Introduction

Un automate programmable est un appareil au contrôle d'une machine ou d'un processus industriel, constitué de composants électronique, comportant une mémoire programmable par un utilisateur non informaticien, à l'aide d'un langage adapté. En d'autres termes, un automate programmable est un calculateur logique, ou ordinateur, au jeu d'instructions volontairement réduit, destiné à la conduite et la surveillance en temps réel de processus industriels.

Trois caractéristiques fondamentales distinguent totalement l'automate programmable (API) des outils informatiques tels que les ordinateurs (PC industriel ou autre) :

- Il peut être directement connecté aux capteurs et pré-actionneurs grâce à ses entrées/sorties industrielles.
- Il est conçu pour fonctionner dans des ambiances industrielles sévères (température, vibration, microcoupures de la tension d'alimentation, parasites, etc.)
- Et enfin, sa programmation à partir de langages spécialement développés pour le traitement de fonctions d'automatisme fait en sorte que sa mise en œuvre et son exploitation ne nécessitent aucune connaissance en informatique.

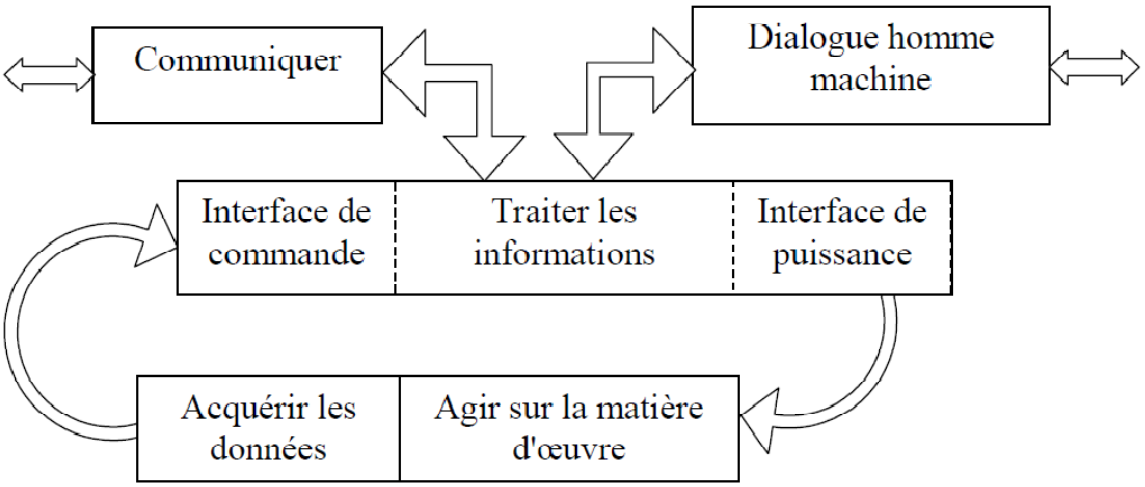
I.1. La structure des systèmes

I.1. 1. Les fonctions de bases

Tout système automatisé comporte les fonctions suivantes :

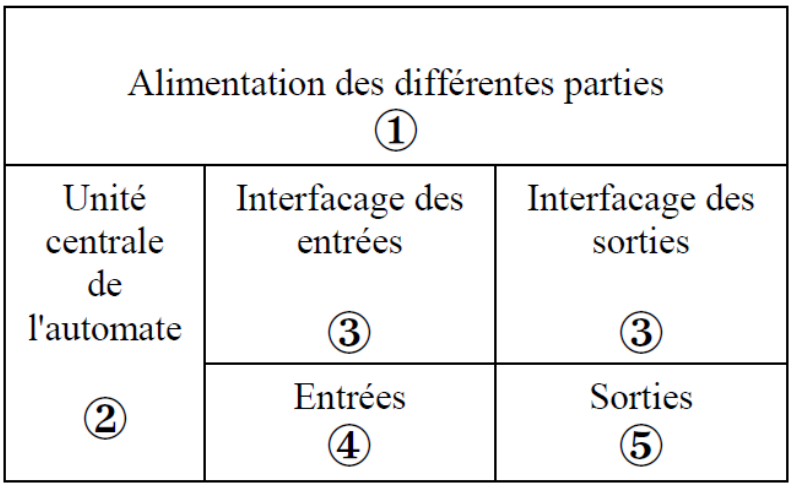
- Agir sur la matière d'oeuvre : C'est la partie opérative qui réalise ce pour quoi le système à été conçu.
- Acquérir les informations : Ce sont les capteurs qui permettent de connaître toutes les informations nécessaires au bon fonctionnement du système.
- Dialoguer avec l'opérateur : C'est les ordres et les comptes-rendus qui permettent à l'opérateur de savoir à chaque instant l'état du système et son évolution.
- Communiquer : C'est tout ce qui permet au système de communiquer avec d'autre système pour une gestion automatisé de la production par exemple.
- Traiter les données : C'est le coeur du système, cette fonction est celle ou l'on effectue tous les calculs nécessaires au bon fonctionnement du système. Les signaux entrants et sortants de cette fonction sont adaptés du point de vue énergétique par des circuits d'interfaçage.

