Chapitre 3 : La programmation des automates

I- Introduction à la logique combinatoire :

I.1- Définition :

La logique combinatoire est une logique de combinaison de variable, c'est-à-dire que pour une combinaison d'entrées donnée, il ne correspond qu'une et une seule combinaison de sortie.

I.2- Conventions :

I.2.1- Etat des contacts et des récepteurs TOR :

Un circuit électrique, pneumatique, hydraulique peut avoir 2 états logiques. Ces états peuvent prendre les valeurs 0 ou 1.

- **Etat 0 :** Les actionneurs tels que : moteurs, vérins, lampe sont à l'état 0 lorsqu'ils ne sont pas alimentés. Le circuit est alors ouvert.
    - Pour un circuit pneumatique ceci correspond à une absence de pression.
    - Pour un circuit électrique cela correspond à une absence de différence de potentiel entre les bornes du composant à alimenter.
    - Pour un contact, c'est l'absence d'action physique intervenant sur un contact qui représente l'état 0. On dit qu'il est au repos.
- **Etat 1 :** Les actionneurs sont à l'état 1 lorsqu'ils sont alimentés.
    - Pour un circuit pneumatique ou hydraulique ceci correspond à une pression d'air ou d'huile dans le circuit.
    - Pour un circuit électrique cela correspond à une différence de potentiel entre les bornes du composant à alimenter.
    - Pour un contact, c'est une action physique qui agit sur le contact qui représente l'état 1. On dit qu'il est au travail.

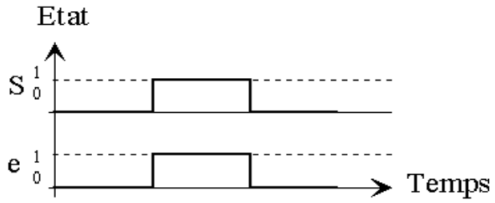
I.2.2- Etat d'un circuit électrique :

- Un circuit électrique est dit **passant** (**ou fermé**), lorsqu'un courant électrique circule dans ce circuit. Cela implique qu'il y ait continuité de ce circuit, c'est-à-dire que les contacts du circuit établissent le circuit.

- Un circuit électrique est **non passant** (**ou ouvert**), si le courant ne peut pas circuler dans ce circuit.
- Un circuit électrique comprend **au minimum**, une source d'énergie, un récepteur et un contact.

I.2.3- Chronogramme :

Un chronogramme est une représentation de l'évolution temporelle des variables d'entrées et de sorties d'un système automatisé.

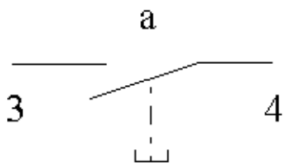


Exemple d'un chronogramme

I.3- Contacts NO et NC :

I.3.1- Contact Normalement Ouvert NO (à fermeture « F ») :

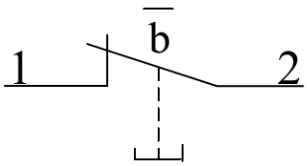
C'est un contact qui est normalement ouvert (Normally Open) au repos. Il se ferme lorsqu'il est actionné. On désigne ce type de contact par des lettres minuscules a, b, c... Ses bornes sont repérées par les chiffres 3 et 4.



Contact ouvert au repos (NO)

I.3.2- Contact Normalement Fermé NC (à fermeture « O ») :

C'est un contact qui est normalement fermé (Normally Closed) au repos et qui s'ouvre lorsqu'il est actionné. On désigne ce type de contact par des lettres $\overline{a}$, $\overline{b}$, $\overline{c}$ (se lit "a barre", "b barre", "c barre", respectivement). Ses bornes sont repérées par les chiffres 1 et 2.



Contact fermé au repos (NC)

II- Les équations logiques et portes logiques :

II.1- Les variables et les fonctions logiques :

II.1.1- Variables logiques :

Une variable logique est une grandeur qui ne peut prendre que deux états logiques. Nous les symbolisons par 0 ou 1.

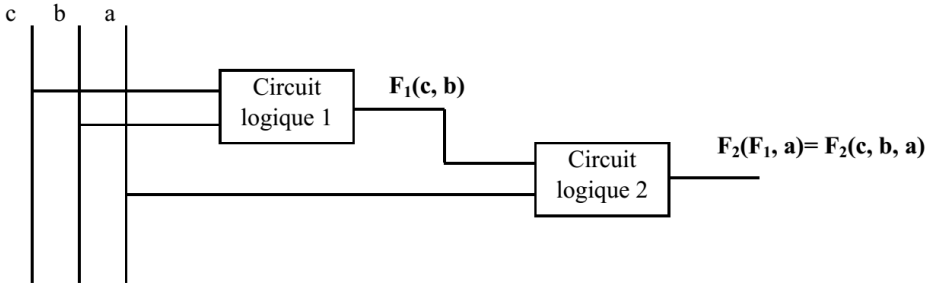
Exemples :

- Un interrupteur peut être soit fermé (1 logique), soit ouvert (0 logique). Il possède donc 2 états possibles de fonctionnement.
- Une lampe possède également 2 états possibles de fonctionnement qui sont éteinte (0 logique) ou allumée (1 logique).

II.1.2- Fonctions logiques :

Une fonction logique est une variable logique dont la valeur dépend d'autres variables,

- Le fonctionnement d'un système logique est décrit par une ou plusieurs propositions logiques simples qui présentent le caractère binaire "**vrai**" ou "**faux**".
- Une fonction logique qui prend les valeurs 0 ou 1 peut être considérée comme une variable binaire pour une autre fonction logique.
- Afin de décrire le fonctionnement d'un système en cherchant l'état de la sortie pour toutes les combinaisons possibles des entrées, on utilisera « La table de vérité ».



Exemple de deux fonctions logiques F_1 et F_2.

II.2- Algèbre de Boole

L'algèbre de Boole porte sur des variables logiques (qui ne peuvent prendre que deux états, vrai ou faux). Elle possède trois opérateurs booléens NOT (NON), AND (ET), OR (OU) présentés dans le tableau suivant :

Opération logique	**Inversion**	**Multiplication**	**Addition**
	NOT (NON)	**AND (ET)**	**OR (OU)**
Notation Algébrique	NOT A = $\overline{A}$	A AND B = A.B	A OR B = A+B

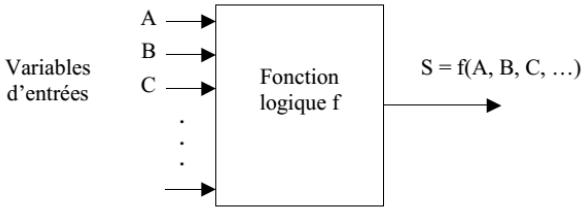
Table de vérité

A	$\bar{A}$
0	1
1	0

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

L'algèbre de Boole permet de réaliser des fonctions à l'aide de variables booléennes et des trois opérateurs de base. Le résultat obtenu est booléen, c'est-à-dire vrai ou faux.



Les lois fondamentales de l'algèbre de Boole se vérifient en écrivant les tables de vérités et en testant tous les cas possibles.

II.2.1- Propriétés des opérations de base :

Quelques propriétés remarquables sont à connaitre :

Fonctions	Commentaire	AND (ET)	OR (OU)
1 variable	Idempotence	$A.A = A$	$A+A = A$
	Elément absorbant	$0.A = 0$	$1+A = 1$
	Elément Neutre	$1.A = A$	$0+A = A$
	Complément	$A.\bar{A} = 0$	$A+\bar{A} = 1$
	Involution	$\bar{\bar{A}} = A$	
2 variables	Commutativité	$A.B = B.A$	$A+B = B+A$
3 variables	Associativité	$A.(B.C) = (A.B).C$	$A+(B+C) = (A+B)+C$
	Distributivité	$A.(B+C) = (A.B)+(A.C)$	$A+(B.C) = (A+B).(A+C)$

II.2.2- Théorèmes de l'algèbre de Boole :

Pour effectuer tout calcul Booléen, on utilise, en plus des propriétés, un ensemble de théorèmes :

Théorèmes	**AND (ET)**	**OR (OU)**
De DEMORGAN	$\overline{A.B} = \overline{A}+\overline{B}$	$\overline{A+B} = \overline{A}.\overline{B}$
	$\overline{A.B. \ldots .Z} = \overline{A}+\overline{B}+ \ldots +\overline{Z}$	$\overline{A+B+ \ldots +Z} = \overline{A}.\overline{B}. \ldots .\overline{Z}$
D'absorption	$A.(A+B) = A$	$A+A.B = A$
D'allègement	$A.(\overline{A}+B) = A.B$	$A+\overline{A}.B = A+B$
	$A.B + \overline{A}.C+BC = A.B+\overline{A}.C$	

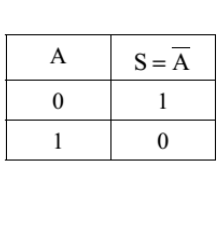
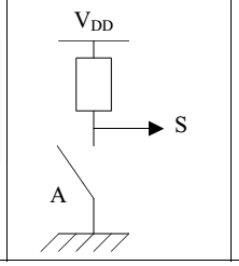
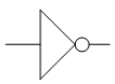
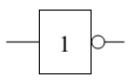
II.3- Matérialisation des opérateurs logiques :

II.3.1- Les portes logiques de base :

Les portes logiques sont des circuits électroniques dont les fonctions de transfert (relations entre les entrées et les sorties) matérialisant les opérations de base appliquées à des variables électriques.

II.3.1.1- La porte NOT (NON) :

C'est une porte à une seule entrée, elle matérialise l'opérateur inverseur. Le tableau suivant résume l'action de cet opérateur. Dans cette section, l'interrupteur ouvert vaut 0 et l'interrupteur fermé vaut 1. Le symbole traditionnel est celui Européen (MIL), tandis que le symbole normalisé est celui International (CEI)

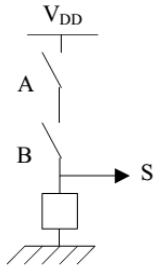
Table de vérité	Montage	Symbole traditionnel	Symbole normalisé
			

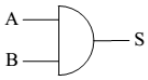
Les circuits intégrés correspondant sont : TTL : 7404, 7405, 7406, 7416 et CMOS : 4009, 4049.

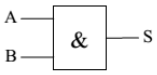
II.3.1.2- La porte AND (ET) :

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	S = A.B
0	0	0
0	1	0
1	0	0
1	1	1





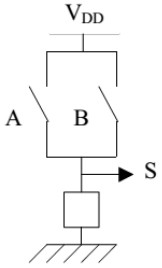


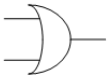
Les circuits intégrés correspondant sont : TTL : 7408, 7409, 7411, 7415, 7421 et CMOS : 4073, 4081, 4082.

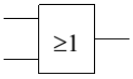
II.3.1.3- La porte OR (OU) :

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	S = A+B
0	0	0
0	1	1
1	0	1
1	1	1







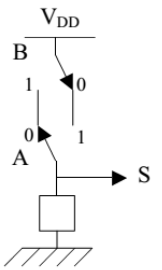
Les circuits intégrés correspondant sont : TTL : 7432 et CMOS : 4071, 4072, 4075.

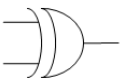
II.3.1.4- La porte XOR (OU-exclusif) :

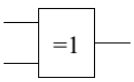
L'opérateur OU-exclusif n'est pas un opérateur de base car il peut être réalisé à l'aide des trois portes précédentes. La fonction XOR vaut 1 si une seule des entrées est à l'état 1 et l'autre est à l'état 0.

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	S = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0







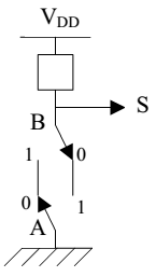
Les circuits intégrés correspondant sont : TTL : 7486-74136 et CMOS : 4030-4070.

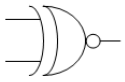
II.3.1.5- La porte XNOR (NON-OU-exclusif) :

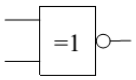
L'opérateur XNOR n'est pas non plus un opérateur de base. En effet, il représente l'inverse de l'opérateur XOR.

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	$S = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1







Les circuits intégrés correspondant sont : TTL : 74266 et CMOS : 4077.

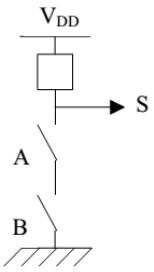
II.3.2- Les portes logiques universelles :

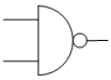
Autre que les portes logiques de base (ou élémentaires), il existe des portes appelées portes logique universelles (complètes), telles que les portes NAND (NON-ET) et NOR (NON-OU). Une porte logique est dite universelle lorsqu'elle permet, à elle seule, d'exprimer les fonctions de base NOT, AND, OR.

