Chapitre 2

Méthodes de programmation des automates programmables industriels

Méthodes de programmation des automates programmables industriels

2.1 Introduction

L'Automate Programmable Industriel (API) est un appareil électronique programmable, adapté à l'environnement industriel, qui réalise des fonctions d'automatisme pour assurer la commande de pré-actionneurs et d'actionneurs à partir d'informations logique, analogique ou numérique.
Le même type d'automate programmable industriel 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.

- **Avant** : utilisation de relais électromagnétiques et de systèmes pneumatiques pour la réalisation des parties commandes ⇒ **Logique câblée**.
- **Inconvénients** : cher, pas de flexibilité, pas de communication possible.
- **Solution** : utilisation de systèmes à base de microprocesseurs permettant une modification aisée des systèmes automatisés ⇒ **Logique programmée**.

2.2 Principaux types de données élémentaires

Dans les systèmes automatisés, la notion de bit, octet (byte), mot (word) ou double mot (double word) est souvent utilisée. Il est donc nécessaire de connaître la signification de ces termes.

- **Bit :** Un bit correspond à une position ou un caractère binaire. Il s'agit de la plus petite unité en matière d'information et elle n'admet que l'un des deux états suivants : "**0**" ou "**1**". Plusieurs bits peuvent être regroupés sous la forme d'unités plus importantes.
- **Octet** (ou byte) **:** un byte est une unité de 8 bits. Il est utilisé par exemple pour regrouper les états logiques de 8 entrées ou de 8 sorties (voir Figure 2.1).

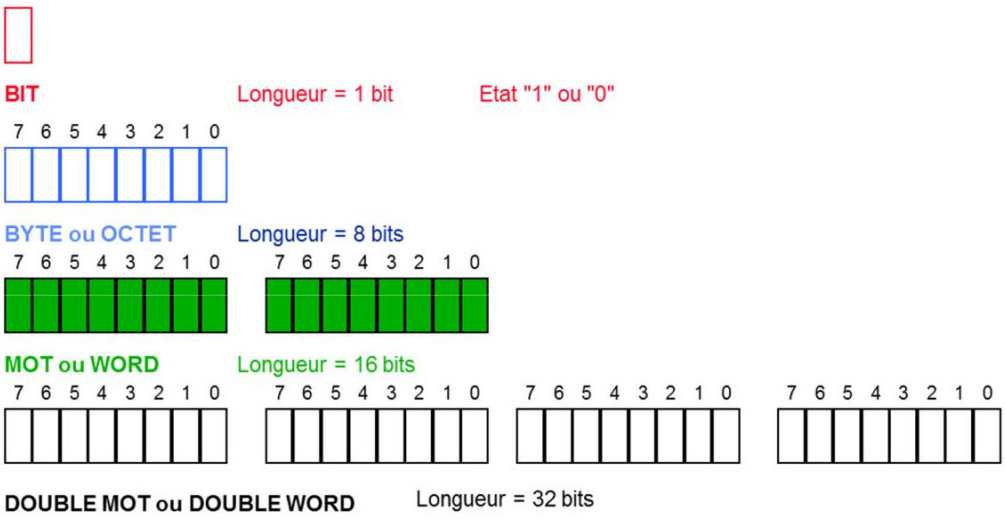


Figure 2.1 – Signification de la répartition des zones mémoires

- **Mot** (ou word) **:** un word se compose de 2 bytes, soit 16 bits. Un mot permet de regrouper 16 entrées ou 16 sorties.
- **Double mot** (ou double word) **:** un double word comporte 2 mots, ou 4 bytes, ou 32 bits. Un double mot est la plus grande unité qu'un automate soit capable de traiter à l'exception de quelques API qui traites le mot long (64 bits).

En résumé :
- 1 Byte = 8 bits ;
- 1 Word = 2 Bytes = 16 bits ;
- 1 Double word = 2 word = 4 Bytes = 32 bits (voir Tableau 2.1).

Tableau 2.1 - Table de types de données élémentaires

Type de Données	Description	Taille ou nombre de bits	Etendue
BOOL	Booléen (bit)	1	0 ou 1
BYTE	Cordon de 8 bits	8	Pas d'étendue numérique pour ce type de données
WORD	Cordon de caractères de longueur 16	16	
DWORD	Cordon de caractères de longueur 32	32	
LWORD	Cordon de caractères de longueur 64	64	
INT	Entier signé	n=16	-2^{n-1} à $+(2^{n+1}-1)$
UINT	Entier non signé	n=16	0 à (2^n-1)
REAL	Nombre réel	32	Virgule flottante
STRING	Cordon de caractères	Encadré par deux apostrophes	
TIME	Durée	Dépend de l'application concernée	

2.3 Représentation des variables

Une variable permet d'identifier des objets de données dont le contenu peut varier (données associes aux entrées, aux sorties ou aux emplacements mémoire de l'API).
Le Tableau 2.2 permet de représenter symboliquement une variable. La variable débute par le symbole %.

Tableau 2.2 - Représentations symboliques des variables

Préfixe	Signification
I	Input : Emplacement d'une entrée automate
Q	Output : Emplacement d'une sortie automate
M	Emplacement de mémento ou mémoire interne
X	Taille d'un seul bit
B	Taille d'un byte ou octet
W	Taille d'un word : mot de 16 bits
D	Taille d'un double word : mot double de 32 bits
L	Taille d'un Mot long : mot de 64 bits

2.4 Affectation des Adresses dans les zones mémoires

- Bits internes : M0.0 à M255.7 dépendants des mots suivants ;
- Octets internes : ensemble de 8 bits MBi ;
- Mots internes : ensemble de 16 bits MWi ;
- Mots doubles : ensemble de 32 bits MDi.

Selon la marque de l'automate programmable (référence constructeur) nous pouvons affecter les entrées/sorties et aussi les étapes comme le montre le tableau 2.3.