I.1. 2. Synoptique générale d'un système



L'automate programmable rempli la fonction traiter les informations.

I.1. 3. La structure matérielle



_ Alimentation des différentes parties : Cette alimentation doit fournir l'énergie nécessaire au fonctionnement correct de l'ensemble de l'automate. Elle sera dimensionnée en fonction des consommations des différentes parties.

_ Unité centrale de l'automate : C'est cette partie qui traite les données. Elle contient en mémoire le programme et élabore donc les ordres de commande.

Son cœur est composé d'un microcontrôleur alimenté en 5 volts.

_ Interfaçage des entrées et des sorties : Ce sont des circuits chargés d'adapter en tension et en courant les signaux entre l'unité centrale et les entrées-sorties.

Ils assurent en outre un isolement entre les entrées-sorties et l'unité centrale.

_ Entrées : Ce sont des circuits spécialisés capables de recevoir en toute sécurité pour l'automate les signaux issus des capteurs. Elles peuvent être logiques

(T.O.R.), analogiques, ou numériques.
_ Sorties : Ce sont des circuits spécialisés capables de commander en toute sécurité pour l'automate les circuits extérieurs. Elles peuvent être logiques (T.O.R.), analogiques, ou numériques.

I.1. 4. La structure logicielle d'un programme

Un programme d'automate peut comporter jusqu'à 3 parties :
- le préliminaire : Il permet de traiter les transitions complexes, les comptages, les transcodages, etc… Les résultats seront conservés dans des bits internes et des mots et pourront être utilisés dans les autres parties du programme.
- le programme "principal" : Il comporte les équations ou le grafcet qui correspondent au fonctionnement désiré.
- le postérieur : Il génère les ordres de commande déterminant l'état des sorties en fonction des étapes actives. C'est dans cette partie que l'on lance les temporisations.

I.2. Architecture d'un API

La structure interne d'un API peut se représenter comme suit :

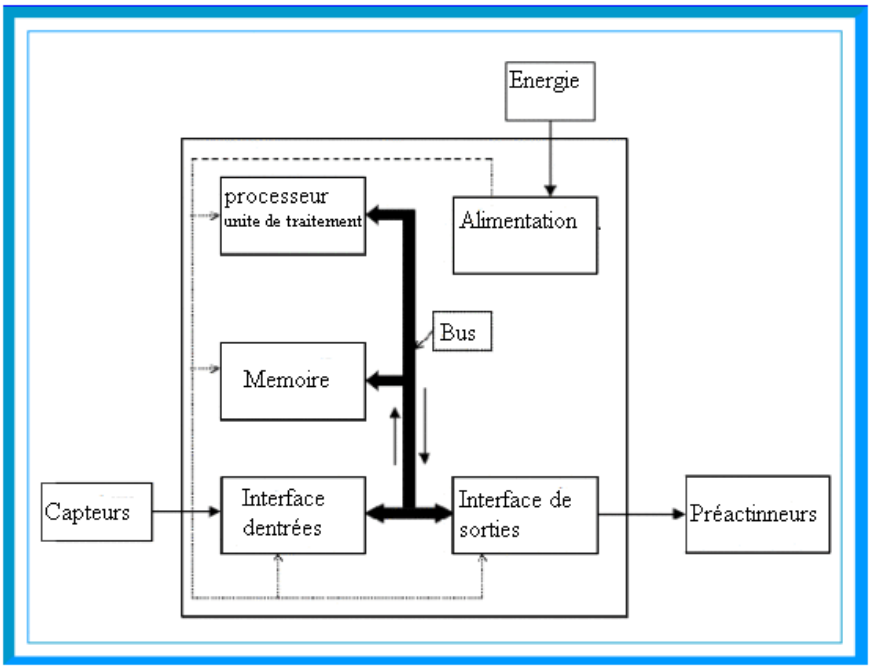


Fig.I.1: Architecture d'un API

L'automate programmable reçoit les informations relatives à l'état du système et puis commande les prés actionneur suivant le programme inscrit dans sa mémoire.

Un API se compose donc de trois grandes parties :

- Le processeur
- La mémoire
- Les interfaces entrées/sorties

Fig.I.2: Exemple d'une architecture réelle d'un API-300

I.2.1.Le processeur

Le processeur, ou l'unité centrale (UC), a pour rôle principale le traitement des instructions qui constituent le programme de fonctionnement de l'application (les fonctions logiques ET, OU, les fonctions d temporisations, de comptage, de calcule PID, etc..). Mais en dehors de cette tache de base, il réalise également d'autres fonctions :

- Gestion des entrées/sorties
- Surveillance et diagnostic de l'automate par une série de tests lancés à la mise sous tension ou cycliquement en cours de fonctionnement.
- Dialogue avec le terminal de programmation, aussi bien pour l'écriture et la mise au point du programme qu'en cours d'exploitation pour des réglages ou des vérifications des données.
- Un ou plusieurs processeurs exécutent ces fonctions grâce à un micro logiciel préprogrammé dans une mémoire de commande, ou mémoire système. Cette mémoire morte définit les fonctionnalités de l'automate. Elle n'est pas accessible à l'utilisateur.