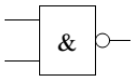
II.3.2.1- La porte NAND (NON-ET) :

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	$S = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0





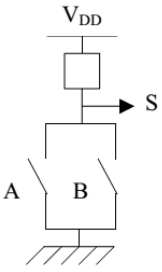


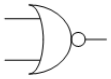
Les circuits intégrés correspondant sont : TTL : 7400, 7401, 7403, 7410, 7430, 74133 et CMOS : 4011, 4012, 4023, 4068, 4093.

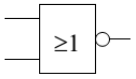
II.3.2.2- La porte NOR (NON-OU) :

Table de vérité	Montage	Symbole traditionnel	Symbole normalisé

A	B	$S = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0







Les circuits intégrés correspondant sont : TTL : 7402, 7427, 7428, 7433 et CMOS : 4000, 4001, 4002, 4025, 4078.

II.4- Simplification des fonctions logiques :

L'objectif de la simplification des fonctions logiques est des minimiser le nombre de termes afin d'obtenir une réalisation matérielle plus simple, donc plus facile à construire et à dépanner et moins couteuse.

Deux méthodes de simplification sont utilisées :

- La simplification algébrique.
- La simplification graphique par tableau de Karnaugh.

II.4.1- Simplification algébrique des fonctions logiques :

Pour obtenir une expression plus simple de la fonction par cette méthode, il faut utiliser les théorèmes et les propriétés de l'algèbre de Boole (voir section II.2).

- La multiplication par 1 ($X + \bar{X}$).
- L'addition d'un terme nul ($X.\bar{X}$).

Exemple : Simplifier la fonction : $F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$$\begin{aligned} F &= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC + ABC + ABC \\ &= BC\left(\bar{A} + A\right) + AC\left(\bar{B} + B\right) + AB\left(\bar{C} + C\right) \\ &= BC + AC + AB \end{aligned}$$

Cependant, cette méthode ne permet jamais de savoir si l'on aboutit ou pas à une expression minimale de la fonction.

II.4.2- Simplification graphique des fonctions logiques (par tableau de Karnaugh):

Le tableau de Karnaugh permet de visualiser une fonction et d'en tirer intuitivement une fonction simplifiée.

La méthode consiste à réaliser des groupements des cases adjacentes. Ces groupements des case doivent être de taille maximale (nombre max de casse) et égale à 2^k (c'est-à-dire 2, 4, 8, 16, …). On cesse d'effectuer les groupements lorsque tous les uns appartiennent au moins à l'un d'eux.

Remarque : Avant de tirer les équations du tableau de Karnaugh, il faut respecter les règles suivantes :

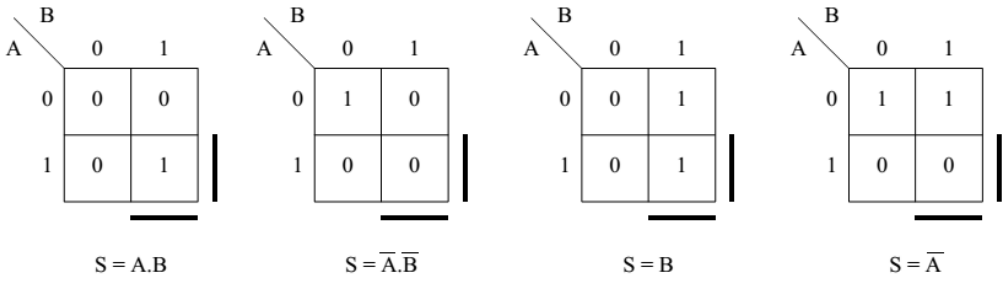
- Grouper tous les uns.
- Grouper le maximum des uns dans un seul groupement.
- Un groupement a une forme rectangulaire.
- Le nombre des uns dans un groupement est une puissance de 2 est égal à 2^k.
- Un 1 peut figurer dans plus qu'un groupement.
- Un groupement doit respecter les axes de symétries du tableau de Karnaugh.

Exemple 1 : deux variables d'entrée A et B

1 case = produit de 2 variables,

2 cases = 1 variable,

4 cases = 1 ou 0.



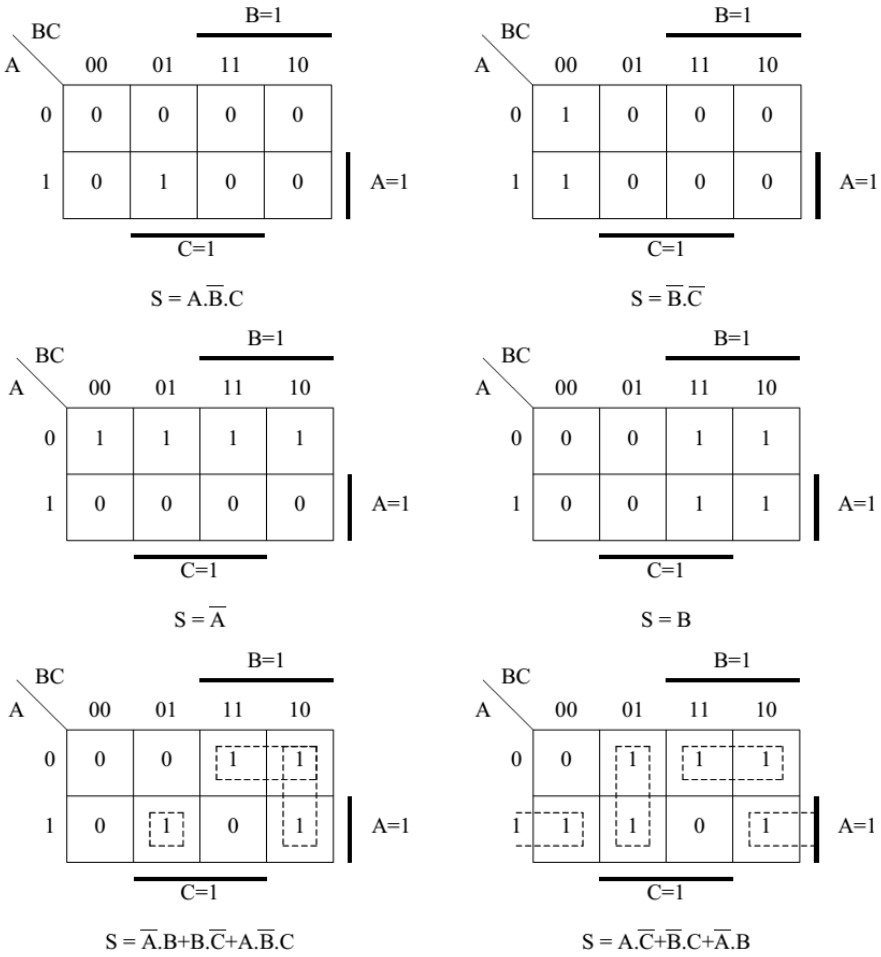
Exemple 2 : trois variables d'entrée A, B et C

1 case = produit de 3 variables,

2 cases = produit de 2 variables,

4 cases = 1 variable,

8 cases = 1 ou 0.



Il y a parfois plusieurs solutions équivalentes.

Exemple 3 : quatre variables d'entrée A, B, C et D

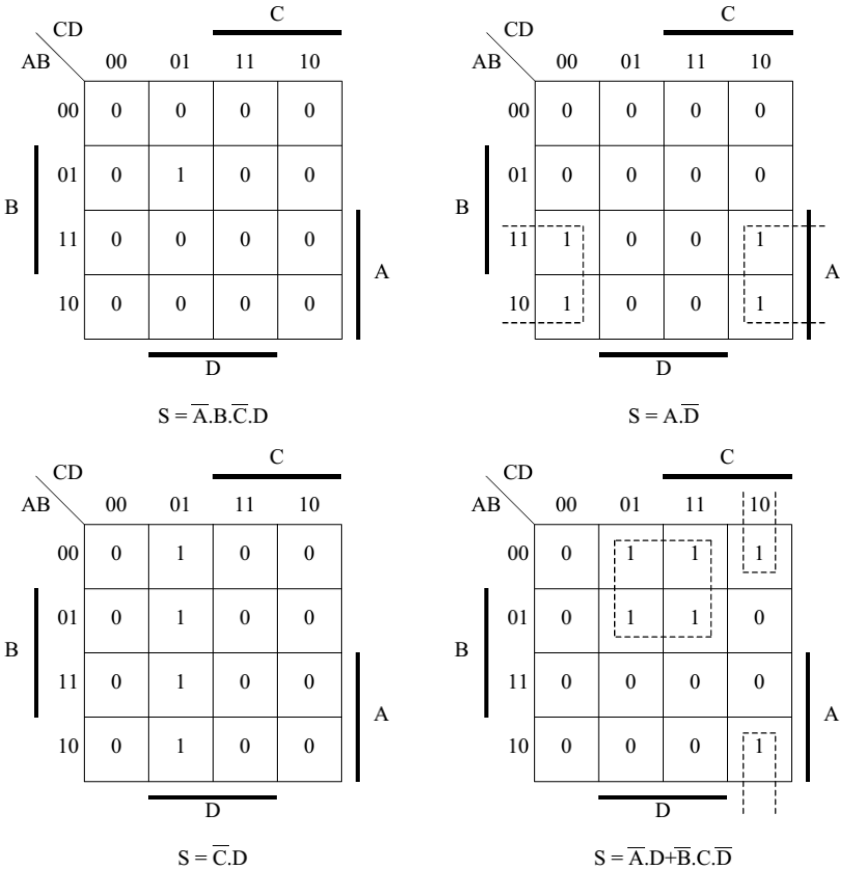
1 case = produit de 4 variables,

2 cases = produit de 3 variables,

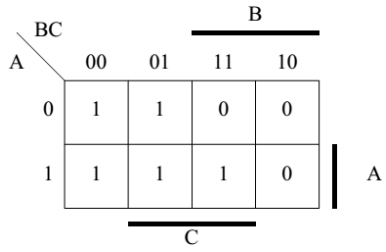
4 cases = produit de 2 variables,

8 cases = 1 variable,

16 cases = 1 ou 0.



Il peut parfois être plus pratique de lire les 0 dans le tableau plutôt que les 1. On obtient alors $\overline{F} =$ somme de produits ce qui implique (par DE MORGAN) que $F =$ produit de somme des variables inversées. Si par exemple on a :



En comptant les 1 : $S = \overline{B} + A.C$. D'autre part, si on compte les 0 au lieu des 1, on obtient : $\overline{S} = B.\overline{C} + \overline{A}.B$. D'où on tire : $S = \overline{B.\overline{C} + \overline{A}.B}$. Ce qui donne finalement : $S = \left(\overline{B} + C\right).\left(A + \overline{B}\right)$.

On peut utiliser la lecture directe sur les 0 quand il y a peu de 0 et beaucoup de 1.

III- Introduction au Grafcet :

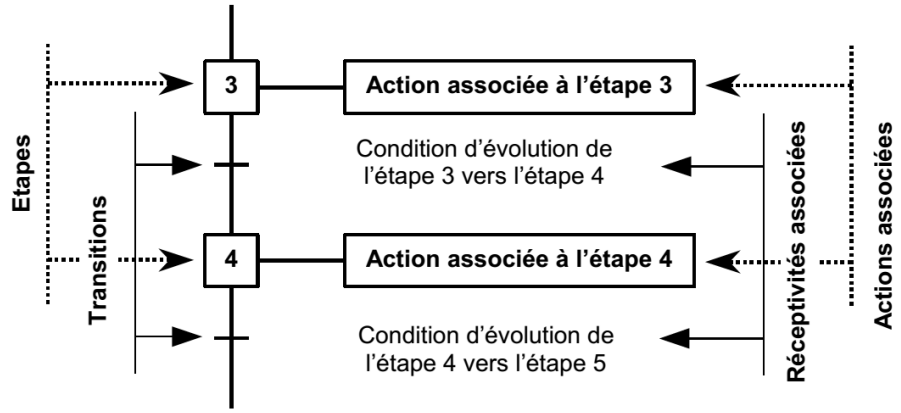
III.1- Définition :

Le GRAFCET (**GRA**phe **F**onctionnel de **C**ommande **E**tape/**T**ransition) est un outil graphique qui permet la description du fonctionnement du système automatisé au cours du temps d'une façon claire et sans ambiguïté.

III.2- Conventions et règles :

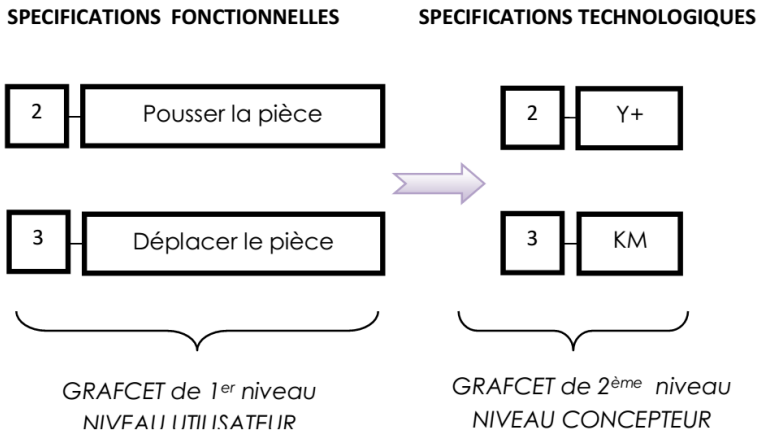
Le fonctionnement d'un système automatisé peut être représenté graphiquement par un ensemble :

- D'**étapes** auxquelles sont associées des **actions**.
- De **transitions** auxquelles sont associées des **réceptivités**.
- Des **liaisons orientées** entre les **étapes** et les **transitions**.



