Cycle de formation : BT Electrotechnique
Niveau : 3ème A
Matière : Automatisme
Rédigé par : M. ADIKO à adikoaugustin@gmail.com

LES AUTOMATES PROGRAMMABLES INDUSTRIELS (API)

1) PROBLEMATIQUE

La partie commande des systèmes automatisés de production (S.A.P) était réalisée en logique câblée, par l'utilisation de relais électromagnétiques, de systèmes pneumatiques, ou de composants électroniques. Avec l'évolution des automatismes, la logique câblée s'est avérée chère, peu souple et peu flexible, et obsolète.

La solution à cette situation est venue de la logique programmée par l'utilisation de systèmes à base de microprocesseur permettant une modification aisée de la partie commande des SAP. La mise en œuvre de la logique programmée, plus souple et flexible, requiert un système à base de microprocesseur :

- Un automate programmable industriel,
- Un micro-ordinateur,
- Un kit à microprocesseur,
- Un microcontrôleur.

2) DEFINITION

Un **Automate Programmable Industriel (A P I)** est une machine électronique programmable, destinée à piloter dans un environnement industriel et en temps réel un **Système Automatisé de Production (S A P)**.

Utilisé comme l'un des matériels de mise en œuvre de la logique programmée, l'API permet de traiter toutes les informations en vue de coordonner les tâches exécutées par la partie opérative d'un système de production. Les automates programmables industriels ont des avantages certains sur les anciennes technologies de commande des S A P. Ce sont entre autres :

- Leur fiabilité
- Leur miniaturisation
- La flexibilité dans leur exploitation
- Leur intégration aisée dans un réseau informatique
- Etc.

EXEMPLES D'API

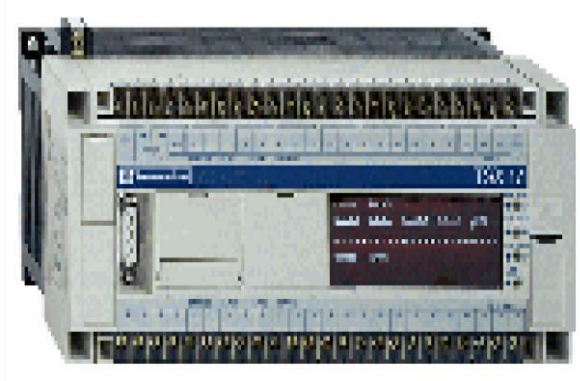
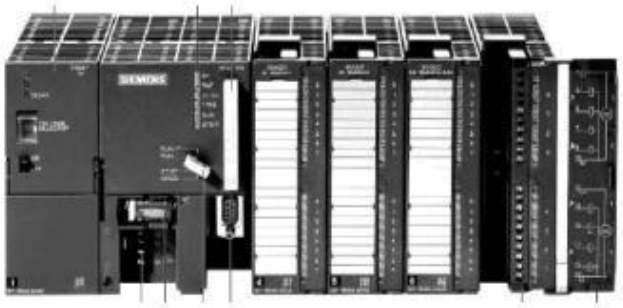