I.2.2.La mémoire

Elle est destinée au stockage des instructions qui constituent le programme de fonctionnement de l'automatisme, ainsi que des données qui peuvent être :

Des informations susceptibles d'évoluer en cours de fonctionnement de l'application. C'est le cas par exemple de résultats de traitements effectués par le processeur et rangés dans l'attente d'une utilisation ultérieure. Ces données sont appelées variables internes ou mots internes.

Des informations qui n'évoluent pas au cours de fonctionnement, mais qui peuvent en cas de besoin être modifiées par l'utilisateur : textes à afficher, valeurs de présélection, etc. Ce sont des mots constants.

Les mémoires d'état des entrées/sorties, mises à jour par le processeur à chaque tour de scrutation du programme.

Deux familles de mémoires sont utilisées dans les automates programmables :

Les mémoires vives, ou mémoires à accès aléatoire « Random Access Memory (RAM) ». Le contenu de ces mémoires peut être lu et modifié à volonté, mais il est perdu en cas de manque de tension (mémoire volatiles). Elles nécessitent par conséquent une sauvegarde par batterie. Les mémoires vives sont utilisées pour l'écriture et la mise au point du programme, et pour le stockage des données.

Elles sont à lecture seule, les informations ne sont pas perdues lors de la coupure de l'alimentation des circuits. On peut citer les types suivants :

- ROM « Read Only Memory » : Elle est programmée par le constructeur et son programme ne peut être modifié.
- PROM « Programmable ROM » : Elle est livrée non enregistrée par le fabricant. Lorsque celle-ci est programmée, on ne peut pas l'effacer
- EPROM « Erasable PROM » : C'est une mémoire PROM effaçable par un rayonnement ultraviolet intense.
- EEPROM « Electrically EPROM » : C'est une mémoire PROM programmable plusieurs fois et
- Mémoire Flash : C'est une mémoire EEPROM rapide en programmation. L'utilisateur peut effacer un bloc de cases ou toute la mémoire.

La mémoire morte est destinée à la mémorisation du programme après la phase de mise au point. La mémoire programme est contenue dans une ou plusieurs cartouches qui viennent s'insérer sur le module processeur ou sur un module d'extension mémoire.

I.2.3. Les interfaces entrées/sorties

Les entrées/sorties TOR (Tout ou Rien) assurent l'intégration directe de l'automate dans son environnement industriel en réalisant la liaison entre le processeur et le processus. Elles ont toutes, de base, une double fonction :

- Une fonction d'interface pour la réception et la mise en forme de signaux provenant de l'extérieur (capteurs, boutons poussoirs, etc.) et pour l'émission de signaux vers l'extérieur (commande de pré-actionneurs, de voyants de signalisation, etc.). La conception de ces interfaces avec un isolement galvanique ou un découplage optoélectronique assure la protection de l'automate contre les signaux parasites.
- Une fonction de communication pour l'échange des signaux avec l'unité centrale par l'intermédiaire du bus d'entrées/sorties.

Donc pour commander un API, l'unité de commande doit envoyer :

- Un « **1** » logique pour actionner une sortie API
- Un « **0** » logique pour stopper la commande d'une sortie API.

