LOGIQUE PROGRAMMÉE

CONTENU DU PROGRAMME

I- Automate Programmable Industriel (A.P.I)

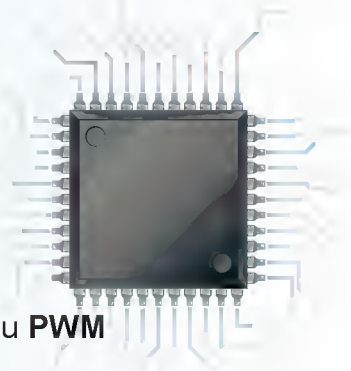
1- **Architecture**
2- **Identification des entrées et des sorties**
3- **Critères de choix d'un A.P.I**
4- **Programmation en mode lignes d'instructions**
5- **Transfert d'un programme vers A.P.I**

II- Microcontrôleurs

1- **Programme en langage évolué**
   a. Structure d'un programme
   b. Algorithme
   c. Instructions spécifiques au compilateur utilisé

2- **Notions d'interruption**
   a. Interruption sur l'entrée INT/RB0
   b. Interruption par changement de la combinaison binaire des broches RB4 à RB7

3- **Applications à base de PIC**
   a. GRAFCET
   b. Comptage
   c. Commande d'un afficheur LCD
   d. Gestion d'un clavier
   e. Conversion analogique numérique
   f. **M**odulation de **L**ageur d'**I**mplulsion **MLI** ou **PWM**



OBJECTIFS DU PROGRAMME

- OS A_{31} - **Identifier les éléments de dialogue d'un système automatisé piloté par A.P.I**
- OS A_{32} - **Traduire un grafcet en langage automate**
- OS A_{33} - **Transférer un programme vers un A.P.I**
- OS A_{34} - **Mettre en œuvre un système piloté par A.P.I**
- OS A_{35} - **Décrire le fonctionnement d'un système par algorithme**
- OS A_{36} - **Traduire un algorithme en programme en langage évolué**
- OS A_{37} - **Elaborer un programme spécifique à une application à base de microcontrôleurs**
- OS A_{38} - **Transférer un programme vers un microcontrôleur**

AUTOMATES PROGRAMMABLES INDUSTRIELS (A.P.I)

A. MISE EN SITUATION

Chaine de montage de véhicules

I- Introduction

Une ligne de montage ou chaîne de montage est un ensemble de postes de travail spécialisés disposés dans un ordre préétabli correspondant à la succession des opérations d'assemblage des composants d'un produit. Une ligne de montage se caractérise généralement par l'emploi d'un convoyeur mécanisé qui transporte le produit en cours de montage d'un poste à un autre. Ce sont les chaînes de convoyage qui lui ont donné le nom de «chaîne de montage» et lui ont consacré l'expression «travail à la chaîne».

Fig. 1

Dans la grande majorité des chaînes de montage actuelles, des robots ont remplacé les ouvriers.

Par l'adoption de ce dispositif, ce ne sont plus les ouvriers qui se déplacent, mais les pièces elles-mêmes; cela a pour concéquence une réduction des temps morts.


Fig. 2

Les lignes de montages se caractérisent par le degré d'automatisation, le nombre de postes et les délais.

Sur une ligne peu automatisée, le transfert des pièces peut se faire sur un simple convoyeur à rouleaux, tandis que sur une ligne fortement automatisée, le transfert est chronométré et paramétré.

À chaque poste, un robot, un ou plusieurs ouvriers réalisent une liste de tâches spécifiques.

La figure 3, représente l'allure d'une ligne de montage de voitures:

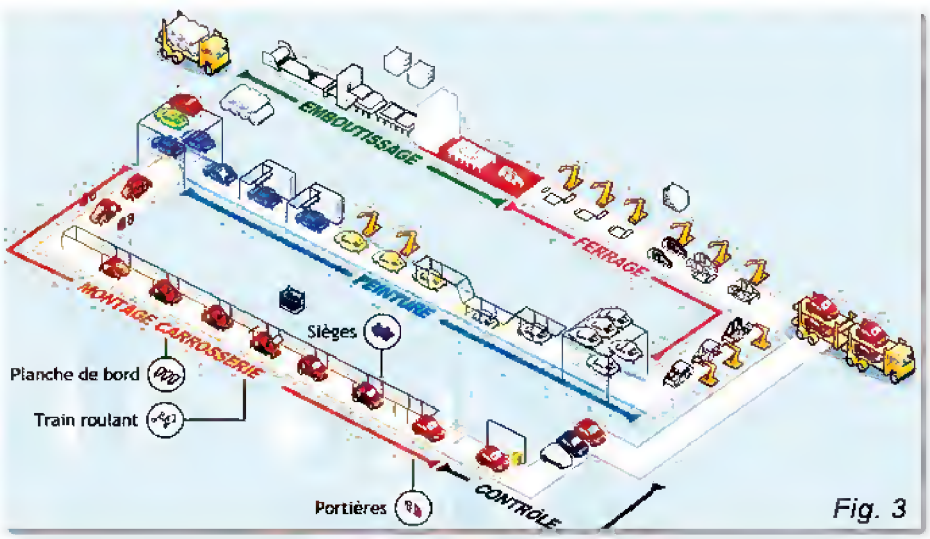
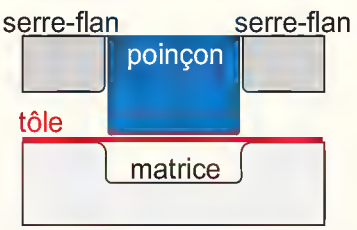
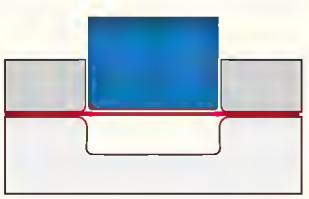


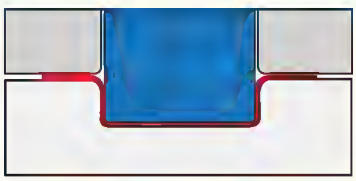
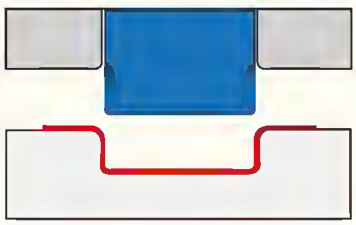
Fig. 3

Parmi les postes de travail équipant cette ligne de montage, on cite celui dédié à **l'emboutissage**. Ce poste permet d'obtenir, à partir d'une feuille de tôle plane et mince, un objet dont la forme n'est pas développable. L'ébauche en tôle est appelée «Becker», c'est la matière brute qui n'a pas encore été emboutie. La température de déformation se situe entre le tiers et la moitié de la température de fusion du matériau.

Fonctionnement

L'opération d'emboutissage typique (double-effet), peut être décrite comme suit :

Phase	Description	Croquis
Phase 1	Poinçon et serre-flan sont relevés. La tôle, préalablement graissée, est posée sur la matrice.	
Phase 2	Le serre-flan descend et vient appliquer une pression bien déterminée, afin de maintenir la tôle tout en lui permettant de glisser.	

Phase	Description	Croquis
Phase 3	Le poinçon descend et déforme la tôle de façon plastique en l'appliquant contre le fond de la matrice.	
Phase 4	Le poinçon revient à sa position initiale, suivi du serre-flan : la pièce conserve la forme acquise (limite d'élasticité dépassée).	

Sachant que chaque poste de travail a ses propres contraintes et exigences, la gestion d'un tel système avec les moyens classiques, peut s'avérer très difficile voire même impossible.

Problématique:

- Comment gérer l'ensemble de ces informations ?
- L'utilisation de la logique câblée est-elle fiable dans ce cas de figure ?
- Par quoi doit-on donc piloter ce type de système ?
- Comment mettre en œuvre ce composant de commande ?

B. AUTOMATE PROGRAMMABLE INDUSTRIEL (A.P.I)

NB : *Vu la diversité des automates équipant nos laboratoires et pour développer les différentes sections de cette partie des programmes, les auteurs ont opté pour le langage des automates Schneider. L'enseignant est donc appelé à adapter le contenu au matériel dont il dispose.*

I- Structure d'un système automatisé et outils d'analyse

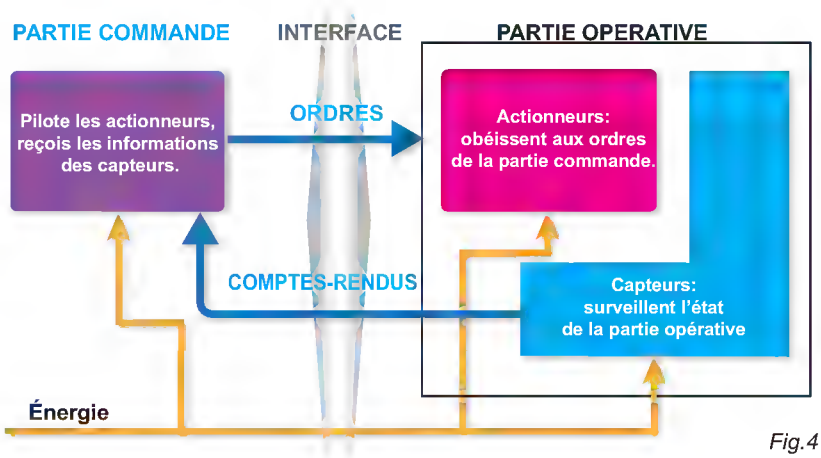


Fig.4

Pour matérialiser la partie commande, deux technologies sont disponibles ayant chacune ses avantages et ses inconvénients, à savoir la technologie câblée et la technologie programmée.

En fonction du résultat attendu, l'étude et l'analyse de cette partie, nécessite la mise en œuvre d'outils. Parmi ces outils, on cite le GRAFCET dédié à l'étude de l'évolution temporelle des systèmes automatisés.

Exemple : *Poste d'emboutissage*

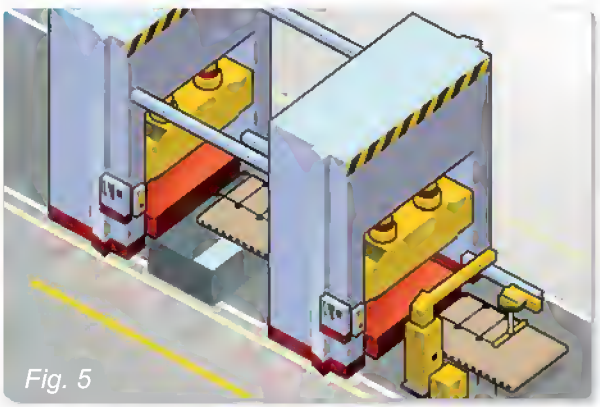


Fig. 5

Ce système, comme il est déjà cité dans la mise en situation permet de créer automatiquement une forme dans de la tôle en quatre étapes.

Dans l'étape 5 on procède au «détourage» de la pièce, c'est-à-dire à l'élimination des parties devenues inutiles, or cette étape a été omise car dans des cas, elle peut être réalisée manuellement.

Avec les choix technologiques suivants :

Fonction	Composants
Départ cycle	Bouton poussoir : **m**
Présence pièce	Détecteur : **S**
Déplacement serre-flan	Vérin à double effet : $\mathbf{C_1}$ Deux capteurs de position $\mathbf{I_{10}}$: position haute $\mathbf{I_{11}}$: position basse
Déplacement poinçon	Vérin à double effet : $\mathbf{C_2}$ Deux capteurs de position $\mathbf{I_{20}}$: position haute $\mathbf{I_{21}}$: position basse

L'analyse de l'évolution temporelle de ce système, nous amène à utiliser le GRAFCET comme outil. On va se contenter du GRAFCET PC représenté à la page suivante.

La matérialisation de ce GRAFCET, peut être assurée soit par une logique câblée mettant en œuvre un séquenceur électronique à base de bascules ou pneumatique à base de modules étapes, ou par une logique programmée par micro-ordinateur ou circuits spécialisés tel qu'un microcontrôleur associés à des cartes d'interface adéquates ou par ***Automate Programmable Industriel (A.P.I)***. L'avantage de la logique programmée réside dans sa puissante flexibilité.

GRAFCET PC

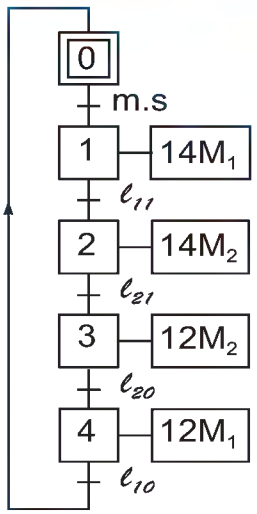


Fig. 6

II- L'automate programmable

1- Définition

Un automate programmable industriel, ou A.P.I, est un dispositif électronique programmable destiné à la commande de processus industriels par un traitement séquentiel. Il envoie des ordres vers les pré-actionneurs à partir de données d'entrées, de consignes et d'un programme écrit dans un langage adapté.

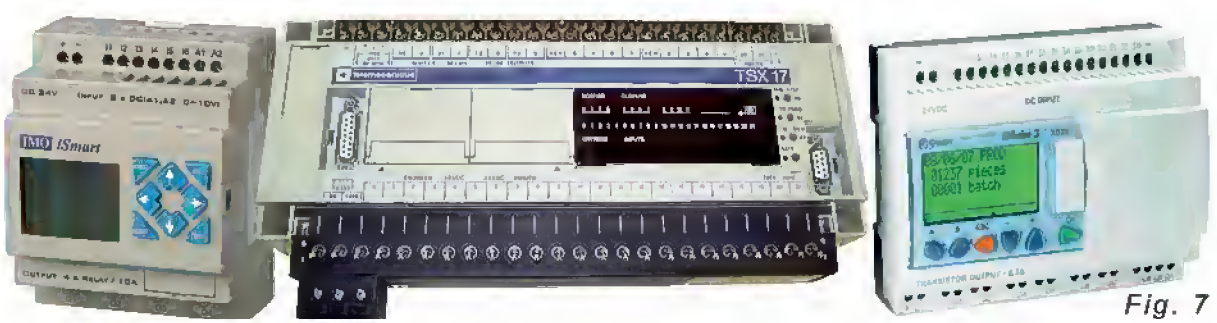


Fig. 7

2- Place de l'automate dans un système automatisé

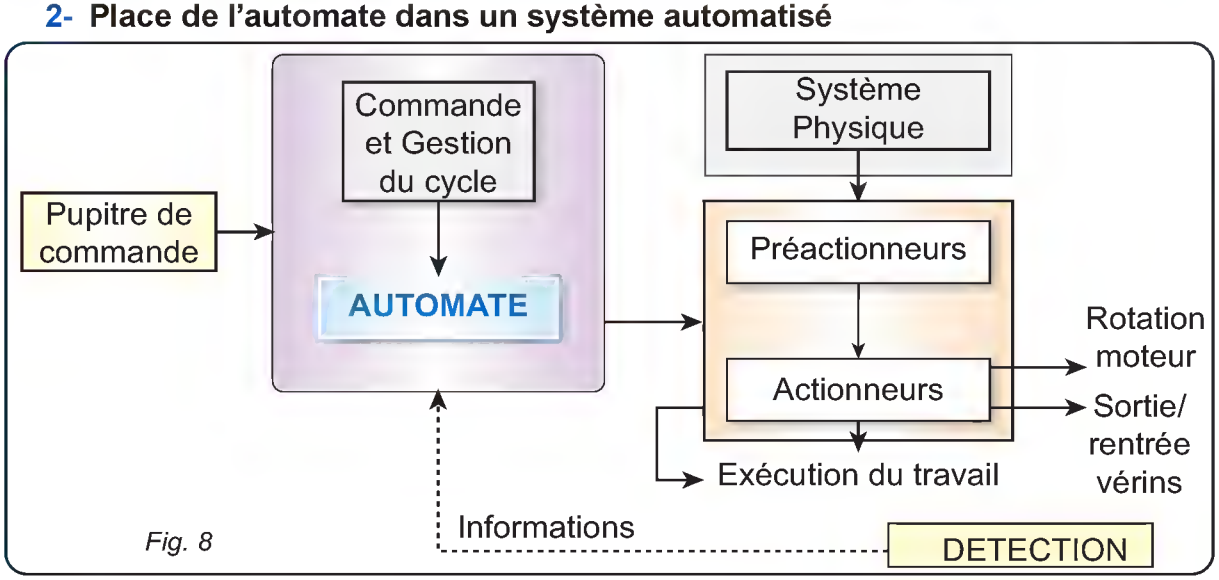
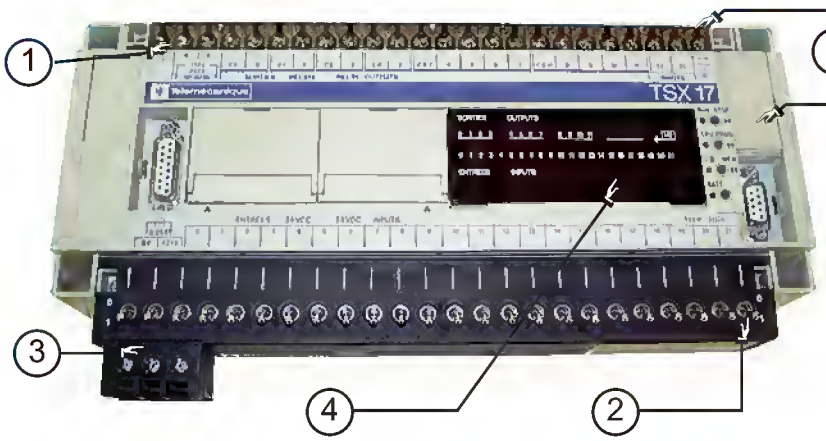


Fig. 8

3- Structure externe d'un A.P.I



1. Alimentation
2. entrées-sorties (E/S)
3. Alimentation capteurs 24 V
4. Visualisation des (E/S)
5. Extension du bus (E/S)

Fig. 9

4- Structure interne d'un A.P.I

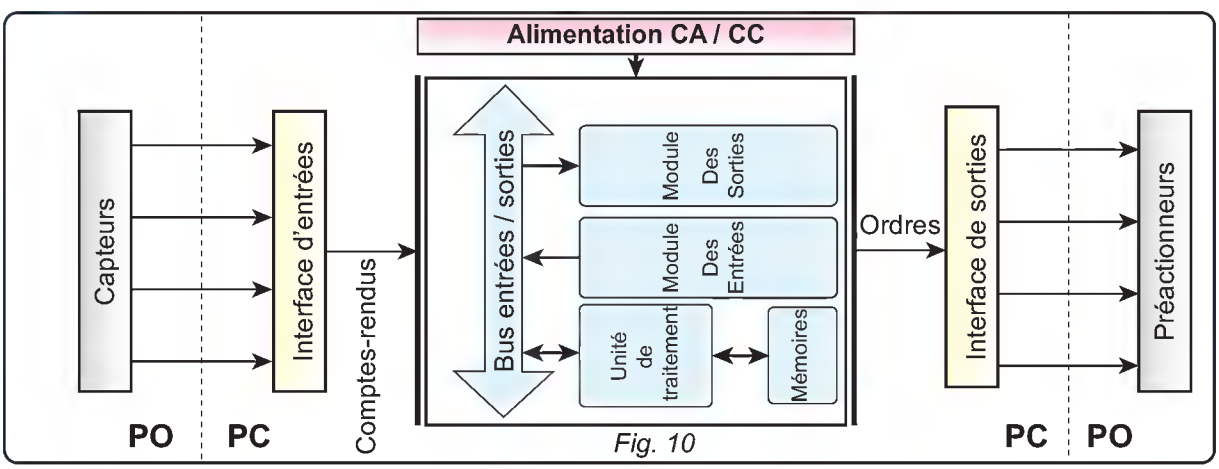


Fig. 10

Quel que soit le type d'automate, ce dernier est composé d'au moins des parties suivantes :

- Mémoire : conçue pour recevoir, gérer, stocker des informations issues des différentes composantes du système qui sont :
  - le terminal de programmation : introduction du programme ;
  - le processeur qui gère et exécute le programme.

  Elle reçoit également des informations en provenance des capteurs. Il existe dans les automates plusieurs types de mémoires qui remplissent des fonctions différentes (RAM, EEPROM, EPROM).
- L'alimentation : permet de fournir l'énergie nécessaire au fonctionnement de l'automate et de l'ensemble de ses cartes.
- Le processeur : représente le cerveau de l'automate. Il traite les données et élabore les ordres de commande.

Les modules d'entrées : ce sont des cartes spécialisées capables de véhiculer en toute sécurité vers l'automate les signaux issus des différents composants d'entrée (capteur, bouton poussoir, clavier, etc...). ces signaux, peuvent être du type tout ou rien, analogique ou numérique.

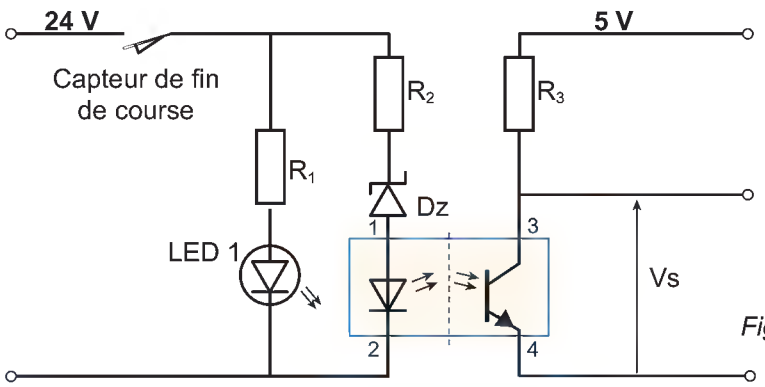


Fig. 11

Les modules de sorties : ce sont aussi des cartes spécialisées, capables de commander en toute sécurité les différents actionneurs par le biais de pré-actionneurs adéquats. Cette commande peut être du type tout ou rien, analogique ou numérique.