Tableau 2.3 - Représentations symboliques des variables

	API Siemens S7-1200/1500	**API Siemens S7-300/400**	**API Omron CQM1 - CPU21**
Entrées	IB0 : I0.0…I0.7 IB1 : I1.0…I1.7	EBO : E0.0…E0.7 EB1 : E1.0…E1.7	000.0X : 000.00…000.07 000.1X : 000.10…000.17
Sorties	QB0 : Q0.0…Q0.7 QB1 : Q1.0…Q1.7	AB0 : A0.0…A0.7 AB1 : A1.0…A1.7	100.0X : 100.00…100.07 100.1X : 100.10…100.17
Etapes	MB0 : M0.0…M0.7 MB1 : M1.0…M1.7	MB0 : M0.0…M0.7 MB1 : M1.0…M1.7	240.0X : 240.00…240.07 240.1X : 240.10…240.17

Exemple :

- %IW125 : Emplacement du mot d'entrée à l'adresse 125 ;
- %QB17 : Emplacement de l'octet de sortie à l'adresse 17 ;
- %MD48 : Emplacement du mot mémoire double à l'adresse 48.

Le Tableau 2.4 propose plusieurs modes de répartition de la zone mémoire allant de M0.0 à M6.7. Cette répartition est faite selon les besoins et en faisons attention aux chevauchements.

Tableau 2.4 - Représentations symboliques des variables

M6.0..M6.7	**M5.0..M5.7**	**M4.0..M4.7**	**M3.0..M3.7**	**M2.0..M2.7**	**M1.0..M1.7**	**M0.0..M0.7**
MB6	MB5	MB4	MB3	MB2	MB1	MB0
MW5		MW3		MB2	MW0	
MB6	MW4		MW2		MW0	
MW5		MB4	MD0			

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 où est stocké l'image de son état 0 ou 1, cela permet d'utiliser plusieurs fois l'entrée ou la sortie dans le programme.

Un automate programmable ayant 8 entrées et 8 sorties, pouvant occuper les adresses suivantes:

- **Entrées :** I0.0 ; I0.1 ; I0.2 ; I0.3 ; I0.4 ; I0.5 ; I0.6 ; I0.7 ;
- **Sorties :** Q1.0 ; Q1.1 ; Q1.2 ; Q1.3 ; Q1.4 ; Q1.5 ; Q1.6 ; Q1.7.

2.5 Adressage absolu et symbolique

Dans un programme, on a le choix d'un adressage absolu, symbolique ou bien les deux à la fois.

3.5.1 Adressage absolu

L'adressage absolu revient à utiliser les adresses physiques de l'automate c'est-à-dire les adresses réelles (voir Figure 2.2).

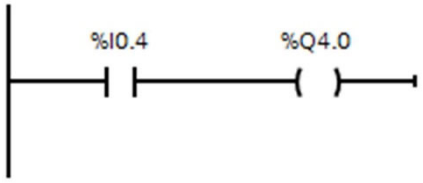


Figure 2.2 – Exemple d'adressage absolu

2.5.2 Adressage symbolique

Pour donner de la clarté au programme, on utilise l'adressage symbolique. Cela revient à substituer chaque adresse physique par une adresse symbolique qui représente au mieux la fonction comme est montré à la Figure 2.3.

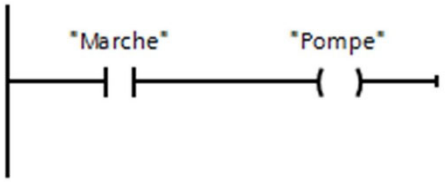


Figure 2.3 – Exemple d'adressage symbolique

2.6 Câblage des entrées / sorties d'un automate programmable

2.6.1 Alimentation de l'automate

L'automate programmable est alimenté généralement par le réseau monophasé 230V/50 Hz mais d'autres alimentations sont possibles (110V par exemple).
La protection sera de type magnétothermique (voir les caractéristiques de l'automate et les préconisations du constructeur). Il est souhaitable d'asservir l'alimentation de l'automate par un circuit de commande spécifique (contacteur KM).

2.6.2 Alimentation des entrées de l'automate

L'automate est pourvu généralement d'une alimentation pour les capteurs/détecteurs (attention au type de la logique utilisée : logique positive ou négative).
Les entrées sont connectées à 24V (borne commune des entrées). Les informations des capteurs/détecteurs sont traitées par les interfaces d'entrées comme est présenté à la Figure 2.2.

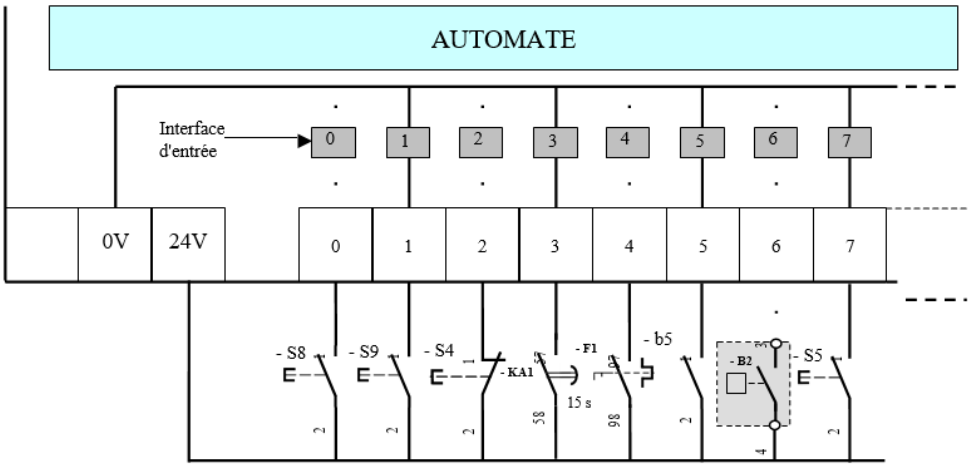


Figure 2.4 – Câblage des entrées de l'automate

2.6.3 Alimentation des sorties de l'automate :

Les interfaces des sorties permettent d'alimenter les divers pré-actionneurs. Il est souhaitable d'équiper chaque pré-actionneur par un relais d'isolation (non représentés à la Figure 2.3).