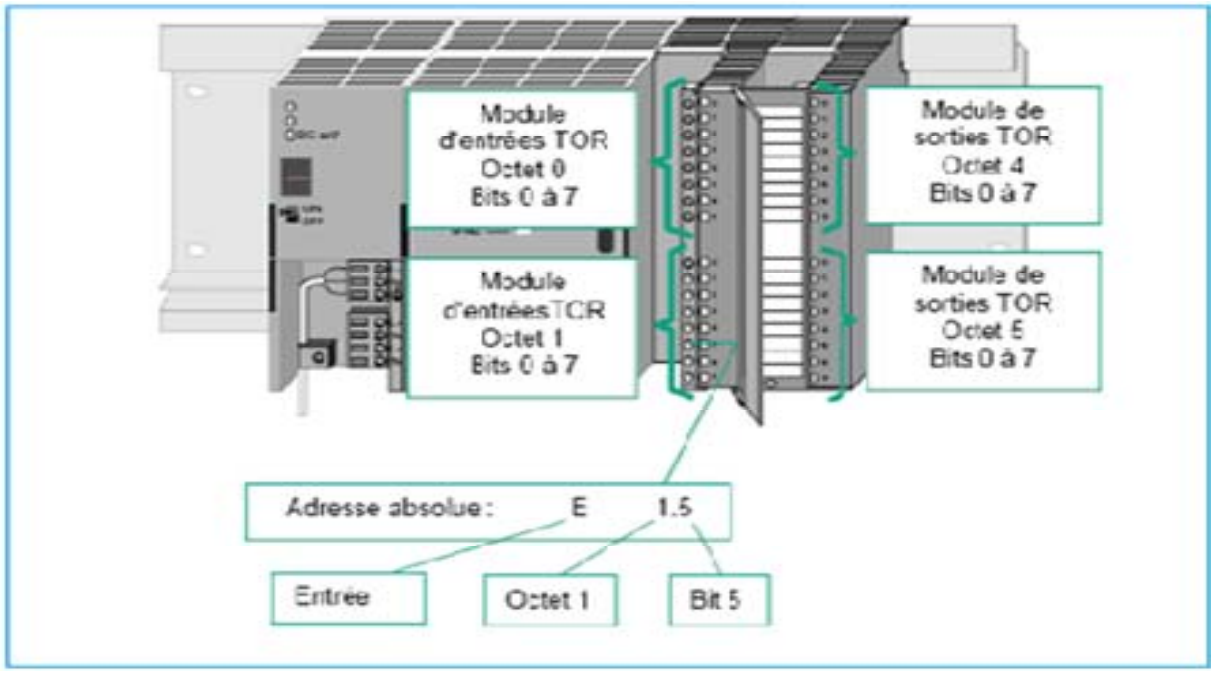


Fig. I.3: Schéma de principe d'un module TOR

I.2.4. Le bus

C'est un ensemble de conducteurs qui réalisent la liaison entre les différents éléments de l'automate. Dans un automate modulaire, il se présente sous forme d'un circuit imprimé situé au fond du bac et supporte des connecteurs sur lesquels viennent s'enficher les différents modules : processeur, extension mémoire, interfaces et coupleurs.

Le bus est organisé en plusieurs sous-ensembles destinés chacun à véhiculer un type bien défini d'informations :

- Bus de données.
- Bus d'adresses.

- Bus de contrôle pour les signaux de service tels que tops de synchronisation, sens des échanges, contrôle de validité des échanges, etc.
- Bus de distribution des tensions issues du bloc d'alimentation.

I.2.5. L'Alimentation

Elle élabore à partir d'un réseau 220V en courant alternatif, ou d'une source 24V en courant continu, les tensions internes distribuées aux modules de l'automate.

Afin d'assurer le niveau de sûreté requis, elle comporte des dispositifs de détection de baisse ou de coupure de la tension réseau, et de surveillance des tensions internes. En cas de défaut, ces dispositifs peuvent lancer une procédure prioritaire de sauvegarde.

I.2.6. Choix d'un API

Le choix d'un API est fonction de la partie commande à programmer. On doit tenir compte de plusieurs critères.

- Nombres d'entrées/sorties intégrés.
- Temps de traitement (scrutation).
- Capacité de la mémoire.
- Nombre de compteurs.
- Nombre de temporisateurs

I.3. Langage de programmation pour API

Il existe différents langages de programmation définis par la norme CEI 61131-3, elle définit cinq langages qui peuvent être utilisés pour la programmation des automates programmables industriels. Ces cinq langages sont :

- **LD** (« LadderDiagram », ou schéma à relais): ce langage graphique est essentiellement dédié à la programmation d'équations booléennes (vraie/faux).
- **IL** (« Instruction List », ou liste d'instructions): ce langage textuel de bas niveau est un langage à une instruction par ligne. Il peut être comparé au langage assembleur.
- **FBD** (« Function Block Diagram », ou schéma par blocs): ce langage permet de programmer graphiquement à l'aide de blocs, représentant des variables, des opérateurs ou des fonctions. Il permet de manipuler tous les types de variables.
- **SFC** (« SequentialFunction Char »): issu du langage GRAFCET, ce langage, de haut niveau, permet la programmation aisée de tous les procédés séquentiels.
- **ST** («StructuredText » ou texte structuré): ce langage est un langage textuel de haut niveau. Il permet la programmation de tout type d'algorithme plus ou moins complexe.

I.3.1. Objets communs à tous les langages

Toute expression, constante ou variable, utilisée dans un programme doit être caractérisée par un type, les types de base sont :

- Booléen : **BOOL** (Vraie ou Faux qui sont équivalent à 1 ou 0).
- Entier : **DINT** (c'est un nombre signé entre -2147483647 et +2147483647. Il exprimé dans l'une des bases suivantes : décimale, hexadécimale, octale ou binaire.)
- Réel : **REAL** (il prend 1 bit de signe +23 bits de mantisse +8 bits d'exposant compris entre -37 et +37.)
- Temporisation : **TIME** (elle ne peut jamais être négative et commencer par **T**# ou TIME#.