Figure 1 : Automate TSX 17

Figure 2 : Automate TSX 37

AUTOMATE SIEMENS S7-200

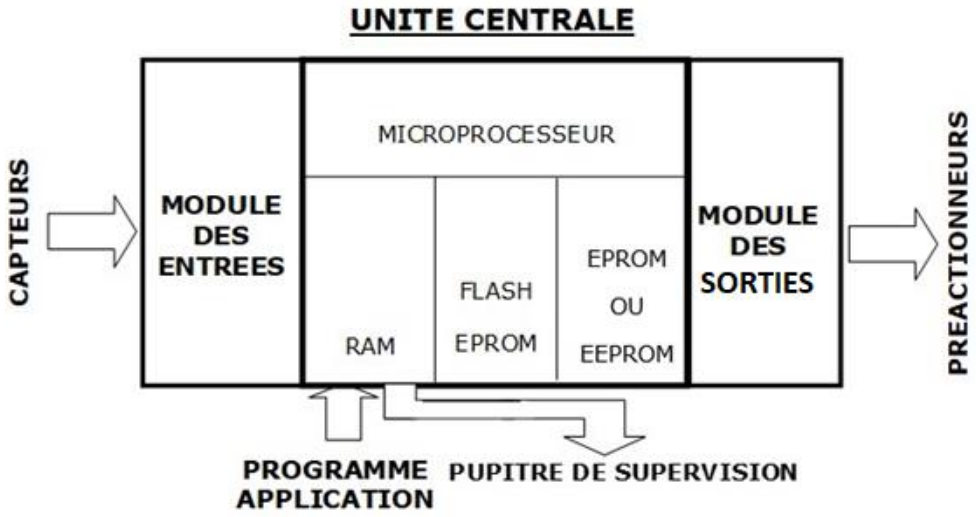


Automate modulaire de siemens

3) DESCRIPTION D'UN API

3.1- STRUCTURE

L'architecture interne de tout API peut se présenter comme suit :



3.2- LE MICROPROCESSEUR

C'est l'unité fonctionnelle de l'API, il interprète et exécute les instructions du programme. Ces instructions sont effectuées les unes après les autres, séquencées ou rythmées par une horloge. Exemple : Si deux actions doivent être simultanées, l'API les traite successivement.

C'est ce circuit intégré, le **microprocesseur, qui** permet à l'API d'effectuer toutes les opérations logiques et arithmétiques ainsi que tous les traitements numériques nécessaires à la conduite du SAP. Il possède des voies de communication avec l'extérieur :

- Raccordement avec l'outil de programmation (console de programmation, ordinateur, …)
- Raccordement sur un réseau de communication inter-automates

3.3- L'ESPACE MEMOIRE

C'est la mémoire centrale ou la zone mémoire. Elle est conçue pour :

- Recevoir les informations issues des capteurs d'entrées,
- Recevoir les informations générées par le processeur et destinées à la commande des sorties.
- Recevoir et conserver le programme du process.

Il comporte trois types de mémoires qui remplissent des fonctions différentes :

a. Mémoire système

Cette mémoire, présente dans le cas d'automates à microprocesseurs, est utilisée pour stocker le système d'exploitation et elle est programmée en usine par le constructeur. Elle peut donc sans problème être réalisée en technologie PROM (c'est-à-dire programmable une seule fois, sans possibilité d'effacement) voire ROM (mémoire morte accessible uniquement en lecture).

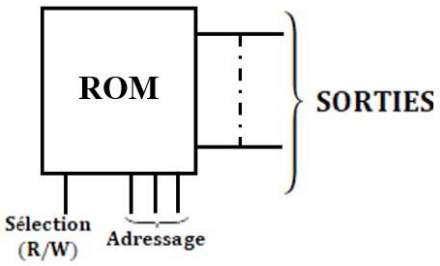
b. Mémoire de programme

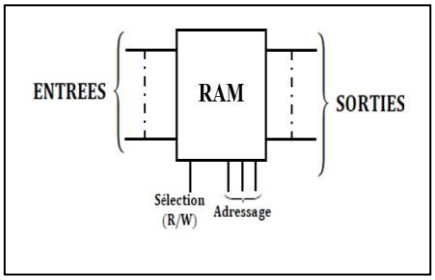
Cette mémoire est utilisée pour stocker le programme. Elle est en général de type EEPROM (electrically erasable PROM : mémoires mortes reprogrammables effacement électrique).

c. Mémoire de données

Elle est utilisable en lecture-écriture des données pendant le fonctionnement. C'est une mémoire de type RAM (mémoire vive dans laquelle on peut lire, écrire et effacer) utilisant une technologie spéciale (CMOS) à très faible consommation électrique du moins, à l'état de repos et elle nécessite pile ou une batterie de sauvegarde.

Elle enregistre les variables d'entrée (états des capteurs), les variables de sortie (ordres aux actionneurs), variable Interne (résultats de fonctions, résultats intermédiaires). On peut, en règle générale, augmenter la capacité mémoire par adjonction de barrettes mémoires type PCMCIA.