III.2.1- Actions associées à l'étape :

On précise pour chaque étape, à l'intérieur d'un rectangle, les actions à effectuer lorsque l'étape est active.

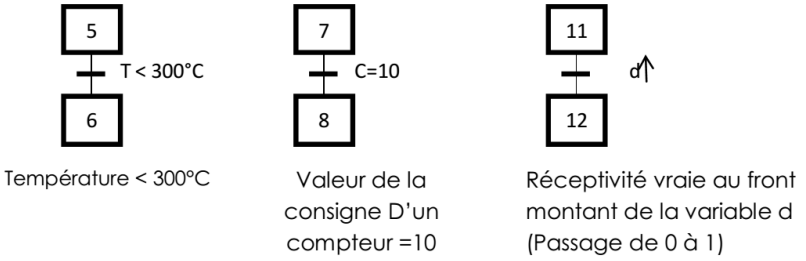


III.2.2- Transition :

Une transition indique la possibilité d'évolution d'une étape à l'étape suivante. A chaque transition, on associe une ou plusieurs conditions logiques qui traduisent la notion de réceptivité.

La réceptivité est une fonction combinatoire d'informations, telles que :

- Etats de capteurs.
- Action de boutons poussoirs par l'opérateur.
- Action d'un temporisateur, d'un compteur.
- Etat actif ou inactif d'autres étapes.
- Comparaison d'une valeur analogique



III.2.3- Liaisons orientées :

Les liaisons indiquent les voies d'évolution du Grafcet. Dans le cas général, les liaisons qui se font du haut vers le bas ne comportent pas de flèches. Dans les autres cas, il faut utiliser des flèches.

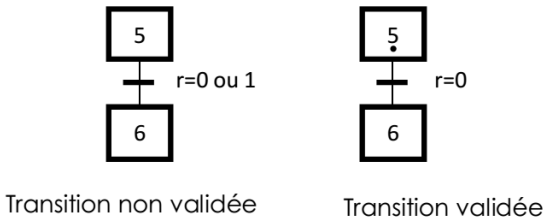
III.2.4- Règles d'évolution :

- **Règle 1 : (Situation initiale)**

La situation initiale caractérise le comportement initial de la partie commande vis-à-vis de la partie opérative et correspond aux étapes actives au début du fonctionnement. L'étape initiale est représentée par un double carré.

- **Règle 2 : (Franchissement d'une transition)**

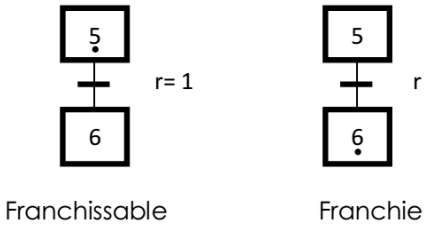
Une transition est validée lorsque toutes les étapes immédiatement précédentes sont actives. Le franchissement ne peut produire que lorsque cette transition est validée et que la réceptivité associée est vraie



- **Règle 3 : (Evolution des étapes actives)**

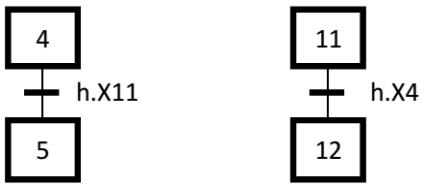
Le franchissement d'une transition provoque simultanément :

- L'activation de toutes les étapes immédiatement suivantes reliées à cette transition.
- La désactivation de toutes les étapes immédiatement précédentes reliées à cette transition.



- **Règle 4 : (Evolution simultanées)**

Plusieurs transitions simultanément franchissables sont simultanément franchies. Cette règle de franchissement permet notamment de décomposer un Grafcet en plusieurs diagrammes indépendants tout en assurant de façon rigoureuse leur interconnexion.



- **Règle 5 : (Activation et désactivation simultanées)**

Si au cours du fonctionnement de l'automatisme une même étape doit être simultanément activée et désactivée, elle reste active.

III.3- Notions de séquence :

III.3.1- Actions aux étapes :

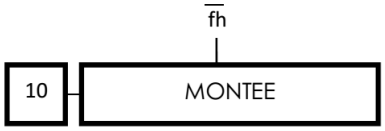
III.3.1.1- Actions continues :

Une action est dite continue lorsque la durée de cette action correspond à la durée d'activation de l'étape. Plusieurs actions continues peuvent être associées à une même étape.

III.3.1.2- Actions conditionnelles :

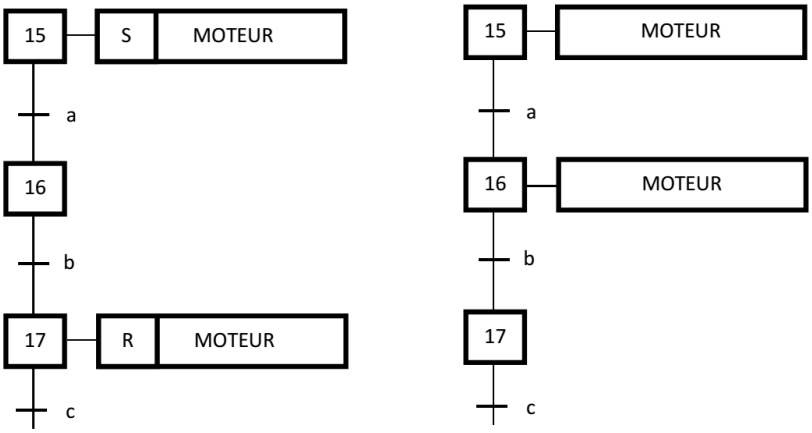
L'exécution de l'action est soumise à une condition logique notée à côté d'un trait vertical au-dessus de l'action.

A l'étape **10**, la montée est effectuée tant que l'on n'a pas atteint le fin de course **fh**



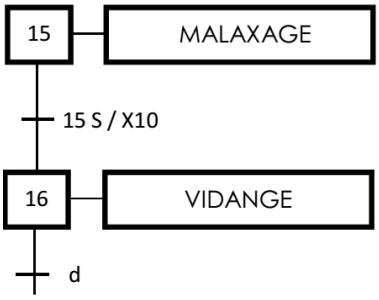
III.3.1.3- Actions mémorisées :

Lorsqu'une action doit être maintenue pendant plusieurs étapes, il suffit d'utiliser les symboles **S** (**Set**) et **R** (**Reset**) ou de la répéter dans toutes les étapes concernées.



III.3.1.4- Durée d'activité d'étape :

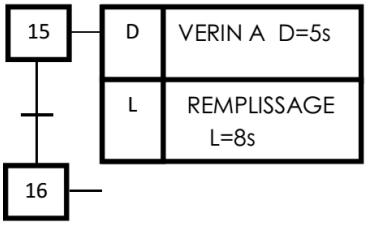
Pour maintenir une étape active et ses actions associées pendant un certain temps ($t = 15s$), il suffit d'utiliser le signal binaire de sortie de l'opérateur à retard comme réceptivité.



III.3.1.5- Actions retardées ou limitées :

L'action peut être retardée, c'est à dire que la condition d'assignation n'est vraie qu'après une durée **D** depuis l'activation de l'étape. Comme elle peut être limitée dans le temps, C'est à dire que la condition d'assignation n'est vraie que pendant une durée **L** depuis l'activation de l'étape.

L'action **VERIN A** est retardée de **5 secondes** et l'action **REMPLISSAGE** est limitée à **8 secondes** à partir de l'activation de l'étape **15**.



III.3.2- Sélection de séquences :

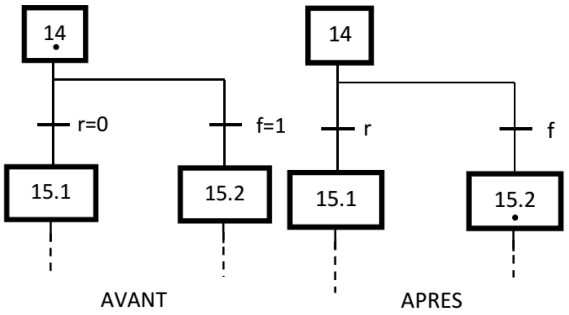
Une sélection de séquence est un choix d'évolution entre une ou plusieurs séquences possibles à partir d'une ou plusieurs étapes. Il est impérative de ne sélectionner qu'une seule évolution et ceci en utilisant des conditions logiques exclusives.

Cette exclusivité peut être :

- Soit d'ordre physique (incompatibilité mécanique ou temporelle)
- Soit d'ordre logique dans l'écriture des réceptivités.

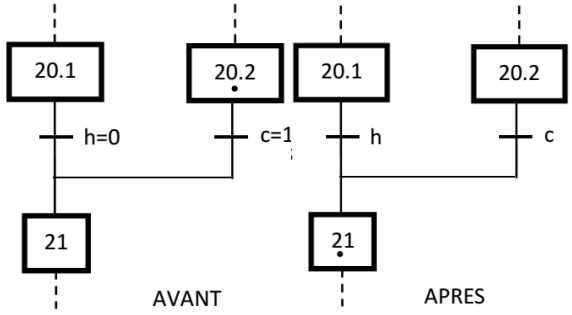
III.3.2.1- Début de sélection (divergence en OU) :

Sur l'exemple, l'étape **14** se trouvant active et la réceptivité « **f** » étant vrais, l'évolution s'effectue vers l'étape **15.2**.



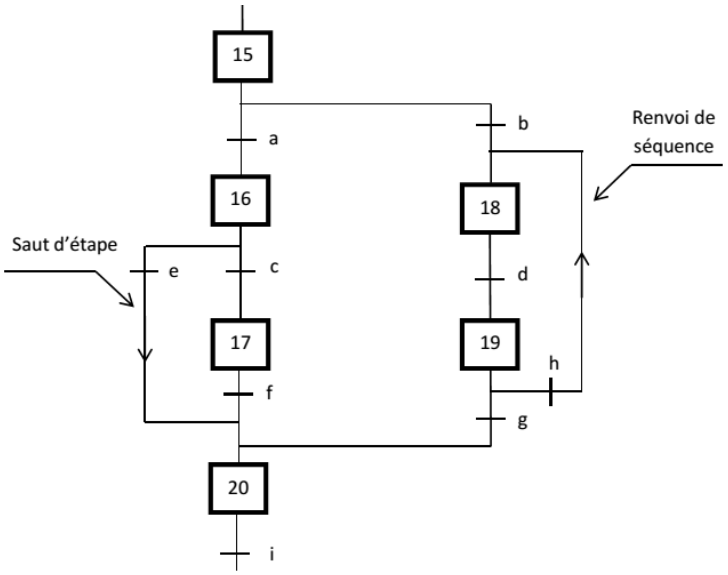
III.3.2.2- Fin de sélection (convergence en OU) :

Lorsque l'étape **20.2** est active et la réceptivité « **c** » est vrais « **c=1** », l'évolution s'effectue vers l'étape **21**.



III.3.2.3- Saut d'étape et reprise de séquence :

Un saut d'étape permet de sauter une ou plusieurs étapes lorsque les actions associées à ces étapes deviennent inutiles (ex: perçage avec ou sans débourrage). Un renvoi de séquence permet d'effectuer plusieurs fois une même séquence tant qu'une condition n'est pas réalisée (ex : remplissage d'un produit). L'exemple suivant résume les principes de saut d'étapes et de renvoi de séquence.

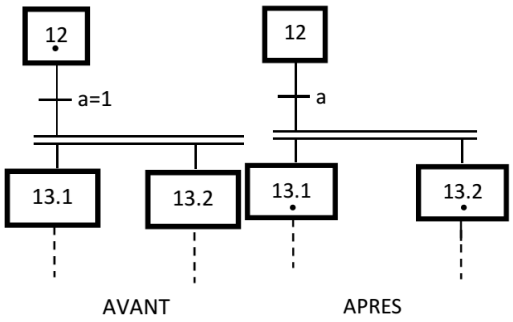


III.3.3- Séquences simultanées :

Les séquences simultanées permettent à partir d'une ou plusieurs étapes d'évoluer vers plusieurs séquences **simultanément.**

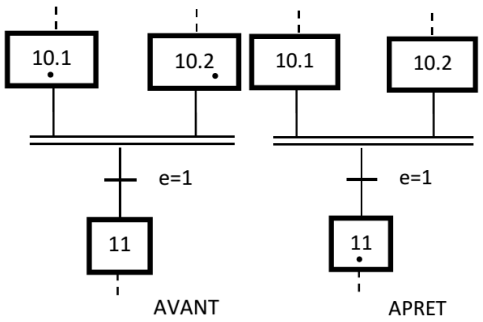
III.3.3.1- Divergence en ET :

Deux ou plusieurs séquences peuvent être simultanément activées à partir de la même transition. Les deux traits parallèles mettent en évidence l'activation simultanée des étapes **13.1** et **13.2** à partir de la réceptivité **a=1**, lorsque l'étape **12** est active.