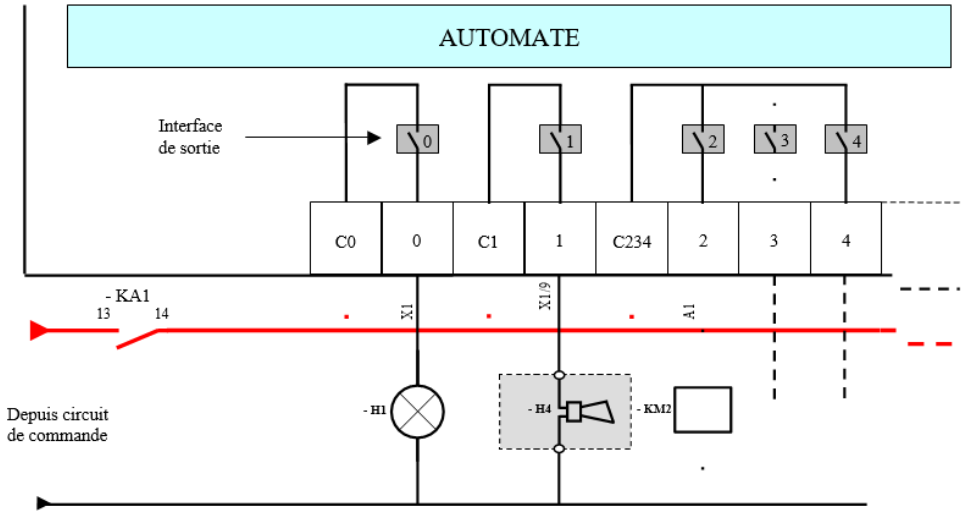


Figure 2.5 – Câblage des sorties de l'automate

2.7 Principe de programmation des API

Chaque automate se programmant via une console de programmation propriétaire ou par un ordinateur équipé du logiciel constructeur spécifique.

Un programme d'un automate 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 et Q pour les sorties ;

- Son adresse géographique sur l'automate (sa position) par exemple 0.5 : 0 étant le numéro du module et 5 étant la voie sur le module.

Tous les automates programmables fonctionnent selon le même mode opératoire comme le montre la Figure 2.4.

On appelle **scrutation** l'ensemble des quatre opérations réalisées par l'automate et le **temps de scrutation** est le temps mis par l'automate pour traiter la même partie de programme. Ce temps est de l'ordre de la dizaine de millisecondes pour les applications standards.

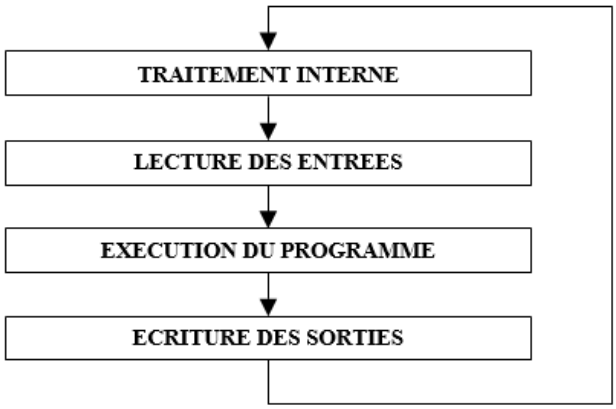


Figure 2.6 – Mode opératoire de fonctionnement d'un automate programmable

- Traitement interne : L'automate effectue des opérations de contrôle et met à jour certains paramètres systèmes (détection des passages en RUN / STOP, mises à jour des valeurs de l'horodateur, ...).
- Lecture des entrées : L'automate lit les entrées (de façon synchrone) et les recopie dans la mémoire image des entrées.
- Exécution du programme : L'automate exécute le programme instruction par instruction et écrit les sorties dans la mémoire image des sorties.
- Ecriture des sorties : L'automate bascule les différentes sorties (de façon synchrone) aux positions définies dans la mémoire image des sorties.
- Ces quatre opérations sont effectuées continuellement par l'automate (fonctionnement cyclique).

2.8 Eléments communs aux différents langages

La structuration d'un programme peut être éditée grâce aux trois types de modules suivants :

- **Fonction :** Module ayant plusieurs entrées possibles, une seule variable de sortie et pas de mémoire interne.
- **Bloc fonctionnel :** Module ayant plusieurs variables d'entrée et de sortie possibles et une mémoire interne.
- **Programme :** Module construit à l'aide de fonctions et de blocs fonctionnels etc.

L'organisation interne d'un programme peut faire intervenir un diagramme fonctionnel de séquence SFC. Les principales fonctions communes à tous les langages de programmation de l'automate programmable sont :

- La fonction de conversion de types : REAL / INT, cette fonction permet la conversion d'une variable d'entrée du type réel en une variable de sortie du type entier.
- La fonction numérique : SQRT, cette fonction permet de calculer la racine carrée d'une variable d'entrée.
- Fonctions cordons de bits : SHL, cette fonction permet le décalage à gauche de N bits d'une variable d'entrée et le remplissage de zéros à droite.
- Fonctions logiques : AND, cette fonction réalise le ET Booléen.
- Fonctions de sélection et de comparaison :

- MAX : cette fonction permet de déterminer la valeur maximale entre trois variables d'entrée ;
- SEL : cette fonction permet de sélectionner une des deux variables d'entrée suivant la variable G ;
- GT : cette fonction réalise la comparaison de supériorité entre deux variables d'entrée (IN1>IN2).

2.9 Langages de programmation des API

La norme CEI 61131-3 est destinée à la réglementer les aspects de la programmation des API.

Cette norme des langages de programmation des automates programmables permet de les classer suivant trois catégories données par le Tableau 2.5.

Tableau 2.5 - Table de vérité de l'activité de l'étape n

Langages littéraux	Langages graphiques	Langage des systèmes séquentiels
Langage IL (Instruction List) **Langage ST** (Structured Text)	**Langage LD** (Ladder Diag) **Langage FBD** (Function Bloc Diag)	**Diagramme SFC** (Sequential Function Chart)

2.9.1 Langage IL (Instruction List ou langage à liste d'instructions)

Comme son nom l'indique ce programme, très peu utilisé par les automaticiens, est constitué d'une suite d'instructions (voir Tableau 2.6) respectant le format suivant :

Etiquette (Non obligatoire)	Opérateur	Opérande(s)	Commentaire (Non obligatoire)

Ce langage est proche du langage de programmation d'un microprocesseur : l'assembleur.