Remarques :

1. **D'une façon générale, on dispose de différents types de mémoire. Ainsi, il existe les mémoires vives (RAM ;** SRAM (RAM statique) ; DRAM (RAM dynamique**) et les mémoires mortes (** ROM (Read Only Memory) ; EAROM (Electrically Alterrable ROM); MROM (ROM programmable par masque) ; PROM (Programmable Read Only Memory) ; FPROM (Flash Programmable Read Only Memory); OTP (One Time PROM); EPROM (Erasable PROM, effacement aux rayons ultraviolets) ; EEPROM (Electrically EPROM); FLASH EPROM (reprogrammable après effacement)).
   A cela, il faut ajouter les mémoires à accès séquentiel (Mémoire FIFO (First In First Out fonctionnement équivalent à celui d'une file d'attente) ; Mémoire LIFO (Last In First Out, fonctionnement équivalent à celui d'une pile d'assiettes).

2. De nos jours, la capacité des mémoires s'exprime comme suit :
   - l'octet = 2^3 bits = 8 bits (noté 1o)
   - le Kilo-octet = 2^{10} octets = 1024 o (noté 1 Ko) - le Méga-octet = 2^{20} octets = $(1024)^2$ o (noté 1 Mo)
   - le Giga-octet = 2^{30} octets = $(1024)^3$ o (noté 1 Go).

3.4- *LE MODULE D'ENTREE*

Le module d'entrée reçoit et transforme les signaux électriques provenant des capteurs, des boutons poussoirs, … en signaux compréhensibles par l'unité centrale de l'automate. Ces signaux électriques peuvent être tout ou rien (TOR) ou analogiques (ANA). Les signaux, avant d'être testés et traités sont stockés dans une **mémoire image des entrées** (**MIE**). Cette mémoire permet de garder l'information d'entrée le temps qu'elle soit utilisée.

3.5- *LE MODULE DE SORTIE*

Il transmet aux pré-actionneurs et aux organes de dialogue (contacteur, relais, …) les ordres de commande et de signalisation du pupitre résultant de l'exécution du programme. Ces ordres peuvent être tout ou rien (TOR) ou analogiques (ANA). Les signaux de sortie avant d'être délivrés sont stockés dans une **mémoire image des sorties** (**MIS**). Cette mémoire permet de garder l'information de sortie le temps qu'elle soit exécutée.