III.3.3.2- Convergence en ET :

La convergence (ou jonction) entre plusieurs branches parallèles ne pourra s'effectuer que lorsque toutes les séquences seront terminées (étapes **10.1** et **10.2** actives) et la réceptivité commune est vraie (**e=1**).

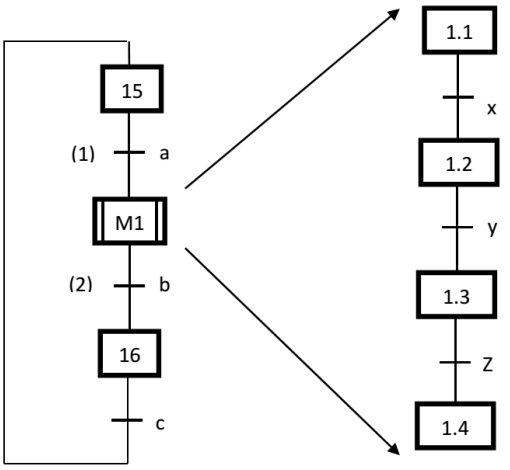


III.3.4- Extension des représentations (Macro-étapes) :

Une macro-étape est **une représentation unique** d'un ensemble d'étapes et de transitions. Le concept de macro-étape permet :

- Lors de l'analyse, de ne pas surcharger la représentation de détails (représentation structurée) ,

- Lors de l'exploitation, une meilleure compréhension du fonctionnement.

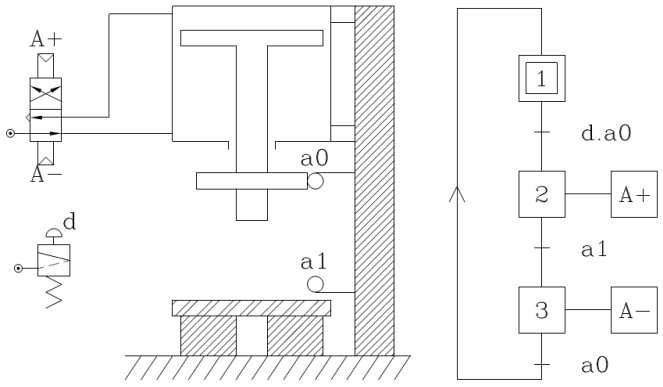


L'expansion de la macro-étape commence par une seule étape d'entrée et finit par une seule étape de sortie.

- Le franchissement de la transition amont **(1)** de la macro-étape active l'étape d'entrée (**1.1**).
- L'étape de sortie (**1.4**) valide la transition aval **(2)** de la macro-étape. Celle-ci sera désactivée lorsque cette transition aval est franchie.

III.4- Exemple d'application

On considère un exemple simple d'une poinçonneuse semi-automatique (voir figure (a) ci-dessous), qui se compose d'une table fixe recevant la tôle à poinçonner et d'un poinçon mobile. Au repos, le poinçon est en position haute. Lorsque l'opérateur appuie sur le bouton départ, le poinçon descend en position basse puis remonte en position haute et retourne au repos. Ce cycle se répète à chaque pression du bouton départ. Pour cet exemple, supposons que les mouvements de descente et de montée sont obtenus par un vérin pneumatique à double effet (A+ et A-), que les informations haut et bas sont obtenues au moyen de fins de course pneumatiques (a0 et a1), et que l'information de départ est donnée par un bouton poussoir pneumatique (d). Le Grafcet correspondant est donné à la figure (b) ci-dessous :

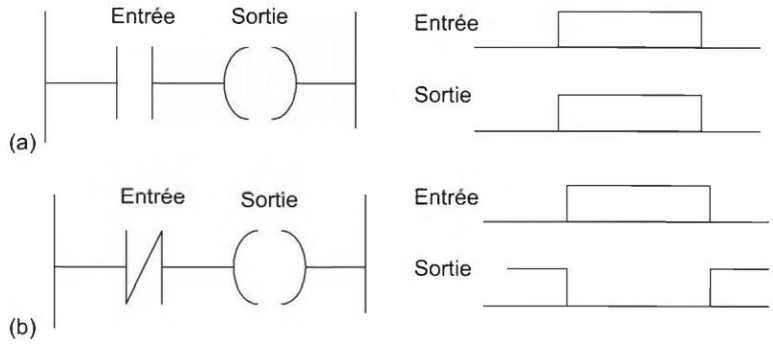


(a) Schéma du système (b) Grafcet

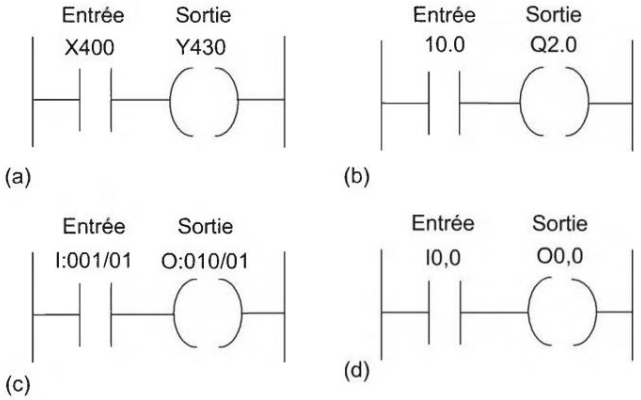
IV- Le langage à contacts (Ladder)

La programmation des API se fonde très souvent sur les *schémas à contacts.* L'écriture d'un programme équivaut à tracer un circuit de commutation. Le schéma à contacts est constitué de deux lignes verticales, qui représentent les barres d'alimentation, les éléments du circuit étant connectés sous forme de lignes horizontales entre ces deux lignes verticales.

Pour illustrer le tracé des lignes horizontales d'un schéma à contacts, prenons le cas où la mise sous tension d'un dispositif de sortie, comme un moteur, est déclenché par la fermeture d'un interrupteur de départ normalement ouvert. La figure suivante présente le schéma a contacts correspondant.

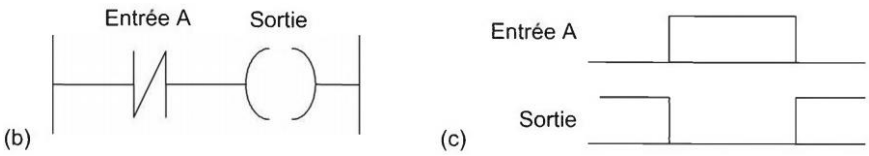


La figure ci-dessous présente le schéma à contacts précédent, tel qu'il serait dessiné avec les adresses au format : (a) Mitsubishi, (b) Siemens, (c) Allen-Bradley et (d) Télémécanique.



IV.1- Fonctions logiques :

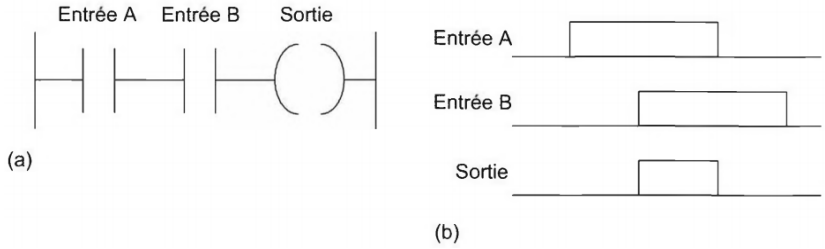
IV.1.1- NOT (NON) :



Une porte NON est par exemple utilisée avec une lampe qui s'allume lorsqu'il fait nuit. Autrement dit, en cas d'absence de lumière sur le capteur de luminosité, la sortie est active.

IV.1.2- AND (ET) :

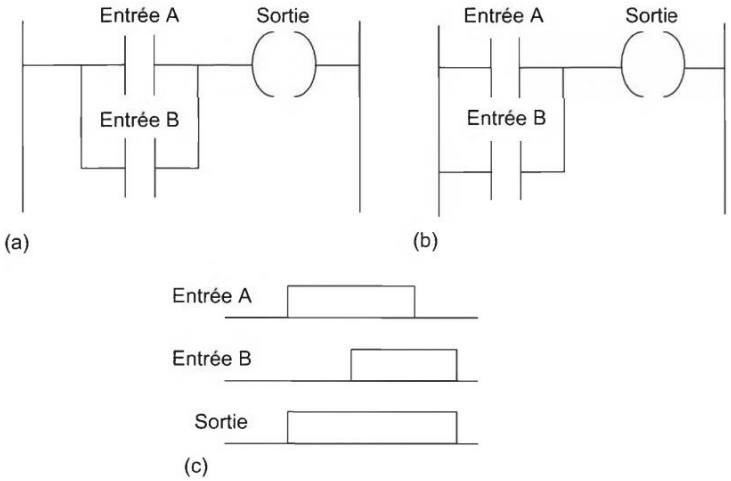
Dans un schéma a contacts, les contacts placés sur une ligne horizontale, c'est-à-dire des contacts en série, représentent les opérations logiques ET.



Une porte ET est par exemple utilisée dans le système de verrouillage d'une machine-outil pour quelle ne démarre que si le dispositif de sécurité est en place et si elle est sous tension.

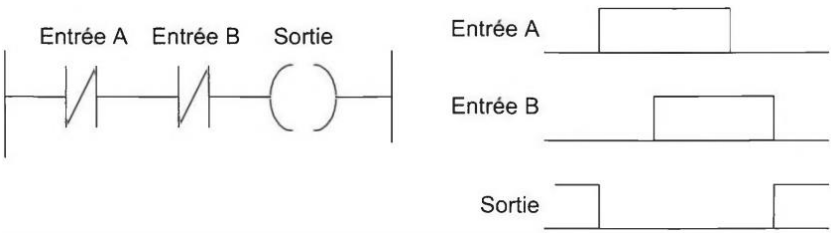
IV.1.3- OR (OU) :

Des chemins alternatifs fournis par des lignes verticales à partir de la ligne principale d'un schéma contacts, c'est-à-dire des chemins en parallèle, représentent les opérations logiques OU.



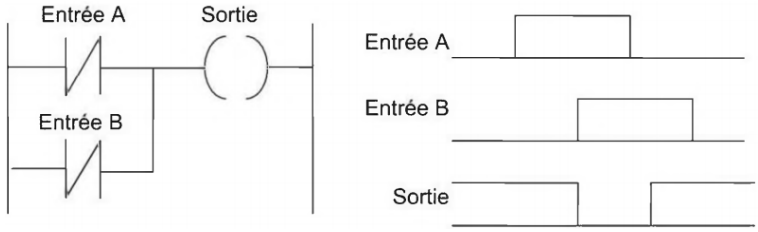
Une porte OU est par exemple utilisée dans une bande transporteuse qui convoie des bouteilles en vue de leur mise en carton. Un déflecteur est active pour dévier une bouteille vers un bac de rebut si son poids n'est pas dans un certain intervalle ou si la capsule n'est pas posée.

IV.1.4- NAND (NON-ET) :

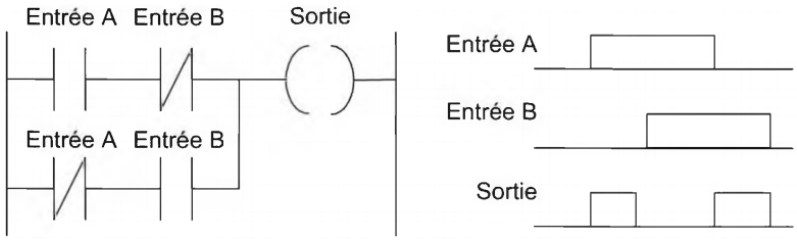


Une porte NON-ET est par exemple utilisée lorsqu'une lampe témoin s'allume si, dans une machine-outil, le dispositif de sécurité et l'interrupteur de fin de course signalant la présence de la pièce ne sont pas actives.