Tableau 2.6 – Instruction du langage Liste

Etiquette	Opérateur	Opérande(s)	Commentaire
Début :	AND (		
	OR	%I0.0	Bp marche (NO)
	OR	%Q4.0	Electrovanne
	)		
	AND	%I0.1	Bp arrêt (NF)
	AND N	%I0.2	Niveau haut (NO)
	ST	%Q4.0	Affectation électrovanne

2.9.1.1 Programmation List d'un API OMRON

- Le mot de passe de la console : Il faut tourner la clé vers la position **Program** puis saisir les instructions suivantes : MONTR / CLR / MONTR / CLR.
- L'exécution d'un schéma se fait toujours de gauche à droite.
- Pour contrôler le fonctionnement d'un programme il suffit de positionner la clé vers la position **Monitor** puis taper **CLR** et défiler la flèche ↓.

- **Programmation d'une temporisation :** il s'agit de traduire les réseaux à contacts donnée par la Figure 2.7, en programme List destiné à un API de type OMRON.

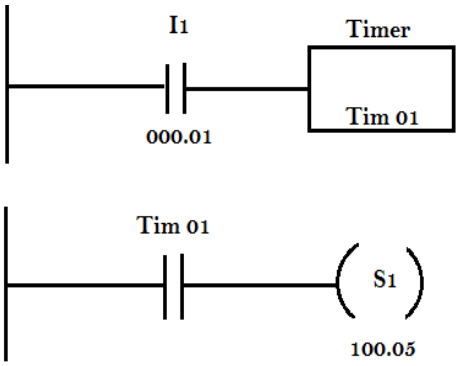


Figure 2.7 – Réseaux à contacts d'une temporisation

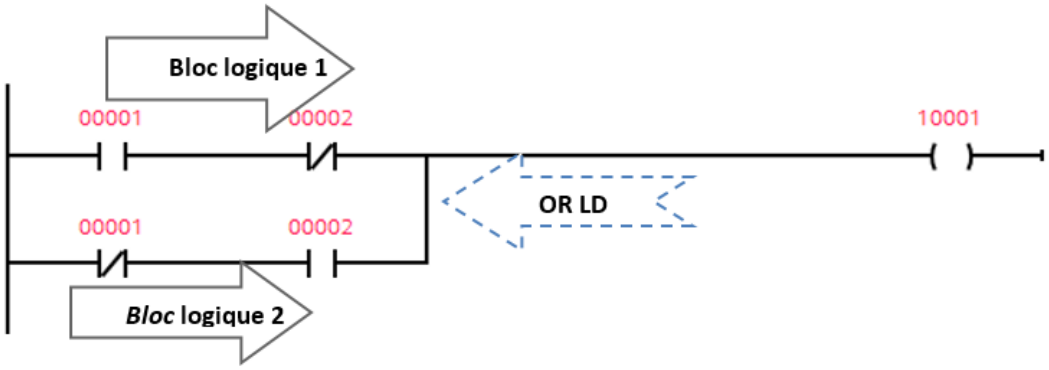
On propose le programme de la temporisation précédente en langage List :

```
LD          000.01   WR
OUT TIM 01           WR
# 50                 WR
LD TIM 01            WR
OUT         100.05   WR
END. «FUNC 01 WR»
```

✓ Pour changer la valeur du timer il suffit de tourner la clé vers la position **Program**, taper **CLEAR** puis pointer la flèche ↓ vers **time** pour changer le temps. Taper SHIFT + CONT et donner la nouvelle valeur et finalement taper **WR**.

✓ Pour voir si les modifications sont validées ou non il faut tourner la clé vers **RUN** et vérifier le résultat de fonctionnement sur la maquette de l'automate.

- **Programmation de la fonction OU EXCLUSIF :** il s'agit de traduire le réseau à contacts suivant, en programme List destiné à un API de type OMRON.

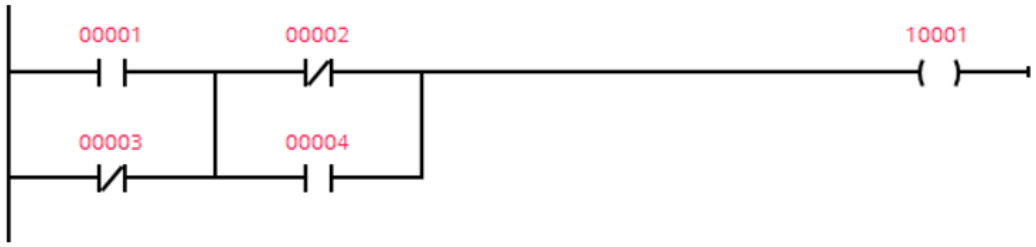


La fonction OU EXCLUSIF est un cas de programmation d'instruction en bloc logique en, effet le schéma à relais correspondant peut être subdivisé en blocs logique. L'instruction **ORLD** permet de définir la condition qui relie le premier au second bloc, dans ce cas les deux blocs sont reliés avec une condition **OU**.

```
LD          000.01    WR
AND NOT     000.02    WR
LD NOT      000.01    WR
AND         000.02    WR
OR LD                 WR
OUT         100.01    WR
END.
```

Application 1 :

Ecrire le programme List équivalent au schéma à contacts suivant :

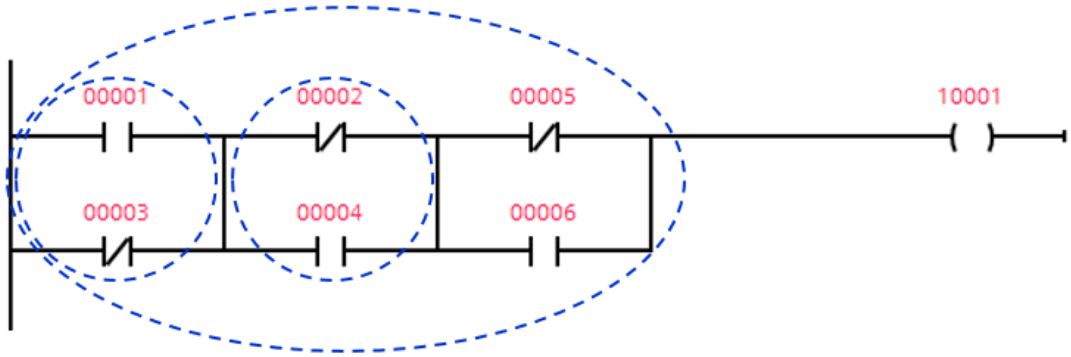


Solution de l'application 1 :

```
LD          000.01    WR
OR NOT      000.03    WR
LD NOT      000.02    WR
OR          000.04    WR
AND LD                WR
OUT         100.01    WR
END.
```