- Chaîne : **STRING** (elle doit être précédée et suivie par une apostrophe, et ne doit jamais excéder 255 caractères). Le caractère spécial (‘$’) est utilisée pour insérer des caractères non imprimables.

I.3.2. Langage LD

Le langage LD (ladderdiagram) est une représentation graphique d’équations booléennes combinant des contacts (en entrée) et des relais (en sortie). Il permet la manipulation de données booléennes, à l’aide de symboles graphiques organisés dans un diagramme comme les éléments d’un schéma électrique à contacts. Les diagrammes LD sont limités à gauche et à droite par des barres d’alimentation.

Les composants graphiques élémentaires d’un diagramme LD sont :

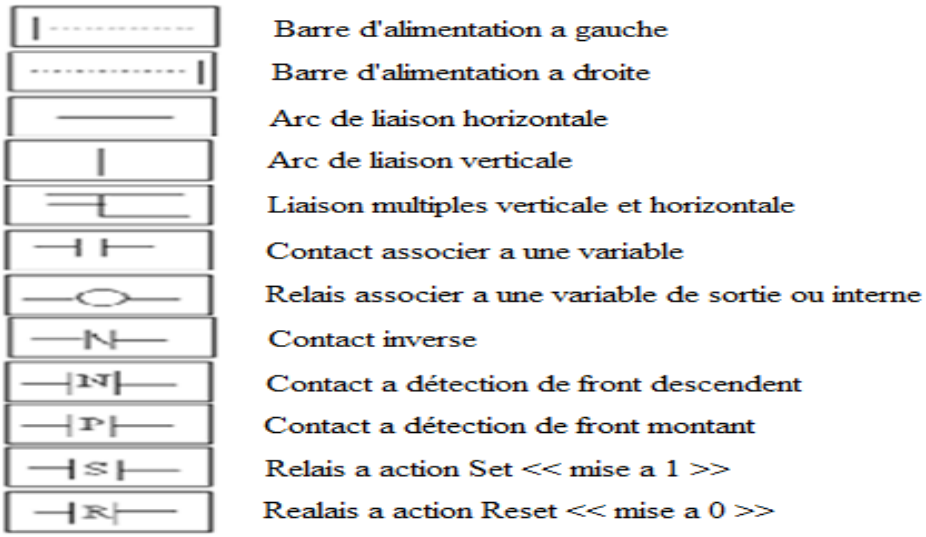


Fig. I.4: Composants graphiques élémentaires d’un diagramme LD

I.3.3. Langage IL

Le langage IL (instruction list), est un langage textuel de bas niveau. Il est particulièrement adapté aux applications de petite taille. Les instructions opèrent toujours sur un résultat courant (ou registre IL). L’opérateur indique le type d’opération à effectuer entre le résultat courant et l’opérande. Le résultat de l’opération est stocké à son tour dans le résultat courant.

Un programme IL est une liste d’instructions. Chaque instruction doit commencer par une nouvelle ligne, et doit contenir un opérateur, complété éventuellement par des modificateurs et, si c’est nécessaire pour l’opération, un ou plusieurs opérandes, séparés par des virgules (’,’). Une étiquette suivie de deux points (’:’) peut précéder l’instruction. Si un commentaire est attaché à l’instruction, il doit être le dernier élément de la ligne. Des lignes vides peuvent être insérées entre des instructions. Un commentaire peut être posé sur une ligne sans instruction.

I.3.4. Langage FBD

Le langage FBD (Function block diagram) est un langage graphique. Il permet la construction d’équations complexes à partir des opérateurs standards, de fonctions ou de clocs fonctionnels.

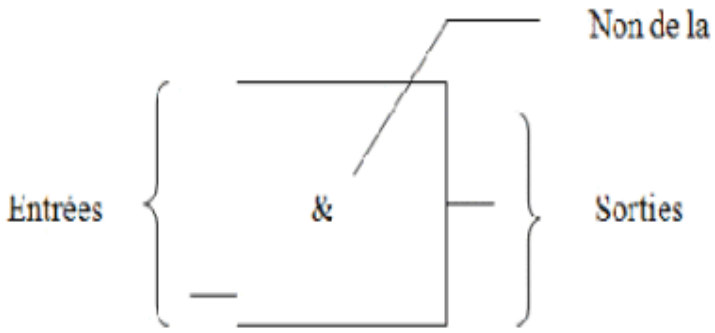


Fig. I.5 : Exemple d'un langage graphique FBD

Les principales fonctions sont :

- l'énonce **RETURN** (peut apparaître comme une sortie du diagramme, si liaison connectée prend l'état booléen **TRUE**, la fin du diagramme n'est pas interprétée.
- Les étiquettes et les sauts conditionnels sont utilisés pour contrôler l'exécution du diagramme. Aucune connexion ne peut être réalisée à droite d'un symbole d'étiquette ou de saut.
- saut à une étiquette (le nom de l'étiquette est « **LAB** »).