IV.1.5- NOR (NON-OU) :

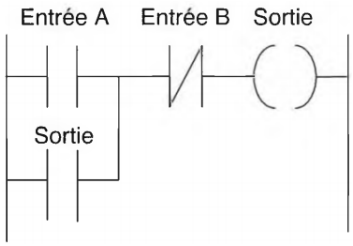


IV.1.6- XOR (OU-exclusif) :



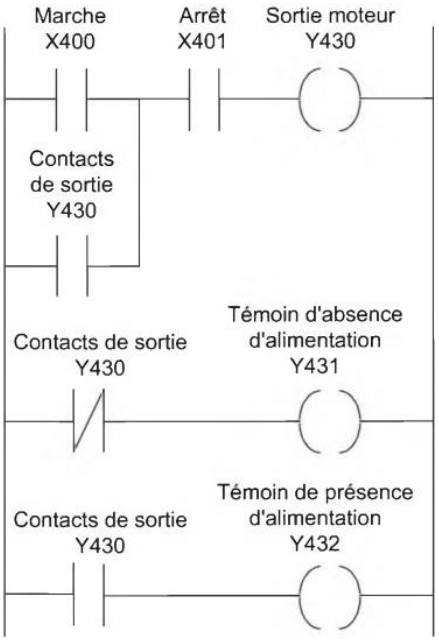
IV.2- Verrouillage :

Dans de nombreuses situations, il est nécessaire de maintenir la sortie alimentée même lorsque l'entrée disparait. C'est par exemple le cas d'un moteur démarré par l'appui sur un bouton poussoir. Bien que les contacts de l'interrupteur ne restent pas fermés, le moteur doit continuer à fonctionner jusqu'à l'appui sur un bouton poussoir d'arrêt. Un *circuit à verrouillage* est un circuit qui permet ce type de fonctionnement. II s'agit d'un circuit de maintien car, après avoir été excité, il conserve cet état jusqu'à la réception d'une autre entrée.



Pour illustrer la mise en application d'un circuit à verrouillage, prenons un moteur commandé par deux boutons poussoirs, un de démarrage et l'autre d'arrêt. Par ailleurs, un témoin lumineux s'allume lorsque le moteur est sous tension, tandis qu'un autre s'allume lorsqu'il ne l'est pas. La figure suivante présente un schéma à contacts avec des adresses au format Mitsubishi.

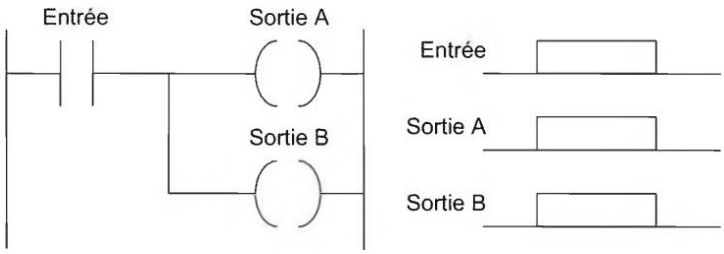
Le verrouillage est souvent employé avec les démarrages afin que l'activation initiale d'une application soit verrouillée.



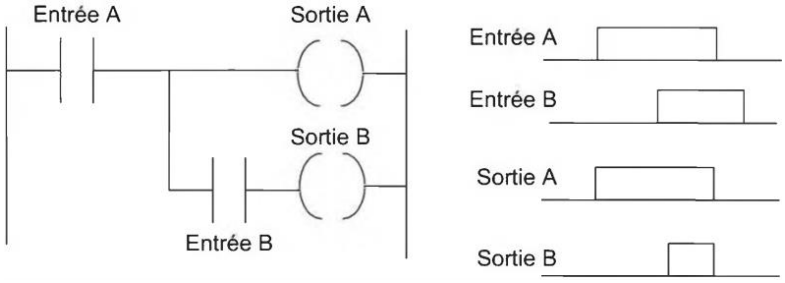
Notant que dans la figure ci-dessus, le contact d'arrêt X401 est représenté ouvert. Si l'interrupteur d'arrêt utilisé est normalement fermé, X401 reçoit un signal de démarrage pour se fermer. Cela permet un fonctionnement plus sûr que de programmer X401 en normalement fermé.

IV.3- Sorties multiples :

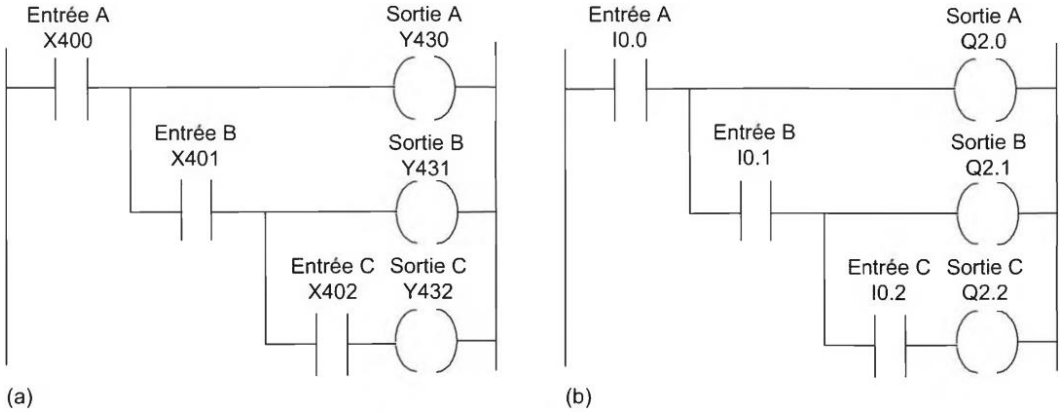
Dans un schéma à contacts, plusieurs sorties peuvent être connectées à un contact. La figure suivante présente un tel schéma avec deux bobines de sortie. Lorsque le contact d'entrée se ferme, les deux bobines produisent des sorties.



Pour le schéma à contacts ci-dessous, la sortie A se produit lorsque l'entrée A se produit. La sortie B se produit uniquement lorsque l'entrée A et l'entrée B sont présentes.

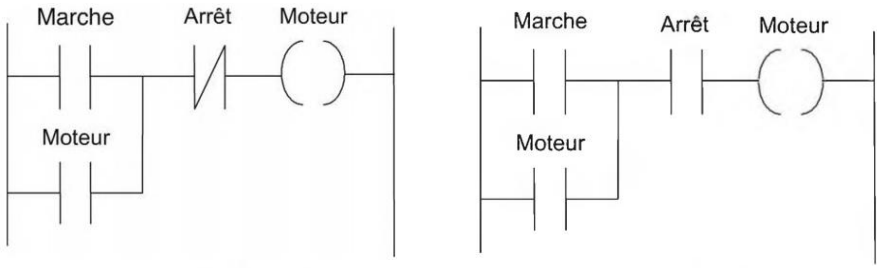


Ce type de disposition permet de produire des sorties en séquence, l'ordre étant celui dans lequel les contacts sont fermés. La figure suivante illustre ce principe avec un programme, donné dans les notations de Mitsubishi (a) et de Siemens (b). Les sorties A, B et C sont activées dans cet ordre lorsque les contacts sont fermés dans l'ordre A, B et C. Tant que l'entrée A n'est pas fermée, aucune des sorties ne peut être activée. Lorsque l'entrée A est fermée, la sortie A est activée. Ensuite, lorsque l'entrée B est fermée, la sortie B est activée. Enfin, lorsque l'entrée C est fermée, la sortie C est activée.



IV.4- Positionnement des interrupteurs d'arrêt :

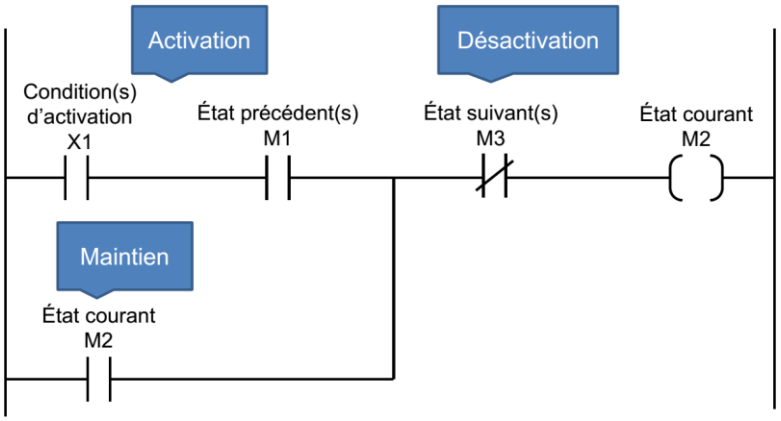
Dans de nombreuses applications, le positionnement des interrupteurs d'arrêt doit être soigneusement étudié afin de garantir la sécurité du système. Un interrupteur d'arrêt ne donne pas un système fiable s'il est normalement fermé et s'il doit être ouvert pour déclencher l'action d'arrêt. En effet, si l'interrupteur ne fonctionne pas correctement et reste fermé, l'arrêt du système est impossible (voir la partie (a) de la figure ci-dessous). Dans le schéma à contacts, il est préférable que l'interrupteur d'arrêt soit programmé ouvert (voir la partie (b) de la même figure).



(a) Un interrupteur d'arrêt non sûr (b) Un interrupteur d'arrêt sûr

VI.5- Principe de séquençage d'états :

Ce principe est illustré par le diagramme à contact suivant :



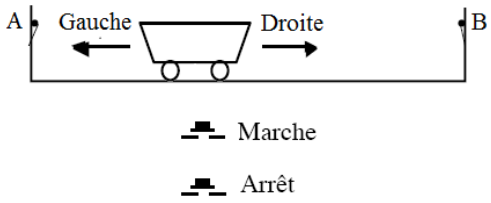
Où X1 représente une entrée de l'automate et les états Mi représentent des relais (mémoires) internes.

V- Traduction d'un Grafcet en Ladder :

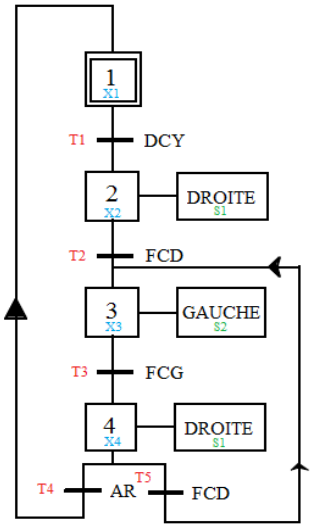
Un Grafcet peut être transformé en un diagramme Ladder. Pour des Grafcet complexes, il faudra travailler de façon systématique afin d'éviter certains problèmes. On écrira d'abord les conditions logiques associées au franchissement des transitions, puis celles d'activation et désactivation des étapes du Grafcet, pour finir avec les conditions pour les actions. L'ordre est important, car cette façon permet d'éviter les problèmes liés à l'ordre d'interprétation des diagrammes Ladder (du haut vers le bas).

V.1- Méthode des conditions de franchissement :

Afin d'illustré cette méthode, on considère l'exemple du système automatisé suivant :



Le chariot fait des navettes entre un point A (détecté par un capteur fin de course FCG) et un point B (détecté par un capteur fin de course FCD). On veut le commander grâce à deux boutons : Marche (DCY) et Arrêt (AR), de façon que le chariot commence à effectuer ses cycles une fois le bouton Marche aura été appuyé et il arrêtera après avoir terminé son cycle si l'on a appuyé sur le bouton Arrêt. Le Grafcet correspondant est donné comme suit :



On obtient les conditions suivantes pour les transitions (T_i correspond à une transition suivant l'étape X_i) :

$$T_1 = X_1.DCY$$
$$T_2 = X_2.FCD$$
$$T_3 = X_3.FCG$$
$$T_4 = X_4.AR$$
$$T_5 = X_4.FCD$$

Pour l'état des étapes, on a :

$$X_1 = T_4 + X_1.\overline{T}_1 + INIT$$
$$X_2 = T_1 + X_2.\overline{T}_2$$
$$X_3 = T_2 + T_5 + X_3.\overline{T}_3$$
$$X_4 = T_3 + X_4.\left(\overline{T_4 + T_5}\right) = T_3 + X_4.\overline{T}_4.\overline{T}_5$$

Si on prend en considération le positionnement des équations en étape initiale, les équations des étapes 2, 3, et 4 (i.e. hors l'étape initiale) deviens :

$$X_2 = \left(T_1 + X_2.\overline{T}_2\right).\overline{INIT}$$
$$X_3 = \left(T_2 + T_5 + X_3.\overline{T}_3\right).\overline{INIT}$$
$$X_4 = \left(T_3 + X_4.\overline{T}_4.\overline{T}_5\right).\overline{INIT}$$

Dans cet exemple, l'action DROITE est actionnée aux étapes 2 et 4. L'action GAUCHE est actionnée uniquement à l'étape 3. En fait, les sorties (actions) sont modélisés comme suit :

$$DROITE = X_2 + X_4$$
$$GAUCHE = X_3$$

Le schéma à contacts doit être réalisé donc avec les conditions précédentes dans l'ordre énoncé.

Lors de l'initialisation du Grafcet, toutes les étapes autres que les étapes initiales sont désactivées. Ceci peut être obtenu pour notre exemple en testant l'état repos de toutes les mémoires d'étape suivantes, pour venir alors systématiquement enclencher la mémoire X_1, ce qui implique que : $INIT = \bar{X}_2.\bar{X}_3.\bar{X}_4$.