Application 2 :

Ecrire le programme List du schéma à contacts suivant :



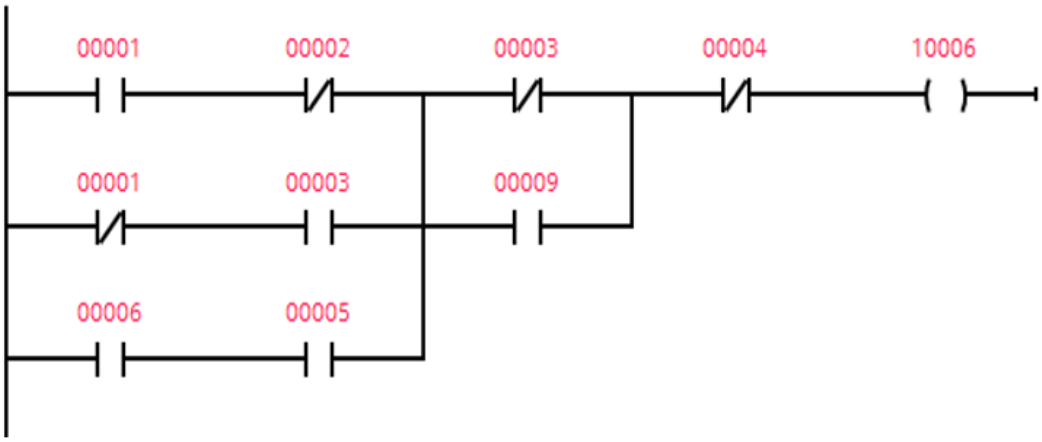
Solution de l'application 2 :

Pour facilite le programme on peut utiliser l'instruction **AND LD** comme suit :

```
 LD          000.01        WR
OR NOT       000.03        WR
LD NOT       000.02        WR
OR           000.04        WR
AND LD                     WR
LD NOT       000.05        WR
OR           000.06        WR
AND LD                     WR
OUT          100.01        WR
END.
```

Application 3 :

Ecrire le programme List équivalent au schéma à contacts suivant :

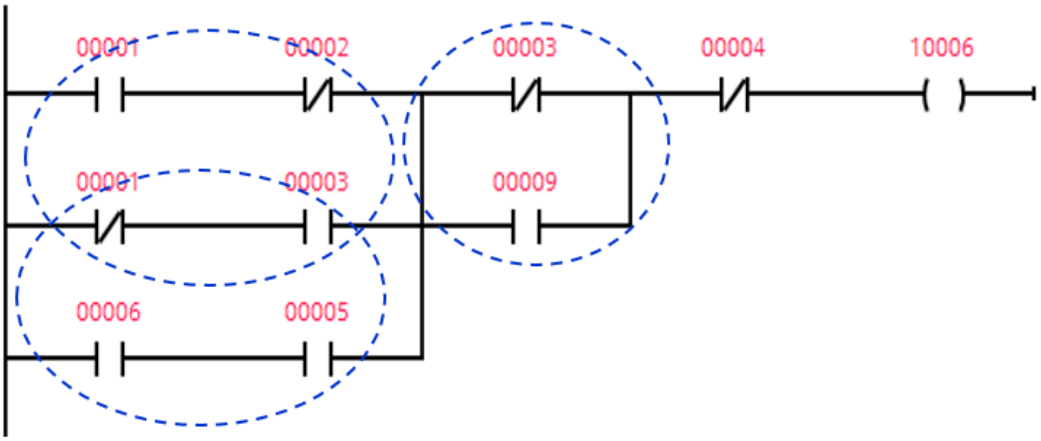


Q1 : Décomposer le schéma en blocs logiques.

Q2 : Ecrire le code mnémonique équivalent en langage List.

Solution de l'application 3 :

Q1 :



Q2:

```
LD           000.01        WR
AND NOT      000.02        WR
```

```
LD NOT      000.01      WR
AND         000.03      WR
OR LD                   WR
LD          000.06      WR
AND         000.05      WR
OR LD                   WR
LD NOT      000.03      WR
OR          000.09      WR
AND LD                  WR
LD NOT      000.04      WR
AND LD                  WR
OUT         100.06      WR
END.
```

2.9.2 Langage ST (Structured Text ou langage littéral structuré)

Ce langage, peu utilisé par les automaticiens, est composé d'expressions littérales constituées d'opérateurs et d'opérandes et d'énoncés. Ce langage est proche d'un langage informatique comme le PASCAL. Il utilise les instructions comme if, then, else (si, alors, sinon ...).

Exemple 1 : Commande d'une électrovanne

```
%L1 (« commande électrovanne »)
IF (%I0.0 OR %Q4.0) AND %I0.1 AND_NOT %I0.2 THEN SET %Q4.0
END_IF;
```

Exemple 2 : Issu d'un programme de démonstration «Machine de dosage et de mélange d'un produits» par le logiciel PL7 de Schneider.

Le contenu de ce bloc écrit en langage ST est le suivant :

```
(*** Routine de Simulation d'un procédé de 1er ordre pour Sortie PID ***)
IF Trs THEN
Sortie : = Tri ;
ELSE
Te : = INT_TO_REAL(%SW0) /1000.0 ;
Sortie : = (Filtrage/(Filtrage+Te)) *Sortie+(Gain*Te) /(Filtrage+Te) *Entree ;
END_IF ;
```

Avec :

%SW0 : Période de scrutation de la tâche maître. Permet de modifier la période de la tâche maître définie en configuration, par le programme utilisateur ou par le terminal. La période est exprimée en ms (1 ÷ 255ms). %SW0=0 en fonctionnement cyclique.

Exemple 3 : La simulation d'un retard écrit en langage ST est le suivant :

```
Nb := REAL_TO_INT(1000.0*Retard/INT_TO_REAL(%SW0));
IF (Nb>=999) THEN
Nb := 999;
END_IF;
IF Trs THEN
```

```
Memoire :=Tri;
Sortie := Tri;
Pos := 0;
ELSE
Sortie := Memoire[Pos];
Memoire[Pos]:= Entree;
Pos := Pos +1;
IF (Pos>= Nb) THEN
Pos := 0;
END_IF;
```

2.9.3 Langage FBD (Function Bloc Diagram) ou langage en blocs fonctionnels

C'est un langage graphique utilisé par les automaticiens où les blocs, sont programmés (bibliothèque) ou programmables, sont représentées par des rectangles avec les entrées à gauche et les sorties à droites. Ce langage se compose de réseaux de blocs fonctionnels qui sont connectés entre eux par des lignes, le flux des signaux se faisant de la sortie d'une fonction vers l'entrée de la fonction raccordée.