Si la liaison à gauche du symbole de saut prend l'état booléen **TRUE**, l'exécution du programme est déroutée après l'étiquette correspondante.
L'inversion booléenne est représentée par un petit cercle.

I.3.5. Langage SFC

Le langage **SFC** (Sequential Function Chart), ou **GRAFCET**, est un langage graphique utilisé pour décrire les opérations séquentielles. Le procédé est représenté comme une suite connue d'étapes (états stables), reliées entre elles par des transitions, une condition booléenne est attachée à chaque transition. Les actions dans les étapes sont décrites avec les langages **ST, IL, LD** ou **FBD**.
Les principales règles graphiques sont :
-un programme **SFC** doit contenir au moins une étape initiale.
-une étape ne peut pas être suivie d'une autre étape.
-une transition ne peut pas être suivie d'une autre transition.
Les composants de base (symboles graphiques) du graphique SFC sont :

- étapes et étapes initiales.
- transitions.
- liaisons orientées et -renvoi à une étape.

Les différents types d'action sont :
- action booléenne (Elle est forcée à chaque fois que le signal d'activité de l'étape change d'état.)
- action impulsionnelle programmée en **ST**, **LD** ou **IL** (c'est une liste d'instructions **ST, IL** ou **LD**, exécutée à chaque cycle pendant toute la durée d'activité de l'étape).
- action normale programmée en **ST**, **LD** ou **IL** ;
-action **SFC** (Une action **SFC** est une séquence fille **SFC**, lancée ou tuée selon les évolutions du signal d'activité de l'étape. Elle peut être décrite avec les qualificatifs d'action **N** (non mémorisée), **S** (set), ou **R** (reset).)

Plusieurs actions (de même type ou de types différents) peuvent être décrites dans la même étape. Un appel de fonctions ou de blocs fonctionnels permet d'intégrer des traitements décrits dans d'autres langages (**FBD**, **LD**, **ST** ou **IL**).

I.3.6. Langage ST

Le langage **ST** (Structurede Text) est un langage textuel de haut niveau dédié aux applications d'automatisation. Ce langage est principalement utilisé pour décrire les procédures complexes, difficilement modélisables avec les langages graphiques. C'est le langage par défaut pour la programmation des actions dans les étapes et des conditions associées aux transitions du langage **SFC**.

Un programme **ST** est une suite d'énoncés. Chaque énoncé est terminé par un point-virgule (« ; »). Les noms utilisés dans le code source (identificateurs de variables, constantes, mots clés du langage...) sont délimités par des séparateurs passifs ou des séparateurs actifs, qui ont un rôle d'opérateur. Des commentaires peuvent être librement insérés dans la programmation.

Les types d'énoncés standard sont :

- assignation (variable := expression;) ;
- appel de fonction ;
- appel de bloc fonctionnel ;
- énoncés de sélection (**IF, THEN, ELSE, CASE**) ;
- énoncés d'itération (**FOR, WHILE, REPEAT**) ;
- énoncés de contrôle (**RETURN, EXIT**) ;
- opérateurs booléens (**NOT, AND, OR, XOR**) ;
- énoncés spéciaux pour le lien avec le langage **SFC**.

Il est recommandé de respecter les règles suivantes quand on utilise les séparateurs passifs, pour assurer une bonne lisibilité du code source :

-ne pas écrire plusieurs énoncés sur la même ligne ;

-utiliser les tabulations pour indenter les structures de contrôle ;

-insérer des commentaires.

I.4.L'automate industriel S7-300

STEP 7 est un logiciel de base pour la configuration et la programmation du système d'automatisation SIMATIC. Il est formé d'un ensemble d'application avec lesquelles nous pouvons aisément réaliser des tâches partielles comme :

- La configuration et le paramètre du matériel.
- La création et le test de programme utilisateur.
- La configuration de réseau et de liaison.

I.4.1. Caractéristiques de S7-300

L'automate programmable S7-300 offre les caractéristiques suivantes :

- Gamme diversifiée de CPU
- Gamme complète de module
- Possibilité d'extension jusqu'à 32modules
- Bus du fond de panier intégré au module
- Raccordement centrale de la PGavec accès à tous les modules
- Possibilité de mise en réseau avec MPI, PROFEBUS, INDUSTRIAL ETHERNET
- Liberté de montage aux différents emplacements