3.6- *LES TYPES D'E/S*

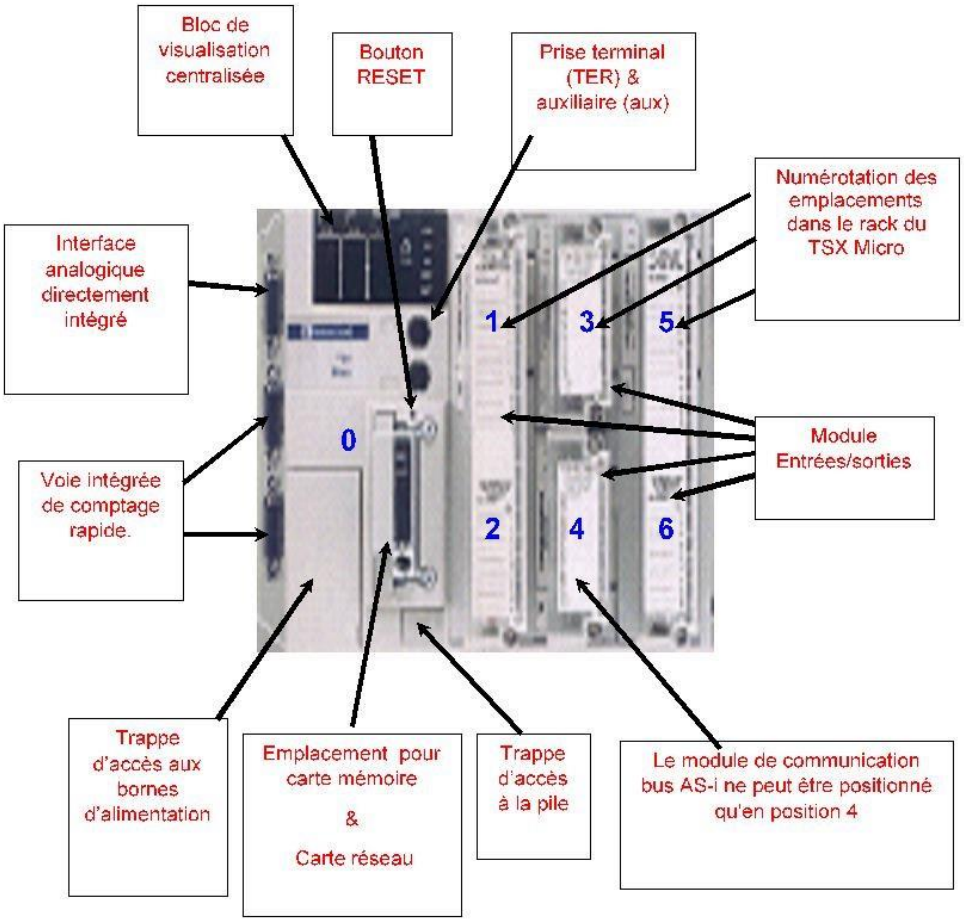
3.6.1- Les systèmes d'E/S locales

Les entrées/sorties sont regroupés sur une même carte de circuits imprimés, directement greffées sur le bus de l'automate.

3.6.2- Les systèmes d'E/S déportées

Les cartes d'entrées/sorties dans les modules comportent leur propre bus et leur propre alimentation. On peut placer ces modules au voisinage des processus ou parties de processus contrôlés, ce qui conduit à une réduction substantielle du câblage.

3.7- EXEMPLE DU AUTOMATE TSX 37



4) CLASSIFICATION

Il existe différentes catégories d'API. La *classification peut se faire selon le nombre d'E/S, selon le type et la taille du processeur, ou la structure. Selon la structure, on distingue l*es automates de type **compact** ou de type **modulaire :**

4.1- Automate de type compact

Il intègre le processeur, l'alimentation, les entrées et les sorties. Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage rapide, E/S analogiques ...) et recevoir des extensions en nombre limité. Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes, tel que la série TSX 17 **(voir ci-dessous).**

- Les automates compacts
  - 10 à 250 entrées /sorties
  - Nombre d'entrées / sorties extensible par blocs
  - Fonctions analogiques et communications limitées
  - Petits automatismes, logique combinatoire et séquentielle

4.2- *Automate de type modulaire*

Le processeur, l'alimentation et les interfaces d'entrées / sorties résident dans des unités séparées (modules) et sont fixées sur un ou plusieurs racks contenant le "fond de panier" (bus plus connecteurs). Ces automates sont intégrés dans les automatismes complexes où puissance, capacité de traitement et flexibilité sont nécessaires **(voir ci-dessous).**

- Les **automates modulaires**
  - <4'000 entrées / sorties par CPU
  - Nombre d'entrées-sorties modulables
  - Cartes "métiers" disponibles
    - Comptage
    - Commandes d'axes
    - Pesage
    - Communications
    - Sécurité
  - Automatismes complexes, régulation numérique, asservissements

5) *PROGRAMMATION*

Afin d'assurer la commande des systèmes automatisés de production, il est nécessaire d'éditer un programme adapté à l'automate destiné à piloter ledit système.

5.1- *Périphériques de programmation*

Ainsi, chaque automate se programme via une console de programmation propriétaire ou par un ordinateur équipé du logiciel constructeur spécifique dans un langage de programmation donné.

Ces consoles ou ordinateurs ont pour rôle de :

- Traduire le programme dans la mémoire de l'automate ;
- Visualiser les causes de panne ;
- Détecter et signaler les erreurs de syntaxe ; contrôler et modifier le programme.