Exemples de blocs fonctionnels standards :

- Bloc fonctionnel compteurs ;
- Bloc fonctionnel temporisateurs ;
- Bloc fonctionnel régulateur PID…

Application : Description du compteur SFB 2

- **CU** : Entrée de comptage, lorsque l'état logique de l'entrée CU passe de l'état 0 à l'état 1 (front montant), le compteur est incrémenté de 1.
- **CD** : Entrée de décomptage, lorsque l'état logique de l'entrée CD passe de l'état 0 à l'état 1 (front montant), le compteur est décrément de 1.
- **R** : Entrée reset, lorsque l'état logique de l'entrée R passe de l'état 0 à l'état 1, la valeur du compteur est forcée à 0 quelle que soit l'état de l'entrée CU.
- **PV :** Réglage de la valeur du compteur. (dans l'exemple ci-dessus 10).
- **LD** : Entrée «chargé», lorsque l'état logique de l'entrée LD passe de l'état 0 à l'état 1, la valeur du compteur (CV) est forcée à la valeur réglée à l'entrée PV quelle que soit l'état de l'entrée CD.
- **QD :** La sortie Q, est à l'état logique 1 si la valeur de comptage actuelle (CV) est égale ou inférieur à 0. ($CV \leq 0$).
- **CV :** MW101 dans l'exemple, mot de mémento pour l'affichage de la valeur du compteur ou l'utilisation de la valeur du compteur ailleurs dans le programme. Attention de ne pas réutiliser les bits de mémoire M101.0 à M102.7 pour autre chose dans le programme.
- **QU :** La sortie Q, est à l'état logique 1 si la valeur de comptage actuelle (CV) est égale ou supérieure à la valeur allouée à l'entrée PV ($CV \geq PV$).

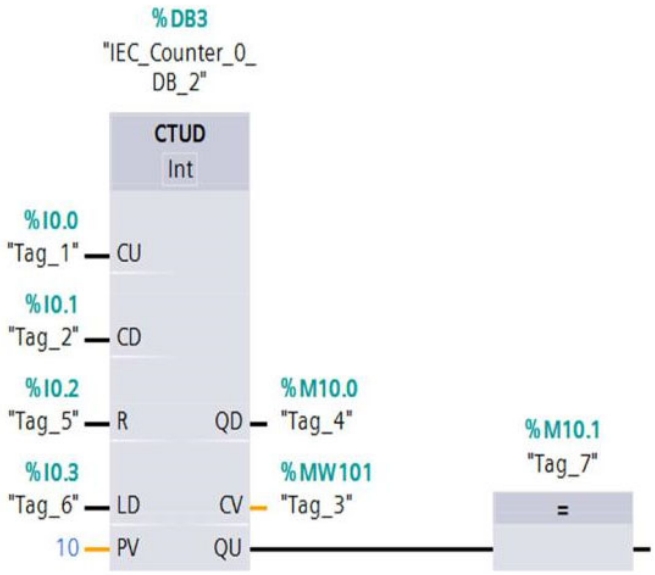


Figure 2.8 – Compteur CTUD (SFB 2)

2.9.4 Langage à contacts (LD : ladder diagram)

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

Des composants graphiques élémentaires d'un diagramme à contacts sont présentés au Tableau 3.7.

Tableau 2.7 – Instruction du langage Liste

┤ ├	Variable d'entrée ou contact à fermeture (I0.2 entrées automate, M1.2 bit mémoire interne, etc.)
┤/├	Variable d'entrée complémentée ou contact à ouverture
-( )-	Variable de sortie (Q0.2 sorties automate, M1.3 bit mémoire interne, etc.)
-(S)-	Sortie mise à 1 mémorisée (S = set)
-(R)-	Sortie remise à 0 mémorisée (R = reset)
┤P├	Un front montant

2.9.4.1 Les principes des schémas à relais

Un schéma à relais comporte une ligne et des branchements. La ligne de gauche est la ligne de bus et les lignes de branchements sont les lignes d'instructions ou lignes secondaires. Deux

lignes d'instructions sont reliées par au moins une condition. L'exécution d'un schéma se fait toujours de gauche à droite.
Les conditions (contacts) sont représentées par deux lignes verticales ou encore par deux lignes verticales traversées par une diagonale. Dans le premier cas la condition est initialement ouverte elle est transcrite par une mnémonique tel que (LD, AND, OR) ; dans le second cas c'est une condition initialement fermée (fermé au repos) elle est transcrite par une mnémonique du type (LDNOT, ORNOT, ANDNOT).

- La ligne de départ verticale placée à l'extrémité gauche du réseau est le point de départ de ce dernier.
- Les lignes verticales parallèles à celle-ci sont des lignes représentant des sous routines par rapport à la ligne principale.
- Les conditions sont représentées par deux lignes verticales.
- La terminologie fondamentale : en schéma à relais à chaque condition est attribuée un bit ce bit est à (un) lorsque la condition est vraie, la condition est alors en position ON. Une condition initialement fermée est initialement en position OFF, elle passera en position ON avec un état complémentaire à celui d'une condition initialement ouverte.
- Les conditions d'exécution : dans un schéma à relais l'exécution d'une instruction se fait à partir du moment où la combinaison des conditions qui la précède est vraie.
- Les bits d'opérande : bit d'entrée/sortie ; drapeau (bit d'état) ; bit de travail ; temporisateur ou compteur.
- Blocs logiques et blocs d'instruction : le bloc logique un groupe d'instructions qui génère un résultat logique. La décomposition d'un programme en plusieurs blocs logiques simplifie la synthèse et le test des applications. Un bloc d'instruction comprend toutes les instructions reliées entre elles dans un même schéma.

2.9.5 Programmation à l'aide du GRAFCET (SFC : Sequential Function Chart)

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 (voir Figure 2.9).
On peut également traduire un Grafcet en langage à contacts et l'implanter sur tout type d'automate.