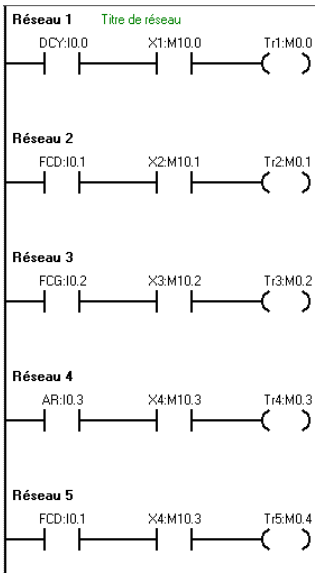
Pour la programmation en Ladder, les réceptivités sont affectées à des entrées, les transitions et l'état des étapes sont affectées à des mémoires internes (mémentos), et les actions sont affectées à des sorties.

En utilisant le logiciel STEP 7 MicroWIN V4.0 de l'automate SIEMENS S7-200, l'adressage correspondant est représenté avec la table des mnémoniques suivante :

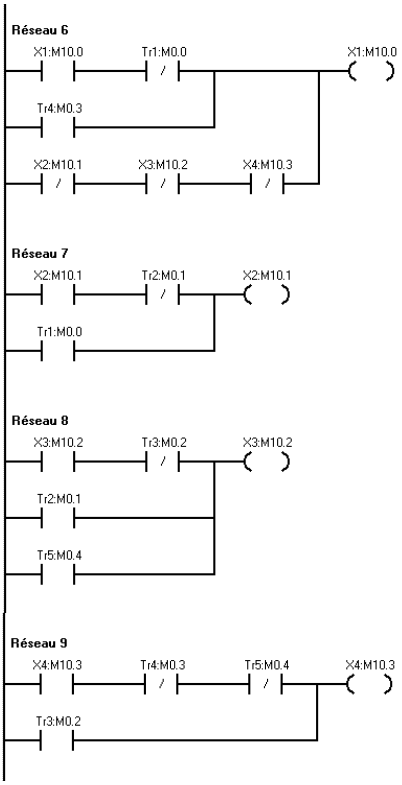
	Mnémonique	Adresse	Commentaire
1	DCY	I0.0	Entrée 1
2	FCD	I0.1	Entrée 2
3	FCG	I0.2	Entrée 3
4	AR	I0.3	Entrée 4
5	DROITE	Q0.0	Sortie 1
6	GAUCHE	Q0.1	Sortie 2
7	Tr1	M0.0	Transition 1
8	Tr2	M0.1	Transition 2
9	Tr3	M0.2	Transition 3
10	Tr4	M0.3	Transition 4
11	Tr5	M0.4	Transition 5
12	X1	M10.0	Etape 1
13	X2	M10.1	Etape 2
14	X3	M10.2	Etape 3
15	X4	M10.3	Etape 4

Suivant cette table, le programme en langage Ladder de cet exemple se donne comme suit :

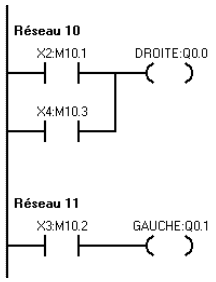
Pour les transitions :



Pour les états des étapes :



Pour les sorties :



V.2- Méthode des conditions d'activation et de désactivation :

Cette méthode peut être exprimée en tenant compte des équations d'activation (S_i , « set memory *i* ») et de désactivation (R_i , « reset memory *i* ») d'une étape, tout en utilisant des sorties (bobines) **Set / Reset**.

Les équations booléennes permettant de matérialiser le Grafcet de l'exemple précédent sont :

$$S_1 = X_4.AR + INIT \qquad R_1 = X_1.DCY$$
$$S_2 = X_1.DCY \qquad R_2 = X_2.FCD$$
$$S_3 = X_2.FCD + X_4.FCD \qquad R_3 = X_3.FCG$$
$$S_4 = X_3.FCG \qquad R_4 = X_4.(AR + FCD)$$

Il ne reste qu'à établir le schéma à contacts équivalent.

Remarque : Il est à noter que l'état d'une étape *i* peut être représenté en utilisant les équations d'activation S_i et de désactivation R_i sous la forme : $X_i = S_i + X_i.\overline{R}_i$.

VI- Les langages de programmation

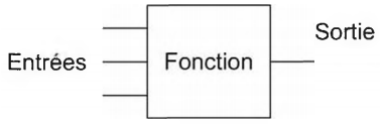
Il existe cinq langages de programmation des automates qui sont normalisés au plan mondial par la norme CEI 61131-3, à savoir :

- **Les langages Graphiques :**
  - Langage à contacts sous le sigle **LD** (Ladder Diagram),
  - Langage en blocs fonctionnels sous le sigle **FBD** (Function Block Diagram).
  - Diagramme de fonctions séquentielles sous le sigle **SFC** (**S**equential **F**unction **C**hart) (Grafcet).
- **Les langages textuels :**
  - Langage à liste d'instructions sous le sigle **IL** (Instruction Lists)
  - Langage littéral structuré sous le sigle **ST** (Structured Text).

En fait, cette section poursuite sur la lancée des précédentes en décrivant les autres langages de programmation définis par la CEI 61131-3, c'est-à-dire, les diagrammes de schémas fonctionnels (FBD**,** *Function Block Diagram*), les listes d'instructions (IL, *Instruction Lists),* et le texte structure (ST, *Structured Text).*

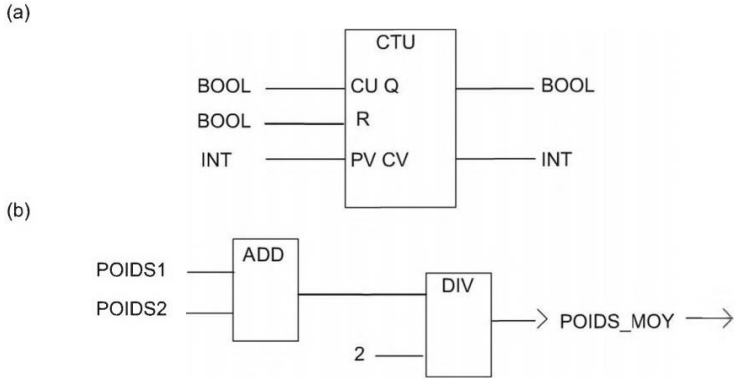
VI.1- Diagrammes de schémas fonctionnels :

Un *diagramme de schémas fonctionnels* est utilisé pour les programmes d'API décrits sous forme de blocs graphiques. II s'agit d'un langage graphique pour représenter les signaux et les données passant au travers de blocs, qui sont des éléments logiciels réutilisables. Un *bloc fonctionnel* est une instruction du programme qui, lorsqu'elle est exécutée, produit une ou plusieurs valeurs de sortie. La figure suivante montre la représentation d'un bloc. Le nom de la fonction est écrit dans la boite.



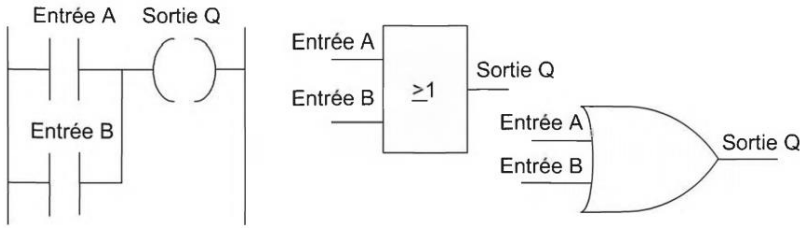
Un exemple de bloc fonctionnel standard (voir figure (a) ci-dessous) représente un compteur CTU qui produit une sortie sur Q lorsque le nombre d'impulsions sur l'entrée CU a atteint la valeur fixée par PV. A chaque impulsion d'entrée, la sortie CV est augmentée de 1. L'entrée R remet à zéro le compteur. Les libelles des entrées et des sorties indiquent le type du signal: BOOL pour booléen et INT pour entier.

Un autre exemple (voir figure (b)) de bloc défini par l'utilisateur pour calculer la moyenne de deux poids. II peut être réutilisé ensuite dans d'autres blocs fonctionnels.

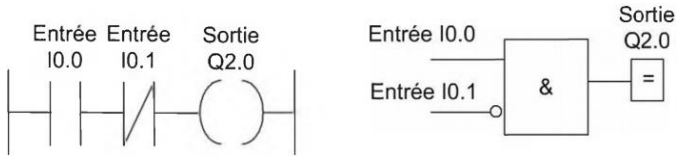


VI.1.1- Portes logiques :

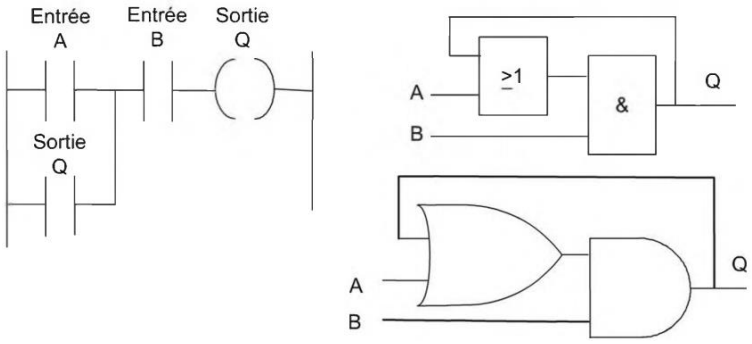
Pour illustrer la forme d'un tel diagramme et son lien avec un schéma à contacts, la figure ci-dessous prend l'exemple d'une porte OU. Lorsque l'entrée A ou l'entrée B est à 1, la sortie Q est présente.



L'exemple suivant montre un schéma à contacts et le bloc fonctionnel équivalent selon la notation de Siemens. Le bloc = indique une sortie du système.

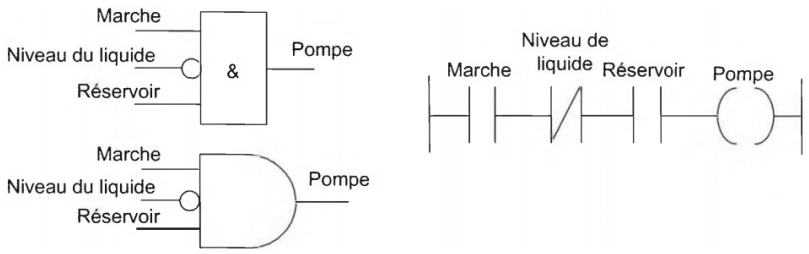


La figure ci-après présente un schéma à contacts qui met en œuvre une sortie dont les contacts servent d'entrée (verrouillage). Le diagramme de schémas fonctionnels équivalent inclut une boucle de retour.



Examinons la création d'un diagramme de schémas fonctionnels et d'un schéma à contacts pour une application dans laquelle une pompe doit être activée pour pomper le contenu d'un réservoir lorsque le bouton marche est fermé, lorsque le niveau du liquide dans

le réservoir est en dessous du niveau requis et lorsque le réservoir contient le liquide. Nous faisons face à une logique ET entre l'entrée de l'interrupteur de démarrage et l'entrée d'un capteur qui est au niveau haut tant que le liquide dans le réservoir se trouve sous le niveau requis. Ces deux conditions se retrouvent également dans une logique ET avec un interrupteur qui signale la présence de liquide dans le réservoir. Supposons que cet interrupteur produise une entrée lorsque du liquide est présent. Le diagramme de schémas fonctionnels qui correspond à ce cas et le schéma à contacts équivalent sont données comme suit :

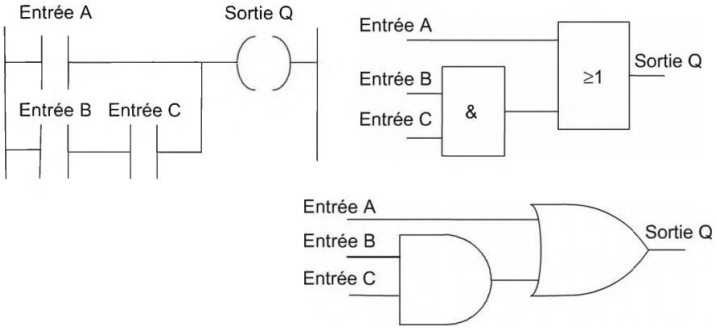


Afin d'illustrer la manière de lier les expressions booléennes aux schémas à contacts, prenons l'expression suivante :