5.2- *Principe d'écriture d'un programme*

Un programme est constitué d'une suite d'instructions, chaque instruction se compose des éléments suivants :

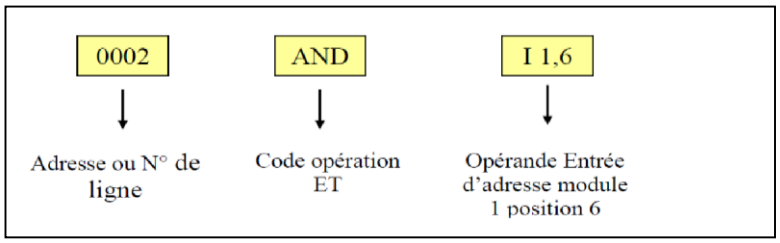
Un numéro de ligne ou une adresse permettant de retrouver une instruction dans le programme

Un code d'opération indiquant le type d'opérateur à exécuter (opération ET (code AND) ; opération OU (code OR)

Un opérande indiquant l'objet sur lequel s'effectue l'opération, il est composé en deux parties :

- Son type par exemple I pour les entrées, Q pour les sorties
- Son adresse géographique sur l'automate (sa position) par exemple 0.5

0 étant le numéro du module, 5 étant la voie sur le module ; ainsi chaque entrée ou sortie à une adresse sur l'automate.



Exemple de repérage des entrées et des sorties :

Le repérage ou adressage, c'est le repère correspondant à l'emplacement de chaque entrée et sortie ainsi son adresse en mémoire ou est stocké son l'image de son état 0 ou 1, cela permet d'utiliser plusieurs fois l'entrée ou la sortie dans le programme.

Un automate ayant 8 entrées et 8 sorties, elles seront aux adresses suivantes :

Entrées : I0,0 ; I0,1 ; I0,2 ; I0,3 ; I0,4 ; I0,5 ; I0,6 ; I0,7

Sorties : Q0,0 ; Q0,1 ; Q0,2 ; Q0,3 ; Q0,4 ; Q0,5 ; Q0,6 ; Q0,7

Ceci est un exemple et dépend du constructeur de l'API il faut donc consulter la notice.

5.3- Les langages de programmation

Le même type d'automate peut être utilisé pour différentes applications, la différence s'effectue avec le programme installé dans celui-ci. Pour réaliser ces programmes on utilise différents langages en fonction de l'automate, de l'utilisateur et du concepteur. Il existe CINQ langages de programmation selon la norme **IEC 611 31-3 :**

- Le langage **littéral structuré (ST),**
- Le langage **schéma à contacts ou LADDER (LD),**
- Le langage **blocs fonction (FBD ou LOG),**
- Le langage **GRAFCET ou SFC,**
- Le langage **liste d'instructions (IL).**

5.3.1- Le langage à contacts (LD : Ladder)

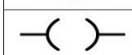
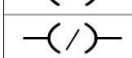
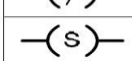
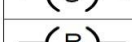
Langage graphique développé pour les électriciens. Il utilise les symboles tels que contacts, relais, et blocs fonctionnels et s'organise en réseaux (labels).

Le langage LADDER est un langage qui utilise les mêmes symboles que les schémas électriques nordaméricains (contacts et bobines). Ce langage est le plus utilisé dans la mise en œuvre des automates.
Le langage à contact est adapté à la programmation de traitements logiques, il utilise le schéma développé.
Nous retrouvons :

- La fonction ET en utilisant des contacts en série
- La fonction OU en utilisant des contacts en parallèle.

Représentation des éléments principaux

Graphe	Désignation	Fonction	Schéma à contact
	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	

Graphe	Désignation	Fonction	Schéma à contact
	bobine directe	la sortie prend la valeur du résultat logique	
	bobine inverse	la sortie prend la valeur inverse du résultat logique	
	bobine d'enclenchement	le bit interne est mis à 1 et garde cet état	
	bobine déclenchement	le bit interne est mis à 0 et garde cet état	

Un bit étant une mémoire interne logique prenant la valeur 0 ou 1

Une bobine d'enclenchement S « set » et bobine de déclenchement R « reset » correspondent à un relais bistable.

En plus des blocs fonctions logiques d'automatisme, il existe les blocs de temporisation, de comptage ...