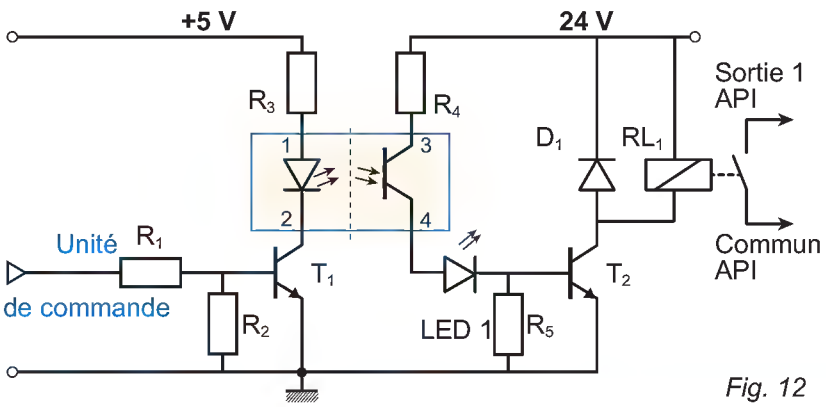
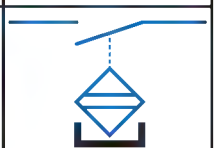
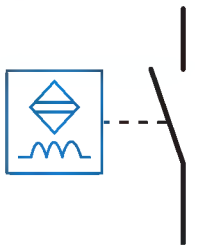
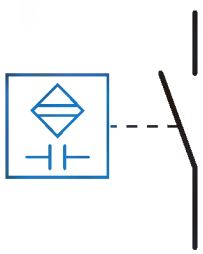
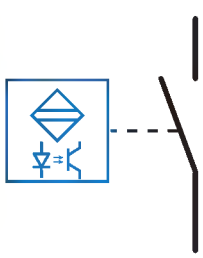


Fig. 12

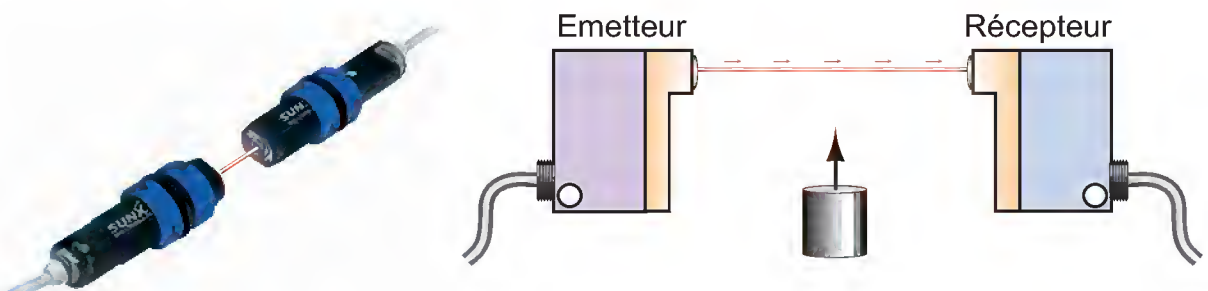
Aperçu sur les capteurs usuels type T.O.R

Désignation	Rôle	Forme commerciale	Symbole
Détecteur de position (fin de course)	Action en cas de contact direct avec l'objet		
Détecteur à lame souple (I.L.S)	Action sous l'effet d'un champ magnétique		

Désignation	Rôle	Forme commerciale	Symbole
Détecteur de proximité inductif	Détection des matériaux conducteurs sans pour autant être en contact direct avec l'objet		
Détecteur de proximité capacitif	Détection des matériaux conducteurs ou isolants sans pour autant être en contact direct avec l'objet		
Détecteur de proximité photoélectrique	Détection lors de la coupure d'un faisceau lumineux par l'objet		

NB : *pour le détecteur photo-électrique, trois types sont disponibles:*

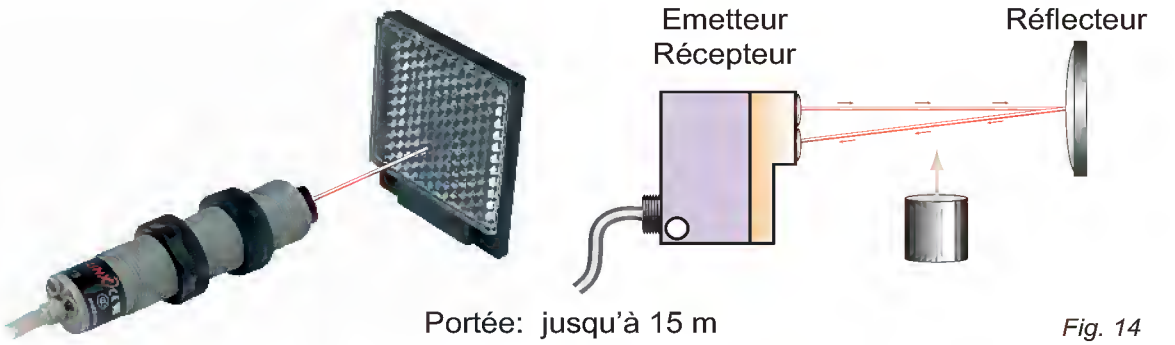
- **Type barrage:** émetteur et récepteur dans deux boitiers différents;



- Portée jusqu'à 50 m. Faisceau laser jusqu'à 100m.
- Détection lors de la coupure du faisceau.

Fig. 13

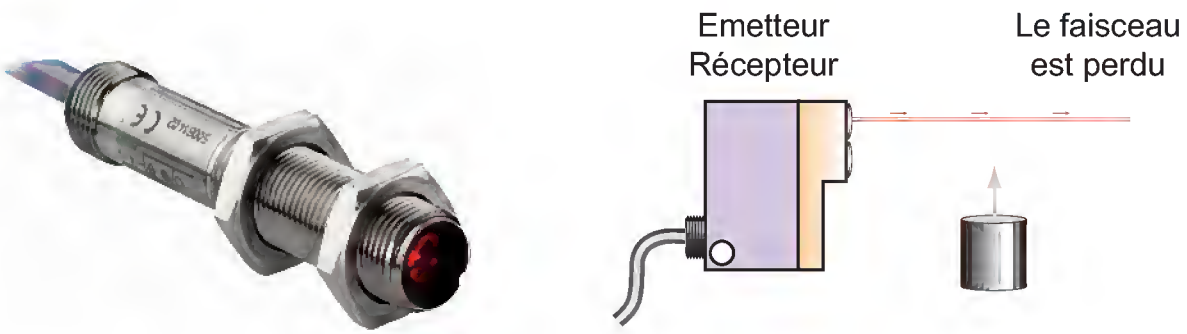
- **Type reflex :** émetteur et récepteur dans le même boitier. Faisceau réfléchi par réflecteur. Détection lors de la coupure du faisceau par l'objet.



Portée: jusqu'à 15 m

Fig. 14

- **Type proximité :** émetteur et récepteur dans le même boitier. Faisceau réfléchi par l'objet. Détection lorsque le faisceau est renvoyé par l'objet.



Portée: jusqu'à 2 m

Fig. 15

5- Critères de choix d'un A.P.I

- Nombre d'entrées / sorties : le nombre de cartes peut avoir une incidence sur le nombre de racks dès que le nombre d'entrées / sorties nécessaires devient élevé.
- Type de processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur.
- Fonctions ou modules spéciaux en option.
- Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (A.P.I, supervision, etc. ...).

6- Domaine d'emploi :

On utilise les A.P.I dans tous les secteurs industriels pour la commande des machines (convoyage, emballage ...) ou des chaînes de production (automobile, agroalimentaire, ...). Ils peuvent également assurer des fonctions de régulation de processus (métallurgie, chimie ...). Ils sont de plus en plus utilisés dans le domaine du bâtiment (tertiaire et industriel) pour la gestion du chauffage, de l'éclairage, de la sécurité ou des alarmes.

Ils conviennent parfaitement pour des systèmes de sécurité ferroviaire, la gestion des feux de croisement, des ascenseurs, des commandes de chaînes de production ou tout autre type d'activité.

III- Langages de programmation

La programmation d'un A.P.I consiste à traduire dans le langage spécifique à l'automate, les équations de fonctionnement du système à automatiser.

L'opération de programmation peut être assurée par une console dédiée ou par micro-ordinateur équipé d'un logiciel approprié.

Parmi les langages usuels, on cite :

- langage à contacts (**Ladder Diagram : LD**);
- langage Liste d'instructions (**Instruction List : IL**);
- langage GRAFCET (**Sequential Function Chart : SFC**).

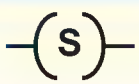
1- Langage à contacts (LD)

a. Description :

Ce type de langage repose sur une représentation graphique des conditions qui assurent l'activation d'une variable. Le langage **LD** est une succession de «réseaux de contacts» véhiculant des informations logiques depuis les entrées vers les sorties. C'est une simple traduction des circuits de commande électriques, seuls les symboles changent.

b. Principaux symboles

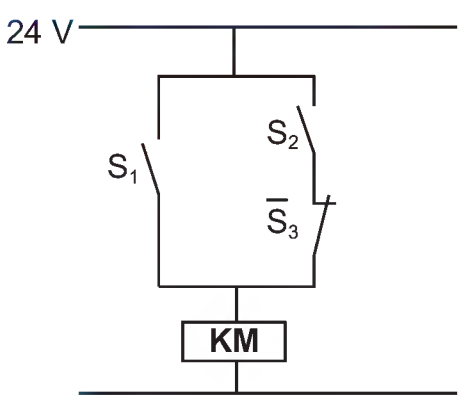
Symbole	Désignation	Fonction	Equivalent électrique
	Contact à fermeture	contact passant quand il est actionné	
	Contact à ouverture	contact passant quand il n'est pas actionné	
	Connexion horizontale	permet de relier les éléments action série	
	Connexion verticale	permet de relier les éléments action en parallèle	
	Bobine directe	la sortie prend la valeur du résultat logique	

Symbole	Désignation	Fonction	Equivalent électrique
	bobine inverse	la sortie prend la valeur complémentée du résultat logique	
	bobine d'enclenchement	la sortie est mise à 1 et garde cet état	
	bobine de déclenchement	la sortie est mise à 0 et garde cet état	

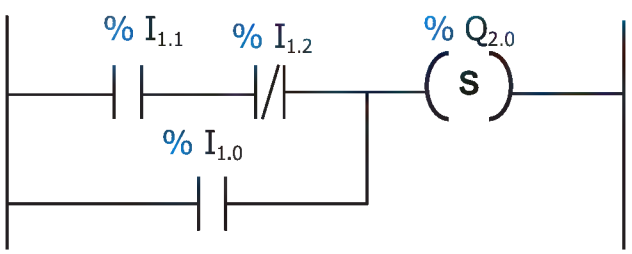
c. Exemple :

Traduction d'un schéma structurel ou à contacts en langage **LD**.

SCHÉMA STRUCTUREL



PROGRAMME EN **LD**



Avec les affectations suivantes :
- KM : %Q2.0
- S1 : %I1.0
- S2 : %I1.1
- S3 : %I1.2

Fig. 16

2- Langage Liste d'instructions (Instruction List)

a. Description :

L'**IL** est un langage dans lequel toutes les opérations sont décrites par des instructions mnémoniques. Ce n'est pas un langage graphique.

Le langage liste d'instruction permet de transcrire sous forme de liste :

- un schéma à contact ;
- un logigramme, des équations booléennes ;
- un grafcet.

Il réalise aussi des fonctions d'automatisme telles que temporisation, comptage, etc...

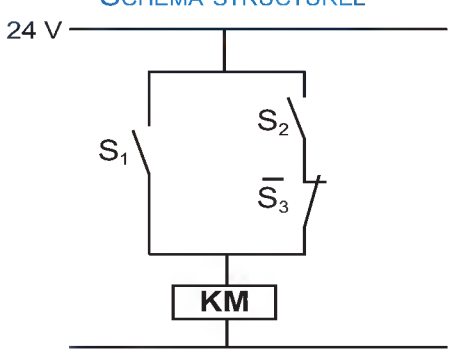
Le tableau suivant présente une liste de quelques commandes de ce langage :

Instructions de test	
Instruction	**Fonction**
LD	Lire une entrée ou une variable interne
LDN	Lire le complément d'une entrée ou d'une variable interne
AND	ET logique entre le résultat de l'instruction précédente et l'état de l'opérande
ANDN	ET logique entre le résultat de l'instruction précédente et l'état complément de l'opérande
OR	OU logique entre le résultat de l'instruction précédente et l'état de l'opérande
ORN	OU logique entre le résultat de l'instruction précédente et l'état complément de l'opérande
XOR XORN	OU exclusif / Coïncidence
Instructions d'action	
Instruction	**Fonction**
N	Complément du résultat de l'instruction précédente
ST	L'opérande associé prend la valeur du résultat de la zone test
STN	L'opérande associé prend la valeur complément du résultat de la zone test
S	L'opérande associé est mis à 1 lorsque le résultat de la zone test est à 1
R	L'opérande associé est mis à 0 lorsque le résultat de la zone test est à 1

NB: *L'adresse ou le code opérande est précédé de %*

Exemple : Traduction du même schéma structurel que celui de la page précédente en un programme en liste d'instruction.

SCHÉMA STRUCTUREL



PROGRAMME IL

Avec les affectations suivantes :

KM :%Q2.0
S1 : %I1.0
S2 : %I1.1
S3 : %I1.2

On obtient le programme ci-dessous :

```
LD %I1.1
ANDN %I1.2
OR %I1.0
ST %Q2.0
```

Fig. 17

3- Langage GRAFCET (Sequential Function Chart : SFC)

a. Description :

Le **SFC** est un langage graphique. Par abus de langage, on l'appelle aussi langage GRAFCET.

Pour éditer un programme **SFC**, il faut suivre les étapes suivantes :

- Construire graphiquement le GRAFCET.
- Traduire les réceptivités dans le langage **IL** ou **LD**.
- Traduire les actions dans le langage **IL** ou **LD.**

NB : *dans ce type de langage, l'activation et la désactivation des étapes se fait automatiquement.*

IV- Exemple de branchement: TSX 17-20 de chez Télémécanique

1- Présentation

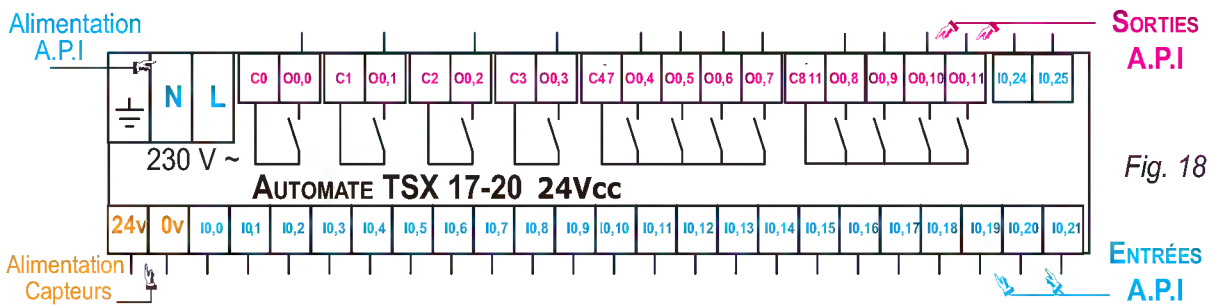


Fig. 18

2- Câblage des entrées/sorties

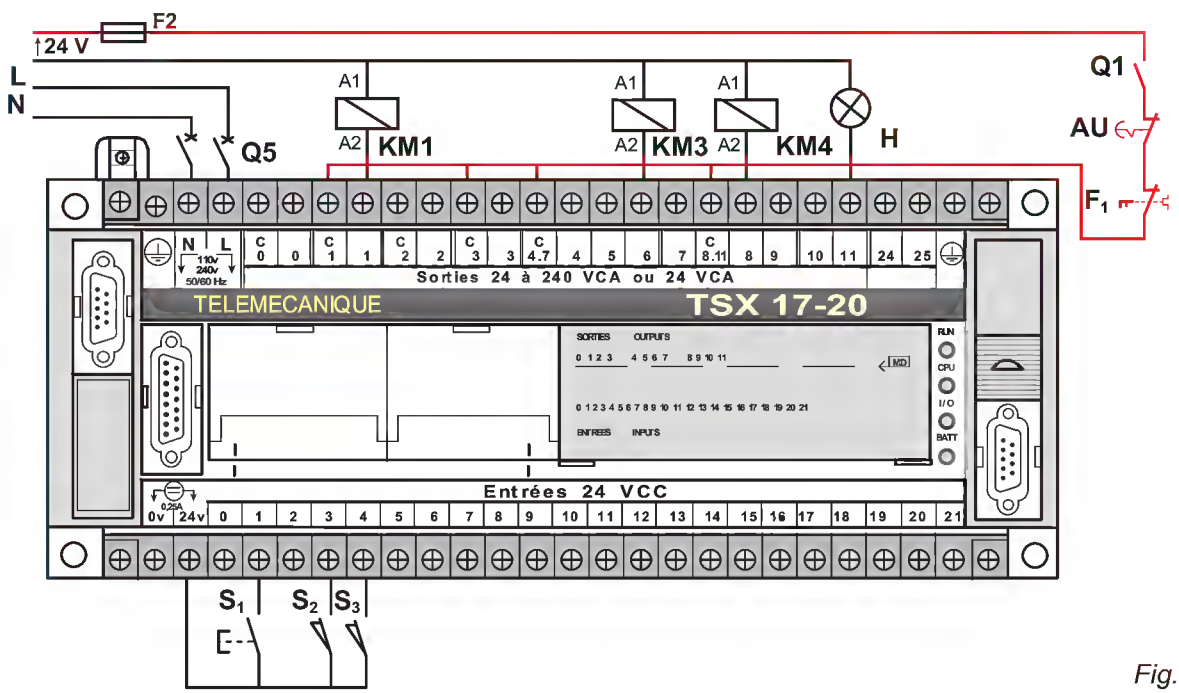


Fig. 19

3- Mise en œuvre d'une application

Comme toute application programmable, la mise en œuvre d'une solution à base d'A.P.I doit transiter au moins par les 4 étapes suivantes :

- Description du système.
- Choix de l'A.P.I.
- Rédaction du programme.
- Transfert du programme.

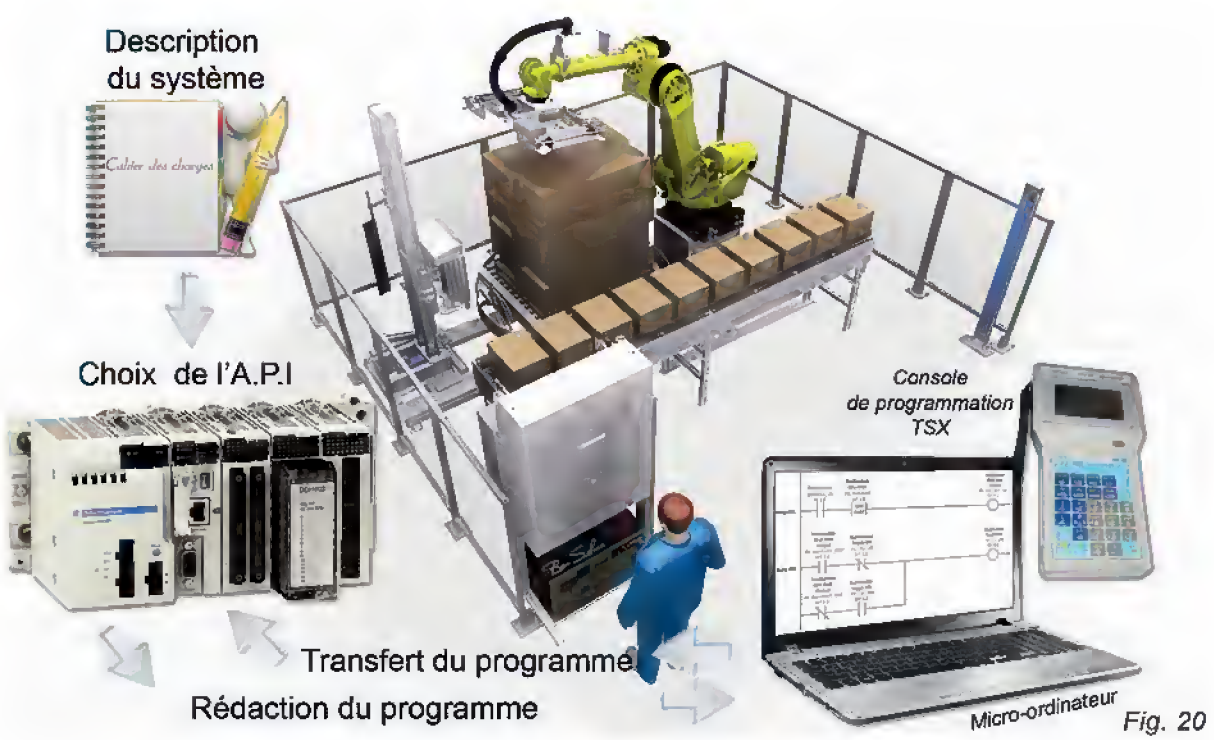


Fig. 20

C. RÉSUMÉ

La programmation du GRAFCET en langage LD, consiste à associer à chaque étape «i» du GRAFCET un bit interne de l'A.P.I «Xi».

Le programme final est donc constitué de deux traitements :

- Traitement séquentiel : cette partie du programme traduit l'évolution séquentielle des étapes en calculant l'état des bits internes «Xi» représentant les étapes.
- Traitement postérieur : cette partie décrit l'état des sorties.

Les différents types de capteurs

- Capteur tout ou rien (TOR) : la sortie présente un niveau bas et un niveau haut.

- Capteur analogique : les informations acquises par le capteur sont délivrées dans la même forme (température, pression, etc....)

- Capteur numérique : les informations acquises sont délivrées dans la même forme et peuvent être traitées directement par le processeur.

Le capteur inductif détecte les pièces en matériaux conducteurs.
Le capteur capacitif détecte les pièces conductrices ou isolantes.

Les langages de programmation usuels des A.P.I

- Langage Ladder diagram (**LD**).
- Langage liste d'instruction (**IL**).
- Langage Sequential Function Chart (**SFC**).

Chaque constructeur a sa propre syntaxe. Il est donc impératif de revenir au document constructeur.

La programmation de l'A.P.I, peut être assurée par une console dédiée ou par micro-ordinateur muni du logiciel approprié.
Les entrées et les sorties de l'A.P.I, sont isolées galvaniquement du monde extérieur.

