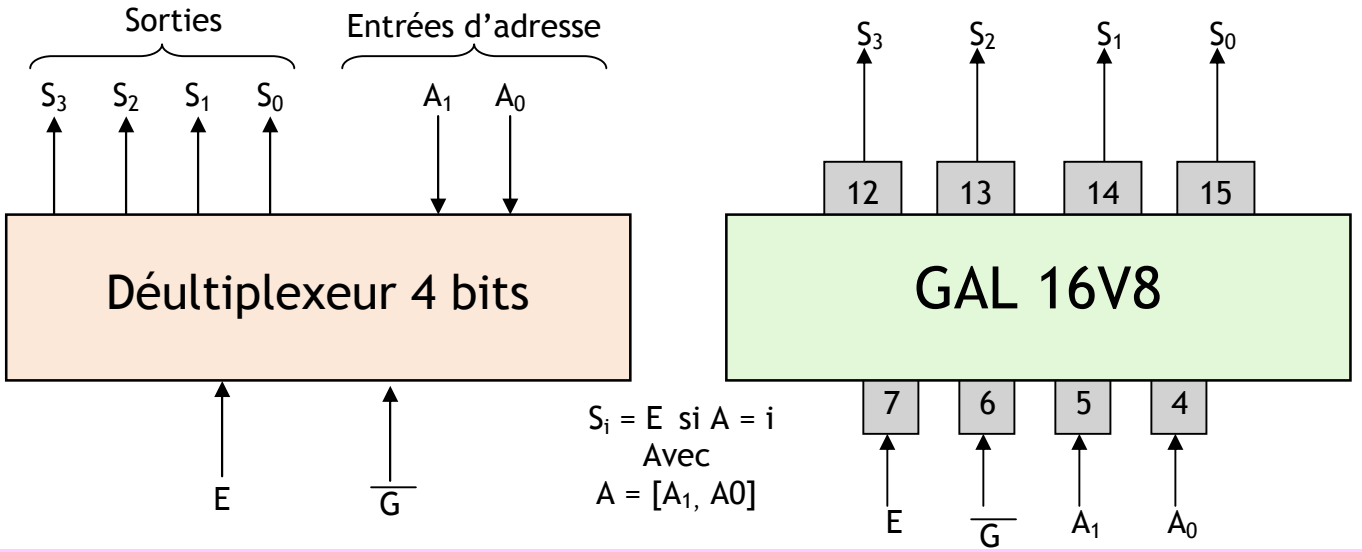
B- Solution 2 :

```
MODULE Multiplexeur
TITLE 'Multiplexeur 4 bits'
DECLARATIONS
        Multiplexeur  device 'P16V8';          // PLD à programmer
        E3, E2, E1, E0, A1, A0 pin 9, 8, 7, 6, 5, 4 ;   // Variables d'entrée
        .....                    // Définition du bus A
        .....                    // Définition du bus E
        S pin 16  istype 'com, buffer ';        // Variables de sortie
EQUATIONS
        .....
        .....
        .....
        .....
```

TEST_VECTORS .....
.....
.....
.....
.....
.....
.....
.....
.....

END Multiplexeur.

3- Démultiplexeur 4 bits :



3.1- Description par équations :

A- Solution 1 :

MODULE .....
TITLE .....
DECLARATIONS
..... // PLD à programmer
..... // Variables d'entrée
..... // Variables de sortie
EQUATIONS
S3 = ...
S2 = ...
S1 = ...
S0 = ...
TEST_VECTORS .....
.....
.....
.....
.....
.....
.....
.....
.....
.....

END Démultiplexeur.

B- Solution 2 :

```
MODULE .....
TITLE 'Multiplexeur 4 bits'
DECLARATIONS
        .....                                          // PLD à programmer
        E, G, A1, A0 pin 7, 6, 5, 4;                   // Variables d'entrée
        S3, S2, S1, S0 pin 12, 13, 14, 15 istype 'com, buffer '; // Variables de sortie
        .....                                          // Définition du bus A
        .....                                          // Définition du bus S
EQUATIONS
        When (G == 0) & (A == 0) then S0 = E;
        .....
        .....
        .....
TEST_VECTORS   ([G, E, A]       -> [S]);
                [1,.X.,.X.]     -> [0];
                .....
                .....
                .....
                .....
                .....
                .....
                .....
                .....
END Démultiplexeur.
```

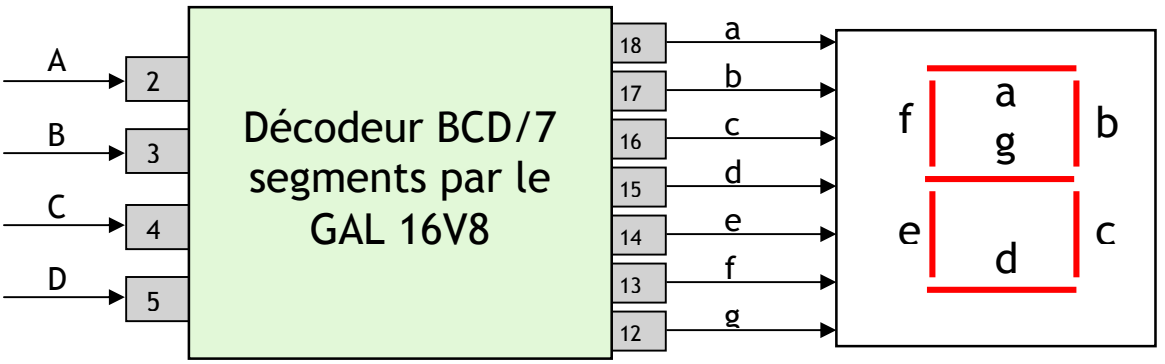
3.2- Description par table de vérité :

```
MODULE Démultiplexeur
TITLE 'Démultiplexeur 4 bits'
DECLARATIONS
        Démultiplexeur device 'P16V8';                 // PLD à programmer
        E, G, A1, A0 pin 7, 6, 5, 4;                   // Variables d'entrée
        S3, S2, S1, S0 pin 12, 13, 14, 15 istype 'com, buffer '; // Variables de sortie
TRUTH_TABLE       ([G, E, A1, A0] -> [S3, S2, S1, S0]);
                [1,.X.,.X.,.X.] -> [0, 0, 0, 0];
                [0, 0, 0, 0]    -> [0, 0, 0, 0];
                [0, 0, 0, 1]    -> [0, 0, 0, 0];
                .....
                .....
                .....
                .....
                .....
                .....
TEST_VECTORS   ([G, E, A1, A0]          -> [S3, S2, S1, S0]);
                [1,.X.,.X.,.X.] -> [0, 0, 0, 0];
                [0, X.,.X.,.X.] -> [0, 0, 0, 0];
                [0, 1, 0, 0]    -> [0, 0, 0, 1];
                [0, 1, 0, 1]    -> [0, 0, 1, 0];
                [0, 1, 1, 0]    -> [0, 1, 0, 0];
                [0, 1, 1, 1]    -> [1, 0, 0, 0];
END Démultiplexeur.
```

Autre solution

```
([G, A1, A0] -> [S3, S2, S1, S0]);
[1,.X.,.X.]     -> [0, 0, 0, 0];
[0, 0, 0]       -> [0, 0, 0, E];
.....
.....
.....
```

4- Décodeur BCD 7 segments :



Description par table de vérité :

MODULE BCD
TITLE 'Décodeur BCD 7 segments '
DECLARATIONS

…… // PLD à programmer
…… // Variables d'entrée
…… // Variables sortie
…… // Définition du bus N

……

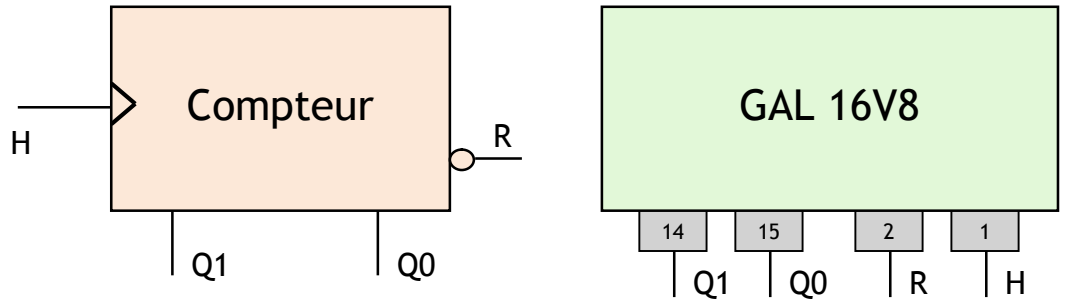
0 -> [1, 1, 1, 1, 1, 1, 0] ; // affichage du 0 dans le cas
1 -> [0, 1, 1, 0, 0, 0, 0] ; // d'un afficheur cathode commune
……
……
……
……
……
……
……
……

……

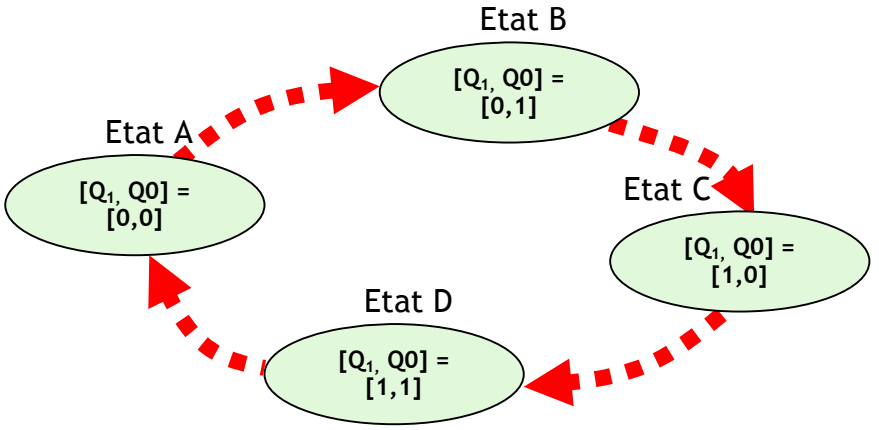
0 -> [1, 1, 1, 1, 1, 1, 0] ;
1 -> [0, 1, 1, 0, 0, 0, 0] ;
……
……
……
……
……
……
……
……

END BCD.