Variable	Repère	Désignation	Exemple
Entrée (I input)	%Ix.i	X : N° module i : N° de voie	%I1.4
Sortie (Q output)	%Qx.i	X : N° module i : N° de voie	%Q2.3
Mémoire bit	%M.i	i : N° du bit interne	%M25
Mémoire mot	%MWi	i : N° du mot interne	%MW11
Temporisateur	%TMi	i : N° du temporisateur	%TM3
Compteur	%Ci	i : N° du compteur	%C4
Variable d'étape	%Xi	X : étape, i : N° étape	%X10

Le réseau à contact s'inscrit entre deux barres verticales représentant la tension d'alimentation

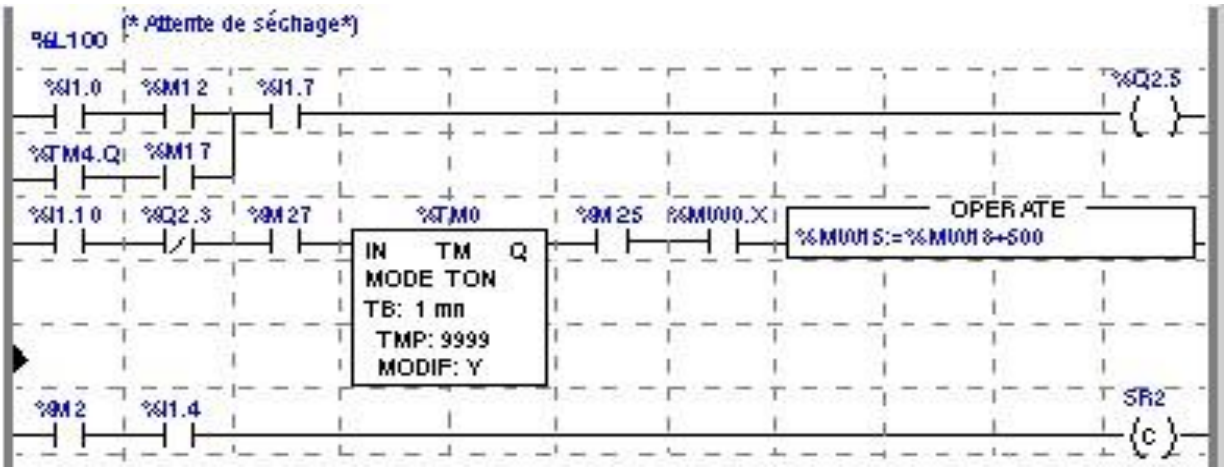


Fig. 1 : Exemple de programme en langage ladder

5.3.2- Le langage Liste d'Instruction (IL)

Langage textuel de même nature que l'assembleur (programmation des microcontrôleurs) très peu utilisé par les automaticiens

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 la temporisation, le comptage, …

a) Instruction de base en langage liste

N° de ligne	Instruction	opérande	commentaire
00	LD	% I0,01	tester l'entrée d'adresse 0,01
01	AND	% I0,02	ET entre l'entrée (I0,01) et l'entrée I0,02
02	ST	% O 0,02	Donner le résultat logique du ET à la sortie Q0,01

Instructions de test	
Désignation	Fonctions
LD	Le résultat est égal à l'opérande (load : lire la valeur).
LDN	Le résultat est égal à l'inverse de l'opérande (contact ouverture).
AND	ET logique entre le résultat et précèdent et l'état de l'opérande.
ANDN	ET logique entre le résultat et précèdent et l'état inverse de l'opérande.
OR	OU logique entre le résultat et précèdent et l'état de l'opérande.
ORN	OU logique entre le résultat et précèdent et l'état inverse de l'opérande.
XOR, XORN	OU exclusif.
Instructions d'action	
ST	L'opérande associé prend la valeur de la zone de test.
STN	L'opérande associé prend la valeur inverse de la zone de test.
S	L'opérande associé est mis à 1 lorsque le résultat de la zone de test est à 1.
R	L'opérande associé est mis à 1 lorsque le résultat de la zone de test est à 1.

b) Fig. 2 : Exemple de programme en langage IL

```
! %L0 :
        LD          %I1.0
        ANDN        %M12
        OR(         %TM4.Q
        AND         %M17
        )
        AND         %I1.7
        ST          %Q2.5
! %L5 :
        LD          %I1.10
        ANDN        %Q2.3
        ANDN        %M27
        IN          %TM0
        LD          %TM0.Q
        AND         %M25
        AND         %M000:X5
        [%M0015 :=  %M0018+500]
```

5.3.3- Le langage grafcet (SFC : Sequential Function Chart)

A partir d'un grafcet fonctionnel ou technologique, on peut transcrire directement en grafcet de programmation. La symbolisation est pratiquement identique les variantes dépendent du type d'automate utilisé.