D. EVALUATION

I- Contrôle de connaissances

1- l'abréviation «A.P.I.» signifie Automate Programmable Industriel.
b. Vrai
c. Faux

2- Les automates programmables interviennent dans un système automatisé en tant que (qu') :
a. Actionneur
b. Pré-actionneurs
c. Capteur numérique
d. Unité de traitement logique

3- La fonction d'un A.P.I est de (d') :
a. Acquérir les informations relatives aux grandeurs physiques externes et de commander des actionneurs.
b. Recevoir les informations relatives à l'état du système et de commander des actionneurs.
c. Recevoir les informations relatives à l'état du système et de commander des pré-actionneurs.

4- Le microprocesseur de l'A.P.I a pour rôle de :
a. Traiter des fonctions logiques
b. Traiter des fonctions logiques et séquentielles
c. Réaliser des fonctions logiques et séquentielles à partir d'un programme contenu dans sa mémoire.

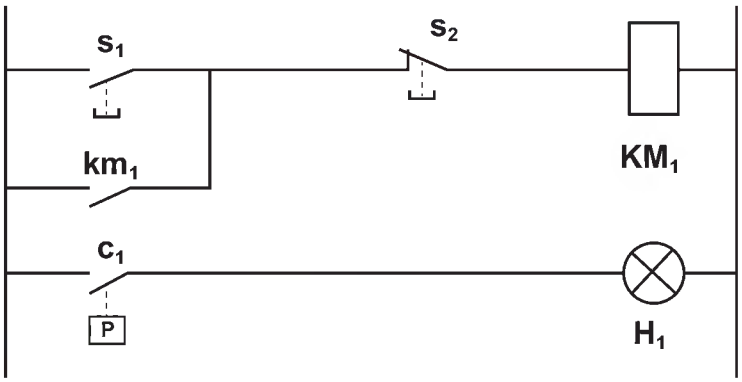
5- Le rôle de l'interface d'entrée dans un A.P.I est de :
a. Recevoir l'information envoyée par les capteurs ou le pupitre de commande.
b. Mettre en forme l'information reçue.
c. Isoler électriquement les circuits de puissance et de commande.
d. Réaliser une isolation galvanique entre le circuit extérieur et l'A.P.I.

6- Le rôle de l'interface de sortie dans un A.P.I est de :
a. Assurer une isolation galvanique entre le circuit extérieur et l'A.P.I.
b. Mettre en forme le signal de sortie et l'adapter au circuit de commande.
c. Commander directement les actionneurs de la partie opérative.
d. Commander les pré-actionneurs de la partie commande.

II- Exercice résolu

Soit le schéma structurel suivant:

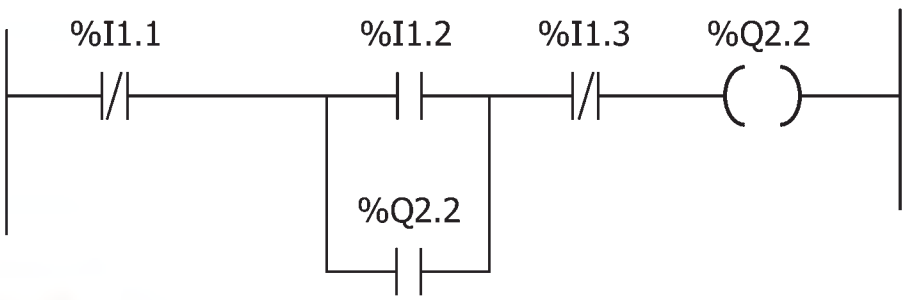
Traduire ce schéma en un programme en langage mnémonique (**IL**) relatif à un automate TSX 3722.



III- Exercices à résoudre

EXERCICE N°1

Traduire le programme LD suivant en son équivalent en liste d'instruction pour un automate TSX 3722.



EXERCICE N°2

Traduire le programme en langage **IL** suivant en son équivalent en langage **LD**.

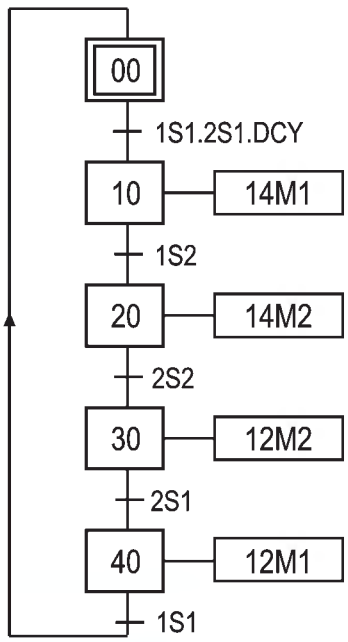
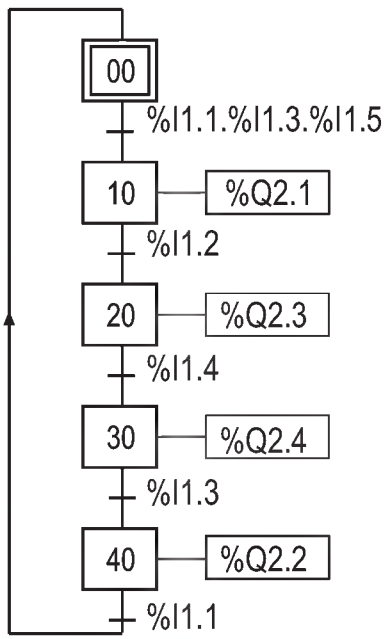
Instruction	Opérande	Commentaire
LD	%I1.1	Lire l'entrée.
AND (	%I1.3	Exécuter un ET, on imbrique une parenthèse.
OR (	%I1.2	Exécuter un OU avec la ligne précédente.
AND	%Q2.2	Exécuter un ET avec la ligne précédente.
)		Fermer la 1ère parenthèse.
)		Fermer la 2ème parenthèse.
ST	%Q2.2	Activer la sortie.

EXERCICE N°3

Traduire le grafcet PC du poste d'emboutissage en un programme type **IL**.

EXERCICE N°4

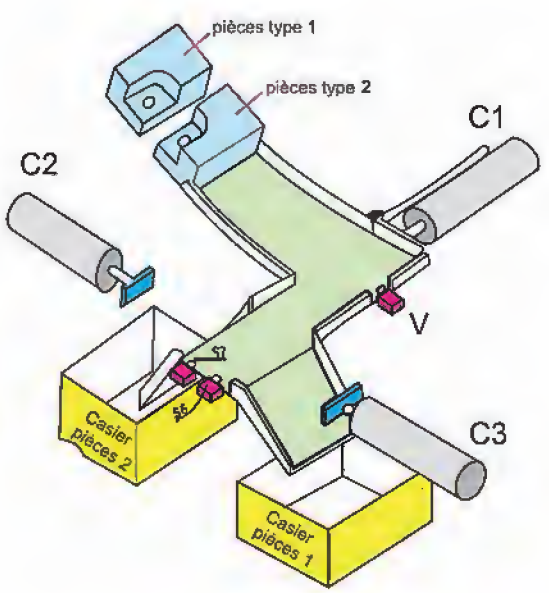
L'analyse d'une application mettant en œuvre un automate (TSX 3722) a abouti aux deux grafcet ci-dessous. Ecrire le programme A.P.I correspondant en langage IL.

Grafcet PC	Grafcet codé Automate TSX
	

EXERCICE N°5 :

Un dispositif de tri de pièces, doit permettre l'aiguillage de deux types de pièces vers deux caisses différentes.

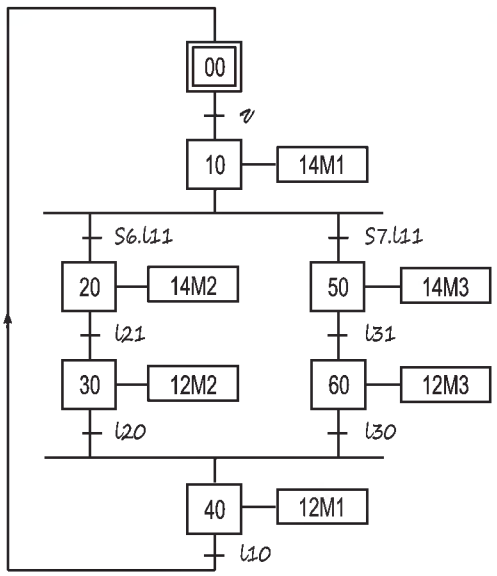
Les pièces arrivant par gravité dans un ordre quelconque, sont détectées par les capteurs **S6** et **S7** puis dirigées en fonction de leur type vers la caisse adéquate.



NB :

- **S6 :** pièce type 1.
- **S7 :** pièce type 2.
- **V :** capteur présence pièce.

On donne le grafcet de point de vue PC gérant ce dispositif



Travail demandé

Sachant que l'automate gérant ce système est de type TSX 3722, on vous demande:

1- Dresser le tableau des affectations correspondant.

2- Etablir le programme en liste d'instructions.

IV- Correction des exercices

Préparatif : Table d'affectations

E/S Système	Désignation	Affectation
S_1	Bouton marche	%I1.0
S_2	Bouton arrêt	%I1.1
C_1	Capteur pression	%I1.2
KM_1	Contacteur pompe	%Q2.0
H_1	Voyant pression	%Q2.1

Programme **IL**

Instruction	Opérande	Commentaire
LD	**%I1.0**	**Tester** le bouton marche **S_1**
OR	**%Q2.0**	Exécuter un **OU** avec le contact **KM_1**
ANDN	**%I1.1**	Exécuter un **ET** avec le bouton arrêt **S_2**
ST	**%Q2.0**	**Activer** la sortie contacteur pompe **KM_1**
LD	**%I1.2**	**Tester** le capteur de pression **C_1**
ST	**%Q2.1**	**Activer** la sortie voyant **H_1**

LES MICROCONTRÔLEURS

A. MISE EN SITUATION

Un robot aspirateur est un aspirateur capable de réaliser le travail de nettoyage de manière autonome, sans intervention de l'être humain.

Fig. 1

C'est en 2009 que s'ouvre le principal marché de robots domestiques dans le monde.

Durant cette période, une société qui fabrique les robots aspirateurs a vendu plus de cinq millions d'exemplaires.

Les principaux modes de fonctionnements de ces robots sont les suivants :

- **aléatoire:** le robot se déplace d'une façon aléatoire en fonction de ce qu'il perçoit de son environnement;
- **contour:** le robot contourne les obstacles et les murs qu'il rencontre;
- **spirale/polygonal:** le robot effectue une spirale ou un polygone à un endroit particulier;
- **aller-retour:** le robot fait un balayage de la zone à nettoyer;
- **mixte:** le robot alterne les modes de fonctionnement (ligne droite, spirale, contour…).

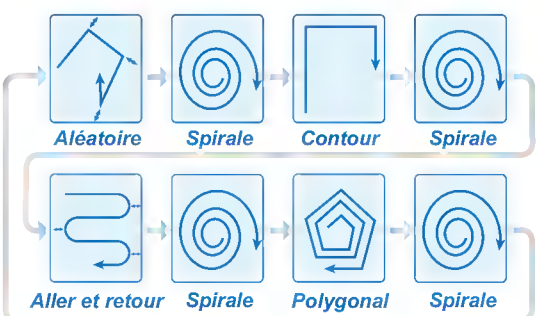


Fig. 2

Plusieurs autres fonctionnalités et accessoires peuvent exister suivant le modèle et la marque tel(le) que :

- un minuteur pour une mise en marche à une heure prédéfinie;
- un afficheur LCD et un clavier incorporé pour la gestion du robot;
- des murs virtuels pour délimiter les surfaces à nettoyer;
- une lampe «ultraviolet» pour stériliser les sols;
- un capteur de température;

Fig. 3

- des détecteurs d'obstacles à infrarouge et à ultrason;
- une station de recharge où l'aspirateur peut se brancher une fois son travail terminé ou sa charge a atteint un seuil minimal;
- une télécommande;
- une caméra pour reconnaître les obstacles;
- un capteur à poussières.

Pour détecter un obstacle, le robot utilise un pare-choc classique. Dès que le pare-choc touche quelque chose le robot s'arrête et repart dans une autre direction.

Il possède également un détecteur infrarouge d'obstacle qui vient rendre la détection d'obstacle plus pertinente. Le détecteur infrarouge identifiant un obstacle indique au robot de réduire la vitesse de déplacement jusqu'à ce que l'objet ne soit touché en douceur par le pare-choc. Cela permet d'apporter de la finesse et de la douceur dans le comportement mais participe également à la qualité du nettoyage aux abords de ces obstacles.

Fig. 4

Problématique:

- Quel composant électronique choisir pour gérer ce robot ?
- Comment traduire le fonctionnement de ce robot de manière structurée ?
- Quelle solution adoptée pour mettre en œuvre les différents constituants de ce robot ?

B. LES MICROCONTRÔLEURS

I- Programme en langage évolué

1- Introduction

Un langage évolué de programmation est une notation conventionnelle destinée à formuler des algorithmes afin de produire des programmes informatiques qui les traduisent.

De manière similaire à une langue naturelle, un langage de programmation évolué est structuré, c'est à dire qu'il compte un alphabet, un vocabulaire, des règles de grammaire, et des significations. Un langage de programmation évolué est mis en oeuvre par un traducteur automatique appelé compilateur ou interpréteur.

Le compilateur est un programme informatique qui transforme un code source écrit dans un langage de programmation évolué en un code cible directement exécutable par un processeur tel que le microcontrôleur, c'est à dire un programme en langage machine (fichier hexadécimal **.hex**).

L'écriture du programme ainsi que sa mise au point sont soumis à une démarche particulière, résumée par le graphique suivant:

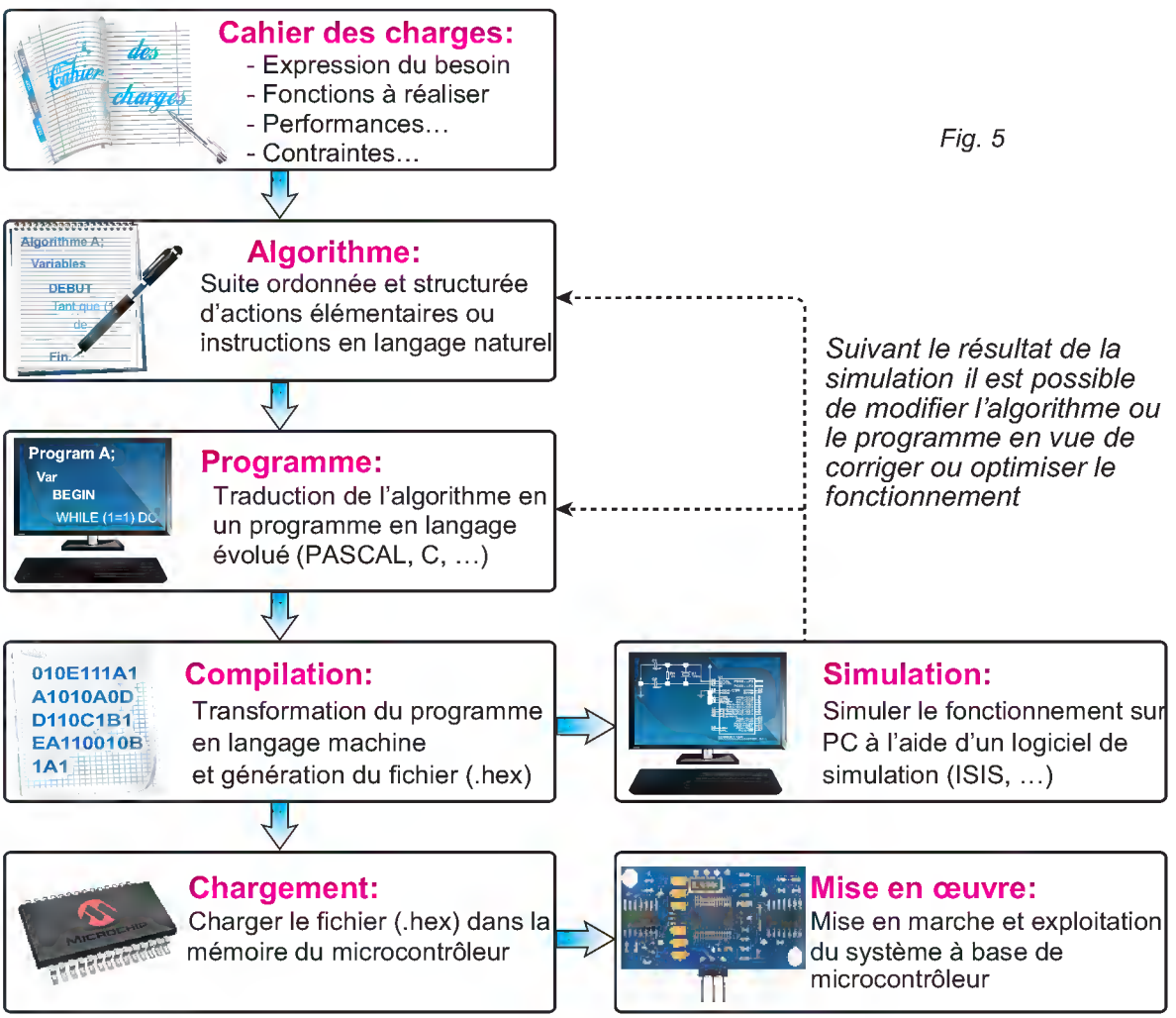


Fig. 5

Il faut traduire le cahier des charges en une suite ordonnée d'opérations que doit réaliser le microcontrôleur, cette suite d'opérations est décomposée en actions élémentaires ou instructions c'est **l'Algorithme**. En fin il suffit de traduire cet algorithme en un langage évolué tel que le langage PASCAL ou le langage C.

Dans la suite du cours on s'intéressera au langage PASCAL *(Compilateur Mikropascal pro de Mikroelektronika)*.

2- Structure d'un programme Mikropascal

Un programme est un texte que le compilateur va traduire en fichier hexadécimal.

Pour cela, il doit avoir une structure particulière.

Le texte d'un programme doit contenir au moins trois parties.

Entête

L'entête constitué d'une seule ligne; commençant par le mot réservé «Program» suivi du nom du programme et d'un point-virgule.

Déclarations

Les déclarations permettent de définir les éléments utilisés dans le programme. En effet, on doit déclarer les variables utilisées pour permettre au compilateur d'effectuer les réservations nécessaires de mémoire ainsi que les sous-programmes (Procédures et fonctions).

Corps du programme

Le corps du programme: Commence par le mot réservé «Begin» et se termine par mot réservé «End» suivi d'un point final. Entre «Begin» et «End» se trouvent les instructions à effectuer par le programme.

Exemple: Programme pour un PIC16F876A

	Algorithmique	*Programme en PASCAL*
Entête	**Algorithme comparison;**	**program comparison;**
Déclarations	**Variables** Na: **octet** affecté au PortA; Nb: **octet** affecté au PortB; inf: **un bit** affecté au PortC.0; ega: **un bit** affecté au PortC.1; sup: **un bit** affecté au PortC.2;	**var** Na: **byte** at porta; Nb: **byte** at portb; Inf: **sbit** at Portc.0; ega: **sbit** at Portc.1; sup: **sbit** at Portc.2;

Corps du programme	**DEBUT**
	TrisA ← $FF; // port A entrées
	TrisB ← $FF; // port B entrées
	TrisC ← $F8; // portc(0,1,2) sorties
	ADCON1 ←$06; // Port A numérique
	TANT QUE (1=1) FAIRE
	DEBUT
	SI (Na < Nb) **ALORS** inf←1
	SINON inf ← 0;
	SI (Na > Nb) **ALORS** sup←1
	SINON sup←0;
	SI (Na = Nb) **ALORS** ega←1
	SINON ega←0;
	FIN TANT QUE;
	FIN.
		BEGIN
		TrisA := $FF; // port A entrées
		Trisb := $FF; // port B entrées
		TrisC := $F8; // portc(0,1,2) sorties
		ADCON1:=$06; // Port A numérique
		WHILE (1=1) DO
		BEGIN
		IF (Na < Nb) **THEN** inf:=1
		ELSE inf:=0;
		IF (Na > Nb) **THEN** sup:=1
		ELSE sup:=0;
		IF (Na = Nb) **THEN** ega:=1
		ELSE ega:=0;
		END;
		END.

3- Les Règles de bases:

- Toute instruction ou action doit se terminer par un point-virgule;
- une ligne de commentaires doit commencer par «**{**» et se terminer par «**}**» ou commence par «**//**»;
- un bloc d'instructions commence par «**Begin**» et se termine par «**End**».