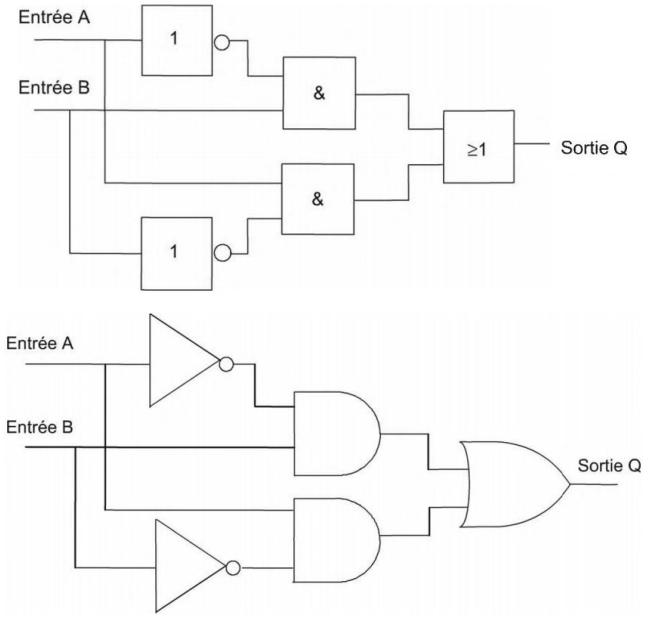
$$A + B.C = Q$$

Pour que la sortie Q soit présente, il faut que les entrées A ou B et C soient présentes. La figure ci-dessous montre le schéma à contacts et le diagramme de schémas fonctionnels correspondants. En utilisant les notations de Mitsubishi et de Siemens, respectivement, l'expression précédente devient :

$$X400 + X401.X402 = Y430$$
$$I0.0 + I0.1.I0.2 = Q2.0$$

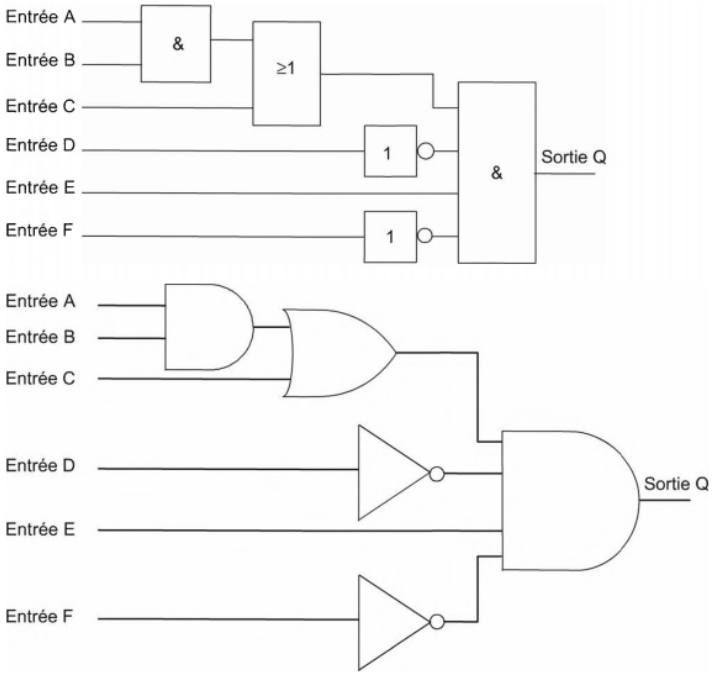


Examinons à présent la porte OU exclusif constituée de portes NON, ET et OU (voir la figure ci-dessous).



Les entrées de la porte ET inferieure sont A et B. Par conséquent, sa sortie est donnée par l'expression : $A.\bar{B} = Q$. Les entrées de la porte ET supérieure sont A et B. Par conséquent, sa sortie est donnée par l'expression : $\bar{A}.B = Q$. L'expression booléenne de la sortie de la porte OU est donc : $A.\bar{B} + \bar{A}.B = Q$.

Etudions un circuit logique avec de nombreuses entrées et sa représentation par une expression booléenne et une ligne d'un schéma à contacts (voir figure ci-dessous). La sortie de la porte ET supérieure répond à l'expression : $A.B$. La sortie de la porte OU suivante est alors : $A.B + C$. La sortie Q de la dernière porte ET est donnée par l'expression : $Q = (A.B + C).\bar{D}.E.\bar{F}$.



Le schéma à contacts qui correspond à cette expression est donné comme suit :

VI.2- Listes d'instructions :

Les listes d'instructions constituent une méthode de programmation que l'on peut comparer à la saisie d'un schéma à contacts sous forme d'un texte. Un programme écrit selon cette méthode est constitué d'une suite d'instructions, chacune placée sur une nouvelle ligne. Chaque instruction est constituée d'un opérateur suivi d'un ou plusieurs opérandes, c'est-à-dire les objets de l'opérateur.

Par exemple, nous pouvons avoir la ligne suivante : `LD A`. Cette ligne indique le chargement de l'opérande `A`, `LD` est l'opérateur utilisé pour effectuer un chargement. Voici une autre instruction : `OUT Q`. Elle indique qu'une sortie doit être produite sur `Q`.

Comparé au langage à contacts, un opérateur peut être vu comme un élément d'une ligne et `LD` équivaut à commencer une ligne par des contacts ouverts pour l'entrée `A`.

Des codes mnémoniques sont utilisés pour les opérateurs, chaque code correspondant à un opérateur/élément d'un schéma à contacts. Ces codes diffèrent d'un fabricant à l'autre, mais la norme CEI 61131-3 a été établie et est à présent largement adoptée. Le tableau suivant recense quelques codes utilisés par les fabricants et les instructions normalisées que nous rencontrerons dans ce chapitre.

CEI 61131-3	Mitsubishi	OMRON	Siemens	Opération	Langage Ladder
LD	LD	LD	LD (A)	Charger l'opérande dans un registre de résultat.	Commencer une ligne avec des contacts ouverts
LDN	LDI	LD NOT	LDN (AN)	Charger l'opérande inverse dans un registre de résultat.	Commencer une ligne avec des contacts fermes.
AND	AND	AND	A	ET Booléen	Eléments en série avec des contacts ouverts.
ANDN	ANI	AND NOT	AN	ET booléen avec un opérande inversé.	Eléments en série avec des contacts fermés.
OR	OR	OR	O	Ou booléen	Eléments en parallèle avec des contacts ouverts.
ORN	ORI	OR NOT	ON	OU booléen avec un opérande inversé.	Eléments en parallèle avec des contacts fermés.
ST	OUT	OUT	=	Stocker un registre de résultats dans un opérande.	Une sortie

Voici une illustration de l'utilisation des opérateurs définis par la norme CEI 61131-3 :

```
LD      A     (*Charger A*)
AND     B     (*ET B*)
ST      Q     (*stocker le résultat dans Q*)
```

Sur la première ligne du programme, `LD` représente l'opérateur, `A` représente l'opérande et les mots placés en fin de lignes, entre parenthèses et précédés et suivis par un caractère `*`, sont des commentaires qui expliquent l'opération et ne font pas partie des instructions du programme de I'API.

II est possible d'utiliser des libellés pour identifier différents points d'entrée dans un programme. Ils sont notamment utiles pour sauter à des endroits d'un programme. Un libellé se place avant l'instruction et en est séparé par des deux-points :

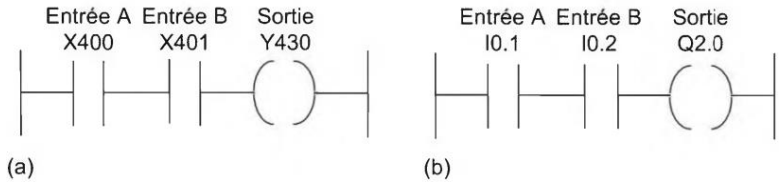
```
P0MPE_0K: LD      C     (*Charger C*)
```

Une instruction dans le programme peut sauter à la ligne `P0MPE_0K` lorsqu'une condition particulière est rencontrée.

VI.2.1- Schémas à contacts et listes d'instructions

Pour correspondre au début d'une ligne horizontale d'un schéma à contacts, nous devons, dans une liste d'instructions, employer un code « débuter une ligne ». II peut s'agir de `LD`, ou peut-être `L`, pour indiquer que la ligne commence par des contacts ouverts, ou de `LDI`, ou peut-être `LDN`, `LD NOT` ou `LN`, pour indiquer quelle commence par des contacts fermés. Toutes les lignes doivent se terminer par une sortie ou un code de stockage du résultat. II peut s'agir de `OUT`, ou peut-être de = ou `ST`.

Nous allons voir comment des lignes individuelles d'un schéma à contacts sont saisies à l'aide des mnémoniques de Mitsubishi et de Siemens pour la porte ET illustrée ci-dessous :

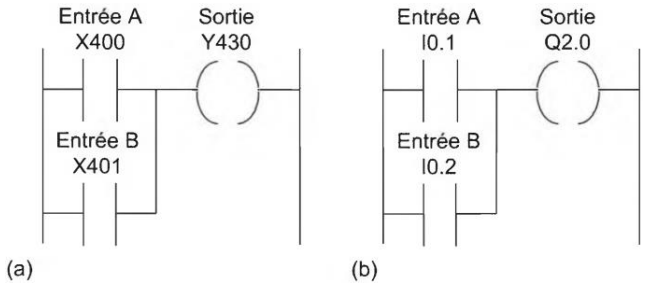


La ligne du schéma à contacts correspond donc à la liste d'instructions suivante :

```
LD    X400          LD    I0.1
AND   X401          A     I0.2
OUT   Y430          =     Q2.0
```

(a) Mitsubishi (b) Siemens

Prenons un autre exemple, celui d'une porte OU. La figure suivante présente cette porte avec les notations de Mitsubishi et de Siemens.

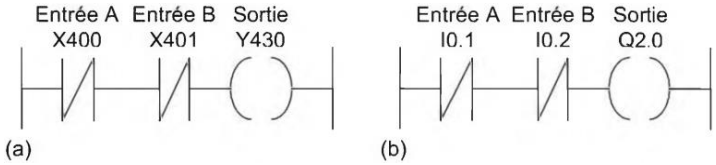


La liste d'instructions est la suivante :

```
LD    X400          LD    I0.1
OR    X401          O     I0.2
OUT   Y430          =     Q2.0
```

(a) Mitsubishi (b) Siemens

La figure ci-dessous présente le schéma à contacts d'une porte NON-OU dans les notations de Mitsubishi et de Siemens.

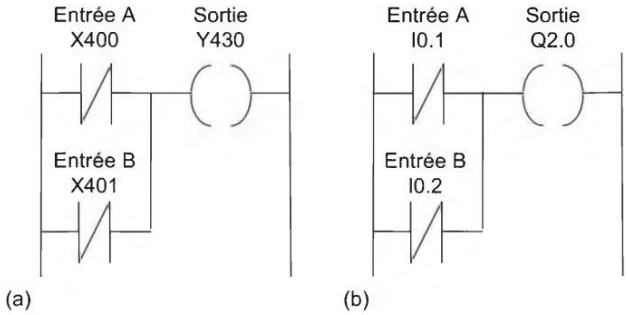


Voici la liste des instructions pour la porte NON-OU du schéma à contacts :

```
LDI   X400          LDN   I0.1
ANI   X401          AN    I0.2
OUT   Y430          =     Q2.0
```

(a) Mitsubishi (b) Siemens

De même, la porte NON-ET est illustrée comme suit :

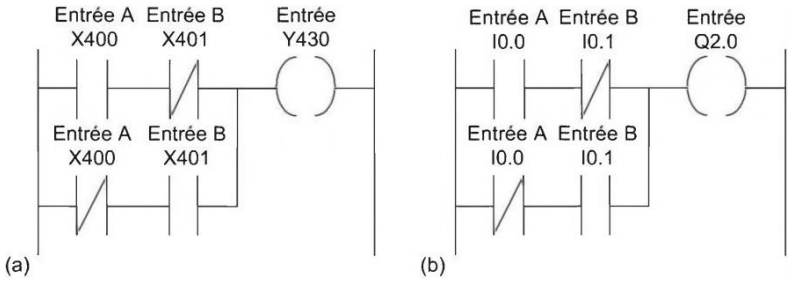


Voici la liste d'instructions complète :

```
LDI   X400          LDN   I0.1
ORI   X401          ON    I0.2
OUT   Y430          =     Q2.0
```

(a) Mitsubishi (b) Siemens

La porte OU-exclusif de la figure ci-dessous comprend deux branches en parallèle avec un ET dans chacune d'elles.