Le GRAFCET, langage de spécification, est utilisé par certains constructeurs d'automate (Schneider, Siemens…) pour la programmation. Parfois associé à un langage de programmation, il permet une programmation aisée des systèmes séquentiels tout en facilitant la mise au point des programmes ainsi que le dépannage des systèmes.

On peut également traduire un grafcet en langage en contacts et l'implanter sur tout type d'automate

Il associe donc les langages IL, LD et ST au graphique du grafcet.

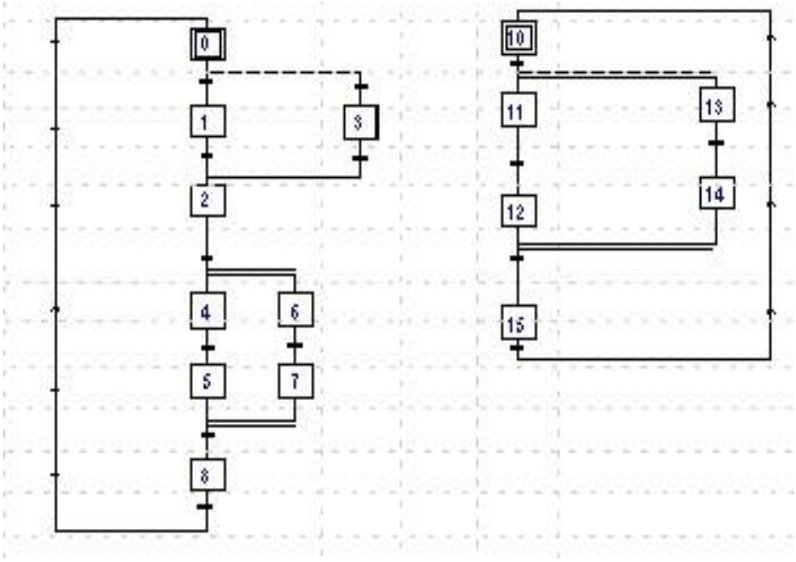


Fig 3 : Exemple de programme en langage SFC

5.3.4- Le langage littéral structuré (ST : Structured Text)

Langage informatique de même nature que le Pascal, il utilise les fonctions comme : if…then…else… (Si…alors…sinon…)

Langage peu utilisé par les automaticiens.

```
IF %M0 THEN
    FOR %MW99 := 0 TO 31 DO
        IF %MW100 [%MW99] <> 0 THEN
            %MW10 := %MW100 [%MW99];
            %MW11 := %MW99;
            %M1 := TRUE;
            EXIT;              (*Sortie de la boucle FOR*)
        ELSE
            %M1 := FALSE;
        END_IF;
    END_FOR;
ELSE
    %M1 := FALSE;
END_IF;
```

Fig 4 : Exemple de programme en langage ST

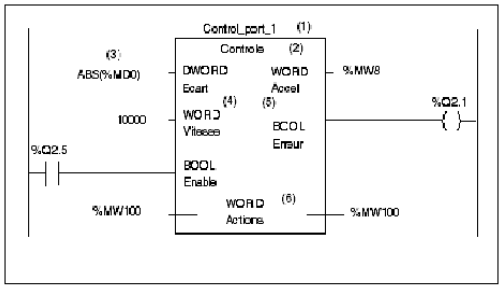


Fig. 5 : Exemple de programme en langage FBD

5.3.5- le langage **en *Blocs Fonctionnels (FBD : Function Bloc Diagram)***

Langage graphique ou des fonctions sont représentées par des rectangles avec les entrées à gauche et les sorties à droites. Les blocs sont programmés (bibliothèque) ou programmables.