5- Compteur modulo 4 :



5.1- Description par diagramme d'état :




```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                    // PLD à programmer
        .....                                        // Variables d'entrée
        .....                                        // Variables de sortie
EQUATIONS
        [Q1, Q0].clk = ......
        [Q1, Q0].AR = .....
QSTATE      =       [Q1, Q0] ;
A           =       [0, 0] ;
B           =       [0, 1] ;
C           =       [1, 0];
D           =       [1, 1];
STATE_DIAGRAM   QSTATE
        State A:    GOTO B;
        ..........................................
        ..........................................
        ..........................................
TEST_VECTORS    ([R, H,]        -> [Q1, Q0])
                 [0, .X.]       -> [0, 0] ;
                 [1, . C.]      -> [0, 1] ;
                 ..........................................
                 ..........................................
                 ..........................................
END Compteur
```

5.2- Description par équation :

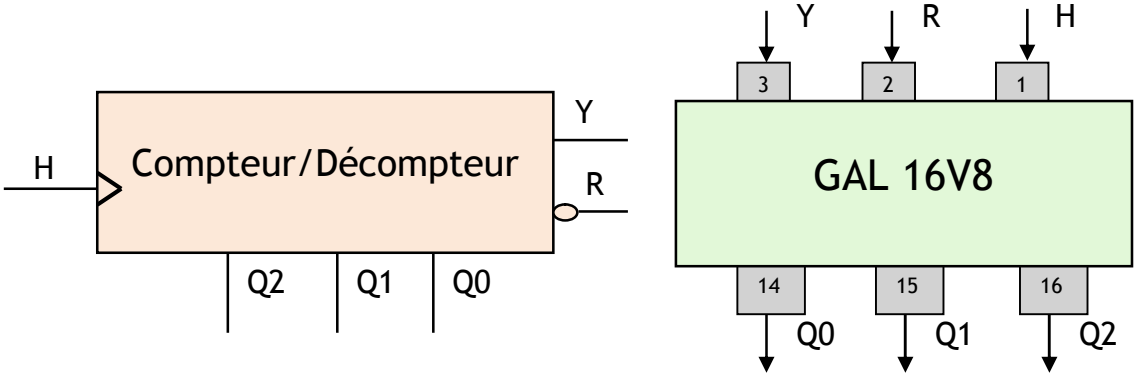
```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                    // PLD à programmer
        H, R pin  1, 2 ;                             // Variables d'entrée
        Q0, Q1 pin 15, 14 Istype 'reg, buffer ';     // Variables de sortie
        .....                                        // Définition du bus Q
EQUATIONS
        ..................................................
        ..................................................
        ..................................................
```

```
TEST_VECTORS    ([R, H,]        -> [Q1, Q0])
                 [0, .X.]       -> [0, 0] ;
                 [1, . C.]      -> [0, 1] ;
                 [1, . C.]      -> [1, 0] ;
                 [1, . C.]      -> [1, 1] ;
                 [1, . C.]      -> [0, 0] ;
END Compteur
```

5.3- Description par Table de vérité :

```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                  // PLD à programmer
        H, R pin  1, 2 ;                           // Variables d'entrée
        Q0, Q1 pin 15, 14 lstype 'reg, buffer ';   // Variables de sortie
EQUATIONS
        .....................................
        .....................................
TRUTH_TABLE     ....................................
                .....................................
                .....................................
                .....................................
                .....................................
                .....................................
TEST_VECTORS    ([R, H,]        -> [Q1, Q0])
                 [0, .X.]       -> [0, 0] ;
                 [1, . C.]      -> [0, 1] ;
                 [1, . C.]      -> [1, 0] ;
                 [1, . C.]      -> [1, 1] ;
                 [1, . C.]      -> [0, 0] ;
END Compteur
```

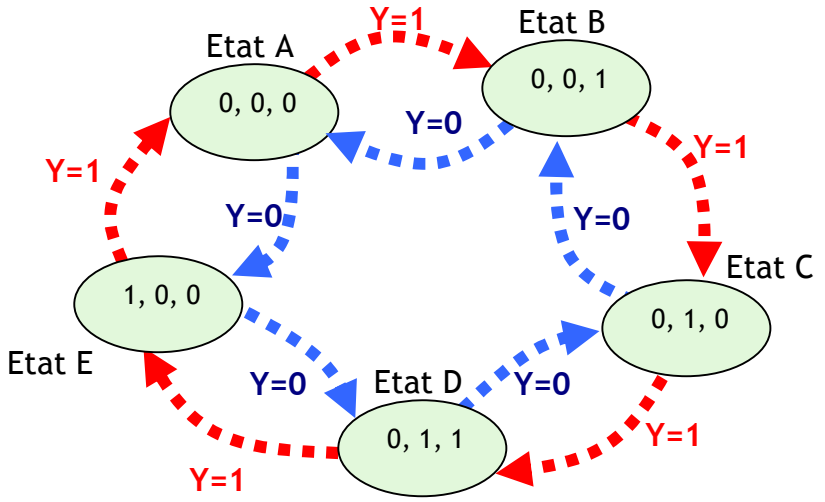
6- Compteur modulo 5 réversible :



Conception d'un compteur modulo 5 réversible suivant une entrée de sélection du mode de fonctionnement Y=comptage/décomptage ; par le GAL P16V8.

- ✓ Si Y=1 c'est le comptage de 000 à 100.
- ✓ Si Y=0 c'est le décomptage de 100 à 000.

6.1- Description par diagramme d'état :




```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        Compteur device 'P16V8' ;                          // PLD à programmer
        H, R, Y pin  1, 2, 3 ;                             // Variables d'entrée
        Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer '; // Variables sortie
EQUATIONS
        .....
        .....
QSTATE      =       [Q2, Q1, Q0];
A           =       [0, 0, 0];
.....
.....
.....
.....
STATE_DIAGRAM   QSTATE
        State A:        When Y = = 1 Then B Else E;
        .....
        .....
        .....
        .....
TEST_VECTORS    ([R, H, Y]      -> [Q2, Q1, Q0])
                 [0, .X., .X.]   -> [0, 0, 0] ;
                 [1, .C., 1]     -> [0, 0, 1] ;
                .....
                .....
                .....
                .....
                 [1, .C., 0]     -> [1, 0, 0] ;
                .....
                .....
                .....
                .....
END Compteur_reversible
```

6.2- Description par table de vérité :

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        .....                                   // PLD à programmer
        .....                                   // Variables d'entrée
        .....                                   // Variables sortie
EQUATIONS
        [Q2, Q1, Q0].clk = H ;
        [Q2, Q1, Q0].AR = ! R;
TRUTH_TABLE     ([R, Y, Q2, Q1, Q0] :> [Q2, Q1, Q0])
                [0, .X., .X., .X., .X.] :> [0, 0, 0] ;
                [1, 1, 0, 0, 0]  :> [0, 0, 1] ;
                [1, 1, 0, 0, 1]  :> [0, 1, 0] ;
                ..................................................
                ..................................................
                ..................................................
                [1, 0, 0, 0, 0]  :> [1, 0, 0] ;
                [1, 0, 1, 0, 0]  :> [0, 1, 1] ;
                ..................................................
                ..................................................
                ..................................................
TEST_VECTORS    ([R, H, Y]      -> [Q2, Q1, Q0])
                 [0, .X., .X.]   -> [0, 0, 0] ;
                 [1, . C., 1]     -> [0, 0, 1] ;
                 [1, . C., 1]     -> [0, 1, 0] ;
                ..................................................
                ..................................................
                ..................................................
                 [1, . C., 0]     -> [1, 0, 0] ;
                 [1, . C., 0]     -> [0, 1, 1] ;
                ..................................................
                ..................................................
                ..................................................
END Compteur_reversible.
```

6.3- Description par équation :

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        Compteur device 'P16V8' ;                       // PLD à programmer
        H, R, Y pin  1, 2, 3 ;                          // Variables d'entrée
        Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer '; // Variables sortie
        .....                                           // Définition du bus Q
EQUATIONS
        ..........................................
        ..........................................
        ..................................................
        ..................................................
        ..................................................
END Compteur_reversible.
```

ROYAUME DU MAROC
MINISTERE DE L'EDUCATION NATIONALE
Académie de Casablanca
DÉLÉGATION DE MOHAMMEDIA
Lycée Technique Mohammedia

Matière :	Science de l'Ingénieur - **A.T.C** -	Pr.MAHBAB
Section :	Sciences et Technologies Électriques	Système n°5

CORRECTION

❖ Sujet :

PARKING AUTOMATIQUE

08 pages

❖ Exercices d'application:

- Fiche cours n°22 « *EEPROM DU 16F84* »

01 page

❖ 2 TD:

- TD n°8 « *Partition d'un GRAFCET et notion de Tâche* »
- TD n°9 « *Programmation des PLD* »