- Les modules peuvent fonctionner sans ventilateur

I.4.2. Constitution de l'automate S7- 300

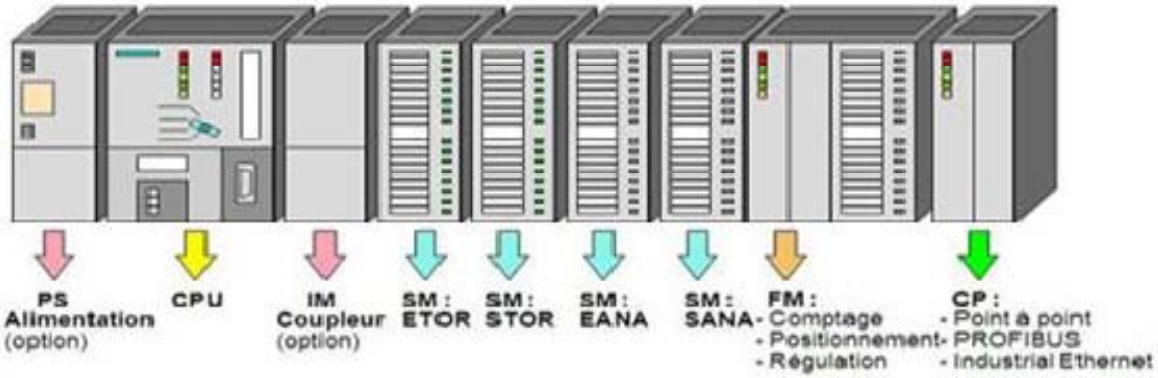


Fig. I.6 : Constitution de l'API S7-300

I.5. Langage de programmation

La programmation en STEP 7 offre trois modes de représentations et assure la conversion d'un mode à un autre.

- **Mode LIST**
  Est un langage de programmation textuel proche de la machine. Dans un programme LIST, les différentes instructions correspondent, dans une programmation LIST a été complétée par quelque structure de langage évalué (comme, par exemple, des paramètres de blocs et accès structures aux données).
- **Mode LOG**
  Est un langage de programmation graphique qui utilise les boites de l'algèbre de Boole pour représenter les opérations logique. Les fonctions complexes, combinées avec les boites logiques[8]
- **Mode CONT**
  Est un langage de programmation graphique. La syntaxe des instructions se fait penser aux schémas de circuits. CONT permet de suivre facilement le trajet du courant entre les barres d'alimentation en passent par les contacts, les éléments complexe, et les bobines.

I.6. Structure de programme utilisateur

La programmation peut être linéaire ou structurée en fonction de la nature de la tâche d'automatisation.

- Programmation linéaire : Utilisée pour les résolutions des tâches d'automatisation simple. Le programme utilisateur est écrit entièrement dans le Bloc organisation(OB1).
- programmation structurée : Utilisée pour la résolution des tâches complexes, le programme utilisateur est subdivisé en différentes tâche, chaque tâche est écrit dans un Bloc programme (FC, FB, OB). La figure suivante représente un exemple d'une programmation linaire et structurée.

I.7. Notion de Blocs dans le programme utilisateur

STEP7 permet une programmation structurée et linéaire, c'est-à-dire une subdivision du programme complet en section appelée blocs, le programme utilisateur comprend divers type de Blocs.

Un bloc correspond à une partie de programme utilisateur, délimité par sa fonction, sa structure et par son application.

Dans le logiciel de programmation step7, on trouve les différents blocs s'illustré ci-dessous :

- **Bloc d'organisation (OB)**
  Un OB est appelé cycliquement par le système d'exploitation et constitue donc une interface entre le programme utilisateur et le système d'exploitation, l'OB contient des instructions d'appel de bloc indiquant à l'unité de commande de l'automate l'ordre dans lequel il doit traiter les blocs.
- **Blocs fonctionnels (FB)**
  Un FB permet de transmettre des paramètres dans le programme utilisateur, il contient un programme qui exécute quand ce bloc est appelé par autre bloc de code. Le bloc FB facilité la programmation des fonctions complexe souvent utilisées. Un FB possède une mémoire (bloc de donnée d'instance)
- **Bloc de données d'instance (SDB)**
  Un SDB d'instance mémorise les paramètres formels et les données statiques de blocs fonctionnels. Il peut être associé à un appel de FB ou a une hiérarchie d'appel de bloc fonctionnel.
- **fonction (FC)**
  Une fonction est un bloc sans mémoire. Les données sont perdues à l'achèvement de la fonction, Les FC peuvent faire appel à des blocs de données globaux pour la sauvegarde des données.
- **Blocs des données (BD)**
  Il s'agit d'une zone dans le programme utilisateur qui contient des données utilisateurs. On distingue d'une part les blocs de données globaux aux quels tout bloc de code peut accéder, et d'autre part les blocs de codes de données d'instances. Les blocs de données ne contiennent aucune instruction.
- **Blocs des données système (SDB)**
  C'est une zone de mémoire dans la CPU contenant des paramètres de blocs.

I.8. Traitement de programme par l'automate

Le traitement du programme dans l'automate est cyclique et se déroule comme suite :

1- après la mise sous tension de l'automate, le processeur qui constitue pour ainsi dire le cerveau de l'automate vérifie si chaque entrée est sous tension ou non. L'état de ces entrées est enregistré dans la mémoire image des entrées (MIE). Si l'entrée est sous tension, l'information 1, si l'entrée n'est pas sous tension l'information 0.