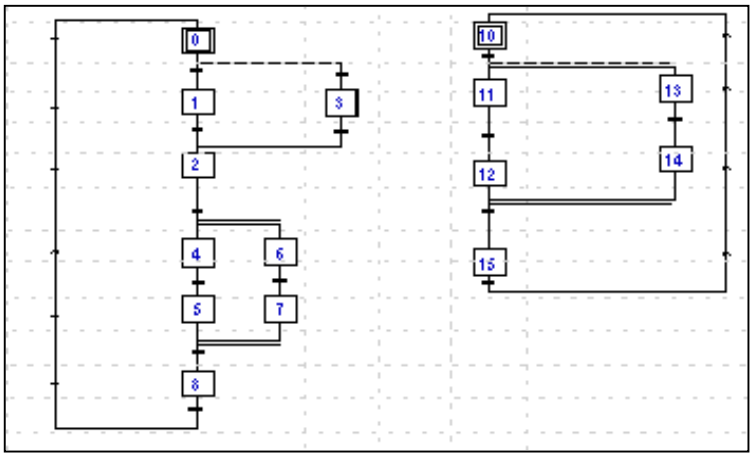


Figure 2.9 – Tranche d'un Grafcet généralisé

2.10 Méthode de traduction des équations de transferts en langage LADDER

Cette méthode de programmation des automates programmables industriels repose essentiellement sur les équations de transferts déterminés à partir d'un Grafcet de fonctionnement. Pour se fixer les idées, on présente une tranche d'un Grafcet généralisé (voir Figure 2.10) à partir duquel on détermine les équations de transferts des étapes X_i et X_{i+1}.

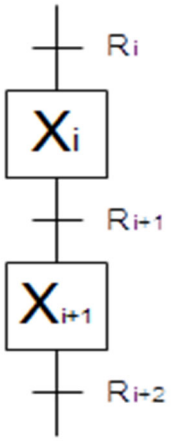


Figure 2.10 – Tranche d'un Grafcet généralisé

Les équations de transfert du Grafcet de fonctionnement seront traduites en réseaux d'étapes et d'actions.

- $\boldsymbol{X_i = (X_{i-1} * R_i + X_i) * \overline{X_{i+1}}}$
- $\boldsymbol{X_{i+1} = (X_i * R_{i+1} + X_{i+1}) * \overline{X_{i+2}}}$

La traduction de ses équations par la méthode des équations de transferts donne le résultat de la Figure 2.11.

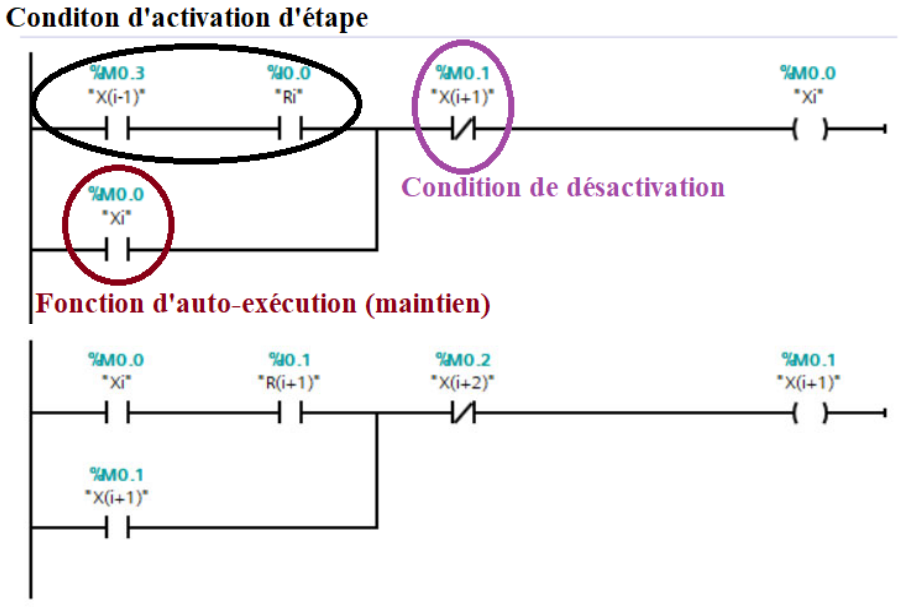


Figure 2.11 – Méthode de traduction des équations de transferts

2.11 Méthode du séquenceur électronique

Cette méthode est basée essentiellement sur l'exploitation des fonctions mise à 1 mémorisée et remise à 0 mémorisée (Set/Reset) et leurs fonctions de maintien qui sont :

- Set : La fonction de mise à 1 logique (Set to One) ;
- Reset : La fonction de remise à 0 (Reset to Zero).

Pour l'application de cette méthode, les étapes de la Figure 2.10 seront traduites par les réseaux de la Figure 2.12.

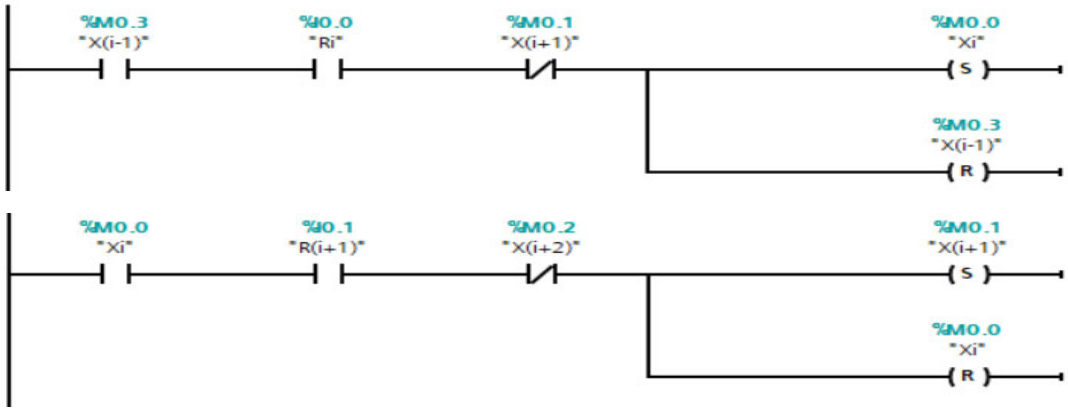


Figure 2.12 – Méthode du séquenceur électronique

2.12 Critères de choix d'un automate

Le choix d'un automate programmable est en premier lieu le choix d'une société ou d'un groupe et les contacts commerciaux et les expériences vécues sont déjà un point de départ.