4- Les types de variables utilisées en Mikropascal pro

Type	Désignation	Taille	Rang
Bit	bit	1 bit	0 ou 1
Bit registre	sbit	1 bit	0 ou 1
Octet	Byte	8 bits	0 ➔ 255
Caractère ASCII	Char	8 bits	0 ➔ 255
Octet signé	short	8 bits	-128 ➔ 127
Mot	word	16 bits	0 ➔ 65535
Entier	integer	16 bits	-32768 ➔ 32767
Mot double	dword	32 bits	0 ➔ 4294967295
Entier long	longint	32 bits	-2147483648 ➔ 2147483647
Réel	real	32 bits	$\pm 1.17549435082*10^{-38}$ ➔ $\pm 6.80564774407*10^{38}$
Tableau	Array[n] of type	n éléments	Rang du type
Chaîne de caractères	string[n]	n caractères	0 ➔ 255

5- Les bases du compilateur Mikropascal pro :

Le décimal: **A:=10** ;
L'hexadécimal: **A :=$0A** ; ou **A:=0x0A** ;
Le binaire: **A:=%00001010** ;

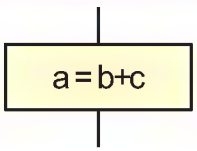
6- Les opérateurs arithmétiques et logiques

Opérateurs arithmétiques		Opérateurs de comparaison		Opérateurs logiques	
Opérateur	Opération	Opérateur	Opération	Opérateur	Opération
+	Addition	**=**	Egalité	**AND**	ET
–	Soustraction	**<>**	Différent	**OR**	OU
*****	Multiplication	**>**	Supérieur	**XOR**	OU exclusif
/	Division	**<**	Inférieur	**NOT**	NON
div	Division entière	**<=**	Inférieur ou égale	**SHL**	Décalage à gauche
mod	Reste de la division entière	**>=**	Supérieur ou égale	**SHR**	Décalage à droite

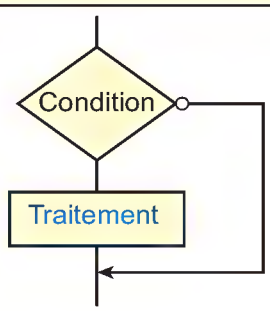
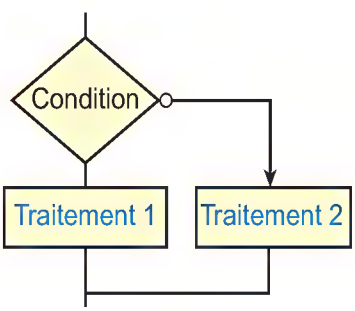
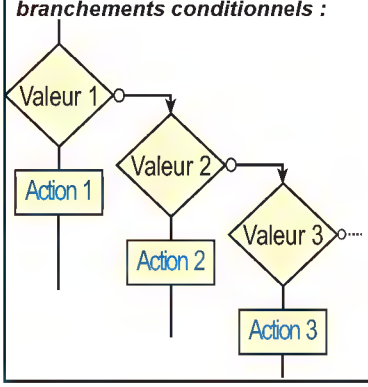
7- Les structures usuelles

a. L'affectation

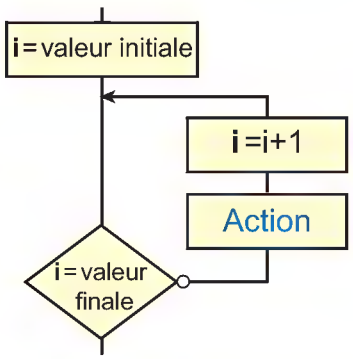
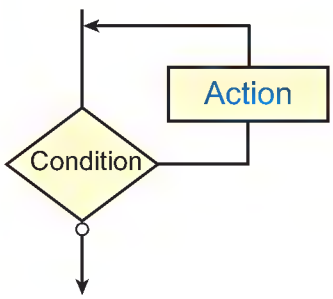
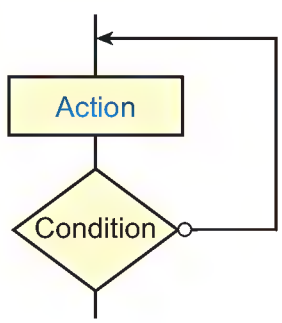
C'est l'action d'assigner une valeur à une variable.

Langage graphique	Langage algorithmique	Langage PASCAL
	a ← b+c	a:=b+c

b. Les structures alternatives

Langage graphique	Langage algorithmique	Langage PASCAL
	**SI** condition **ALORS** **DEBUT** *Traitement ;* **FINSI**;	**IF** condition **THEN** **BEGIN** *Traitement ;* **END ;**
	**SI** condition **ALORS** **DEBUT** *Traitement 1;* **FIN** **SINON** **DEBUT** *Traitement 2;* **FINSI ;**	**IF** condition **THEN** **BEGIN** *Traitement 1;* **END** **ELSE** **BEGIN** *Traitement 2;* **END ;**
A construire avec les branchements conditionnels : 	**SELON** expression Valeur 1: action 1; Valeur 2: action 2; .....; Valeur n: action n; **Autrement**: action 0; **FINSELON** ;	**CASE** expression **OF** Valeur 1:action 1; Valeur 2:action 2; .....; Valeur n:action n; **ELSE** action 0 **END**;

c. Les structures itératives ou répétitives

Langage graphique	Langage algorithmique	Langage PASCAL
A construire avec un branchement conditionnel :  	**I:** entier; **POUR I** variant de <valeur initiale> **JUSQU'A** <valeur finale> **FAIRE** **DEBUT** *Action;* **FINFAIRE;**	**I: integer;** **FOR I:=**<valeur initiale> **To** <Valeur finale> **Do** **BEGIN** **Action;** **END;**
A construire avec un branchement conditionnel :  	**TANQUE** condition **FAIRE** **DEBUT** *Action;* **FINFAIRE;**	**WHILE** condition **DO** **BEGIN** *Action;* **END;**
A construire avec un branchement conditionnel :  	**REPETER** **DEBUT** *Action;* **FIN;** **JUSQU'A condition**	**REPEAT** **BEGIN** *Action;* **END;** **UNTIL condition;**

8- Les procédures et les fonctions :

Une suite d'instructions peut être rassemblée en un seul bloc qui peut être appelé depuis plusieurs endroits d'un programme.

Ceci donne lieu aux notions de sous-programmes appelés aussi procédures ou fonctions.

a. Les Procédures

Ce sont des groupes d'instructions qui vont former une nouvelle instruction simple utilisable dans un programme. En Pascal il faut les définir avant de les utiliser.

Ceci se traduit par une structure similaire à celle d'un programme.

	Structure	Exemple
Entête	**Procedure** Identificateur (P1:Type1, P2:Type2,...); *Identificateur est le nom de la procédure;* *P1, P2 ... sont des paramètres que le programme fournit à la procédure sous forme de constantes, de variables ou d'expressions;* *Type1, Type2 ... sont les types de ces paramètres.*	**Procedure** testP(var a : byte) ; *Ici la procédure testP reçoit du programme principal une variable «a» de type octet*
Déclarations	**Var** Variable: type; **Const** constante: type = valeur; *Déclarations de constantes et variables utilisées à l'intérieur de la procédure*	**Var** s: sbit at portb.0; **Const** b:byte = 5; *Déclaration interne à la procédure*
Corps de la procédure	**Begin** **Instruction1;** **Instruction2;** **...** **End;** *Il s'agit des instructions exécutées par le programme à l'appel de la procédure. Une procédure peut appeler d'autres procédures définies avant elle.*	**Begin** **IF a=b then s:=1** **else s:=0 ;** **End;** *Cette procédure compare la valeur de la variable reçue du programme principal «a» à la valeur 5 et met à 1 la broche RB0 du microcontrôleur en cas d'égalité.*
Appel	**P1 := 30;** **P2 := 20;** **Identificateur (P1, P2:,...);**	**a:=20;** **testP(a);**

L'appel d'une procédure se fait en écrivant son nom suivi des paramètres nécessaires entre parenthèses.

NB: *il est possible d'écrire des procédures sans paramètres.*

EXEMPLE

La signalisation lumineuse du robot aspirateur cité précédemment dans la mise en situation est assurée par une série de 8 diodes LED de couleur rouge.

Ces diodes permettent à l'utilisateur de connaitre l'état du robot à distance sans être obligé de s'approcher pour lire l'affichage LCD.

Le fonctionnement des diodes est le suivant :

- à l'arrêt les diodes sont toutes éteintes;
- en fonctionnement le robot allume les diodes l'une après l'autre (chenillard);
- lorsque sa batterie est à la limite de la décharge, le robot effectue un clignotement des diodes ;
- durant la charge de la batterie toutes les diodes sont constamment allumées.

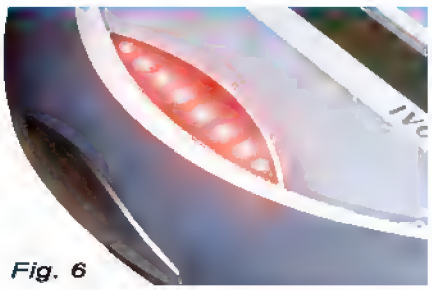
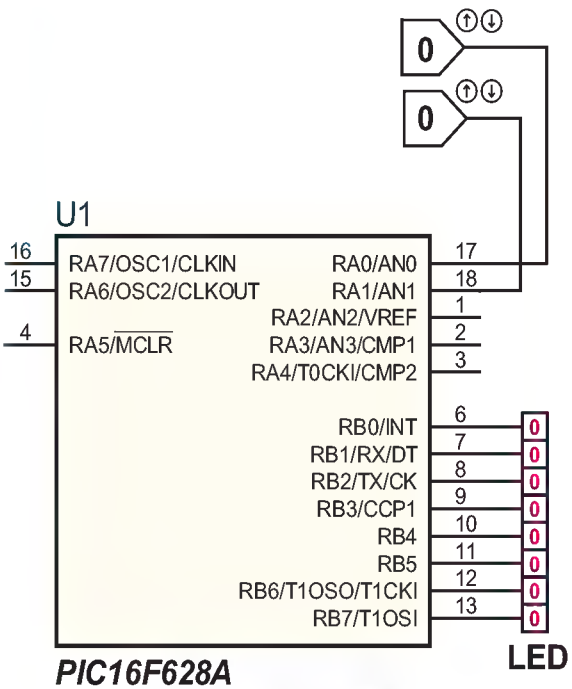
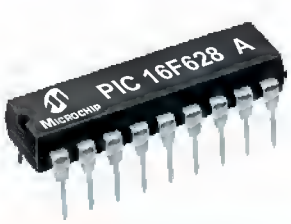


Fig. 6

On peut mettre en œuvre la fonction signalisation lumineuse du robot en utilisant un microcontrôleur de type PIC16F628 comme suit :





RA1	RA0	Etat des diodes
0	0	Arrêt
0	1	Clignotant
1	0	Chenillard
1	1	Marche

Fig. 7

```
program diodeled;
 Var
diodes: byte at portb;
Entree: byte at Porta;
procedure marche (const etat:byte);
  begin
   diodes:= etat;
  end;
procedure chenillard();
  var i : byte;
  begin
  diodes:=%10000000 ;
  For i:=0 to 7 do
    begin
    delay_ms(500);
    diodes := diodes SHR 1;
    end;
  end;
procedure clignotant();
  begin
   marche($FF); delay_ms(500);
   marche(0); delay_ms(500);
  end;
begin
 trisb:=$00; trisa:=$FF;
 cmcon:=$07; //port A numérique
 while true do
     begin
     if (Entree = 0) then marche(0);
     if (Entree = 1) then clignotant();
     if (Entree = 2) then chenillard();
     if (Entree = 3) then marche($FF);
     end;
end.
```

b. Les Fonctions

Une fonction est un sous-programme qui doit fournir un résultat de type numérique ou chaîne de caractères. La définition se fait en utilisant une structure similaire à celle de la procédure.

	Structure	Exemple
Entête	**Fonction Identificateur (P1:Type1, P2:Type2, ...): Type_R;** *Identificateur est le nom de la fonction;* *P1, P2 ... sont des paramètres que le programme fournit à la fonction sous forme de constantes, de variables ou d'expressions;* *Type1, Type2 ... sont les types de ces paramètres.* *Type_R est le type du résultat fourni par la fonction.*	**Function testF(var a : byte): byte ;** *Ici la fonction testF reçoit du programme principal une variable de type octet et renvoie en fin de traitement un résultat de type octet*
Déclarations	**Var** Variable : type ; **Const** constante : type = valeur; *Déclarations de constantes, types, variables utilisés à l'intérieur de la fonction*	**Var** s: byte ; **Const** b: byte = 4; *Déclaration interne à la fonction*
Corps de la fonction	**Begin** **Instruction1;** **Instruction2;** **...** **Identificateur:=résultat;** **End;** *Il s'agit des instructions exécutées par le programme à l'appel de la fonction. L'une de ces instructions doit fournir le résultat de la fonction en l'affectant au nom de la fonction.*	**Begin** **s:= a mod b** **testF:=s;** **End;** *Cette fonction renvoie le résultat du calcul a mod b (Reste de la division entière de a par b).*
Appel	**P1:=30 ;** **P2:=20 ;** **...** **V:= Identificateur (P1, P2, ...)**	**a:= 40 ;** **portb:= testF(a) ;**

L'appel d'une fonction se fait en écrivant son nom suivi des paramètres nécessaires entre parenthèses.

A la différence de la procédure une fonction représente une expression (valeur numérique, caractère, chaine de caractère, ...) semblable au type du résultat fourni.

EXEMPLE :

La détection des obstacles par le robot aspirateur est assurée par une chaine de capteurs infrarouges et à ultrason.

L'un des capteurs, placé à l'avant du robot, est un capteur à ultrason, le principe de son fonctionnement est basé sur l'émission à intervalles réguliers de courtes impulsions sonores à haute fréquence (40 KHz non audible).

Ces impulsions se propagent dans l'air à la vitesse du son (**340,29 m/s**). Lorsqu'elles rencontrent un objet, elles se réfléchissent et reviennent sous forme d'écho au capteur.

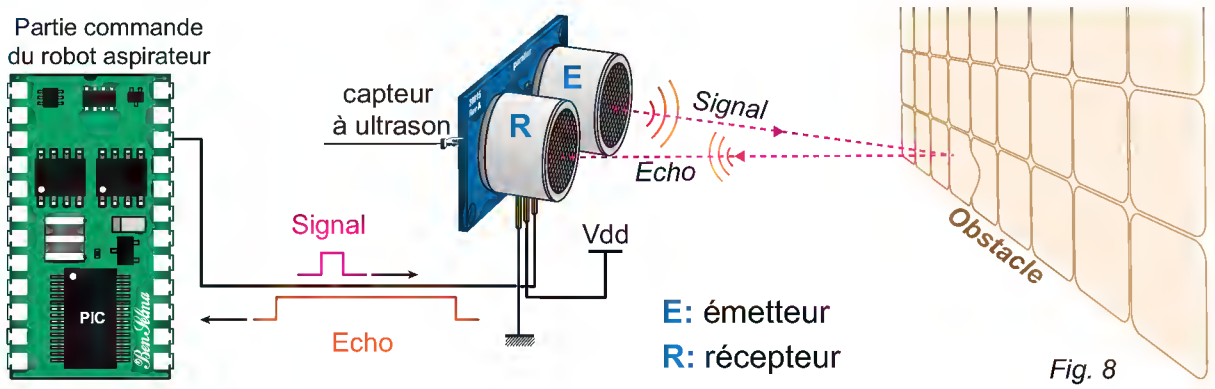


Fig. 8

La partie commande calcule la distance séparant le robot de l'obstacle sur la base du temps écoulé entre l'émission du signal et la réception de l'écho.

En appliquant la formule suivante:

distance (d) = vitesse x temps

Pour un aller-retour du signal ultrason:

- 2 x d = vitesse du signal ultrason x temps;
- vitesse du signal ultrason: **V = 340,29 m/s**;
- temps mesuré = T_m en secondes;
- **2 x d =** $V \times T_m \Rightarrow$

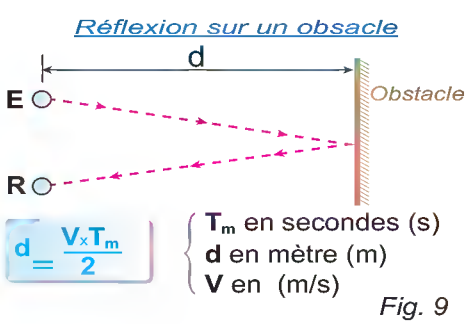


Fig. 9

Pour concrétiser cette fonction on peut utiliser un microcontrôleur de type PIC16F876A, ce microcontrôleur a pour mission d'(e) :

- envoyer un signal ultrason ;
- recevoir l'écho ;
- mesurer le temps écoulé entre l'émission et la réception ;
- calculer et afficher la distance qui sépare le robot de l'obstacle.

Pour mesurer le temps écoulé entre l'émission du signal et sa réception on utilise le ***TIMER0*** du microcontrôleur cadencé avec une horloge de ***1Mhz*** (période = 1µs).

Pour cela on choisit :

- une horloge de 16 MHz pour le microcontrôleur.
- l'horloge du cycle instruction (Fosc/4) comme entrée au pré-diviseur du *Timer0*
- un rapport 1:4 pour le pré-diviseur (prescaler).

La sélection du mode de fonctionnement et la configuration du **TIMER0** sont assurées par le registre **OPTION_REG**. (Voir tableau à la fin de la page).

bit7: RBPU = Pull up Enable bit on Port B.
1 = Pull up désactivé sur le Port B.
0 = Pull up activé.

Bit6: INTEDG = Interrupt Edge select bit.
1 = Interruption si front montant sur RB0/INT
0 = Interruption si front descendant sur RB0/INT.

Bit5 : TOCS = Timer TMR0 Clock Source select bit.
1 = L'horloge du Timer est l'entrée RA4/Clk
0 = Le Timer utilise l'horloge interne du PIC.

Bit4: TOSE = Timer TMR0 Source Edge select bit.
1 = Le Timer s'incrémente à chaque front montant sur RA4/Clk.
0 = Le Timer s'incrémente à chaque front descendant sur RA4/Clk.

Bit3: PSA = Prescaler Assignement bit.
1 = Le pré-diviseur est affecté au watchdog..
0 = Le pré-diviseur est affecté au Timer TMR0.

PS2	PS1	PS0	Prédiv
0	0	0	1:2
0	0	1	1:4
0	1	0	1:8
0	1	1	1:16
1	0	0	1:32
1	0	1	1:64
1	1	0	1:128
1	1	1	1:256

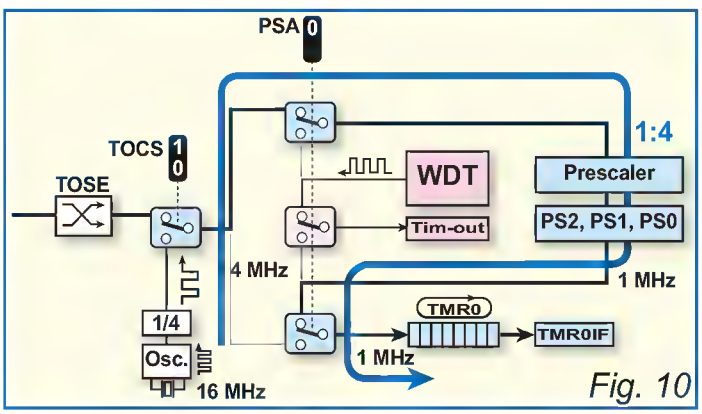


Fig. 10

NB: *Le bit 6 (**INTEDG**) et le bit 4 (**TOSE**) ne sont pas utilisés dans notre cas on leur affecte la valeur 0 par défaut.*

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

OPTION_REG = %10000001 = $81

SCHÉMA:

Pour simuler la fonction du capteur ultrason sous Isis, on utilise le composant **«DELAY_1»** réglé à 100µs pour créer le retard du signal sortant de la broche RB1 du microcontrôleur.

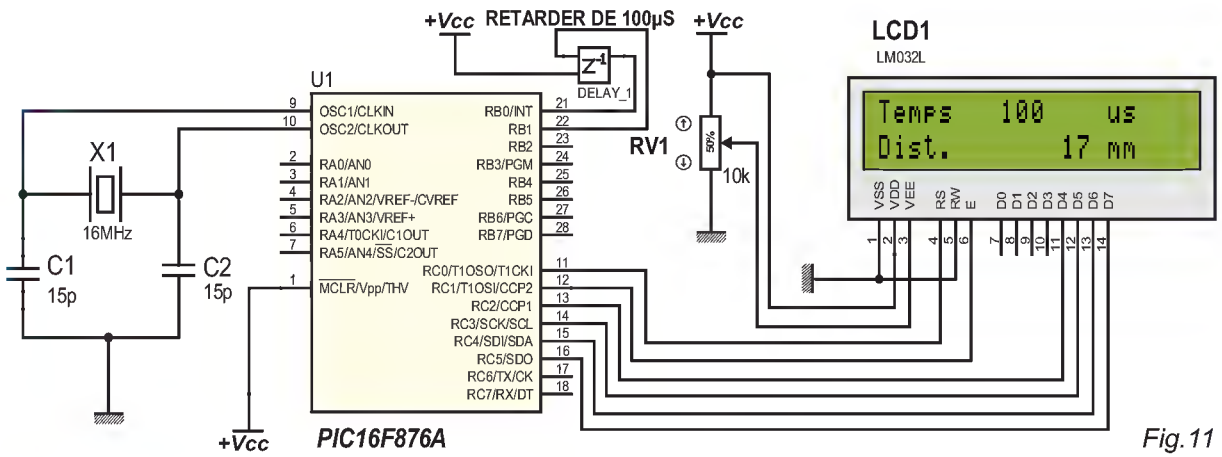


Fig.11

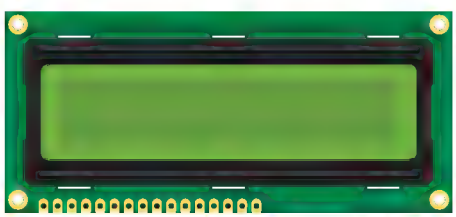
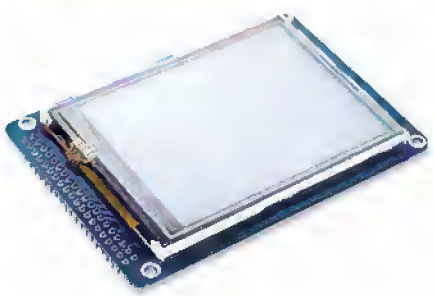
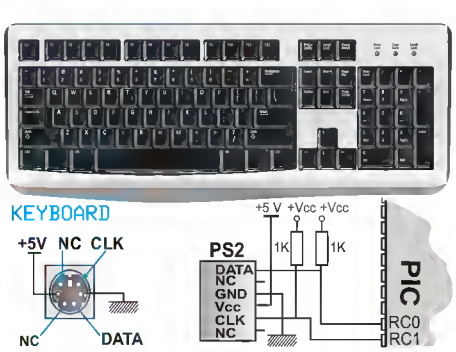
PROGRAMME:

```
program ultrason;
var
   Tmesure : byte;
   Distance : integer;
   temps_affichage : string[3];
   distance_affichage : string[5];
   signal : sbit at portb.1;
   echo   : sbit at portb.0;
   // Connections de l'LCD
   LCD_RS : sbit at portc.0;  // RS est connectée à RC0
   LCD_EN : sbit at portc.1;  // EN est connectée à RC1
   LCD_D4 : sbit at portc.2;  // D4 est connectée à RC2
   LCD_D5 : sbit at portc.3;  // D5 est connectée à RC3
   LCD_D6 : sbit at portc.4;  // D6 est connectée à RC4
   LCD_D7 : sbit at portc.5;  // D7 est connectée à RC5
   LCD_RS_Direction : sbit at TRISC.0;
   LCD_EN_Direction : sbit at TRISC.1;
   LCD_D4_Direction : sbit at TRISC.2;
   LCD_D5_Direction : sbit at TRISC.3;
   LCD_D6_Direction : sbit at TRISC.4;
   LCD_D7_Direction : sbit at TRISC.5;

function calcul_distance_mm(var temps: byte):integer;
  var temps_s : real;
      Distance_m : real;
```