09 pages

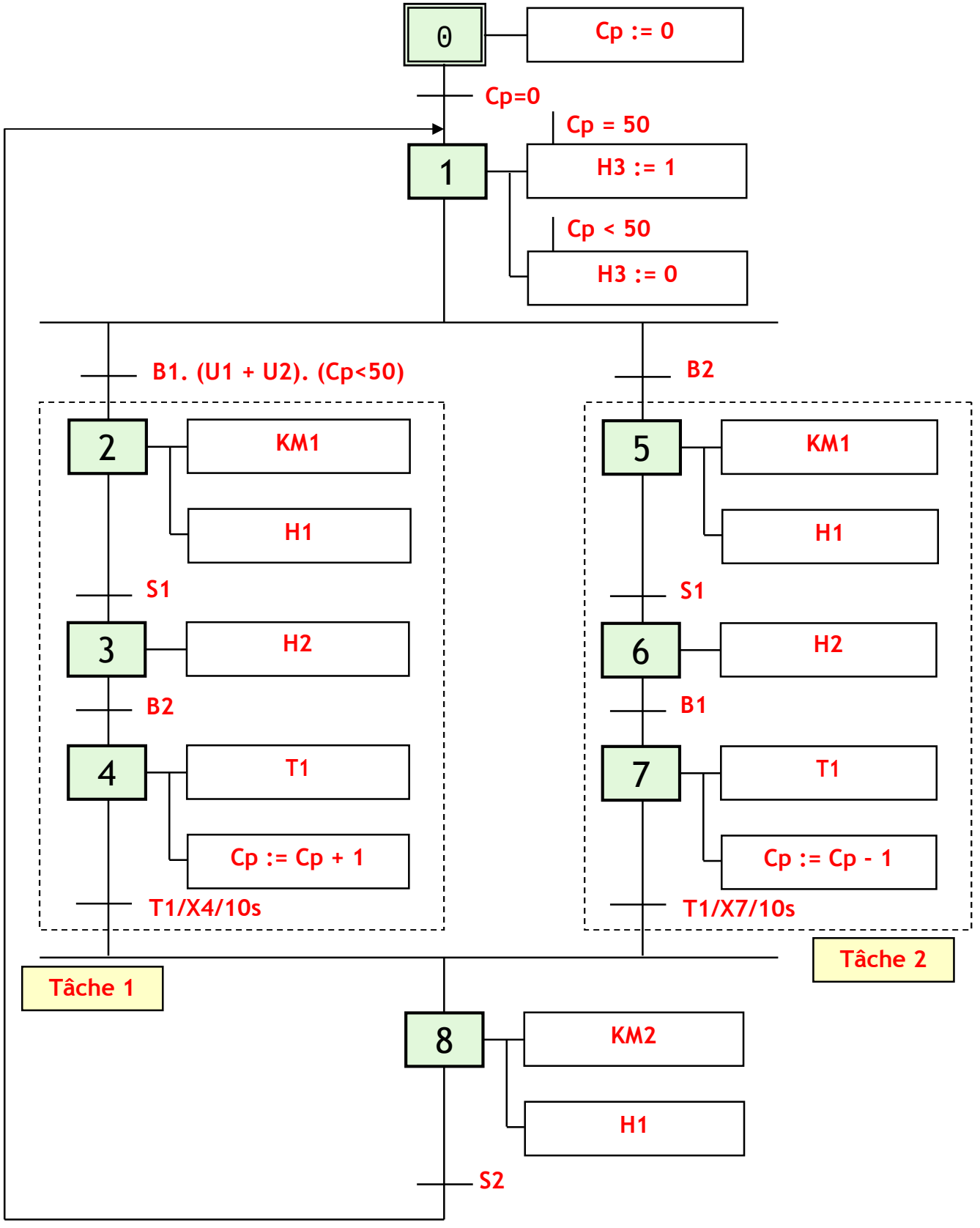
Nombre de pages : **19**

CLASSES : 2 STE

DREP 01

CORRECTION

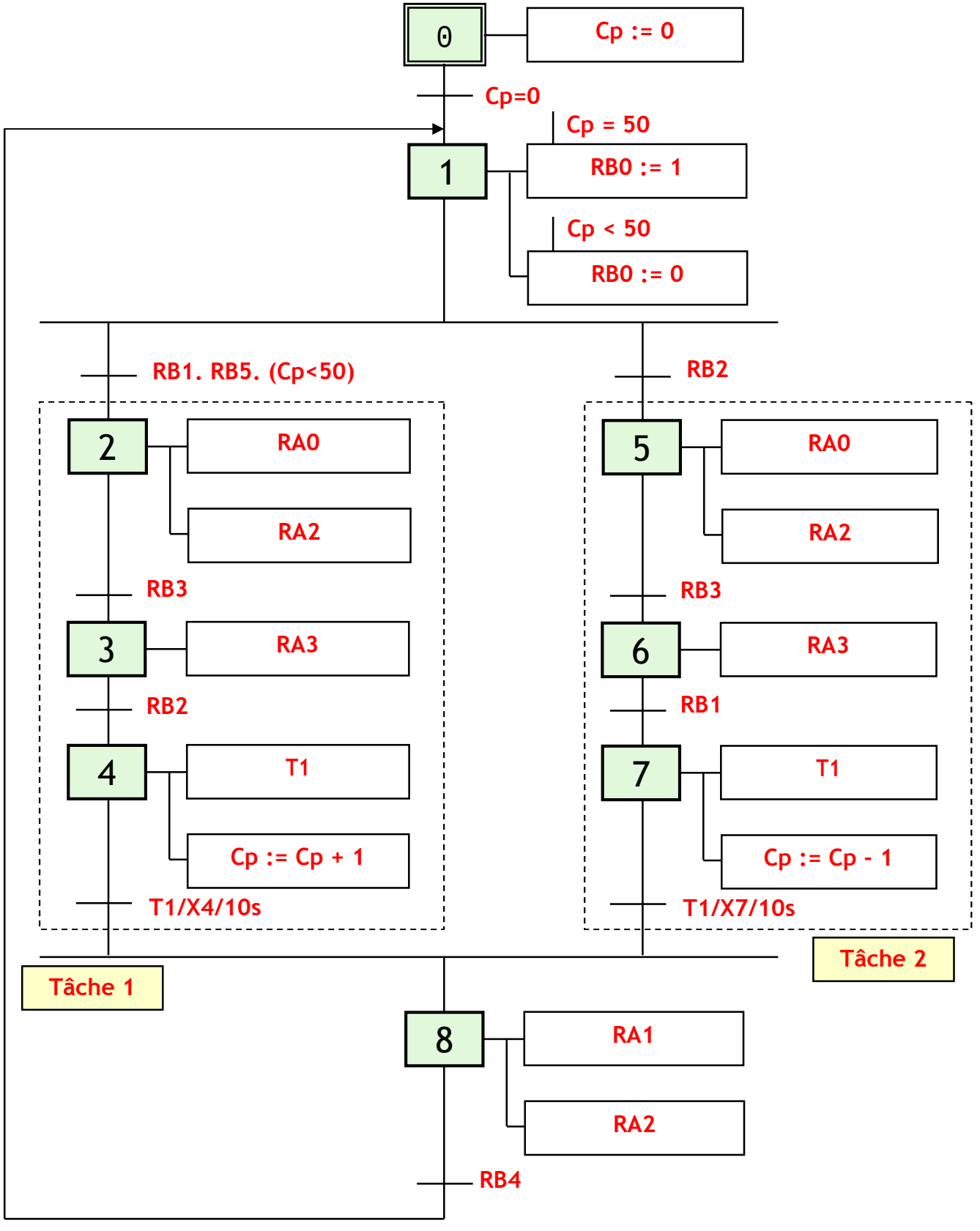
GRAFCET point de vue partie commande



DREP 02

CORRECTION

GRAFCET point de vue partie commande codé PIC 16 F 84

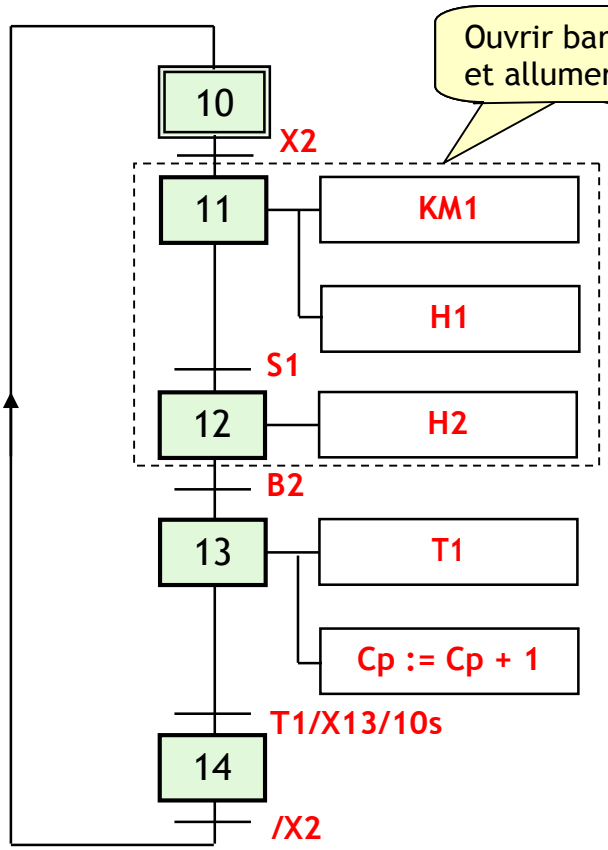


DREP 03

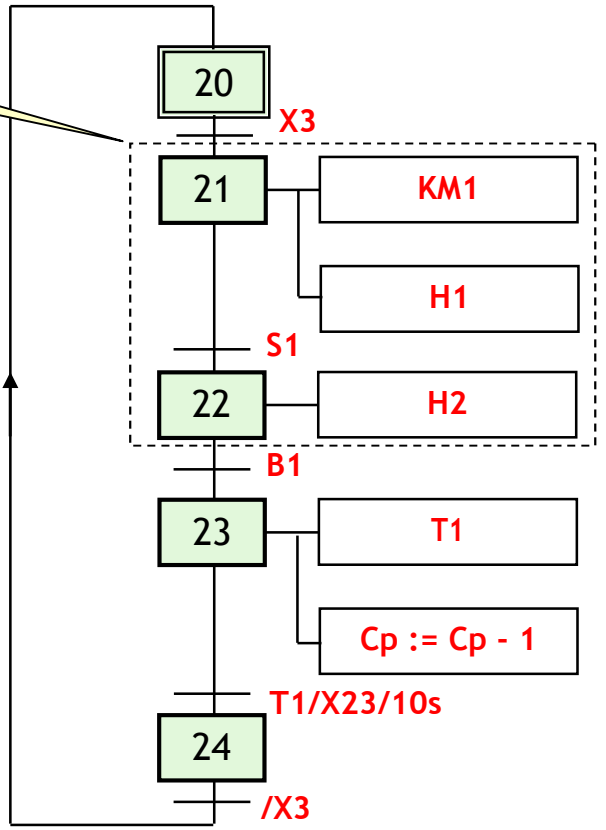
CORRECTION

Partition du GRAFCET global (P.C)