2- Ce processeur exécute le programme stocké en mémoire de programme. Celui- est constitué d'une liste d'instructions et d'opérations logiques exécutées de manière séquentielle. L'information d'entrée requise à cet effet est prélevée dans la mémoire image des entrées lue auparavant et les résultats logiques sont écrits dans une mémoire image des sorties (MIS). Durant l'exécution du programme le processeur accède également aux zones de mémoires des compteurs, temporisations et mémentos.

3- Dans la dernière étape, l'étape est transmise après l'exécution du programme utilisateur de la MIS aux sorties, activant ou désactivant celles- ci- l'exécution du programme reprend au point 1.

I.8. Adressage des modules S7- 300

On a deux types d'adressages :

- **Adressage lie à l'emplacement**
  Il s'agit d'un adressage par défaut, autrement dit, à chaque numéro d'emplacement correspond une adresse de début de module bien définie.
- **Adressage libre**
  Contrairement à l'adressage lie à l'emplacement, on peut choisir librement l'adresse d'un module de signaux. Lors de la programmation, on n'est pas obligé de connaitre l'endroit où il a été implanté et le numéro de cet emplacement, c'est avec STEP7 qu'on fait la corrélation entre l'emplacement et l'adresse choisie.

I.9. Les mémentos

Ces derniers sont des éléments constituant de la mémoire système de la CPU qui mémorisent des résultats intermédiaires. Ils sont également considérés comme des éléments électroniques bistables servant à mémoriser les états logiques « 0 »ou « 1 ».

I.10.Simulation du programme a l'aide de S7-plcsim

I.10.1. Présentation du S7-plcsim

S7-plcsim est une application qui permet d'exécuter et de tester le programme utilisateur élaboré dans un automate programmable simulé dans l'ordinateur ou dans une console de programmation.

S7-plcsim dispose d'une interface simple permettant de visualiser et de forcer les différents paramètres utilisé par le programme (activer et désactiver des entres).tout en exécutant le programme dans CPU simulée, il procure la possibilité de mettre en œuvre les diverses application du logiciel step7, comme par exemple la table des variables (VAT) afin d'y visualiser et d'y forcer des variable.

I.10.2. Interface homme machine

Le montage suivant montre comment charger le programme via une console de programmation, dans le système d'automatisation pour ensuite le tester.

I.10.3. Chargement de programme

Le chargement du programme dans le système automatisé comporte quatre étapes, à savoir :

1. Application de liaison.
2. Effacement général de la CPU.
3. Chargement de programme.
4. Mise en marche de la CPU.

I.10.4. Fonctionnent de la CPU

I.10.4.1.Etat de marche (RUN-P)

La CPU exécute le programme tout en vous permettant de le modifier, de même que ces paramètres. Afin de pouvoir utiliser les applications de step7 pour forcer un paramètre quelconque de programme durant son exécution.

I.10.4.2.Etat de marche (RUN)

La CPU exécute le programme en lisant les entrées, exécutant le programme, puis en actualisant les sorties. Lorsque la CPU se trouve à l'état de marche (RUN), vous ne pouvez ni charger un programme, ni utiliser les applications du STEP7 pour forcer un paramètre quelconque.

I.10.4.3. Etat d'arrêt (STOP)

La CPU n'exécute pas le programme. Contrairement à l'état d'arrêt (STOP) de CPU réel, les sorties ne prennent pas de valeurs de sécurité prédéfinies, mais elles conservent l'état auquel elles étaient lorsque la CPU a passé l'état arrêt (STOP).

Vous pouvez charge des programmes dans le CPU lorsque elle est à l'arrêt. Le passage de l'état d'arrêt (STOP) à celui de marche (RUN) démarre l'exécution de programme à partir de la première opération.

I.10.4.4. Effacement générale de la mémoire de CPU

Pour effacer la mémoire de la CPU de simulation, vous pouvez sélectionnée la commande (CPU puis effacement générale), ou cliquer sur le boutons MRES dans la fenêtre CPU. Les zones de mémoires sont alors réinitialisées et les blocs de code ainsi que la configuration matérielle de l'API de la simulation sont effacés.

La CPU passe automatiquement a l'état d'arrêt (STOP) lorsque vous effectuer un effacement générale.

I.10.4.5. Test du programme

- **Test du programme avec la fonction visualisation**

La fonction de visualisation permet de tester le bloc d'un programme pour cela :

- une liaison en ligne doit être établie à la CPU.
- La CPU doit être en mode de fonctionnement.
- Le programme doit être chargé dans la CPU.
- Il faut ouvrir le bloc OB1 dans la fenêtre en ligne de projet.
- Il faut activer la fonction TEST > VISUALISER.

- **Test avec la table des variables**

On teste les variables du programme, en les visualisant et en les forçant à partir de la table des variables, pour cela, il faut :

- Que la liaison en ligne à la CPU existe.
- Que la CPU se trouve en mode de fonctionnement.
- Que le programme soit chargé.