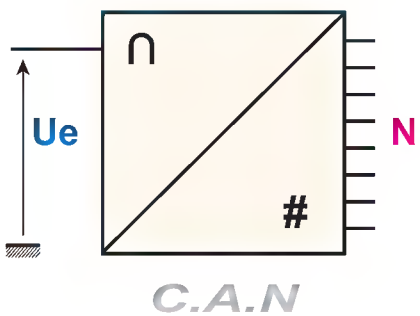
```
begin
  temps_s := temps*pow(10,-6); // transformation du temps de µs en s
  Distance_m := (temps_s*340.29)/2;  // calcul de la Distance en m
  calcul_distance_mm:=Distance_m * pow(10,3); //transformation en mm
end;

begin
lcd_init();              // initialisation de l'LCD
lcd_cmd(_LCD_CURSOR_OFF); // désactivation du curseur de l'LCD
lcd_out(1,1,'Temps ');    // préparation de l'affichage
lcd_out(2,1,'Dist. ');
trisB:=$FD;         // RB1 : sortie, RB0 : entrée
portb.1:=0;         // initialisation de RB1 à 0
OPTION_REG := $81;  // Timer0
                    // Horloge Fosc/4
                    // Prescaler = 1:4 donc 1MHz ou 1µs
while true do
begin
  Tmesure:=0; // initialisation du temps mesuré à 0
   signal:=1; // envoie du signal
   TMR0 := 0; // initialisation du Timer 0  à 0
    // attente de l'echo
while (TMR0 <= 250) do
  begin
      if (echo = 1) then
      begin
      Tmesure:=TMR0; // réception de l'écho alors lecture du TIMER0
      break;         // sortir de la boucle
      end;
  end;
signal:=0;                      // remise à zéro du signal
bytetostr(TMesure,temps_affichage); // conversion du temps mesuré en
                                // texte pour l'affichage
lcd_out(1,9,temps_affichage);   // affichage du temps mesuré en µs
lcd_out(1,16,'us');             // affichage de l'unité µs
Distance := calcul_distance_mm(TMesure); //fonction de calcul de la
                                                    // distance
intToStr(Distance,distance_affichage); // conversion de la distance
                               // calculée en texte
lcd_out(2,9,distance_affichage);   // affichage de la distance
                                        // calculé en mm
lcd_out(2,16,'mm');     // affichage de l'unité mm
end;
end.
```

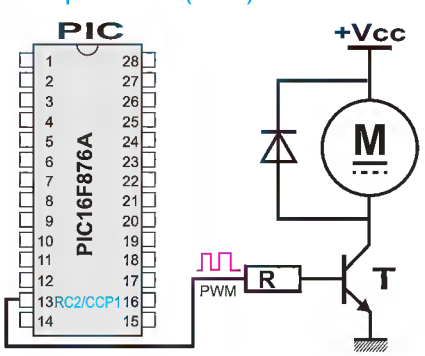
9- Instructions spécifiques au compilateur MikroPascal pro

Le compilateur mikropascal pro offre un large éventail de procédures et fonctions adaptées aux microcontrôleurs de la famille PIC de MICROCHIP. Ces fonctions sont classées par catégorie et accessibles dans l'aide du logiciel néanmoins on va citer quelques-unes.

Catégorie	Type	Exemple
Affichage	Ecran LCD 	*//Initialisation de l'LCD* ***lcd_Init();*** *// Ecriture de «Bonjour!» sur l'Lcd* *// à partir de la ligne 1, colonne 3:* ***Lcd_Out**(1, 3, ''Bonjour!'');*
	Ecran TFT 	*//Initialisation du TFT avec une* *//résolution de 240x320* ***TFT_Init**(240, 320) ;* … *// traçage d'un rectangle* ***TFT_Rectangle**(20, 20, 219, 107);*
Périphérique d'entrée	Clavier Matriciel 	***var** keypadPort : byte at PORTD* ***var** kp : byte;* ... *// initialisation du clavier* ***Keypad_Init();*** *// lecture de la touche appuyée* ***kp := Keypad_Key_Press();***
	Clavier PS2 	*// Brochage du clavier PS2* ***var** PS2_Data : sbit at RC0_bit;* ***var** PS2_Clock : sbit at RC1_bit;* ***var** PS2_Data_Direction : sbit at TRISC0_bit;* ***var** PS2_Clock_Direction : sbit at TRISC1_bit;* ... ***Ps2_Config();** //Init PS2 Keyboard* ***Ps2_Key_Read**(value, special, pressed) ; // lecture d'une touche*

Catégorie	Type	Exemple
Périphérique d'entrée	Bouton poussoir 	*// lecture d'un bouton poussoir monté* *// sur la broche RB0 du microcontrôleur* *// avec un temps d'appui minimal de* *// 10ms, état actif haut* ***If Button****(PORTB, 0, 10, 1)* ***then ...***

Signaux et conversion	Conversion analogique numérique 	// initialization du module convertisseur
		// analogique numérique
		ADC_Init();
		// lecture de la dernière valeur lue par le
		// convertisseur analogique numérique
		//sur le canal 1
		var adc_value : word
		...
		adc_value := ADC_Get_Sample(1);
		// lecture après initialisation et
		//démarrage de la conversion
		adc_value:= ADC_Read(1);

	Modulation de la largeur d'impulsion (MLI) 	// initialization du module MLI (PWM)
		//avec une fréquence de 5kHz
		PWM1_Init*(5000);*
		// régler le rapport cyclique à 75%
		PWM1_Set_Duty*(192);*
		// démarrage de la modulation de largeur
		// d'impulsion
		PWM1_Start();
		// arret de la modulation
		PWM1_Stop();

10- Conversion analogique numérique

a. Introduction

Afin de protéger la lampe ultraviolet le robot mesure constamment la température, il arrête la stérilisation du sol lorsque la température de la lampe atteint 80°C.

D'autre part, pour des raisons d'efficacité contre les allergènes, le robot ne commence le cycle stérilisation du sol que lorsque la température de la lampe atteint 40°C.

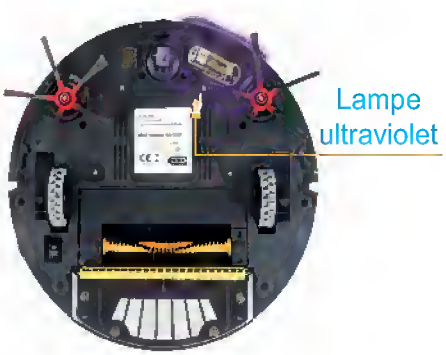


Fig. 12

Le robot affiche en permanence la température de la lampe sur l'afficheur LCD.

La détection de la température est assurée par le capteur LM35, ce capteur fournit à sa sortie une tension proportionnelle à la température (10mV/°C).

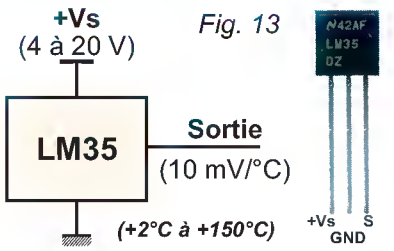


Fig. 13

Les données du capteur de température ont besoin d'être interprétées par la partie commande du robot. Pour cela, on a eu recours à un convertisseur analogique numérique (CAN), ou en anglais ADC pour (Analog to Digital Converter) dont la fonction est de traduire une grandeur analogique (tension) en une valeur numérique (codée sur plusieurs bits), proportionnelle au rapport entre la grandeur analogique d'entrée et la valeur maximale du signal.

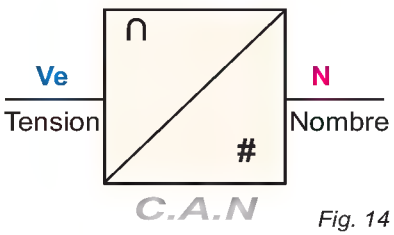


Fig. 14

Certains microcontrôleurs de type PIC (PIC16F876, PIC16F877,…) ont l'avantage de posséder un module convertisseur analogique numérique (CAN) interne.

b. Module convertisseur CAN du **PIC16F87X**

Les microcontrôleurs PIC16F87X tel que le PIC16F876 et le PIC16F877 possèdent un convertisseur analogique numérique sur **10 bits**, ce dernier permet de convertir une tension analogique comprise entre **Vref-** et **Vref+** en une valeur numérique comprise entre **0** et **1023**.

Pour exploiter ce convertisseur il est nécessaire de ***configurer certains registres dans le microcontrôleur***, dans notre cas on s'intéressera uniquement au registre **ADCON1** pour sélectionner et activer les entrées analogiques multiplexées avec le **port A** et le **port E** du **PIC16F877**.

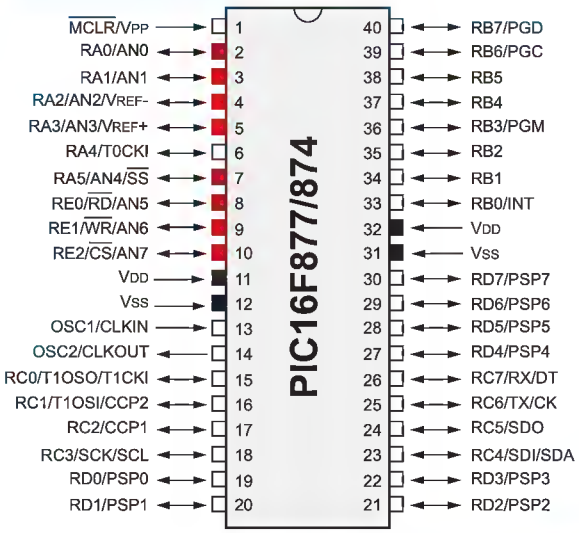
ADCON1

ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

Les bits PCFG3 … PCFG0

Ces **4 bits** permettent la sélection et la configuration des entrées analogiques à utiliser conformément au tableau de la page suivante :

CONFIGURATION



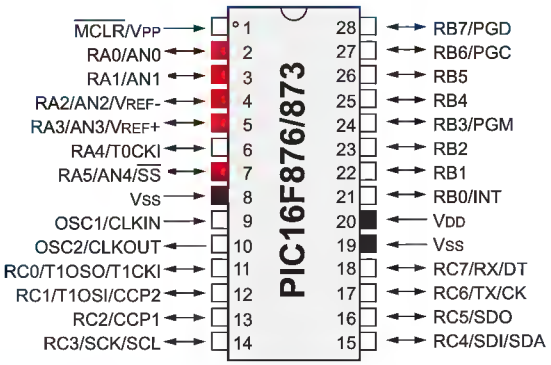


Fig.15

	PIC16F876									
	PIC16F877									
ADCON1								PORTE			PORTA					Tensions de références	
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	AN7/RE2	AN6/RE1	AN5/RE0	AN4/RA5	AN3/RA3	AN2/RA2	AN1/RA1	AN0/RA0	VREF+	VREF-
1	-	-	-	0	0	0	0	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}
1	-	-	-	0	0	0	1	A	A	A	A	VREF+	A	A	A	RA3	VSS
1	-	-	-	0	0	1	0	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}
1	-	-	-	0	0	1	1	D	D	D	A	VREF+	A	A	A	RA3	VSS
1	-	-	-	0	1	0	0	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}
1	-	-	-	0	1	0	1	D	D	D	D	VREF+	D	A	A	RA3	VSS
1	-	-	-	0	1	1	x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}
1	-	-	-	1	0	0	0	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2
1	-	-	-	1	0	0	1	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}
1	-	-	-	1	0	1	0	D	D	A	A	VREF+	A	A	A	RA3	VSS
1	-	-	-	1	0	1	1	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2
1	-	-	-	1	1	0	0	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2
1	-	-	-	1	1	0	1	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2
1	-	-	-	1	1	1	0	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}
1	-	-	-	1	1	1	1	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2

$V_{DD}=V_{CC}= 5V$
$V_{CC}=GND= 0V$

A: analogique **D: numérique**

Fig. 16

NB :

- On s'intéressera uniquement au cas où VREF- =VSS=0V et VREF+=VDD = 5V (Cas des lignes coloriées en rouge dans le tableau précedent).

- Les entrées analogiques du « PORTA » sélectionnées doivent être configurées en « Entrée » via le registre « TRISA ».

Le bit ADFM :

Le convertisseur CAN fournit un nombre binaire naturel de 10 bits (B9 B8 B7 B6 B5 B4 B3 B2 B1 B0).

Deux registres (2 x 8 bits) sont nécessaires pour stocker le résultat de la conversion. Ce sont les registres :

- ADRESH
- ADRESL

Deux formats sont disponibles suivant la valeur du bit ADFM (bit 7 du registre ADCON1) :

- ***ADFM = 1 :*** le résultat de la conversion est justifié à droite

ADRESH								ADRESL							
0	0	0	0	0	0	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

- ***ADFM = 0 :*** le résultat de la conversion est justifié à gauche

ADRESH								ADRESL							
B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	0	0	0	0	0	0

La fonction prédéfinie de mikropascal pro ADC_Init() permet de configurer automatiquement le module convertisseur analogique numérique du microcontrôleur avec les réglages suivants :

- *Vref- = 0V*
- *Vref+ = 5V*
- *Utilisation de l'horloge RC interne pour la conversion.*

Cette fonction doit être appelée **après** avoir sélectionné les entrées analogiques avec le registre **ADCON1.**

Deux autres fonctions **ADC_Get_Sample(canal)** et **ADC_Read(canal)** permettent de lire automatiquement le résultat de la conversion à partir des registres **ADRESH et ADRESL**.

La fonction **ADC_Get_Sample(canal)** permet de lire le résultat de la conversion sur le canal fournit sous forme d'un mot sur 16 bits, la justification des 10 bits du résultat dépend du bit **ADFM** du registre **ADCON1.**

EXEMPLE :

Lecture du résultat de la conversion sur le canal 0 (RA0/AN0)

```
var v : word
...
v := ADC_Get_Sample(0);
```

La fonction **ADC_Read(canal)** permet d'initialiser le convertisseur, démarrer une opération de conversion puis lire le résultat de conversion sur le canal fourni sous forme d'un mot sur 16 bits, la justification des 10 bits du résultat dépend du bit ADFM du registre ADCON1

EXEMPLE :

Initialisation puis lecture de la valeur fournie par le convertisseur analogique numérique sur le canal 0 (RA0/AN0)

```
var v : word
...
v := ADC_Read(0);
```

c. Application pour le robot aspirateur

Pour lire et afficher la température de la lampe « ultraviolet » du robot on utilise un microcontrôleur PIC16F876A, un afficheur LCD et un capteur de température LM35 cité précédemment.

Schéma

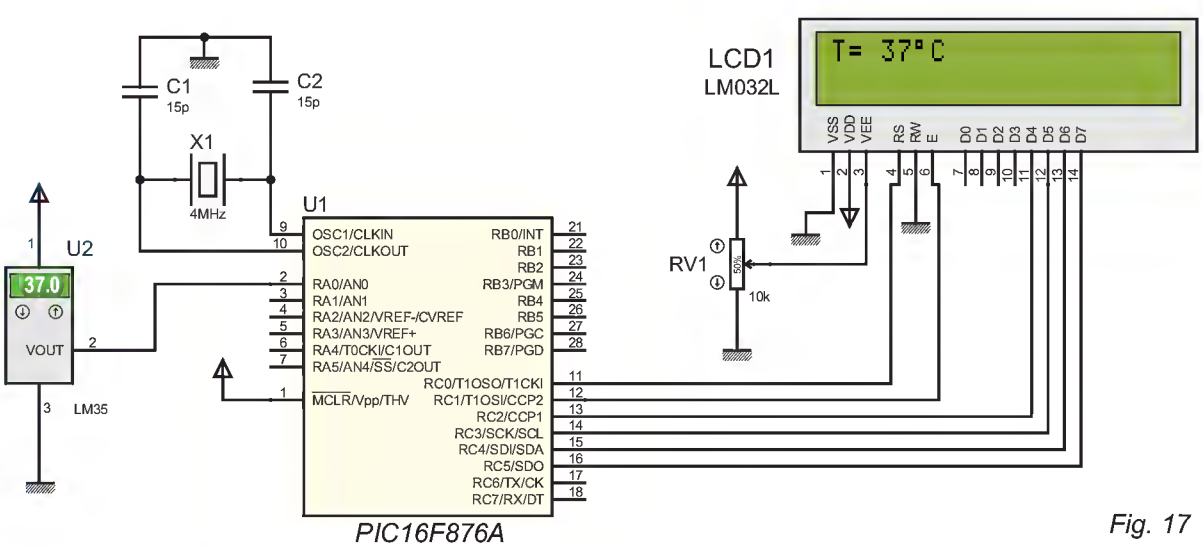


Fig. 17

Le capteur **LM35** fournit à sa sortie une tension proportionnelle à la température (10mV/°C), en appliquant la règle de 3 on obtient :

$$T = \frac{V}{10}$$

T en °C
V en mV

Le convertisseur **CAN** du microcontrôleur **PIC16F876A** converti toute tension comprise entre **0** et **5V** (**0** à **5000 mV**) en un nombre sur **10 bits** (de **0** à 2^{10}**-1** ou de **0** à **1023**). En appliquant la règle de 3 on obtient :

$$V = \frac{N \times 5000}{1023} \quad \{ V \text{ en mV} \quad \Rightarrow \quad T = \frac{N \times 500}{1023} \quad \{ T \text{ en °C}$$

PROGRAMME :

```
program conversion;
var
valeur_conversion : word; //2 octets car le résultat de conversion est sur 10 bits
variable_calcul : real ;  // Type réel pour le calcul afin ne pas avoir un
                          // dépassement de taille lors de la multiplication
                          // ou une perte de précision lors de la division
temperature : byte; // 1 octet car la température est comprise entre 2 et 150
valeur_affichage : string[3]; //chaine de 3 caractères pour afficher la température
// Connections de l'LCD
   LCD_RS : sbit at portc.0;
   LCD_EN : sbit at portc.1;
   LCD_D4 : sbit at portc.2;
   LCD_D5 : sbit at portc.3;
   LCD_D6 : sbit at portc.4;
   LCD_D7 : sbit at portc.5;
   LCD_RS_Direction : sbit at TRISC.0;
   LCD_EN_Direction : sbit at TRISC.1;
   LCD_D4_Direction : sbit at TRISC.2;
   LCD_D5_Direction : sbit at TRISC.3;
   LCD_D6_Direction : sbit at TRISC.4;
   LCD_D7_Direction : sbit at TRISC.5;

begin
lcd_init(); // initialisation de l'LCD
lcd_cmd(_LCD_CURSOR_OFF); // désactivation du curseur de l'LCD
lcd_out(1,1,'T=');                  // préparation de l'affichage
adcon1:=%10000100 ;                 // choix de RA0/AN0 en tant qu'entrée analogique
adc_init();                         // initialisation du module CAN
while true do
 begin
   valeur_conversion := adc_read(0); // lecture du convertisseur
  variable_calcul  := (valeur_conversion * 500)/1023 ; // calcul
  temperature:= byte(variable_calcul);  //transformation en octet(partie entière)
 byteToStr(temperature,valeur_affichage); //conversion de la temp. en texte
 lcd_out(1,3, valeur_affichage);   // affichage de la valeur de la température
 lcd_chr(1,6,%11011111);         // affichage du symbole degré: °
```

```
  lcd_chr(1,7,'C');              // affichage de C pour Celsius
  delay_ms(500);                 // attente de 500ms puis rafraichissement de
                                 // l'affichage

  end;
end.
```

11- Modulation de Largeur d'Impulsion MLI

a. Introduction :

Le déplacement du robot aspirateur est assuré par deux roues motrices entrainées en rotation par deux moteurs à courant continu.

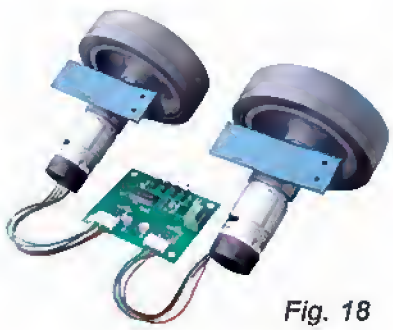


Fig. 18

Chaque moteur est équipé d'un réducteur à engrenages et d'un encodeur.

L'encodeur a pour rôle de fournir à la partie commande (microcontrôleur) une information sur la position et la vitesse de la roue selon le besoin.

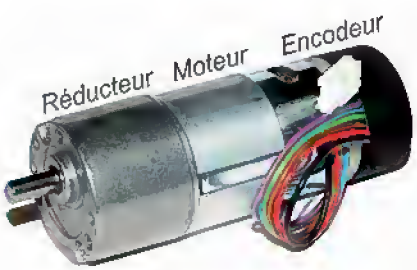
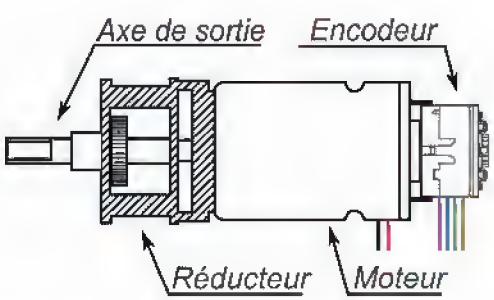
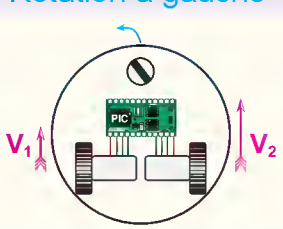
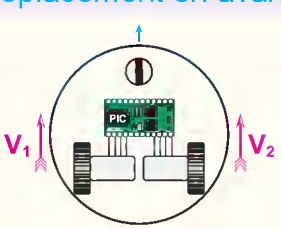
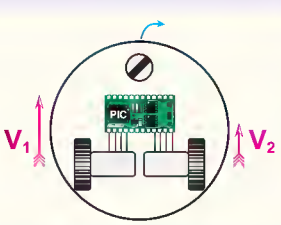


Fig. 19



En variant le sens de rotation et la vitesse de chaque roue, le robot peut se déplacer dans toutes les directions.