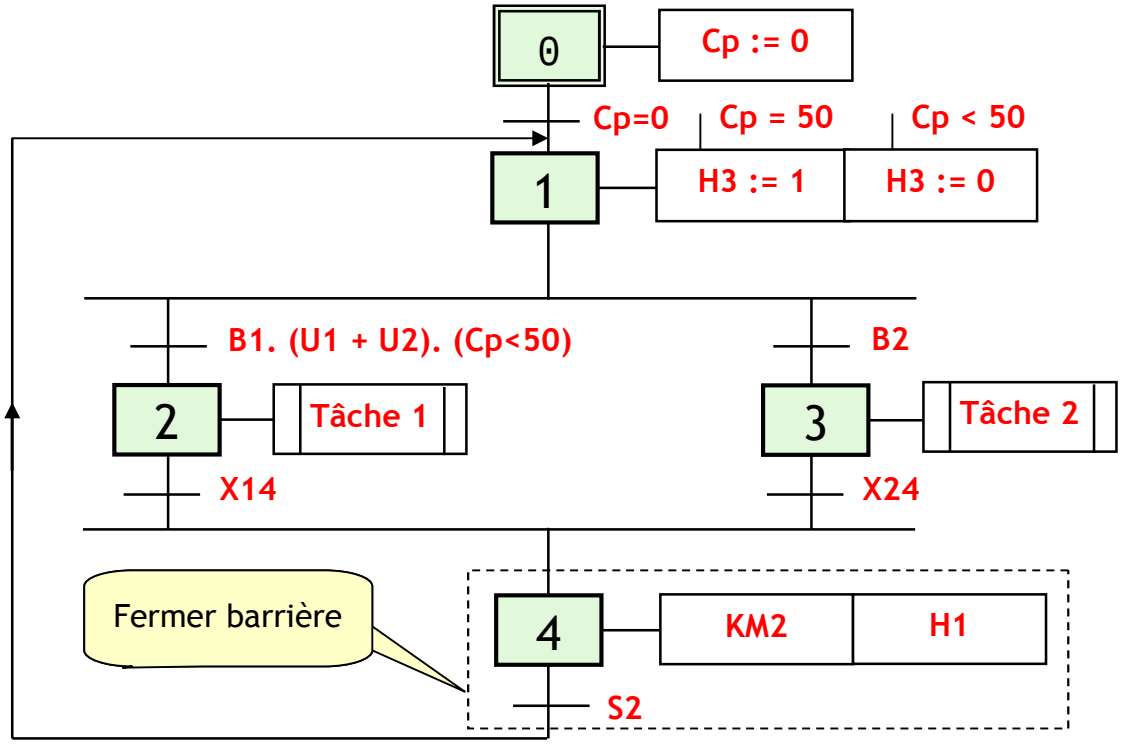
GRAFCET Tâche 1



GRAFCET Tâche 2



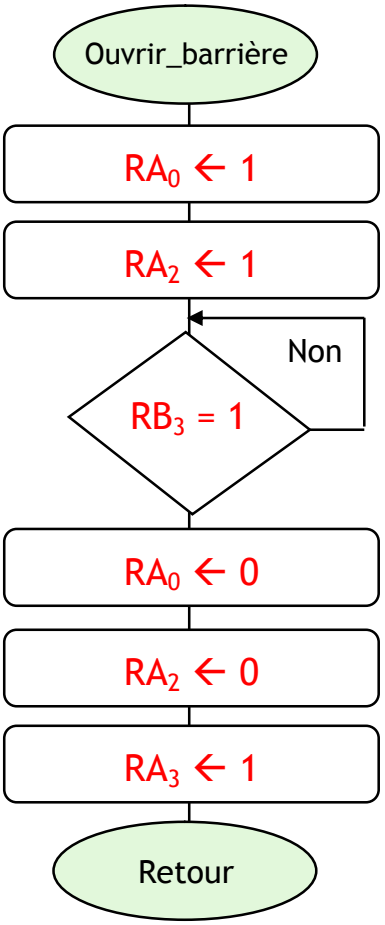
GRAFCET PRINCIPAL



DREP 04

CORRECTION

Sous programme ouvrir_Barrière

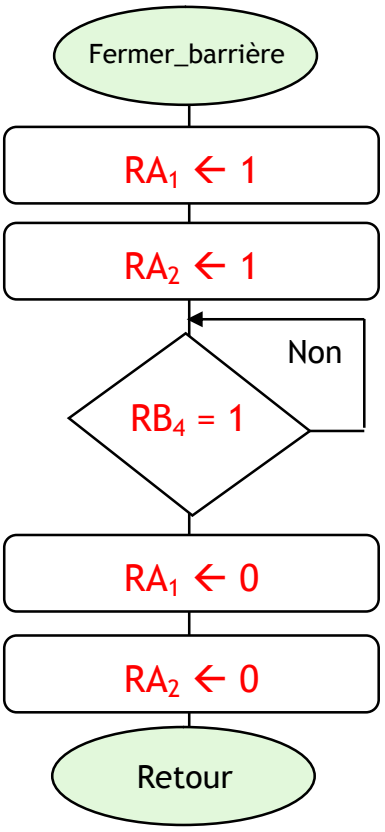


Organigramme

Programme

```
Ouvrir_barriere   BSF     PORTA, 0
                  BSF     PORTA, 2
LAB5              BTFSS   PORTB, 3
                  GOTO    LAB5
                  BCF     PORTA, 0
                  BCF     PORTA, 2
                  BSF     PORTA, 3
                  RETURN
```

Sous programme fermer_Barrière



Organigramme

Programme

```
Fermer_barriere   BSF     PORTA, 1
                  BSF     PORTA, 2
LAB6              BTFSS   PORTB, 4
                  GOTO    LAB6
                  BCF     PORTA, 1
                  BCF     PORTA, 2
                  RETURN
```

DREP 05

CORRECTION

Sous programme Lecture_EEPROM

```
Lecture_EEPROM  MOVLW   0x02          ;
                MOVWF   EEADR         ; l'adresse à lire
                BSF     STATUS, RP0   ; Bank 1
                BSF     EECON1, RD    ; lecture EPROM
                BCF     STATUS, RP0   ; Bank 0
                MOVF    EEDATA, W     ; EEDATA dans W
                MOVWF   Cp            ; W dans Cp
                RETURN                ;
```

Sous programme Ecriture_EEPROM

```
Ecriture_EEPROM MOVLW   0x02          ;
                MOVWF   EEADR         ; définition de l'adresse
                MOVF    Cp, W         ;
                MOVWF   EEDATA        ; définition de la donnée
                BSF     STATUS, RP0   ; Bank 1
                BSF     EECON1, WREN  ; autorisation de l'écriture
                MOVLW   0x55          ;
                MOVWF   EECON2        ; écriture de 0x55
                MOVLW   0xAA          ;
                MOVWF   EECON2        ; écriture de 0xAA
                BSF     EECON1, WR    ; écriture dans EEPROM
LAB7            BTFSS   EECON, EEIF   ;
                GOTO    LAB7          ; écriture terminée
                BCF     EECON, EEIF   ; remise à zéro du témoin EEIF
                BCF     STATUS, RP0   ; Bank 0
                RETURN                ;
```

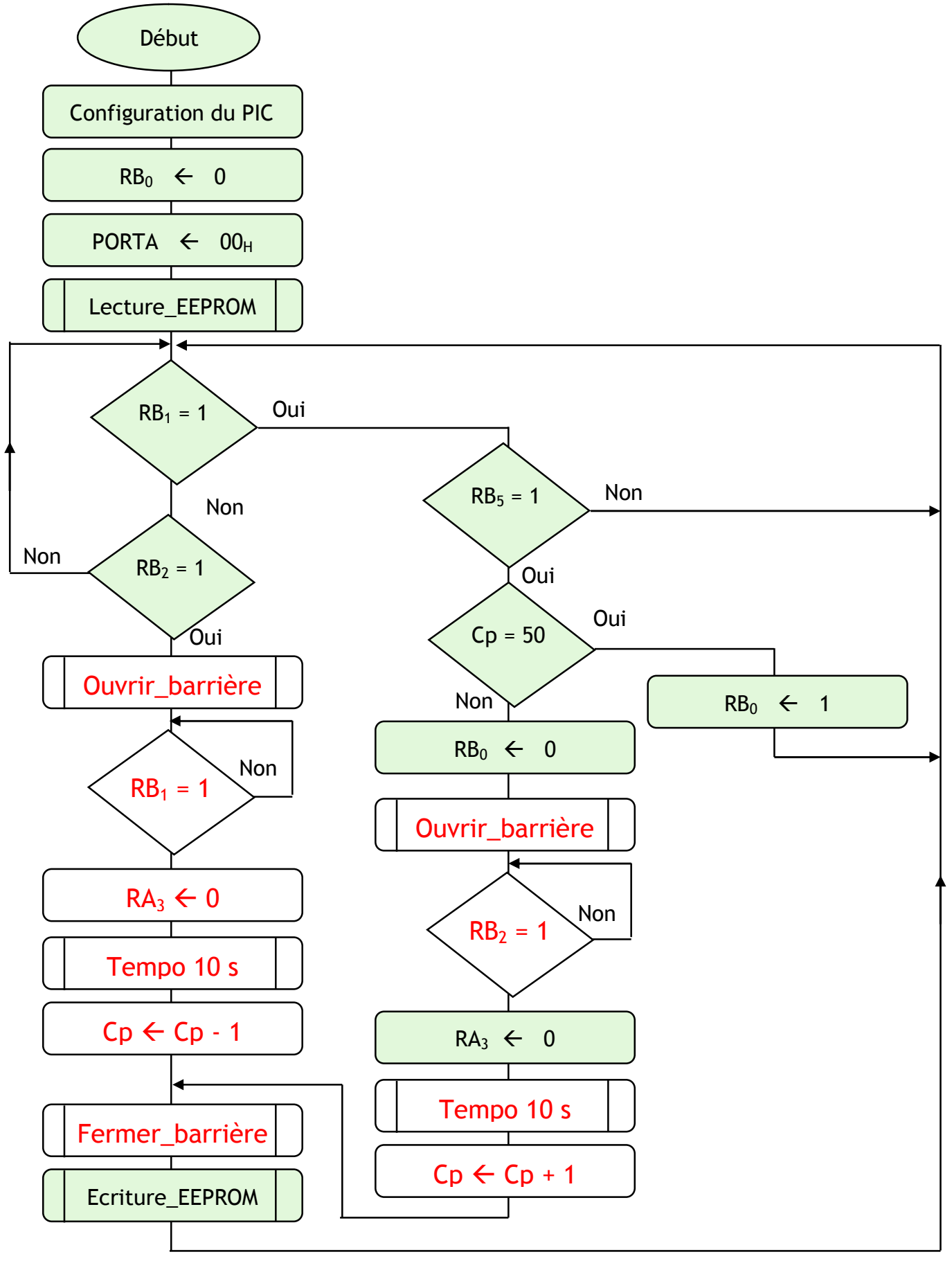
Sous programme TEMPO

```
TEMPO   MOVLW   D'100'        ; doit compter 156 impulsions
        MOVWF   TMR0          ; écriture de 100
LAB8    BTFSS   INTCON, TOIF  ; fin de comptage
        GOTO    LAB8          ;
        BCF     INTCON, TOIF  ; remise à zéro du témoin TOIF
        RETURN                ;
```