Les grandes sociétés privilégieront deux fabricants pour faire jouer la concurrence et pouvoir se retourner en cas de perte de vitesse de l'une d'entre elles.

Le personnel de maintenance doit toutefois être formé sur ce matériel et une trop grande diversité de matériel peut avoir de graves répercussions. Un automate programmable utilisant des langages de programmation de type Grafcet est également préférable pour assurer les mises au point et les dépannages dans les meilleures conditions.

La possession d'un logiciel de programmation est aussi source d'économies. Des outils permettant une simulation des programmes sont également souhaitables.

Il faut ensuite quantifier les besoins :

- Nombre d'entrées/sorties : le nombre des modules peut avoir une incidence sur le nombre de racks dès que le nombre d'entrées/sorties nécessaires devient élevé.
- Type du processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur permettront le choix dans la gamme souvent très étendue.
- Fonctions ou modules spéciaux : certaines cartes (commande d'axe, pesage, etc.) permettront de soulager le processeur et devront offrir les caractéristiques souhaitées (résolution, etc.).
- Fonctions de communication : l'automate programmable doit pouvoir communiquer avec les autres systèmes de commande (API, OP, PC, etc.) et offrir des possibilités de communication avec des standards normalisés (Profibus, Profinet, Modbus, etc.).

2.13 Signalisations dans un automate programmable

On considère la CPU d'un automate programmable Simatic S7-1200 comme un exemple concret d'un API récent. La CPU de cet API fournit les indicateurs d'état suivants:

- STOP/RUN
  - Jaune continu indique l'état ARRET ;
  - Vert continu indique l'état MARCHE ;
  - Vert et jaune clignotant en alternance indiquent que la CPU est à l'état MISE EN ROUTE.
- ERROR
  - Rouge clignotant indique une erreur, telle qu'une erreur interne dans la CPU, une erreur avec la carte mémoire ou une erreur de configuration (modules non concordants) ;
  - Rouge continu signale un matériel défectueux ;
  - Toutes les DEL clignotent si la défaillance est détectée dans le Firmware.
- La LED MAINT (Maintenance) clignote dès que vous insérez une carte mémoire. La CPU passe alors à l'état ARRET. Une fois que la CPU est passée à l'état ARRET, exécuter l'une des actions suivantes pour déclencher l'évaluation de la carte mémoire :
  - Faire passer la CPU à l'état MARCHE ;
  - Effectuer un effacement général (MRES) ;
  - Mettre la CPU hors tension puis sous tension.

La CPU fournit également deux LEDs qui indiquent l'état de la communication. Ouvrez le cache du bornier inférieur pour les voir :

- Link (vert) s'allume pour signaler qu'une connexion a été établie avec succès.
- Rx/Tx (jaune) s'allume pour signaler une activité de transmission est en cours.

La CPU et chaque module d'entrées/sorties TOR fournissent une LED I/O Channel pour chacune des entrées et sorties. La LED I/O Channel (verte) s'allume ou s'éteint pour indiquer l'état de l'entrée ou de la sortie correspondante (Voir Tableau 2.8).

Tableau 2.8 – Indicateurs d'état de la CPU d'un API Simatic S7-1200

Description	**STOP/RUN** Jaune/Vert	**ERROR** Rouge	**MAINT** Jaune
Hors tension	Éteint	Éteint	Éteint
Démarrage, auto-test ou actualisation du firmware	Clignotant Jaune vert en alternance	-	Éteint
État ARRET	Allumé (jaune)	-	-
État MARCHE	Allumé (vert)	-	-
Enlever la carte mémoire	Allumé (jaune)	-	Clignotant
Erreur	Allumé Soit jaune soit vert	Clignotant	-
Maintenance requise - E/S forcées - Remplacer la pile (si le Battery Board est installé)	Allumé Soit jaune soit vert	-	Allumé
Matériel défectueux	Allumé (jaune)	Allumé	Eteint
Test des LEDs ou firmware CPU défectueux	Clignotant (jaune vert en alternance)	Clignotant	Clignotant

Version de configuration de la CPU inconnue ou incompatible	Allumé (jaune)	Clignotant	Clignotant

2.14 Sécurité des systèmes automatisés

Les systèmes automatisés sont, par nature, source de nombreux dangers (tensions utilisées, déplacements mécaniques, jets de matière sous pression, etc.).

La défaillance d'un automate programmable pouvant avoir de graves répercussions en matière de sécurité, les normes interdisent la gestion des arrêts d'urgence par l'automate programmable ; celle-ci doit être réalisée en technologie câblée.

Placé au cœur du système automatisé, l'automate se doit d'être un élément fiable car :

- Un dysfonctionnement de celui-ci pourrait avoir de graves répercussions sur la sécurité des personnes.
- Les coûts de réparation de l'outil de production sont généralement très élevés.
- Un arrêt de la production peut avoir de lourdes conséquences sur le plan financier.

Aussi, l'automate programmable fait l'objet de nombreuses dispositions pour assurer la sécurité :

- Contraintes extérieures : l'automate programmable est conçu pour supporter les différentes contraintes du monde industriel et a fait l'objet de nombreux tests normalisés (tenue aux vibrations, CEM, etc.).
- Coupures d'alimentation : l'automate programmable est conçu pour supporter les coupures d'alimentation et permet, par programme, d'assurer un fonctionnement correct lors de la réalimentation (reprises à froid ou à chaud).
- Mode RUN/STOP : seul un technicien peut mettre en marche ou arrêter un automate programmable et la remise en marche se fait par une procédure d'initialisation (programmée).
- Contrôles cycliques :
  - Procédures d'autocontrôle des mémoires, de l'horloge, de la batterie, de la tension d'alimentation et des entrées/sorties.
  - Vérification du temps de scrutation à chaque cycle appelée Watch-Dog (chien de garde) et enclenchement d'une procédure d'alarme en cas de dépassement de celui-ci (réglé par l'utilisateur).
- Visualisation : les automates offrent un écran de visualisation où l'on peut voir l'évolution des entrées/sorties.