Exemple pour le déplacement en marche avant:

Rotation à gauche	Déplacement en avant	Rotation à droite
		
$V_1 < V_2$	$V_1 = V_2$	$V_1 > V_2$

V_1 : vitesse de la roue gauche

V_2 : vitesse de la roue droite

Pour varier la vitesse de chaque moteur du robot, on utilise la technique de **M**odulation de **L**argeur d'**I**mpulsion (**MLI**).

b. La Modulation de Largeur d'Impulsion (**MLI**)

La Modulation de Largeur d'Impulsion MLI (en anglais **P**ulse **W**idth **M**odulation **PWM**) est une technique qui consiste à générer un signal à période constante mais à rapport cyclique variable.

Cette technique est largement utilisée pour faire varier la vitesse des moteurs à courant continu ou pour faire varier la luminosité d'une lampe ou d'une diode LED.

La variation de vitesse d'un moteur à courant continu par MLI consiste à alimenter ce moteur de façon discontinue avec un hacheur et faire ainsi varier la tension moyenne à ses bornes.

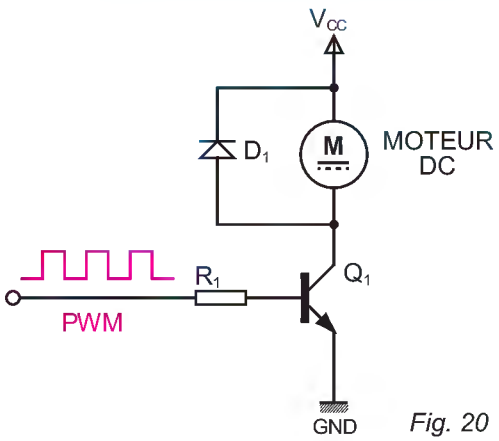
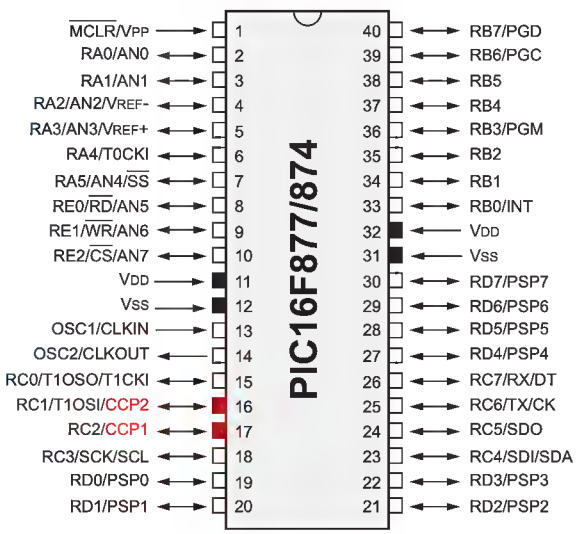


Fig. 20

Afin de faciliter l'utilisation de la technique de modulation de largeur d'impulsion, plusieurs microcontrôleurs possèdent des sorties capables de générer automatiquement des signaux MLI généralement appelées sorties **PWM**.

Les microcontrôleurs PIC16F876A et PIC16F877A possèdent deux sorties PWM notées CCP1 et CCP2 (**CCP** : **C**apture **C**ompare **P**wm).



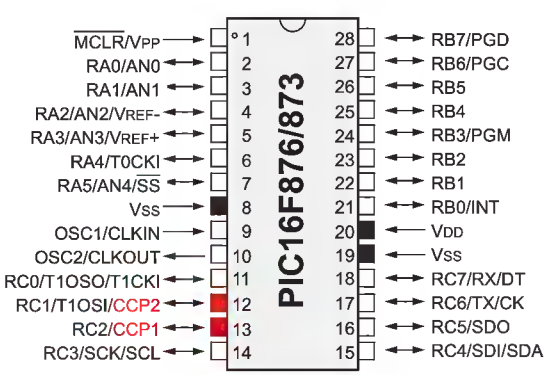


Fig.21

Le compilateur Mikropascal propose des procédures prédéfinies pour gérer les signaux PWM :

Pour chaque sortie CCPx on a :

PWMx_init : Cette procédure permet d'initialiser le module PWM de la sortie CCPx

PWMx_start : Démarrage du module PWM et sortie du signal sur la broche CCPx

PWMx_Set_duty(N) : Change le rapport cyclique α du signal sortant sur la broche CCPx avec α = N / 255 et N variant de 0 à 255.

Exemple :

PWM1_change_duty(255) : change le rapport cyclique α à 255/255 = 1 donc alimente le moteur avec la tension maximale.

PWM1_change_duty(64) : change le rapport cyclique α = 64/255 = 0,25 donc alimente le moteur au quart de la tension.

PWMx_stop : Arrête le module PWM de la sortie CCPx.

c. Application

On désire varier la vitesse d'un moteur à courant continu à aimant permanant en actionnant un potentiomètre rotatif conformément au schéma ci-dessous:

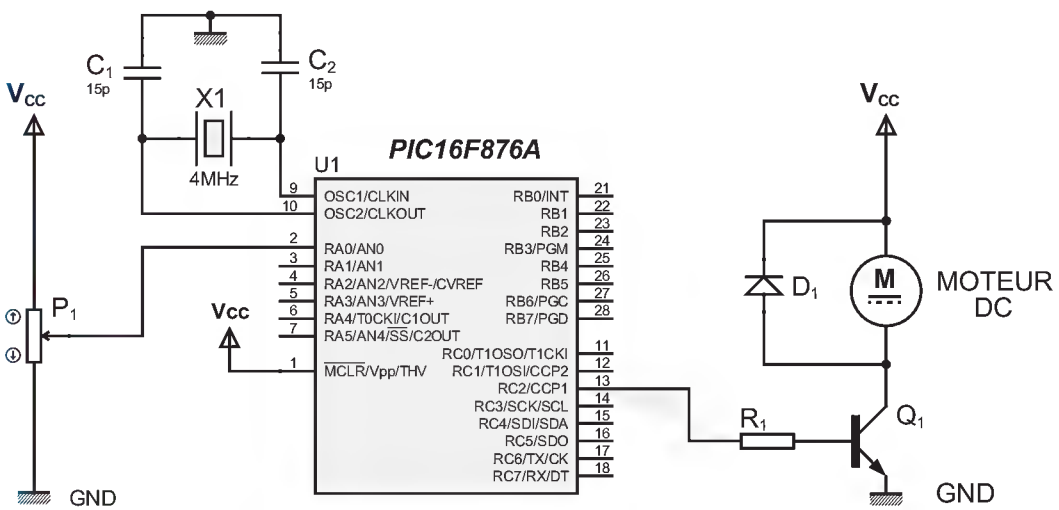


Fig.22

Pour varier la vitesse du moteur **M** de **0** à sa vitesse maximale, on a besoin de faire varier le rapport cyclique du signal **MLI** appliqué à la base du transistor Q_1 de **0** à **1**. Ceci implique une variation du paramètre (**N**) de la procédure ***PWMx_change_duty(N)*** de **0** à **255**.

La rotation du potentiomètre **P1** entraine une variation de la tension sur son curseur de **0** à **5V** .

La lecture de cette tension par le convertisseur analogique numérique **CAN** du microcontrôleur nous donne une valeur qu'on notera (**t**) variant de **0** à **1023**.

En appliquant la règle de trois on obtient :

$$T = \frac{t \times 255}{1023}$$

ALGORITHME

```
Algorithme vitesse ;
Variables :
t : mot de 16 bits;
N : octet ;
Calc : réel ;
DEBUT
Sélection de l'entrée analogique AN0 ;
Initialisation du convertisseur ;
Initialisation du module PWM à une fréquence de 1000 Hz ;
Démarrage du signal PWM ;
TANTQUE (vrai) FAIRE
   DEBUT
   t ← Lecture du convertisseur ;
   Calc ← (t*255)/1023
   N ← octet (Calc) ;
   Modification du rapport cyclique avec la nouvelle valeur de N ;
   Attente(10 ms) ;
   FIN ;
FIN.
```

PROGRAMME

```
program vitesse;
Var
t : word;
N : byte ;
Calc : real ;

begin
adcon1:=%10000100 ;     // choix de RA0/AN0 en tant qu'entrée analogique
adc_init();                  // initialisation du module CAN
PWM1_init(1000);       // initialisation du module PWM à 1000Hz
PWM1_start ;           // Démarrage du signal PWM
while true do
   begin
   t:= adc_get_sample(0);        // Lecture du convertisseur
   Calc := (t*255)/1023;          // calcul
   N := byte(Calc) ;              // transformation du résultat de calcul en octet
   PWM1_Set_Duty(N);        // changement du rapport cyclique
   delay_ms(10);
   end;
end.
```

12- Notions d'interruption

a. Introduction

Le Robot aspirateur est destiné à fonctionner dans des locaux équipés de meubles, objets de décoration, etc....

Fig. 23

Pour assurer une sécurité accrue et éviter d'endommager les objets fragiles (verrerie, porcelaine...) installés dans son environnement, le robot est équipé par des capteurs à distance de type ultrason, infrarouge et des capteurs de contact.

La détection de collision est un évènement d'importance majeure, le microcontrôleur qui commande le robot doit obligatoirement abandonner momentanément toutes les tâches en court (nettoyage, affichage température, cycle de stérilisation...) et envoyer un ordre d'arrêt immédiat aux moteurs suivi d'un recul de 2cm et d'une rotation de 90°.

Une fois ces ordres exécutés, le robot reprend sa tache habituelle

On dit que le microcontrôleur a interrompu son fonctionnement normal pour traiter l'évènement de la détection de collision.

Ce type de fonctionnement fait apparaître la nécessité d'interrompre momentanément le déroulement d'un programme principal (nettoyage, affichage température, cycle de stérilisation...) pour exécuter un autre programme (ordre d'arrêt aux moteurs, recul de 2 cm et une rotation de 90°) à la fin duquel le programme principal reprend son exécution à l'endroit où il a été interrompu.

La solution à ce problème fait appel à ***l'interruption***.

b. Interruption

Une interruption est un événement imprévisible qui provoque l'arrêt d'un programme en cours d'exécution pour aller exécuter un autre programme appelé programme (ou routine) d'interruption.

A la fin du programme d'interruption, le microcontrôleur reprend le programme principal à l'endroit où il s'est arrêté.

On distingue deux types d'interruptions:

Les interruptions externes, qui sont déclenchées lorsqu'un événement extérieur se produit tels que le changement d'état d'une entrée destinée à l'interruption.

Les interruptions internes, qui sont déclenchées par le déroulement du programme tel que le résultat d'un calcul ou le débordement d'un Timer.

Toute interruption est gérée à l'aide de 3 bits :

- Un bit indicateur ou drapeau (***Flag bit***). Ce bit est mis à 1 lorsque l'interruption correspondante survient.
- Un bit d'activation (***Enable bit***). Ce bit permet d'activer ou de désactiver l'interruption correspondante.
- Un bit d'activation globale (***Global Enable bit***). Ce bit permet d'activer ou de désactiver toutes les interruptions.

Ces bits sont regroupés suivant le microcontrôleur cible dans des registres appelés registres de configuration des interruptions tels que: **INTCON, PIE1, PIE2, PIR1** et **PIR2.** Le nombre de sources d'interruptions dépend du microcontrôleur utilisé.

c. Registre de configuration des interruptions (INTCON):

Le registre **INTCON** (**INT**errupt **CON**troller) est le registre principal de contrôle et de gestion des interruptions.

Suivant le type du microcontrôleur donc du nombre de sources d'interruptions, le registre INTCON est parfois accompagné par d'autres registres tels que (PIE, PIR1..) pour gérer la totalité des sources d'interruptions disponibles.

Le registre **INTCON** est parfois différent d'un **PIC** à un autre il est impératif de revenir au document constructeur pour chaque type de microcontrôleur.

EXEMPLES

- Registre INTCON pour PIC16F876A :

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**PEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

- Registre INTCON pour PIC16F84A

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 7: GIE = Global Interrupt Enable bit
1 = Autorise toutes les interruptions non masquées par leur bit individuel.
0 = Désactive toutes les interruptions.

Bit 6 (PIC16F876A): PEIE = Peripheral Interrupt Enable bit.
1 = Autorise les interruptions causées par les périphériques non masquées par leur bit individuel dans les registres PIE1 et PIE2 (PIE : Peripheral Interrupts Enable)
0 = Désactive les interruptions causées par les périphériques.
Bit 6 (PIC16F84A): EEIE = EEPROM Interrupt Enable bit.
1 = Autorise les interruptions causées par la fin d'écriture dans l'EEPROM
0 = Désactive les interruptions causées par la fin d'écriture dans l'EEPROM

Bit 5: TOIE = Timer TMR0 Overflow Interrupt Enable bit.
1 = Autorise l'interruption du Timer TMR0.
0 = Désactive l'interruption du Timer TMR0.

Bit 4: INTE = RB0/Int Interrupt Enable bit.
1 = Autorise l'interruption sur la broche : RB0
0 = Désactive l'interruption sur la broche : RB0

Bit 3: RBIE = RB Port Change Interrupt Enable bit.
1 = Autorise l'interruption par changement d'état du Port B (RB4 à RB7).
0 = Désactive l'interruption par changement d'état du Port B (RB4 à RB7).

Bit 2: TOIF = Timer TMR0 Overflow Interrupt Flag bit.
Ce bit est un indicateur ou drapeau (Flag); il est mis à 1 si une interruption est générée par le débordement du TMR0.
1 = Le Timer a débordé.
0 = Le Timer n'a pas débordé.
Ce drapeau doit être remis à zéro par le programme de traitement de l'interruption.

Bit 1: INTF = RB0/Int Interrupt Flag bit.
1 = Une interruption sur la broche RB0 est survenue.
0 = Pas d'interruption sur la broche RB0.
Ce drapeau doit être remis à zéro par le programme de traitement de l'interruption.

Bit 0: RBIF = RB Port Change Interrupt Flag bit. Ce drapeau doit être remis à zéro par le programme.
1 = Quand au moins une entrée du port B (de RB4 à RB7) a changé d'état.
0 = Aucune entrée de RB4 à RB7 n'a changé d'état.

NB :

- *Par défaut toutes les interruptions sont désactivées I**NTCON = 0000000X***
- *Lorsqu'une interruption survient et durant l'exécution du sous-programme d'in-*

terruption le bit GIE est mis à 0 automatiquement pour interdire le déclenchement d'autres interruptions dans la routine d'interruption en cours.

- *L'utilisateur doit impérativement réactiver l'interruption désirée et mettre à zéro les indicateurs correspondants à la fin du sous-programme d'interruption.*

d. Mise en œuvre d'une routine d'interruption en mikropascal

En mikropascal, le sous-programme d'interruption est déclaré en tant que procédure avec le nom spécial « Interrupt ». Cette procédure s'exécute automatiquement en réponse aux évènements déclencheurs des interruptions activées par l'utilisateur.

	Algorithmique	Programme en PASCAL
Entête	**Algorithme testint;**	**program testint;**
Déclarations	**Variables** **déclaration des variables** **…**	**var** **déclaration des variables** **…**
Routine d'interruption	*Procédure interruption ;* *DEBUT* *Instruction 1* *Instruction 2* *…..* *Instruction n* *Réactivation de l'interruption et remise à zéro de l'indicateur correspondant* *FIN ;*	*Procedure interrupt ;* *BEGIN* *Instruction 1* *Instruction 2* *…..* *Instruction n* *Réactivation de l'interruption et remise à zéro de l'indicateur correspondant* *END ;*
	…	**…**
Corps de la procédure	**DEBUT** **…** *Activation de l'interruption dans le programme principal* … TANT QUE (1=1) FAIRE DEBUT …. FIN TANT QUE; FIN.	**BEGIN** **…** *Activation de l'interruption dans le programme principal* … WHILE (1=1) DO BEGIN …. END; END.

13- Interruption RB0/INT

L'interruption externe **RB0/INT** se produit sur le front montant ou descendant d'une impulsion appliquée sur l'entrée **RB0**.

- Il s'agit du front montant si le **bit 6** (**INTEDG**) du registre d'option **OPTION_REG** est à **1**.

Il s'agit du front descendant si le **bit 6** (**INTEDG**) du registre d'option **OPTION_REG** est à **0**.

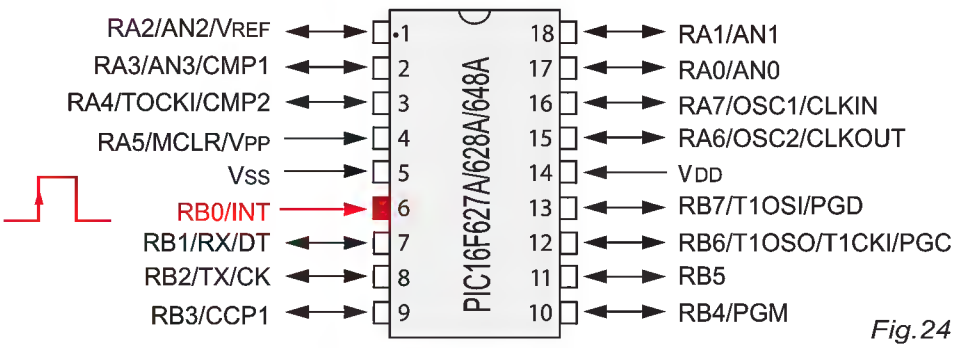


Fig.24

L'interruption externe **RB0/INT** est gérée par les **4 bits** suivants:

- **GIE :** bit de validation globale de toutes les interruptions (1=oui, 0=non).
- **INTE :** bit de validation de l'interruption externe **RB0/INT** (1=oui, 0=non) .
- **INTF :** indicateur ou drapeau correspondant à l'interruption externe **RB0/INT**.
- **INTEDG :** type du front de déclenchement de l'interruption externe **RB0/INT**.

INTCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

OPTION_REG

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

L'interruption RB0/INT est autorisée si le registre **INTCON = $90**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
1	0	0	1	0	0	0	0

Lorsque l'interruption a eu lieu, le registre **INTCON = $12**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
0	0	0	1	0	0	1	0

Il faut impérativement mettre à **1** le bit **GIE** et mettre à zéro l'indicateur **INTF** à la fin du sous-programme d'interruption pour pouvoir revenir au programme principal et autoriser de nouvelles interruptions sur la broche **RB0/INT**.

A la sortie de la routine d'interruption on écrit la valeur **$90** dans le registre **INTCON**.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**PEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**
1	**0**	**0**	**1**	**0**	**0**	**0**	**0**

EXEMPLE

Pour simplifier l'étude du phénomène d'interruption dans le robot aspirateur, on se limite dans le programme principal du microcontrôleur à la fonction signalisation lumineuse (Chenillard avec les diodes **LED** montées sur la face avant du robot).

L'évènement extérieur (détection d'une collision avec un obstacle) est généré par l'appui sur un bouton poussoir **P1** branché sur la broche **RB0** (le bouton poussoir **P1** joue le rôle du capteur à contact).

En réponse à cet évènement le microcontrôleur doit allumer immédiatement une diode LED **D0** branchée sur la broche **RB1** (D0 joue le rôle de l'ordre d'arrêt des moteurs), faire une attente de 500ms puis il doit continuer son programme principal qui est dans notre cas le chenillard des diodes

SCHÉMA

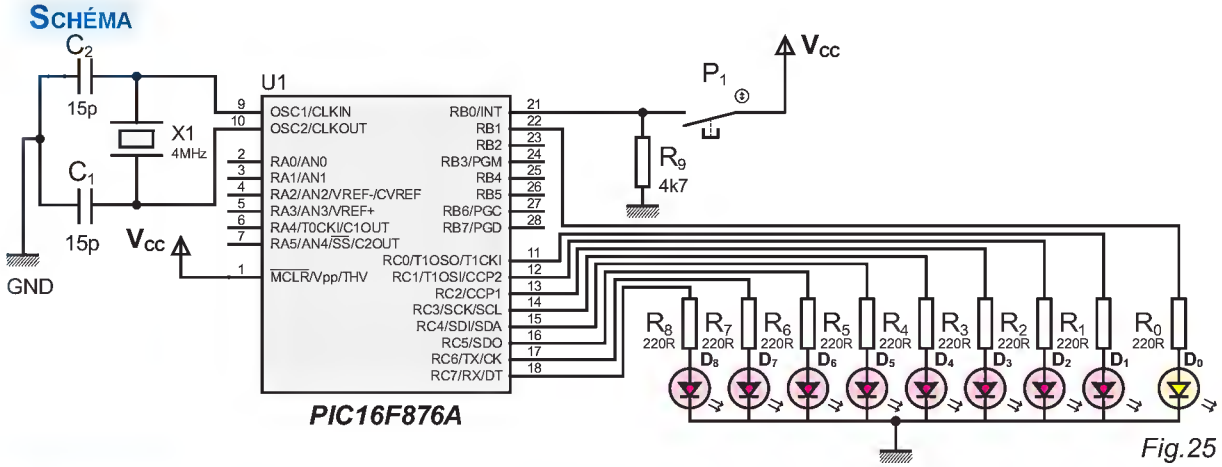


Fig.25

PROGRAMME

Programme en PASCAL

Entête	**program testint;**
Déclara-tions	**var** **D0 : sbit at portb.1;** **diodes : byte at portc;** **i : byte;**

Routine d'interruption	Procedure Interrupt;
	begin
	D0:=1; // allumer la diode LED D0
	delay_ms(500); // attente de 500ms
	D0:=0; // éteindre la diode LED D0
	INTCON.GIE:=1; // Réactivation Globale des interruptions
	INTCON.INTF:=0; // Remise à zéro de l'indicateur INTF
	end;

```
begin
 trisc:=0;   // Portc : sorties.
 trisb.1:=0; // RB1 sortie
 D0:=0;      // Etat initial de la diode LED D0
INTCON:=$90;     // Validation Globale des interruptions
           //validation de RB0/INT
OPTION_REG.INTEDG:=1; // Interruption sur front montant
```

Programme principal

```
while true do
  begin
  // programme principal
  diodes:=%10000000 ;
  for i:=0 to 7 do
     begin
     delay_ms(500);
     diodes := diodes SHR 1;
     end;
  end;
end.
```

14- Interruption RB4 à RB7

L'interruption RB4 à RB7 est provoquée par un changement d'état sur au moins une des entrées **RB4** à **RB7** du port **B**.