DREP 06

CORRECTION

Organigramme à compléter



DREP 07

CORRECTION

Programme à compléter

Initialisation du PIC 16 F 84

```
        BSF     STATUS, 5       ; accès à la BANK 1
        MOVLW   B'00001000'     ;
        MOVWF   TRISA           ; configuration du PORTA
        MOVLW   B'00111110'     ;
        MOVWF   TRISB           ; configuration du PORTB
        MOVLW   B'00110101'     ;
        MOVWF   OPTION          ; configuration du TIMER 0
        BCF     STATUS, 5       ; accès à la BANK 0
```

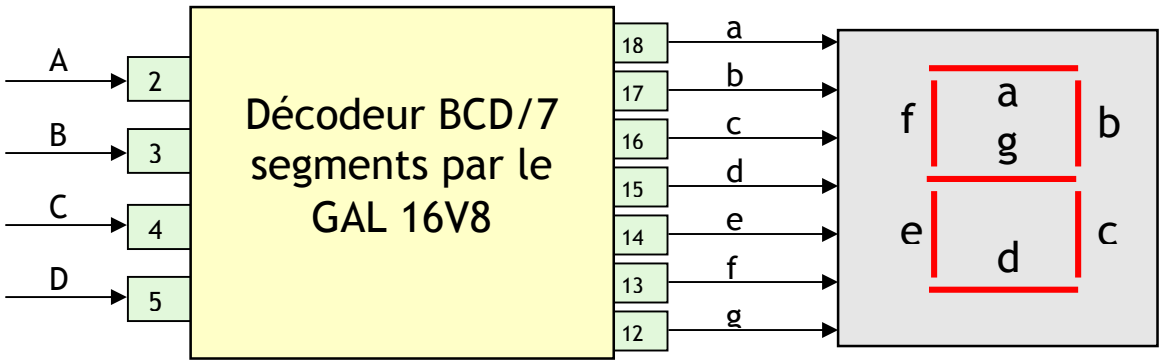
Programme principal

```
        CLRF    PORTA           ; moteur et voyants au repos
        CALL    Lecture_EPROM   ; nombre de véhicule dans Cp
DEBUT   BTFSS   PORTB, 1        ; test si entrée
        GOTO    SORTIE          ;
        GOTO    ENTRÉE          ;
SORTIE  BTFSS   PORTB, 2        ; test si sortie
        GOTO    DEBUT           ;
        CALL    Ouvrir_barriere ; ouvrir la barrière
LAB1    BTFSS   PORTB, 1        ; test si véhicule est sortie
        GOTO    LAB1            ;
        BCF     PORTA, 3        ; éteindre vert
        CALL    TEMPO           ; Temporisation T1
        DECF    Cp, 1           ; décrémenter compteur
        GOTO    LAB4            ;
ENTREE  BTFSS   PORTB, 5        ; test si autorisation
        GOTO    DEBUT           ;
        MOWLW   D'50'           ;
        SUBWF   Cp, W           ; comparer Cp et 10
        BTFSS   STATUS, Z       ; saut si Cp = 10
        GOTO    LAB2            ; parking disponible
        BSF     PORTB, 0        ; allumer jaune
        GOTO    DEBUT           ; parking complet
LAB2    BCF     PORTB, 0        ; éteindre jaune
        CALL    Ouvrir_barriere ; ouvrir la barrière
LAB3    BTFSS   PORTB, 2        ; test si véhicule est entré
        GOTO    LAB3            ;
        BCF     PORTA, 3        ; éteindre vert
        CALL    TEMPO           ; Temporisation T1
        INCF    Cp, 1           ; incrémenter compteur
LAB4    CALL    Fermer_barriere ; fermer la barrière
        CALL    Ecriture_EPROM  ; enregistrer compteur
        GOTO    DEBUT           ;
        END                     ; directive de fin de programme
```

DREP 08

CORRECTION

Décodeur BCD/7 segments par le GAL 16V8



Programme 'ABEL'

```
MODULE BCD
TITLE 'Décodeur BCD 7 segments '
DECLARATIONS
        BCD  device 'P16V8;                          // PLD à programmer
        A, B, C, D pin   2, 3, 4, 5 ;                // Variables d'entrée
        a, b, c, d, e, f, g  pin 18, 17, 16, 15, 14, 13, 12
        istype 'com,buffert' ;                       // Variables sortie
        N = [D, C, B, A] ;                           // Définition du bus N
        S = [a, b, c, d, e, f, g] ;                  // Définition du bus S
EQUATIONS
        When N = 0 THEN S = [1, 1, 1, 1, 1, 1, 0] ELSE
        When N = 1 THEN S = [0, 1, 1, 0, 0, 0, 0] ELSE
        When N = 2 THEN S = [1, 1, 0, 1, 1, 0, 1] ELSE
        When N = 3 THEN S = [1, 1, 1, 1, 0, 0, 1] ELSE
        When N = 4 THEN S = [0, 1, 1, 0, 0, 1, 1] ELSE
        When N = 5 THEN S = [1, 0, 1, 1, 0, 1, 1] ELSE
        When N = 6 THEN S = [1, 0, 1, 1, 1, 1, 1] ELSE
        When N = 7 THEN S = [1, 1, 1, 0, 0, 0, 0] ELSE
        When N = 8 THEN S = [1, 1, 1, 1, 1, 1, 1] ELSE
        When N = 9 THEN S = [1, 1, 1, 1, 0, 1, 1];
TEST_VECTORS    (N -> [a, b, c, d, e, f, g])
                0 -> [1, 1, 1, 1, 1, 1, 0] ;
                1 -> [0, 1, 1, 0, 0, 0, 0] ;
                2 -> [1, 1, 0, 1, 1, 0, 1] ;
                3 -> [1, 1, 1, 1, 0, 0, 1] ;
                4 -> [0, 1, 1, 0, 0, 1, 1] ;
                5 -> [1, 0, 1, 1, 0, 1, 1] ;
                6 -> [1, 0, 1, 1, 1, 1, 1] ;
                7 -> [1, 1, 1, 0, 0, 0, 0] ;
                8 -> [1, 1, 1, 1, 1, 1, 1] ;
                9 -> [1, 1, 1, 1, 0, 1, 1] ;
END BCD.
```

EEPROM DU 16 F 84

Ecriture et lecture de l'EEPROM :

Lecture d'une donnée :

- Placer l'**adresse** de la donnée à lire dans **EEADR**.
- Mettre le bit **RD** de EECONN1 à **1**.
- Lire le contenu du registre **EEDATA.**

```
        BCF     STATUS, RP0     ; Bank 0
        MOVLW   Adresse         ;
        MOVWF   EEADR           ; l'adresse à lire
        BSF     STATUS, RP0     ; Bank 1
        BSF     EECON1, RD      ; lecture EPROM
        BCF     STATUS, RP0     ; Bank 0
        MOVF    EEDATA, W       ; W ← EEDATA
```

Ecriture d'une donnée :

- Placer l'adresse de la donnée à écrire dans EEADR.
- Placer la donnée à écrire dans EEDATA.
- Mettre le bit WREN de EECONN1 à 1 pour autoriser l'écriture.
- Placer 0x55 dans EECON2.
- Placer 0xAA dans EECON2.
- Mettre le bit WR de EECONN1 à 1.
- Attendre que le bit EEIF soit à 1.
- On peut utiliser l'interruption produite par EEIF en la validant par le bit EEIE de INTCON.
- N'oublier pas de remettre EEIF à 0.

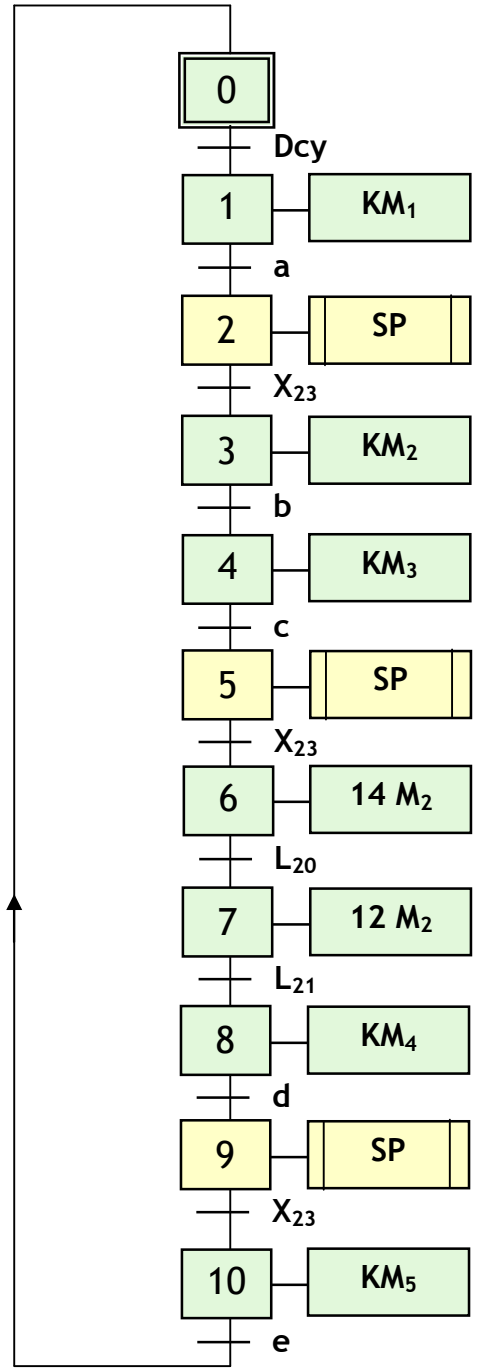
```
        BCF     STATUS, RP0     ; Bank 0
        MOVLW   Adresse         ;
        MOVWF   EEADR           ; définition de l'adresse
        MOVLW   Donnée          ;
        MOVWF   EEDATA          ; définition de la donnée
        BSF     STATUS, RP0     ; Bank 1
        BSF     EECON1, WREN    ; autorisation de l'écriture
        MOVLW   0x55            ;
        MOVWF   EECON2          ; écriture de 0x55
        MOVLW   0xAA            ;
        MOVWF   EECON2          ; écriture de 0xAA
        BSF     EECON1, WR      ; écriture dans EEPROM
Lab     BTFSS   EECON, EEIF     ;
        GOTO    Lab             ; écriture terminée
        BCF     STATUS, RP0     ; Bank 0
```

PARTITION D'UN GRAFCET

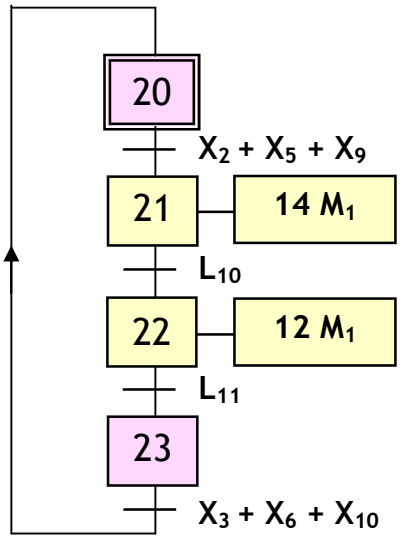
Coordination Verticale asynchrone :

GRAFCET n°1 :

Le GRAFCET n°1 contient une séquence répétitive. On veut réécrire ce GRAFCET d'une manière simplifiée en considérant la séquence répétitive comme une tâche (sous programme) intitulée SP. Compléter le GRAFCET **principal** et le GRAFCET de tâche **SP**.



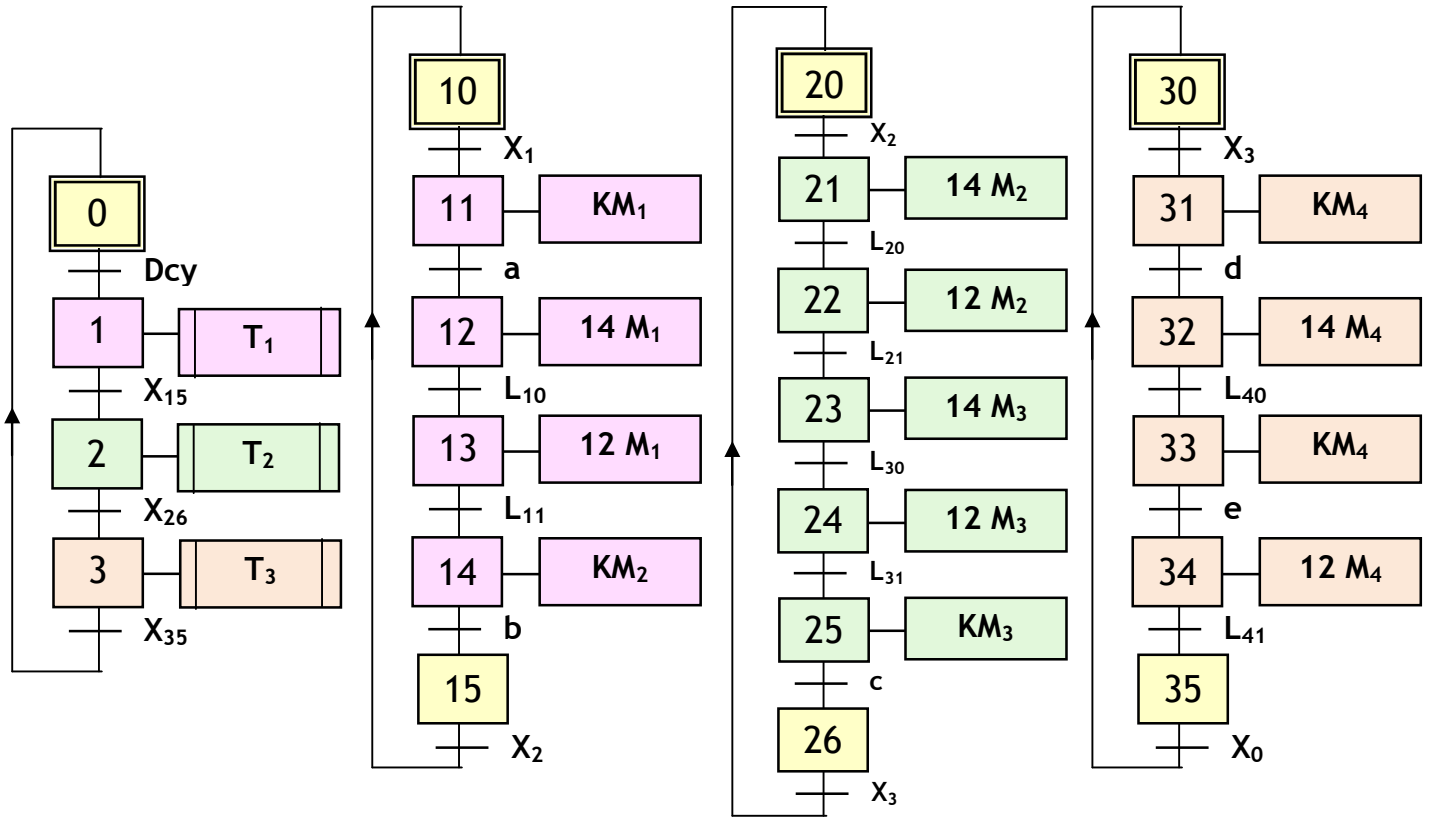
GRAFCET principal



GRAFCET de tâche SP

GRAFCET n°2 :

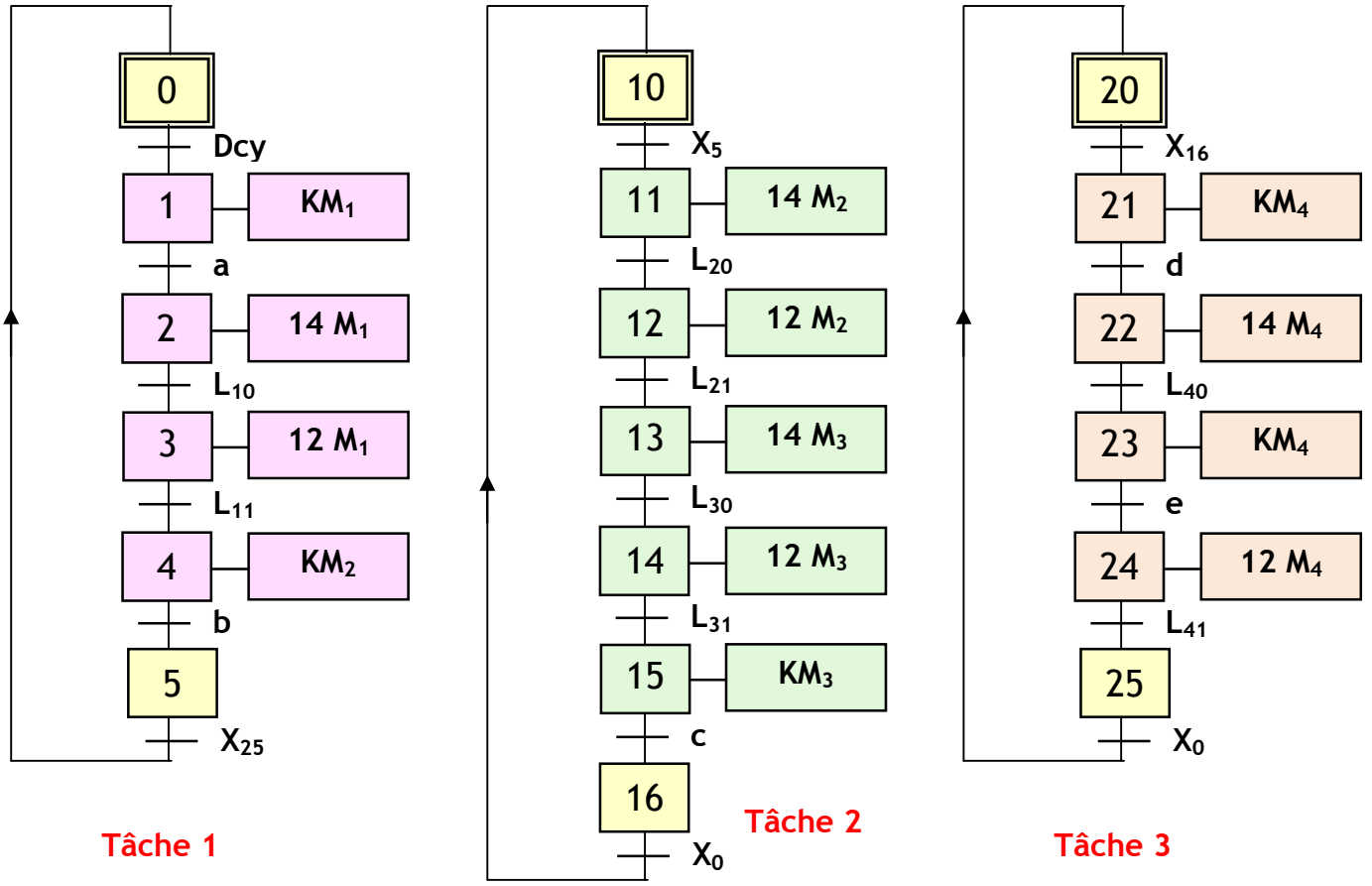
On veut réécrire le GRAFCET n°2 d'une manière simplifiée. Compléter le GRAFCET **principal** et les GRAFCETS de tâche **1, 2 et 3**.