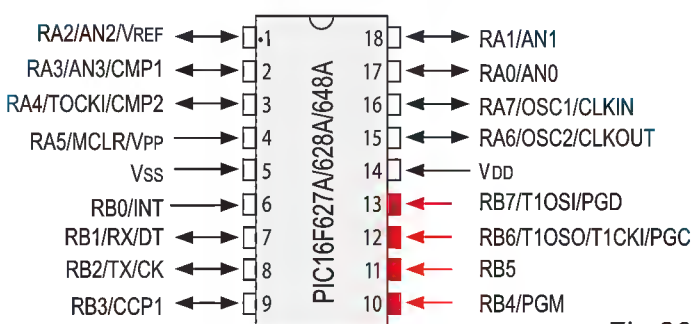


Fig.26

Seules les broches configurées en entrées via le registre **TRISB** peuvent causer cette interruption. (C'est à dire, toute broche de RB4 à RB7 configurée en sortie sera exclue de la comparaison de changement d'état du mécanisme de l'interruption).

Les broches (**RB4** à **RB7**) configurées en entrée sont comparées périodiquement à l'ancienne valeur mémorisée par la dernière lecture du **PORTB** pour générer l'interruption.

L'interruption par changement d'état des broches RB4 à RB7 est gérée par les 3 bits suivants:

- **GIE :** bit de validation globale de toutes les interruptions (1=oui, 0=non).
- **RBIE :** bit de validation de l'interruption externe RB4 à RB7 (**1=oui**, **0=non**)
- **RBIF :** indicateur ou drapeau correspondant à l'interruption externe RB4 à RB7

INTCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**PEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Lorsque le mécanisme de l'interruption **RB4** à **RB7** se déclenche le microcontrôleur verrouille le bit indicateur **RBIF** à **1**.

Une opération de lecture sur le port B est nécessaire pour déverrouiller l'accès au **bit RBIF** afin de pouvoir le remettre à **0**.

L'interruption **RB4 à RB7** est autorisée si le registre **INTCON = $88**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
1	0	0	0	1	0	0	0

Lorsque l'interruption a eu lieu, le registre **INTCON = $09**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
0	0	0	0	1	0	0	1

NB : *Il faut impérativement lire le port* ***B****,* ***mettre à 1*** *le bit* ***GIE*** *et* ***mettre à zéro*** *l'indicateur* ***RBIF*** *à la fin du sous-programme d'interruption pour pouvoir revenir au programme principal et autoriser d'autres interruptions.*

A la sortie de la routine d'interruption et après avoir effectué une lecture du port B, on écrit la valeur **$88** dans le registre **INTCON**.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
1	0	0	0	1	0	0	0

EXEMPLE

Le robot aspirateur est en réalité équipé de 4 capteurs à contacts pour détecter toute collision avec un objet dans les 4 directions (avant, arrière, droite et gauche), ces capteurs à contact sont placés sous les 4 morceaux de pare-choc qui entoure le robot.

Les contacts **CAV**, **CAR**, **CD** et **CG** sont reliés au **port B** (**RB4 à RB7**), le changement d'état d'au moins un des capteurs déclenche une routine d'interruption.

On reprend le même exemple cité dans le paragraphe «***Interruption RB0/INT*** ».

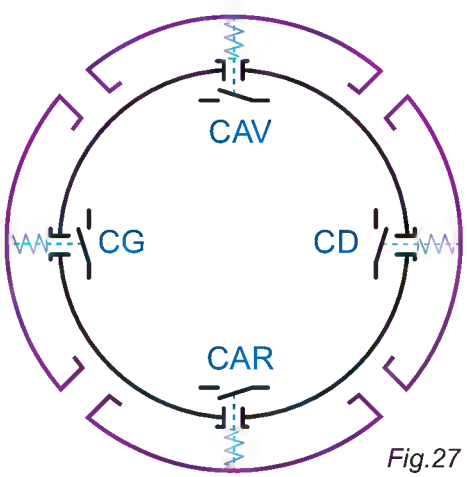


Fig.27

SCHEMA

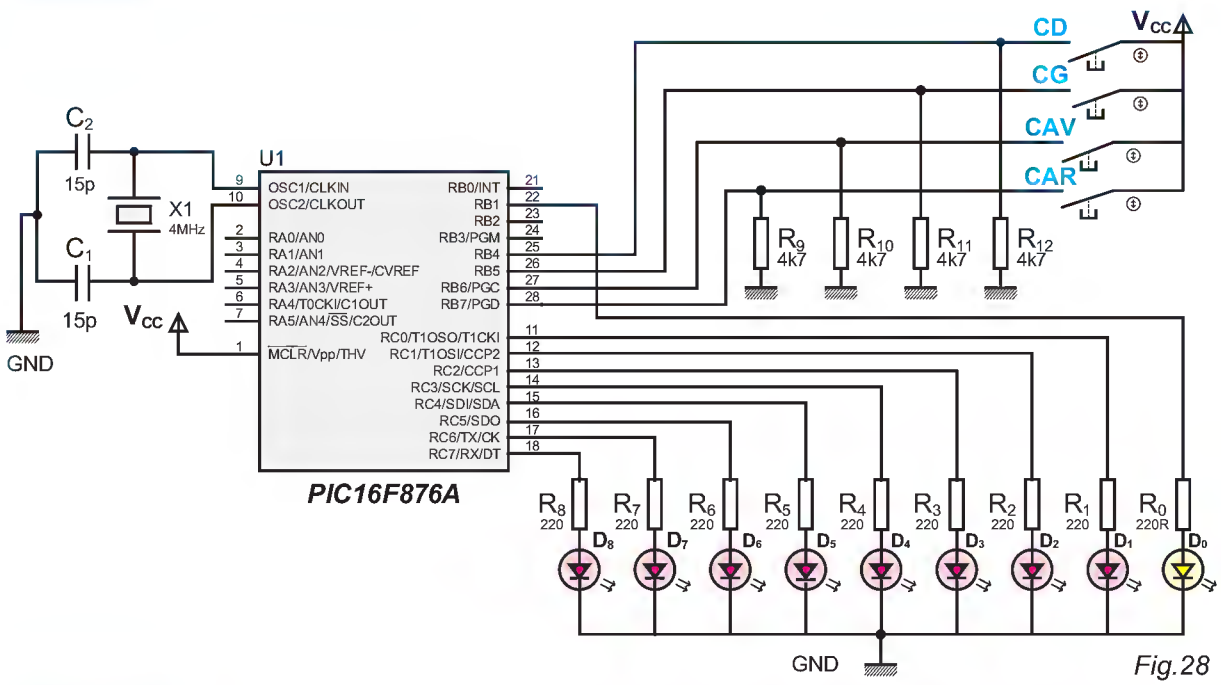


Fig.28

PROGRAMME

Programme en PASCAL

Entête	**program rbi;**
Déclara-tions	**var** **D0 : sbit at portb.1;** **etat : byte;** **diodes : byte at portc;** **i : byte;**
Routine d'interruption	Procedure Interrupt; begin etat:=portb; *// lecture du port B pour déverrouiller* *// l'accès au bit RBIF et pour tester* *// éventuellement le(s) capteur(s) actionné(s)* D0:=1; *// allumer la diode LED D0* delay_ms(500); *// attente de 500ms* D0:=0; *// éteindre la diode LED D0* INTCON.RBIF:=0; *// Remise à zéro de l'indicateur RBIF* INTCON.GIE:=1; *// Réactivation Globale des interruptions* end;

Programme principal

```
begin
TrisC:=0; // Portc : sorties.
 trisb.1:=0; // RB1 sortie
 D0:=0;      // état initial de la diode LED D0
INTCON:=$88;     // Validation Globale des interruptions
              //validation de l'interruption RB4 à RB7
while true do
  begin
  // programme principal
  diodes:=%10000000 ;
  for i:=0 to 7 do
     begin
     delay_ms(500);
     diodes := diodes SHR 1;
     end;
  end;
end.
```

15- Applications à base de PIC

a. Chariot de soudage mobile sur rail fixe

Une société de conception et fabrication de systèmes industriels a reçu une demande de la part de trois clients pour concevoir et réaliser la partie commande d'un chariot de soudage mobile sur rail fixe.

Fig. 29

Pour le même chariot les trois clients ont souhaité chacun un mode de fonctionnement différent.

Mode de fonctionnement pour le client **N1**:

- Un appui sur le bouton poussoir «**m**» provoque le déplacement du chariot et de la torche de soudage «**MIG**».
- Lorsqu'il atteint l'extrémité droite de son rail, il rebrousse chemin.

Mode de fonctionnement pour le client **N2**:

- Un appui sur le bouton poussoir «**m**» provoque le déplacement du chariot et de la torche de soudage «**MIG**»..
- Lorsqu'il atteint l'extrémité droite de son rail, il marque une attente de 5 secondes puis il rebrousse chemin en activant la torche de soudage «**MIG**».

Mode de fonctionnement pour le client **N3**:

- Un appui sur le bouton poussoir «**m**» provoque le déplacement du chariot et active la torche de soudage «**MIG**».
- Le chariot effectue 4 cycles de va et vient puis s'arrête.

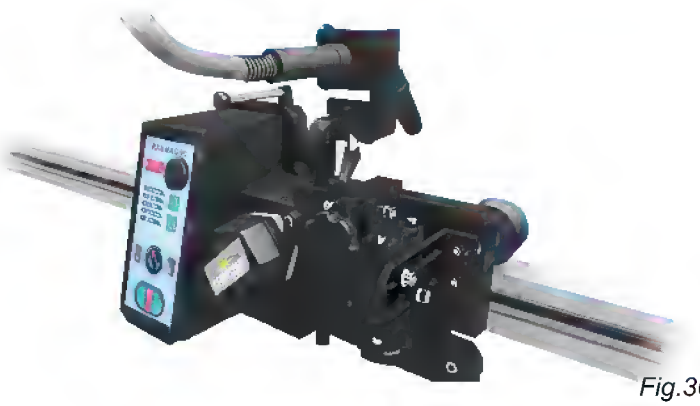


Fig.30

Liste des capteurs et pré-actionneurs :

PREACTIONNEURS		CAPTEURS	
Désignation	Symbole	Désignation	Symbole
Translation du chariot	KM_1	Bouton mise en marche	m
Torche de soudage MIG	KM_2	Fin de course droite	a_1
Sens avant pour la translation du chariot	AV	Fin de course gauche	a_0
Sens arrière pour la translation du chariot	AR		

Pour des raisons de sécurité les capteurs de fin de course $\mathbf{a_0}$ et $\mathbf{a_1}$ sont de type contact fermé au repos (en cas de détachement ou coupure de leurs câbles de connexions, la partie commande traite cet état en tant que contact actionné et arrête le déplacement du chariot)

Une analyse des solutions possibles a abouti à l'adoption d'une solution qui consiste à utiliser une seule carte de commande à base de microcontrôleur **16F84A** avec trois programmes différents (un programme par client).

Table d'affectation des entrées/sorties : **PIC16F84A**

Entrées		Sorties	
Système	microcontrôleur	Système	microcontrôleur
m	RA0	KM_1	RB0
a_1	RA1	KM_2	RB1
a_0	RA2	AV	RB2
		AR	RB3

NB : *seules la première et la deuxième solution seront traitées dans la suite de ce cours. La possibilité du client* ***N3*** *sera traitée dans le manuel d'activités.*

Mode de fonctionnement pour le client **N1**

Grafcet de point de vue partie commande:

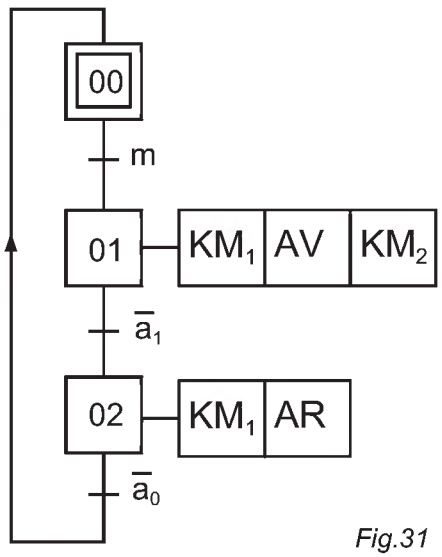


Fig.31

ALGORITHME

Algorithme chariot ;
Variables :
m : bit du registre PORTA affecté à RA0 ;
a0 : bit du registre PORTA affecté à RA1 ;
a1 : bit du registre PORTA affecté à RA2 ;
KM1 : bit du registre PORTB affecté à RB0 ;
KM2 : bit du registre PORTB affecté à RB1 ;
X0, X1, X2 : bit mémoire ; // étapes du Grafcet
DEBUT
TrisA ← $FF ; // tout le porta est configuré en entrée
TrisB ← $00 ; // tout le portb est configuré en sortie
KM1 ← 0 ; // état initial des sorties
KM2 ← 0 ;
X0 ← 1 ; // état initial des étapes du Grafcet
X1 ← 0 ;
X2 ← 0 ;
TANTQUE vrai FAIRE
DEBUT
// Traitement des étapes
SI (X0 = 1) ET (m = 1) ALORS
DEBUT
X0 ← 0 ;

```
                X1 ← 1 ;
               FIN ;
    SI (X1 = 1) ET (a1 = 0)  ALORS
               DEBUT
                X1 ← 0 ;
                X2 ← 1 ;
               FIN ;

    SI (X2 = 1) ET (a0 = 0)  ALORS
               DEBUT
                X2 ← 0 ;
                X0 ← 1 ;
               FIN ;
    // Traitement des sorties
    SI (X1 = 1) OU (X2 = 1)  ALORS KM1 ← 1 SINON KM1 ← 0 ;
    SI (X1 = 1) ALORS AV  ← 1 SINON AV  ← 0 ;
    SI (X1 = 1) ALORS KM2 ← 1 SINON KM2 ← 0 ;
    SI (X2 = 1) ALORS AR  ← 1 SINON AR  ← 0 ;

    FIN ;
FIN.
```

PROGRAMME

```
program chariot;
var
m   : sbit at RA0_bit ;
a0  : sbit at RA1_bit ;
a1  : sbit at RA2_bit ;
KM1 : sbit at RB0_bit ;
KM2 : sbit at RB1_bit ;
AV  : sbit at RB2_bit ;
AR  : sbit at RB3_bit ;
X0, X1, X2 : bit;  // étapes du grafcet
begin
     Trisa := $FF ; // tout le porta est configuré en entrée
     Trisb := $00 ; //tout le portb est configuré en sortie
     KM1 := 0 ;     // état initial des sorties
     KM2 := 0 ;
     AV := 0 ;
     AR := 0 ;
     X0 :=  1 ;       // état initial des étapes du grafcet
     X1 :=  0 ;
     X2 :=  0 ;
while true do
```

```
begin
  // traitement des étapes
      if (X0 = 1) and (m = 1)  then
          begin
            X0 := 0 ;
            X1 := 1 ;
          end ;
       if (X1 = 1) and (a1 = 0)  then
          begin
            X1 := 0 ;
            X2 := 1 ;
          end ;
      if (X2 = 1) and (a0 = 0)  then
          begin
            X2 := 0 ;
            X0 := 1 ;
          end ;
  // traitement des sorties
    if (X1 = 1)OR (X2 = 1) then KM1 := 1 else KM1 := 0 ;
    if (X1 = 1) then AV := 1 else AV := 0 ;
    if (X1 = 1) then KM2 := 1 else KM2 := 0 ;
    if (X2 = 1) then AR := 1 else AR := 0 ;
  end ;
end.
```

Mode de fonctionnement pour le client **N2**:
Grafcet de point de vue partie commande

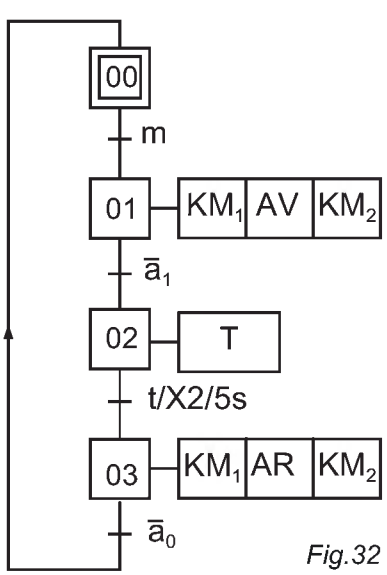


Fig.32

ALGORITHME 2

```
Algorithme chariot ;
Variables :
   m   : bit du registre port A affecté à RA0 ;
  a0   : bit du registre port A affecté à RA1 ;
  a1   : bit du registre port A affecté à RA2 ;
  KM1  : bit du registre port B affecté à RB0 ;
 KM2   : bit du registre port B affecté à RB1 ;
 AV    : bit du registre port B affecté à RB2 ;
 AR    : bit du registre port B affecté à RB3 ;
 X0, X1, X2,x3 : bit ;  // étapes du Grafcet
 T : bit ;              // temporisateur T
DEBUT
 TrisA ← $FF ; // tout le porta est configuré en entrée
 TrisB ← $00 ; // tout le portb est configuré en sortie
 KM1 ← 0 ;  KM2 ← 0 ; AV ← 0 ;AR ← 0 ;  // état initial des sorties
X0 ← 1 ; X1 ← 0 ; X2 ← 0 ; X3 ← 0 ; T ← 0 ;  // état initial des étapes du Grafcet
TANTQUE vrai FAIRE
   DEBUT
   // Traitement des étapes
   SI (X0 = 1) ET (m = 1)  ALORS
              DEBUT
              X0 ← 0 ;
              X1 ← 1 ;
              FIN ;
   SI (X1 = 1) ET (a1 = 0)  ALORS
              DEBUT
              X1 ← 0 ;
              X2 ← 1 ;
              FIN ;
   SI (X2 = 1) ET (T = 1)  ALORS
              DEBUT
              X2 ← 0 ;
              X3 ← 1 ;
              FIN ;
   SI (X3 = 1) ET (a0 = 0)  ALORS
              DEBUT
              X3 ← 0 ;
              X0 ← 1 ;
              FIN ;
   // Traitement des sorties
   SI (X1 = 1) OU (X3 = 1) ALORS KM1 ← 1 SINON KM1 ← 0 ; // équation KM1
   SI (X1 = 1) OU (X3 = 1) ALORS KM2 ← 1 SINON KM2 ← 0 ; // équation KM2
   SI (X1 = 1) ALORS AV ← 1 SINON AV ← 0 ;          // équation AV
```

```
   SI (X2 = 1) ALORS T ← 1 SINON T ← 0 ;          // équation T
   SI (X3 = 1) ALORS AR ← 1 SINON AR ← 0 ;       // équation AR
   // Traitement des temporisateurs
   SI (T = 1) ALORS Attente(5s);
   FIN ;
FIN.
```

PROGRAMME 2

```
program chariot2;
var
m   : sbit at RA0_bit ;
a0  : sbit at RA1_bit ;
a1  : sbit at RA2_bit ;
kM1 : sbit at RB0_bit ;
kM2 : sbit at RB1_bit ;
AV : sbit at RB2_bit ;
AR : sbit at RB3_bit ;
X0, X1, X2, X3 : bit;  // étapes du grafcet
T : bit ;                  // temporisateur T
begin
trisa := $ff ;    // tout le porta est configuré en entrée
trisb := $00 ;    // tout le portb est configuré en sortie
kM1 := 0 ;  kM2 := 0 ; AV  := 0 ; AR  := 0 ;  // état initial des sorties
X0 :=  1 ;  X1 :=  0 ; X2 :=  0 ; X3 :=  0 ; T:=0;  // état initial des étapes du grafcet
while true do
   begin
   // traitement des étapes
   if (X0 = 1) and (m = 1)  then
               begin
               X0 := 0 ;
               X1 := 1 ;
               end ;
   if (X1 = 1) and (a1 = 0)  then
               begin
               X1 := 0 ;
               X2 := 1 ;
               end ;
   if (X2 = 1) and (T = 1)  then
               begin
               X2 := 0 ;
               X3 := 1 ;
               end ;
```

```
    if (X3 = 1) and (a0 = 0)  then
                begin
                X3 := 0 ;
                X0 := 1 ;
                end ;
    // Traitement des sorties
      if (X1 = 1) OR (X3 = 1) then KM1 := 1 else KM1 := 0 ;    // équation KM1
      if (X1 = 1) OR (X3 = 1) then KM2 := 1 else KM2 := 0 ;  // équation KM2
      if (X1 = 1) then AV := 1 else AV := 0 ;                  // équation AV
      if (X2 = 1) then T := 1 else T := 0 ;                   // équation T
      if (X3 = 1) then AR := 1 else AR := 0 ;                 // équation AR
    // Traitement des temporisateurs
     if (T = 1) then delay_ms(5000) ;
    end ;
end.
```

b. Système de gestion d'une file d'attente

Les salles d'attente de certaines administrations publiques, disposent généralement d'un système électronique de gestion de la file d'attente.

Ce système est généralement composé d':

- un distributeur de tickets ;
- un afficheur à deux digits placé en haut du guichet ;
- un bouton poussoir **B** pour incrémenter le numéro à servir ;
- un avertisseur sonore incorporé dans l'afficheur ;
- un bouton **RAZ** de remise à zéro.