GRAFCET principal **Tâche 1** **Tâche 2** **Tâche 3**

Coordination Horizontale :

On veut réécrire ce GRAFCET n°2 d'une manière simplifiée. Compléter les GRAFCETS **Tâche 1, 2 et 3**.



Tâche 1 **Tâche 2** **Tâche 3**

PROGRAMMATION DES PLD

1- Comparateur binaire 4 bits :

Description par équations :

```
MODULE comparateur
TITLE 'comparateur logique 4 bits'
DECLARATIONS
        Comparateur  device 'P16V8';                    // PLD à programmer
        B3, B2, B1, B0 pin 9, 8, 7, 6 ;                 // Variables d'entrée
        A3, A2, A1, A0 pin 5, 4, 3, 2 ;                 // Variables d'entrée
        A = [A3, A2, A1, A0] ;                          // Définition du bus A
        B = [B3, B2, B1, B0] ;                          // Définition du bus B
        S, I, E pin 15, 16, 17 istype 'com, buffer ';   // Variables de sortie
EQUATIONS
        S = A > B     ;
        E = A = = B   ;
        I = A < B     ;
TEST_VECTORS   ([A, B] -> [S, E, I]);
                        [5, 5]   -> [0, 1, 0];
                        [9, 12] -> [0, 0, 1];
                        [7, 6]   -> [1, 0, 0];
END comparateur
```

Utilisation de **When …. Then ….**

```
EQUATIONS
        When A > B then S = 1     ;
        When A = = Bthen E = 1    ;
        When A < B then I =1      ;
```

2- Multiplexeur 4 bits :

Description par équations :

A- Solution 1 :

```
MODULE Multiplexeur
TITLE 'Multiplexeur 4 bits'
DECLARATIONS
        Multiplexeur  device 'P16V8';          // PLD à programmer
        E3, E2, E1, E0 pin 9, 8, 7, 6 ;        // Variables d'entrée
        A1, A0 pin 5, 4;                       // Variables d'entrée
        S pin 16 istype 'com, buffer ';        // Variables de sortie
EQUATIONS
        S = E3 & A1 & A0 # E2 & A1 &! A0 # E1 &! A1 & A0 # E0 &! A1 &! A0 ;
TEST_VECTORS   ([E3, E2, E1, E0, A1, A0] -> [S]);
                        [0, 0, 0, 0, 0, 0] -> [0];
                        [0, 0, 0, 1, 0, 0] -> [1];
                        [0, 0, 0, 0, 0, 1] -> [0];
                        [0, 0, 1, 0, 0, 1] -> [1];
                        [0, 0, 0, 0, 1, 0] -> [0];
                        [0, 1, 0, 0, 1, 0] -> [1];
                        [0, 0, 0, 0, 1, 1] -> [0];
                        [1, 0, 0, 0, 1, 1] -> [1];
END Multiplexeur.
```

B- Solution 2 :

```
MODULE Multiplexeur
TITLE 'Multiplexeur 4 bits'
DECLARATIONS
        Multiplexeur  device 'P16V8';                  // PLD à programmer
        E3, E2, E1, E0, A1, A0 pin 9, 8, 7, 6, 5, 4 ;  // Variables d'entrée
        A = [A1, A0] ;                                 // Définition du bus A
        E = [E3, E2, E1, E0] ;                         // Définition du bus E
        S pin 16  istype 'com, buffer ';               // Variables de sortie
EQUATIONS
        When A = [0, 0] then S = E0; // ou When A = 0 then S = E0;
        When A = [0, 1] then S = E1; // ou When A = 1 then S = E1;
        When A = [1, 0] then S = E2; // ou When A = 2 then S = E2;
        When A = [1, 1] then S = E3; // ou When A = 3 then S = E3;

TEST_VECTORS   ([E3, E2, E1, E0, A1, A0] -> [S]); // ou ([E, A] -> [S]);
                 [0, 0, 0, 0, 0, 0] -> [0];       // ou   [0, 0] -> [0];
                 [0, 0, 0, 1, 0, 0] -> [1];       // ou   [1, 0] -> [1];
                 [0, 0, 0, 0, 0, 1] -> [0];       // ou   [0, 1] -> [0];
                 [0, 0, 1, 0, 0, 1] -> [1];       // ou   [2, 1] -> [1];
                 [0, 0, 0, 0, 1, 0] -> [0];       // ou   [0, 2] -> [0];
                 [0, 1, 0, 0, 1, 0] -> [1];       // ou   [4, 2] -> [1];
                 [0, 0, 0, 0, 1, 1] -> [0];       // ou   [0, 3] -> [0];
                 [1, 0, 0, 0, 1, 1] -> [1];       // ou   [8, 3] -> [1];
END Multiplexeur.
```

3- Démultiplexeur 4 bits :

3.1- Description par équations :

A- Solution 1 :

```
MODULE Demultiplexeur
TITLE 'Démultiplexeur 4 bits'
DECLARATIONS
        Demultiplexeur device 'P16V8';                          // PLD à programmer
        E, G, A1, A0 pin 7, 6, 5, 4;                            // Variables d'entrée
    S3, S2, S1, S0 pin 12, 13, 14, 15 istype 'com, buffer '; // Variables de sortie
EQUATIONS
        S3 = E & A1 & A0 &! G;
        S2 = E & A1 &! A0 &! G;
        S1 = E &! A1 & A0 &! G;
        S0 = E &! A1 &! A0 &! G;
TEST_VECTORS   ([G, E, A1, A0]      -> [S3, S2, S1, S0]);
                 [1,.X.,.X.,.X.]    -> [0, 0, 0, 0];
                 [0, 0, 0, 0]       -> [0, 0, 0, 0];
                 [0, 1, 0, 0]       -> [0, 0, 0, 1];
                 [0, 0, 0, 1]       -> [0, 0, 0, 0];
                 [0, 1, 0, 1]       -> [0, 0, 1, 0];
                 [0, 0, 1, 0]       -> [0, 0, 0, 0];
                 [0, 1, 1, 0]       -> [0, 1, 0, 0];
                 [0, 0, 1, 1]       -> [0, 0, 0, 0];
```

```
                    [0, 1, 1, 1]        -> [1, 0, 0, 0];
END Demultiplexeur.
```

B- Solution 2 :

```
MODULE Demultiplexeur
TITLE 'Demultiplexeur 4 bits'
DECLARATIONS
        Demultiplexeur  device 'P16V8';                          // PLD à programmer
        E, G, A1, A0 pin 7, 6, 5, 4;                             // Variables d'entrée
        S3, S2, S1, S0 pin 12, 13, 14, 15 istype 'com, buffer '; // Variables de sortie
        A = [A1, A0] ;                                           // Définition du bus A
        S = [S3, S2, S1, S0] ;                                   // Définition du bus S
EQUATIONS
        When (G == 0) & (A == 0) then S0 = E;
        When (G == 0) & (A == 1) then S1 = E;
        When (G == 0) & (A == 2) then S2 = E;
        When (G == 0) & (A == 3) then S3 = E;
TEST_VECTORS   ([G, E, A]       -> [S]);
                [1,.X.,.X.]     -> [0];
                [0, 0, 0]       -> [0];
                [0, 1, 0]       -> [1];
                [0, 0, 1]       -> [0];
                [0, 1, 1]       -> [2];
                [0, 0, 2]       -> [0];
                [0, 1, 2]       -> [4];
                [0, 0, 3]       -> [0];
                [0, 1, 3]       -> [8];
END Demultiplexeur.
```

3.2- Description par table de vérité :

```
MODULE Demultiplexeur
TITLE 'Demultiplexeur 4 bits'
DECLARATIONS
        Demultiplexeur device 'P16V8';                           // PLD à programmer
        E, G, A1, A0 pin 7, 6, 5, 4;                             // Variables d'entrée
        S3, S2, S1, S0 pin 12, 13, 14, 15 istype 'com, buffer '; // Variables de sortie
TRUTH_TABLE       ([G, E, A1, A0] -> [S3, S2, S1, S0]);
                [1,.X.,.X.,.X.] -> [0, 0, 0, 0];
                [0, 0, 0, 0]    -> [0, 0, 0, 0];
                [0, 0, 0, 1]    -> [0, 0, 0, 0];
                [0, 0, 1, 0]    -> [0, 0, 0, 0];
                [0, 0, 1, 1]    -> [0, 0, 0, 0];
                [0, 1, 0, 0]    -> [0, 0, 0, 1];
                [0, 1, 0, 1]    -> [0, 0, 1, 0];
                [0, 1, 1, 0]    -> [0, 1, 0, 0];
                [0, 1, 1, 1]    -> [1, 0, 0, 0];
TEST_VECTORS   ([G, E, A1, A0]          -> [S3, S2, S1, S0]);
                [1,.X.,.X.,.X.] -> [0, 0, 0, 0];
                [0, 0, 0, 0]    -> [0, 0, 0, 0];
                [0, 0, 0, 1]    -> [0, 0, 0, 0];
                [0, 0, 1, 0]    -> [0, 0, 0, 0];
```

Autre solution

```
([G, A1, A0] -> [S3, S2, S1, S0]);
[1,.X.,.X.]     -> [0, 0, 0, 0];
[0, 0, 0]       -> [0, 0, 0, E];
[0, 0, 1]       -> [0, 0, E, 0];
[0, 1, 0]       -> [0, E, 0, 0];
[0, 1, 1]       -> [E, 0, 0, 0];
```

```
                [0, 0, 1, 1]   -> [0, 0, 0, 0];
                [0, 1, 0, 0]   -> [0, 0, 0, 1];
                [0, 1, 0, 1]   -> [0, 0, 1, 0];
                [0, 1, 1, 0]   -> [0, 1, 0, 0];
                [0, 1, 1, 1]   -> [1, 0, 0, 0];
END Demultiplexeur.
```

4- Décodeur BCD 7 segments :

Description par table de vérité :

```
MODULE BCD
TITLE 'Décodeur BCD 7 segments '
DECLARATIONS
        BCD  device 'P16V8;                                  // PLD à programmer
        A, B, C, D pin   2, 3, 4, 5 ;                        // Variables d'entrée
        a, b, c, d, e, f, g pin 18, 17, 16, 15, 14, 13, 12;  // Variables sortie
        N = [D, C, B, A] ;                                   // Définition du bus N
TRUTH_TABLE     (N -> [a, b, c, d, e, f, g])
                0 -> [1, 1, 1, 1, 1, 1, 0] ; // affichage du 0 dans le cas
                1 -> [0, 1, 1, 0, 0, 0, 0] ; // d'un afficheur cathode commune
                2 -> [1, 1, 0, 1, 1, 0, 1] ;
                3 -> [1, 1, 1, 1, 0, 0, 1] ;
                4 -> [0, 1, 1, 0, 0, 1, 1] ;
                5 -> [1, 0, 1, 1, 0, 1, 1] ;
                6 -> [1, 0, 1, 1, 1, 1, 1] ;
                7 -> [1, 1, 1, 0, 0, 0, 0] ;
                8 -> [1, 1, 1, 1, 1, 1, 1] ;
                9 -> [1, 1, 1, 1, 0, 1, 1] ;
TEST_VECTORS    (N -> [a, b, c, d, e, f, g])
                0 -> [1, 1, 1, 1, 1, 1, 0] ;
                1 -> [0, 1, 1, 0, 0, 0, 0] ;
                2 -> [1, 1, 0, 1, 1, 0, 1] ;
                3 -> [1, 1, 1, 1, 0, 0, 1] ;
                4 -> [0, 1, 1, 0, 0, 1, 1] ;
                5 -> [1, 0, 1, 1, 0, 1, 1] ;
                6 -> [1, 0, 1, 1, 1, 1, 1] ;
                7 -> [1, 1, 1, 0, 0, 0, 0] ;
                8 -> [1, 1, 1, 1, 1, 1, 1] ;
                9 -> [1, 1, 1, 1, 0, 1, 1] ;
END BCD.
```

5- Compteur modulo 4 :

5.1- Description par diagramme d'état :

```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                    // PLD à programmer
        H, R pin  1, 2 ;                             // Variables d'entrée
        Q0, Q1 pin 15, 14 Istype 'reg, buffer ';     // Variables de sortie
EQUATIONS
```

```
        [Q1, Q0].clk = H ;
        [Q1, Q0].AR = ! R;
QSTATE      =       [Q1, Q0] ;
A           =       [0, 0] ;
B           =       [0, 1] ;
C           =       [1, 0];
D           =       [1, 1];
STATE_DIAGRAM   QSTATE
        State A:    GOTO B;
        State B:    GOTO C;
        State C:    GOTO D;
        State D:    GOTO A;
TEST_VECTORS    ([R, H,]        -> [Q1, Q0])
                 [0, .X.]       -> [0, 0] ;
                 [1, . C.]      -> [0, 1] ;
                 [1, . C.]      -> [1, 0] ;
                 [1, . C.]      -> [1, 1] ;
                 [1, . C.]      -> [0, 0] ;
END Compteur
```

5.2- Description par équation :

```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                   // PLD à programmer
        H, R pin  1, 2 ;                            // Variables d'entrée
        Q0, Q1 pin 15, 14 Istype 'reg, buffer ';    // Variables de sortie
        Q = [Q1,Q0] ; // Définition du bus Q
EQUATIONS
        Q.clk = H ;
        Q.AR = ! R;
        When R == 0 then Q := 0 ELSE Q := Q + 1 ;

TEST_VECTORS    ([R, H,]        -> [Q1, Q0])
                 [0, .X.]       -> [0, 0] ;
                 [1, . C.]      -> [0, 1] ;
                 [1, . C.]      -> [1, 0] ;
                 [1, . C.]      -> [1, 1] ;
                 [1, . C.]      -> [0, 0] ;
END Compteur
```

5.3- Description par Table de vérité :

```
MODULE Compteur
TITLE 'Compteur  modulo 4'
        Compteur device 'P16V8'
DECLARATIONS
        Compteur device 'P16V8' ;                   // PLD à programmer
        H, R pin  1, 2 ;                            // Variables d'entrée
        Q0, Q1 pin 15, 14 Istype 'reg, buffer ';    // Variables de sortie
EQUATIONS
        [Q1, Q0].clk = H ;
```

```
        [Q1, Q0].AR = ! R;
TEST_VECTORS    ([R, H,]      :> [Q1, Q0])
                 [0, .X.]     :> [0, 0] ;
                 [1, . C.]    :> [0, 1] ;
                 [1, . C.]    :> [1, 0] ;
                 [1, . C.]    :> [1, 1] ;
                 [1, . C.]    :> [0, 0] ;
TEST_VECTORS    ([R, H,]      -> [Q1, Q0])
                 [0, .X.]     -> [0, 0] ;
                 [1, . C.]    -> [0, 1] ;
                 [1, . C.]    -> [1, 0] ;
                 [1, . C.]    -> [1, 1] ;
                 [1, . C.]    -> [0, 0] ;
END Compteur
```

6- Compteur modulo 5 réversible :

6.1- Description par diagramme d'état :

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        Compteur device 'P16V8' ;                          // PLD à programmer
        H, R, Y pin  1, 2, 3 ;                             // Variables d'entrée
        Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer '; // Variables sortie
EQUATIONS
        [Q2, Q1, Q0].clk = H ;
        [Q2, Q1, Q0].AR = ! R;
QSTATE      =       [Q2, Q1, Q0];
A           =       [0, 0, 0];
B           =       [0, 0, 1];
C           =       [0, 1, 0];
D           =       [0, 1, 1];
E           =       [0, 1, 1];
STATE_DIAGRAM   QSTATE
        State A:    When Y = = 1 Then B Else E;
        State B:    When Y = = 1 Then C Else A;
        State C:    When Y = = 1 Then D Else B;
        State D:    When Y = = 1 Then E Else C;
        State E:    When Y = = 1 Then A Else D;
TEST_VECTORS    ([R, H, Y]      -> [Q2, Q1, Q0])
                 [0, .X., .X.]  -> [0, 0, 0] ;
                 [1, .C., 1]    -> [0, 0, 1] ;
                 [1, .C., 1]    -> [0, 1, 0] ;
                 [1, .C., 1]    -> [0, 1, 1] ;
                 [1, .C., 1]    -> [1, 0, 0] ;
                 [1, .C., 1]    -> [0, 0, 0] ;
                 [1, .C., 0]    -> [1, 0, 0] ;
                 [1, .C., 0]    -> [0, 1, 1] ;
                 [1, .C., 0]    -> [0, 1, 0] ;
                 [1, .C., 0]    -> [0, 0, 1] ;
                 [1, .C., 0]    -> [0, 0, 0] ;
END Compteur_reversible
```

6.2- Description par table de vérité :

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        Compteur device 'P16V8' ;                          // PLD à programmer
        H, R, Y pin  1, 2, 3 ;                             // Variables d'entrée
        Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer ';   // Variables sortie
EQUATIONS
        [Q2, Q1, Q0].clk = H ;
        [Q2, Q1, Q0].AR = ! R;
TRUTH_TABLE      ([R, Y, Q2, Q1, Q0] :> [Q2, Q1, Q0])
                 [0, .X., .X., .X., .X.] :> [0, 0, 0] ;
                 [1, 1, 0, 0, 0]  :> [0, 0, 1] ;
                 [1, 1, 0, 0, 1]  :> [0, 1, 0] ;
                 [1, 1, 0, 1, 0]  :> [0, 1, 1] ;
                 [1, 1, 0, 1, 1]  :> [1, 0, 0] ;
                 [1, 1, 1, 0, 0]  :> [0, 0, 0] ;
                 [1, 0, 0, 0, 0]  :> [1, 0, 0] ;
                 [1, 0, 1, 0, 0]  :> [0, 1, 1] ;
                 [1, 0, 0, 1, 1]  :> [0, 1, 0] ;
                 [1, 0, 0, 1, 0]  :> [0, 0, 1] ;
                 [1, 0, 0, 0, 1]  :> [0, 0, 0] ;
TEST_VECTORS     ([R, H, Y]      -> [Q2, Q1, Q0])
                  [0, .X., .X.]   -> [0, 0, 0] ;
                  [1, . C., 1]     -> [0, 0, 1] ;
                  [1, . C., 1]     -> [0, 1, 0] ;
                  [1, . C., 1]     -> [0, 1, 1] ;
                  [1, . C., 1]     -> [1, 0, 0] ;
                  [1, . C., 1]     -> [0, 0, 0] ;
                  [1, . C., 0]     -> [1, 0, 0] ;
                  [1, . C., 0]     -> [0, 1, 1] ;
                  [1, . C., 0]     -> [0, 1, 0] ;
                  [1, . C., 0]     -> [0, 0, 1] ;
                  [1, . C., 0]     -> [0, 0, 0] ;
END Compteur_reversible.
```

6.3- Description par équation :

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
        Compteur device 'P16V8' ;                          // PLD à programmer
        H, R, Y pin  1, 2, 3 ;                             // Variables d'entrée
        Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer ';   // Variables sortie
        Q = [Q2, Q1, Q0] ;                                 // Définition du bus Q
EQUATIONS
        Q.clk  = H ;
        Q.AR  = ! R;
        When R == 0 then Q := 0 Else When H == 1 then Q := Q + 1
                                     Else Q := Q - 1 ;
        When Q > 4 then Q := 0 ;
END Compteur_reversible.
```