Fig. 33

SCHÉMA

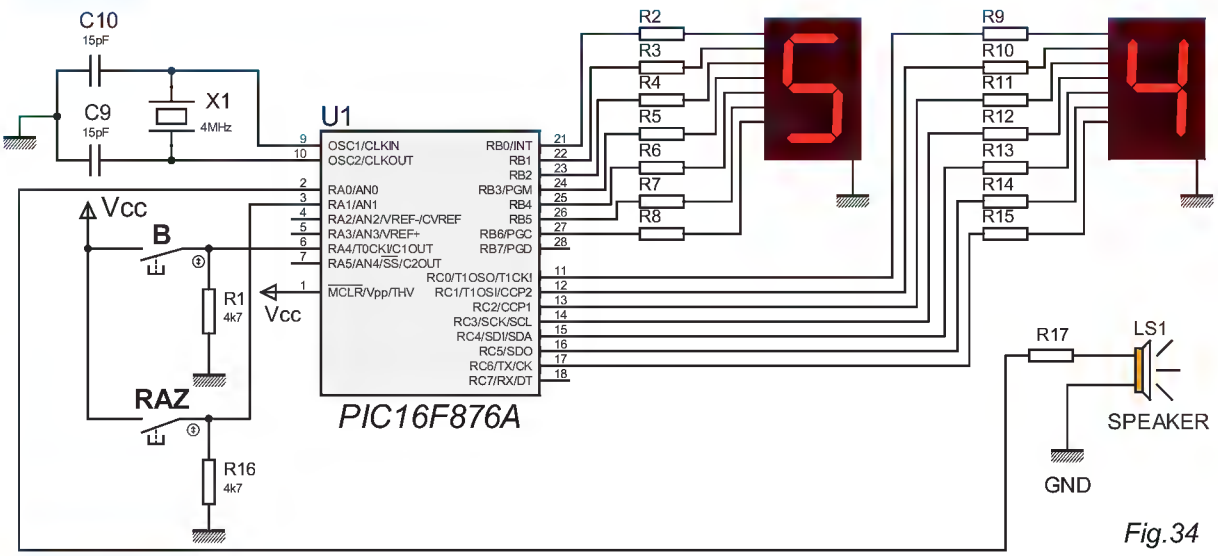


Fig.34

PROGRAMME

```
program comptage;
const code7seg : array[10] of byte  = ($3F,$06,$5B,$4F,$66,$6D,$7D,$07,$7F,
$6F) ; // code 7 segment
var
   uni : byte;
   diz : byte;
   RAZ : sbit at RA1_bit; // bouton de remise à zéro
   B  : sbit at RA4_bit; // bouton B
begin
  TrisA:=$FE;  // RA0 sortie le reste entrées
  ADCON1:=$87; // Tout le portA est numérique
  TrisB:=$80;  // RB7 entrée le reste Sortie
  TrisC:=$80;  // RC7 entrée le reste Sortie
  OPTION_REG := $F8;     // TMR0 compteur à front montant
                         // sur RA4 pas de pré-diviseur
  Sound_Init(PORTA,0);   // initialisation de la procédure son
  while true do
    begin
      if TMR0 > 99 then TMR0:=0;
      uni := TMR0 mod 10;  // extraction des dizaines du nombre dans TMR0
      diz := (TMR0 div 10) mod 10; // extraction des unités
      Portc:= code7seg[uni];       // affichage unités
      Portb:= code7seg[diz];       // affichage dizaines
      if (RZ = 1) then TMR0:=0;    // remise à zéro
      if (B = 1) then sound_play(1200,1000); // émettre un son (de 1,2KHz durant 1s)
    end;
end.
```

C. RÉSUMÉ

✍ **La structure générale** d'un programme en Mikropascal pro est comme suit :

Entête

Program nom_programme ;

Déclarations

-Déclarations de constantes, types, variables utilisés dans le programme.
-Déclarations des procédures et fonctions utilisées dans le programme.

Corps du programme

```
Begin
- Activation des interruptions utilisées
- Configuration des entrées/sorties.
- Initialisation des sorties utilisées.
While true do
        begin
            Instruction 1;
                Instruction 2;
            ……….
        end ;
End.
```

✍ **La conversion analogique numérique :**

Le registre **ADCON1** permet de sélectionner et d'activer les entrées analogiques.

Exp : adcon1:=%10000100 ; // choix de RA0/AN0 (entrée analogique)

- La procédure adc_init() permet d'initialiser le module convertisseur CAN
- La fonction adc_read(0) permet la lecture du convertisseur

✍ **La Modulation de Largeur d'Impulsion (MLI)**

Pour chaque sortie PWM appelée CCPx on a :

PWMx_init : permet d'initialiser le module PWM de la sortie CCPx

PWMx_start : Démarrage du PWM et sortie du signal (la broche CCPx)

PWMx_Set_duty(N) : Change le rapport cyclique a du signal sortant sur la broche CCPx avec a = N / 255 et N variant de 0 à 255.

PWMx_stop : Arrête le module PWM de la sortie CCPx

✍ **Les Interruptions**

L'interruption externe RB0/INT se produit sur le front montant ou descendant d'une impulsion appliquée sur l'entrée RB0.

L'interruption RB4 à RB7 est provoquée par un changement d'état sur au moins une des entrées RB4 à RB7 du port B.

Le registre INTCON (INTerrupt CONtroller) est le registre principal de contrôle et de gestion des interruptions.

D. EVALUATION

I- Contrôle des connaissances

1- Quelle est la ou les différences entre procédure et fonctions ?
2- Quelle est la taille en bits d'une variable de type WORD ?
3- Quelles sont les différents types d'interruptions ?
4- Donner la liste des instructions nécessaires pour générer un signal MLI sur la broche CCP1 avec un rapport cyclique égale à 0.75 ?
5- Quel est le rôle du bit ADFM du registre ADCON1 ?

II- Exercice résolu : calculatrice basique

Certaines calculatrices basiques, sous forme de jouets éducatifs ou gadgets destinés aux enfants, ne réalisent que les opérations mathématiques de base à savoir : addition, soustraction, multiplication et division des nombres entiers.

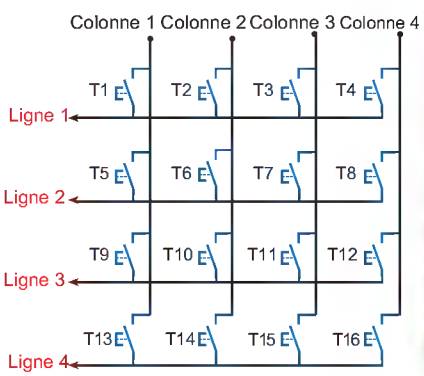
Elles sont constituées généralement d'un afficheur LCD et d'un clavier 16 touches.

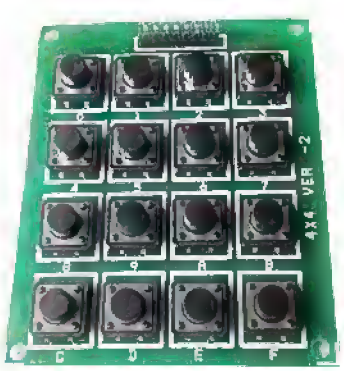
On désire gérer les différentes fonctions de ce type de calculatrice en utilisant un microcontrôleur de type **PIC16F628A**.

Clavier matriciel

Le schéma du clavier matriciel est à 16 touches. Il dispose uniquement de 8 broches pour la gestion de ses touches.

Les 8 broches sont organisées en 4 colonnes et 4 lignes.



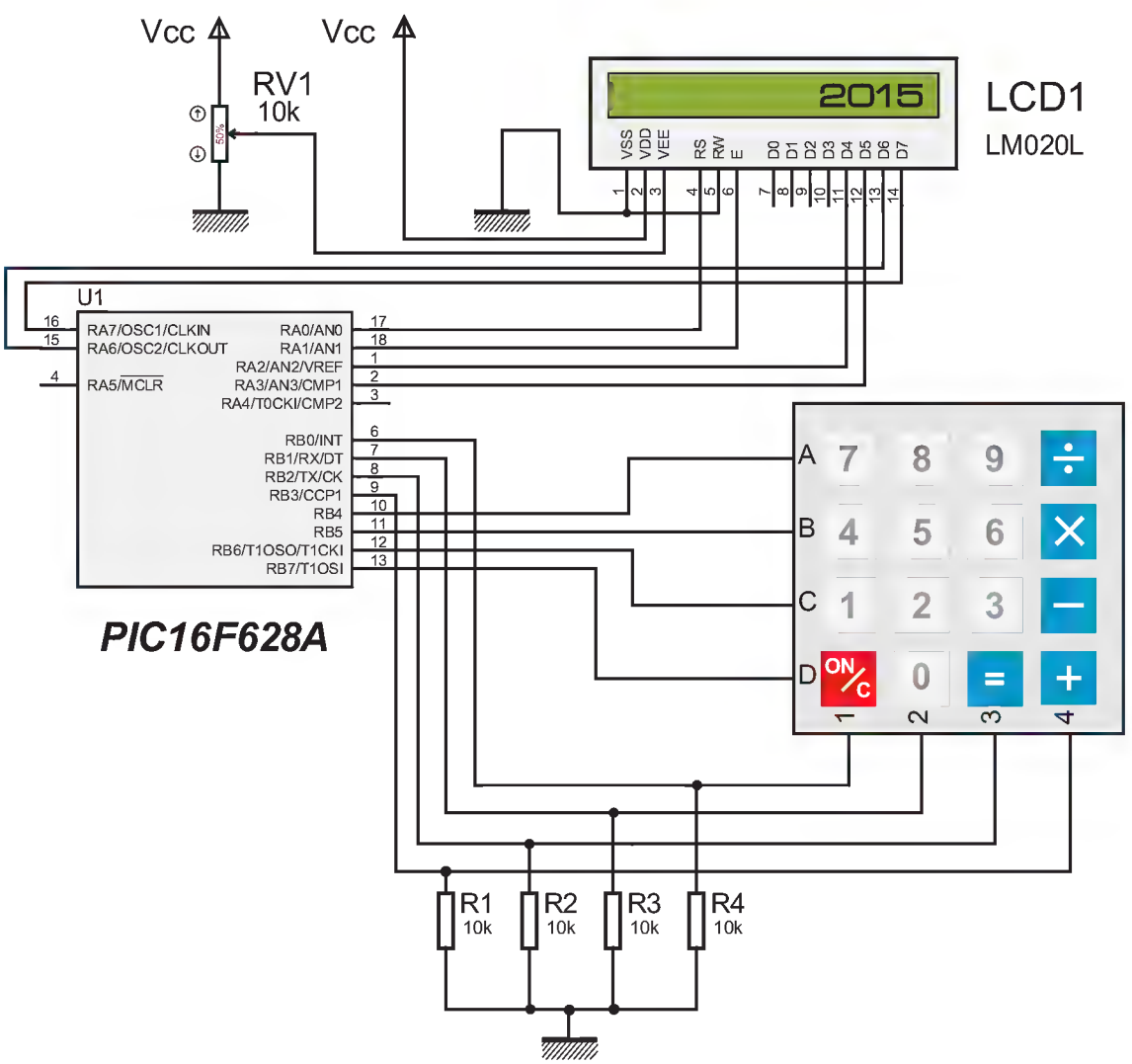


L'appui sur une touche réalise un contact direct entre la ligne et la colonne de cette touche.

Pour détecter l'action sur une touche on procède comme suit :

- on fait un balayage sur les lignes c'est-à-dire on envoie les séquences suivantes (1000, 0100, 0010, 0001);
- on lit l'état des colonnes sur 4 bits.

Schéma



Ecrire un programme en mikropascal pour gérer ce système.

III- Exercices à résoudre

EXERCICE N°1:GRAFCET

A partir du programme ci-après, déterminer la table d'affectation des entrées/sorties et tracer le grafcet correspondant :

```
program Grafcet;
Var X0,X1,X2,X3:byte ; // Déclaration des étapes du GRAFCET
    Dcy : sbit at Portb.0;
    a1  : sbit at Portb.1;
    a2  : sbit at Portb.2;
    a3  : sbit at Portb.3;
    KM1 : sbit at porta.0;
    KM2 : sbit at porta.1;
    KM3 : sbit at porta.2;
begin
CMCON:= $07; // Désactivation du comparateur
TrisA := $F8 ; // RA0,RA1,RA2 sont des sorties ; les autres broches sont des entrées.
TrisB := $FF ; // Toutes les broches du port B sont des entrées.
PortA:= 0 ; // Etat initial des sorties.
X0:=1; // Initialement l'étape « X0» est active.
X1:=0; // Initialement l'étape « X1» est non active.
X2:=0; // Initialement l'étape « X2» est non active.
X3:=0; // Initialement l'étape « X3» est non active.
while (1=1) do // Boucle infinie.
begin
if ((X0 =1) and (Dcy=1)) then // Condition de franchissement de la
// première transition qui est :
// étape «X0» active et réceptivité «Dcy» vraie.
begin
X0:=0; // Désactivation de l'étape «X0».
X1:=1; // Activation de l'étape «X1».
end ;
if ((X1 =1) and (a1=1)) then // Condition de franchissement de la
// deuxième transition qui est :
// étape «X1» active et réceptivité «a1» vraie.
begin
X1:=0; // Désactivation de l'étape « X1».
X2:=1; // Activation de l'étape « X2».
end ;
if ((X2 =1) and (a2=1)) then // Condition de franchissement de la
// troisième transition qui est :
```

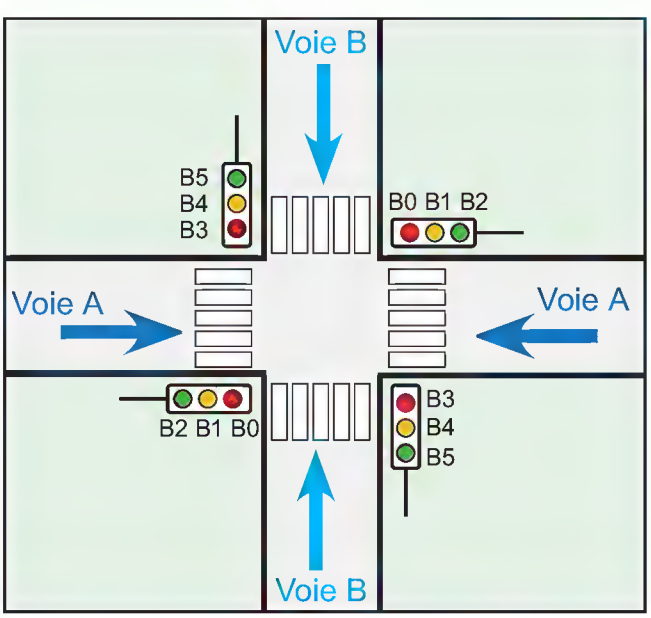
```
// étape «X2» active et réceptivité «a2» vraie.
begin
X2:=0; // Désactivation de l'étape «X2».
X3:=1; // Activation de l'étape «X3».
end ;
if ((X3 =1) and (a3=1)) then // Condition de franchissement de la
// quatrième transition qui est :
// étape «X3» active et réceptivité «a3» vraie.
begin
X3:=0 ; // Désactivation de l'étape «X3».
X0:=1 ; // Activation de l'étape «X0».
end ;
if (X1=1) then KM1:=1 else KM1:= 0 ; // KM1
if (X2=1) then KM2:=1 else KM2:= 0 ; // KM2
if ((X1=1) OR (X3=1))  then KM3:=1 else KM3:= 0 ; // KM3
end;
end.
```

EXERCICE N°2: FEUX DE CROISEMENT

On désire gérer les feux de croisement de deux voies **A** et **B** selon les deux modes de fonctionnement **Jour** et **Nuit**.

Les différents feux sont associés au Port B d'un microcontrôleur de type 16F628A.

Un commutateur (**J**) Jour/nuit est connecté à la broche **RA0**.



Mode Jour (J=1)

Voie A	V	O	R	R	R	R
Voie B	R	R	R	V	O	R
Temps	4s	2s	1s	4s	2s	1s

Mode nuit (J=0)

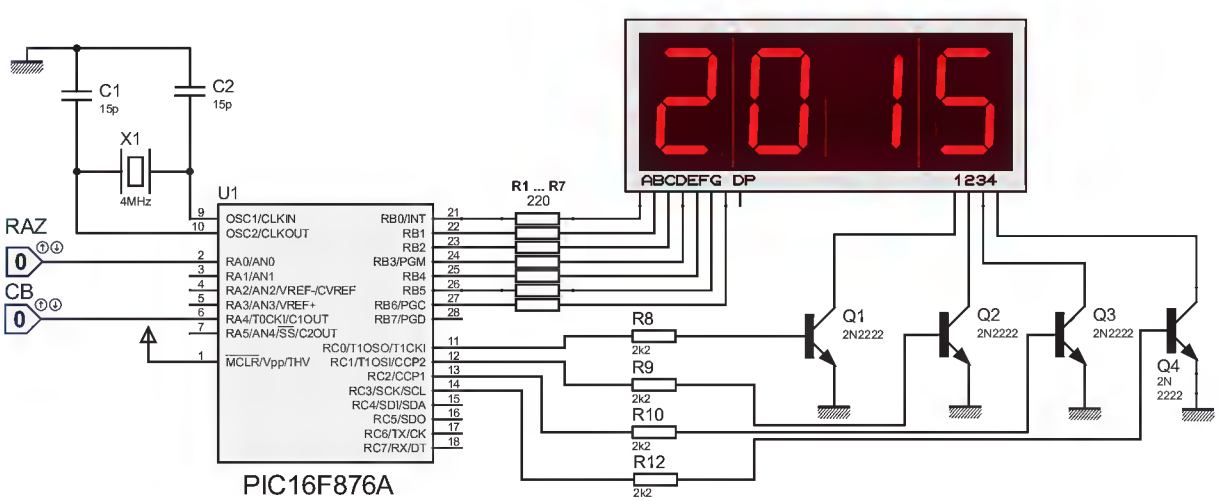
Voie A	O	-
Voie B	O	-
Temps	1s	1s

1- Proposer un schéma de simulation.
2- Ecrire un programme en mikropascal pour gérer ce système.

Exercice N°3 : Compteuse de billets de banque.

Les compteuses de billets sont des systèmes électromécaniques destinées au comptage de billets de banques. Elles sont généralement utilisées dans les secteurs professionnels manipulant de grandes quantités de billets tels que les transporteurs de fonds, les stations-services, les grandes surfaces commerciales, les restaurants, etc.

Ce système est géré par une carte de commande à base de microcontrôleur, dont le schéma structurel est le suivant :



Le capteur de billets **CB** génère une impulsion de courte durée lors du passage d'un billet, incrémentant ainsi le dispositif de comptage. Un bouton **RAZ** permet à l'utilisateur de remettre le compteur à zéro.

1- Justifier l'emploi de la broche RA4 du microcontrôleur **PIC16F876A** comme entrée de comptage par le biais du capteur de billets **CB**.

2- Décrire le fonctionnement de ce système par un algorithme tenant compte des deux fonctions (comptage et affichage).

3- Traduire l'algorithme en un programme Mikropascal.

IV- Correction de l'exercice résolu

PROGRAMME :

```
program piccalc;
var
// connections de l'afficheur Lcd

 LCD_RS : sbit at RA0_bit;
 LCD_EN : sbit at RA1_bit;
 LCD_D4 : sbit at RA2_bit;
 LCD_D5 : sbit at RA3_bit;
 LCD_D6 : sbit at RA6_bit;
 LCD_D7 : sbit at RA7_bit;
 LCD_RS_Direction : sbit at TRISA0_bit;
 LCD_EN_Direction : sbit at TRISA1_bit;
 LCD_D4_Direction : sbit at TRISA2_bit;
 LCD_D5_Direction : sbit at TRISA3_bit;
 LCD_D6_Direction : sbit at TRISA6_bit;
 LCD_D7_Direction : sbit at TRISA7_bit;
 keypadPort : byte at PORTB; // connection du clavier
 V1, V2, R : dword; // variables de calcul
 txt : string[10]; //  Texte pour affichage
 kp,touche, operation : byte; // variables pour le clavier
procedure decodeclavier; //procédure de décodage de la touche appuyée
  begin
  case kp of
     1 : touche := 7;
     2 : touche := 8;
     3 : touche := 9;
     4 : touche := '/';
     5 : touche := 4;
     6 : touche := 5;
     7 : touche := 6;
     8 : touche := '*';
     9 : touche := 1;
     10: touche := 2;
     11: touche := 3;
     12: touche := '-';
     13: touche := 'c';
     14: touche := 0;
     15: touche := '=';
     16: touche := '+';
    end;
  end;
```

```
begin
cmcon:=$07;    // désactivation du comparateur
lcd_init();    // initialisation de l'afficheur LCD
Keypad_Init(); // initialisation du clavier
lcd_cmd(_LCD_CURSOR_OFF); // désactivation du curseur de l'LCD
kp:=0;
operation:=0;
V1:=0;V2:=0;
LongWordToStr(V1,txt);  // transformation du nombre V1 en texte
lcd_out(1,7,txt);       // affichage de V1 sur l'LCD
while true do
  begin
  kp := Keypad_Key_Click();  // Appui sur une touche du clavier
  if kp<>0 then
    begin
    decodeclavier;
    case touche of
    'c' : begin
            V1:=0;
            V2:=0;
            operation:=0;
            LongWordToStr(V1,txt);
            lcd_out(1,7,txt);
          end;
    '+' : begin
            operation:='+';
            touche:=0;
            V2:=0;
            LongWordToStr(V2,txt);
            lcd_out(1,7,txt);
          end;
    '-' : begin
            operation:='-';
            touche:=0;
            V2:=0;
            LongWordToStr(V2,txt);
            lcd_out(1,7,txt);
          end;
    '*' : begin
            operation:='*';
            touche:=0;
            V2:=0;
            LongWordToStr(V2,txt);
            lcd_out(1,7,txt);
```

```
        end;
    '/' : begin
            operation:='/';
            touche:=0;
            V2:=0;
            LongWordToStr(V2,txt);
            lcd_out(1,7,txt);
          end;
    end;
    if (operation = 0) and (touche <> 'c') and (touche <> '=') then
       begin
         V1:=10*V1+touche;
         LongWordToStr(V1,txt);
         lcd_out(1,7,txt);
        end;
    if (operation <> 0) and (touche <> '=') and (touche <> 0) then
        begin
         V2:=10*V2+touche;
         LongWordToStr(V2,txt);
         lcd_out(1,7,txt);
        end;
    if touche = '=' then
      begin
      if operation='+' then R:=V1+V2;
      if (operation='-') and (V1 > V2) then R:=V1-V2;
      if (operation='-') and (V1 < V2) then
                                  begin
                                    R:=0;
                                    V2:=0;
                                  operation:=0;
                                end;
      if operation='*' then R:=V1*V2;
      if operation='/' then R:=V1/V2;
      LongWordToStr(R,txt);
      lcd_out(1,7,txt);
      V1:=R;
      end;
    end;
  end;
end.
```