La figure (a) montre la version Mitsubishi. Dans ce genre de situation, Mitsubishi utilise une instruction `ORB` (`OLD` dans Siemens) pour indiquer un « OU entre des branches en parallèle ». Rappelons que les nouvelles lignes commencent toutes par `LD` ou `LDI` (`LDN` dans Siemens). L'intégralité de la liste d'instructions correspondante est illustrée avec la notation de Mitsubishi comme suit :

```
LD      X400    (*début de la première branche*)
ANI     X401
LDI     X400    (*début de la deuxième branche*)
AND     X401
ORB             (*OU entre les deux branches*)
OUT     Y430
```

Pour la notation de Siemens, on a :

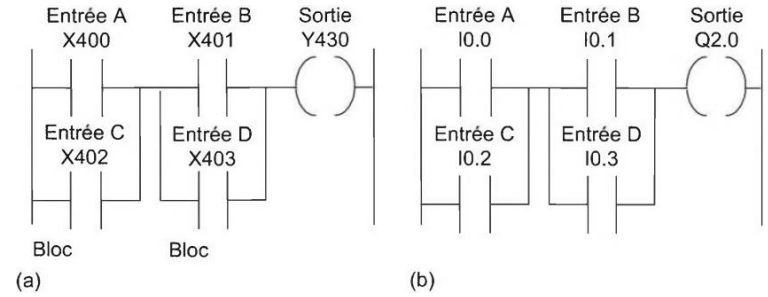
```
LD      I0.0    (*début de la première branche*)
AN      I0.1
LDN     I0.1    (*début de la deuxième branche*)
A       I0.1
OLD             (*OU entre les deux branches*)
=       Q2.0
```

Les parenthèses sont utilisées également dans la norme CEI 61131-3 (comme dans la notation de Siemens), indiquant que certaines instructions forment un bloc. Elles jouent le même rôle que les parenthèses dans une équation mathématique. Par exemple, (2 + 3) / 4 signifie que 2 et 3 doivent être additionnés avant que le résultat ne soit divisé par 4. Ainsi, la liste d'instructions CEI suivante :

```
LD      X
ADD(    B
MUL(    C
ADD     D
)
)
```

correspond à l'expression : `X + (B × (C + D))`.

La figure suivante présente un circuit effectuant un ET entre des blocs à deux branches. La version Mitsubishi est donnée à la partie (a), tandis que celle de Siemens est illustrée à la partie (b).



Voici la liste d'instructions correspondante :

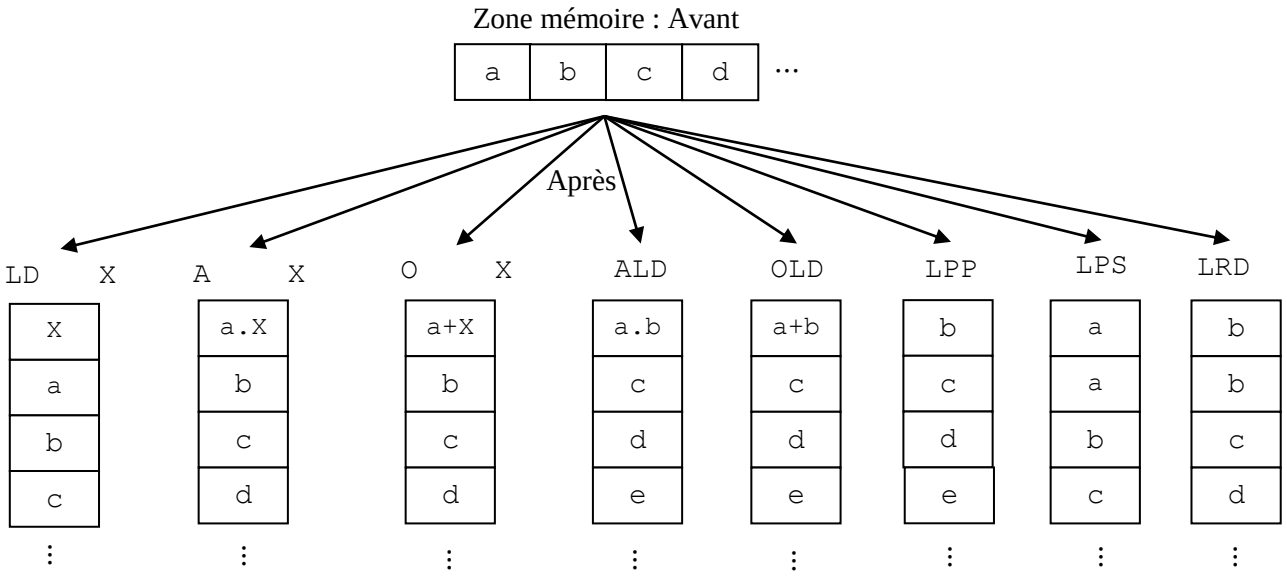
```
LD      X400            LD      I0.0
OR      X402            O       I0.2
LD      X401            LD      I0.1
OR      X403            O       I0.3
ANB                     ALD
OUT     Y430            =       Q2.0
```

(a) Mitsubishi (b) Siemens

Pour la version Siemens S7-200, quelques instructions de base sont données dans le tableau suivant :

Instruction	Description
LD	Lire (Load)
LDN	Lire inverse (Load NOT)
A	Faire le ET (AND)
AN	Et avec inverse (AND NOT)
ALD	Faire le ET entre les deux premières cases (branches)
O	Faire le OU (OR)
ON	OU avec inverse (OR NOT)
OLD	Faire le OU entre les deux premières cases (branches)
NOT	Faire l'inverse (NOT)
S @,N	Mise à 1 des N sorties successives à partir de la sortie @
R @,N	Mise à 0 des N sorties successives à partir de la sortie @
LPS	Doubler la 1ère case (branche)
LPP	Enlever la 1ère case (branche)
LRD	Enlever la 1ère case et doubler la 2ème
= @	Mettre le résultat dans l'adresse **@**
EU	Prendre le front montant
ED	Prendre le front descendant

Un exemple exploitant ces instructions est donné comme suit :



VI.3- Texte structuré

Le texte structuré est un langage de programmation qui ressemble énormément au langage Pascal. Les programmes sont écrits sous forme d'une suite d'instructions séparées par des points-virgules. Il s'agit d'instructions prédéfinies et de sous-routines qui permettent de modifier des variables, celles-ci étant des valeurs définies, des valeurs mémorisées de manière interne ou des entrées et des sorties.

Une instruction d'affectation permet de modifier la valeur d'une variable :

```
Lampe := InterrupteurA;      (i.e. Y := X;)
```

Ici, la lampe est allumée par l'interrupteur A. Voici un autre exemple :

```
Lampe := InterrupteurA OR InterrupteurB;
```

La lampe est allumée par l'interrupteur A ou par l'interrupteur B. La fonction ET existe également :

```
Marche := Vapeur AND Pompe;
```

La mise en marche se produit lorsque le détecteur de vapeur et la pompe sont actifs.

Le tableau suivant recense quelques opérateurs utilisés dans un texte structuré :

Opérateur	Description
(...)	Expression parenthésée
Function(...)	Liste des paramètres d'une fonction
**	Elévation a une puissance
-, NOT	Négation, NON booléen
*, /, MOD	Multiplication, division, modulo
+, -	Addition, soustraction
<, >, <=, >=	Inferieur à, supérieur à, inférieur ou égal à, supérieur ou égal à
=, <>	Egalité, inégalité
AND, &	ET booléen

OR	OU booléen
XOR	Ou exclusif booléen

Les parenthèses permettent de regrouper des expressions au sein d'autres expressions afin de fixer l'ordre d'évaluation. Par exemple :

```
EntreeA := 6;
EntreeB := 4;
EntreeC := 2;
SortieQ := EntreeA/3 + EntreeB/(3 - EntreeC);
```

Dans cet exemple, la valeur de `SortieQ` est : `2 + 4 = 6`.

Les identités des variables représentées directement commencent par le caractère `%` comme indiqué dans l'exemple suivant :

```
%IX100      (*Bit mémoire de l'entrée 100*)
%ID200      (*Mot mémoire de l'entrée 200*)
%QX100      (*Bit mémoire de la sortie 100*)
```

La première lettre est un `I` pour un emplacement mémoire d'entrée, `Q` pour une sortie ou `M` pour un emplacement mémoire interne. La deuxième lettre est `X` pour un bit, `B` pour un octet (8 bits), `W` pour un mot (16 bits), `D` pour un mot double (32 bits) ou `L` pour un mot long (64 bits).

L'opérateur `AT` permet de fixer l'emplacement mémoire d'une variable, par exemple :

```
Entreel  AT  %IX100;     (*Entreel correspond au bit mémoire*)
                         (*de l'entrée 100*)
```

VI.3.1- Instructions conditionnelles

Avec l'instruction `IF` suivante, si la variable de température du fluide est active, c'est-à-dire à 1, les actions qui viennent après cette ligne dans le programme sont exécutées :

```
IF temp_fluide THEN
```

Avec l'instruction `IF NOT`, si la variable de température du fluide n'est pas à 1, les actions qui viennent après cette ligne dans le programme sont effectuées :

```
IF NOT temp_fluide THEN
```

Dans le cas suivant :

```
IF temp_fluidel OR temp_fluide2 THEN
```

les actions qui viennent après l'instruction `IF` dans le programme sont exécutées si la variable 1 de température du fluide est à 1 ou si la variable 2 de température du fluide est active (i.e. à 1).

La combinaison `IF ... THEN ... ELSE` est utilisée pour exécuter des instructions sélectionnées lorsque certaines conditions se réalisent :

```
IF (Fin_de_coursel AND Piece_presente) THEN
     Portel := Ouvert;
     Porte2 := Ferme;
```

```
ELSE
     Portel := Ferme;
     Porte2 := Ouvert;
END_IF;
```

Voici un autre exemple avec des adresses d'API :

```
IF (I:000/00 = 1) THEN
     O:001/00 := 1;
ELSE
     O:000/01 := 0;
END_IF ;
```

Ici, si l'entrée `I:000/00` est à 1, la sortie `O:001/00` est mise à 1; sinon elle est mise à 0.

L'instruction `CASE` est utilisée pour exécuter des instructions lorsqu'une valeur entière particulière est rencontrée, sinon d'autres instructions sont exécutées. Par exemple, pour un contrôle de température, nous pouvons avoir le code suivant :

```
CASE (Temperature) OF
     0 ... 40:      Interrupteur_four := Marche;
     40 ... 100:    Interrupteur_four := Arret;
ELSE
                    Interrupteur_four  :=  Arret;
END_CASE;
```

Dans l'exemple suivant, les ventilateurs d'un moteur doivent tourner à des vitesses différentes en fonction de la position d'un interrupteur :

```
CASE Configuration_vitesse OF
     1:    vitesse := 5;
     2:    vitesse := 10;
     3:    vitesse := 15;   ventilo1 := Marche;
     4:    vitesse := 20;   ventilo2 := Marche;
ELSE
           Vitesse := 0;   defaut_vitesse := TRUE;
END_CASE;
```

VI.3.2- Instructions d'itération

Ces instructions permettent de répéter une ou plusieurs instructions un certain nombre de fois, en fonction de l'état d'une autre variable. La boucle `FOR ... DO` permet de répéter une suite d'instructions en fonction de la valeur d'une variable d'itération entière :

```
FOR Entree := 10 TO 0 BY -1
     DO
     Sortie := Entree;
END_F0R;
```

Ici, la sortie diminue de 1 chaque fois que l'entrée, qui va de 10 à 0, diminue de 1.

La boucle `WHILE ... DO` permet d'exécuter une ou plusieurs instructions tant qu'une expression booléenne reste vraie :

```
SortieQ := 0;
WHILE EntreeA AND EntreeB
     DO
     SortieQ := SortieQ + 1;
END_WHILE;
```

La boucle `REPEAT ... UNTIL` permet d'exécuter une suite d'instructions et de la répéter tant qu'une expression booléenne reste vraie :

```
SortieQ := 0;
REPEAT
     SortieQ := SortieQ + 1;
UNTIL (Entreel = Arret) OR (SortieQ > 5)
END_REPEAT;
```

VI.3.3- Ecriture des programmes

Les programmes doivent commencer par définir les types qui représentent les données :

```
TYPE Moteur: (Arrete, EnFonction);
END_TYPE;

TYPE Vanne: (Ouverte, Fermee);
END_TYPE;

TYPE Pression: REAL; (*La pression est une valeur analogique*)
END_TYPE;
```

Ensuite viennent les variables, c'est-à-dire les signaux issus des capteurs et les signaux de sortie qui seront utilisés dans le programme :

```
VAR_IN               (*Entrées*)
Defaut_Pompe: BOOL; (*Le défaut de fonctionnement de la pompe*)
END_VAR;             (*est une variable booléenne*)

VAR_0UT                (*Sorties*)
Vitesse_Moteur: REAL; (*la vitesse du moteur est une variable*)
END_VAR;               (*analogique*)

VAR_IN
Valeur: INT; (*La valeur est un entier*)
END_VAR;

VAR
Entreel AT %IX100; (*Entreel correspond au bit mémoire*)
END_VAR;           (*de l'entrée 100*)
```

Des valeurs initiales doivent être données aux variables :

```
VAR
Temp: REAL = 100; (*La valeur initiale de la variable Temp*)
END_VAR;          (*est un nombre analogique 100*)
```

Ce n'est qu'alors que vous pouvez ajouter les instructions.

Le code suivant est un exemple de bloc fonctionnel qui peut apparaitre dans un programme plus vaste ; il vérifie des tensions :

```
FUNCTION_BLOCK TEST_VOLTAGE
     VAR_INPUT
     VOLTS1, VOLTS2, VOLTS3
     END_VAR;
     VAR_OUTPUT
     SURTENSION: BOOL;
     END_VAR;
     IF VOLTS1 > 12 THEN
     SURTENSION := TRUE; RETURN;
     END_IF;
```

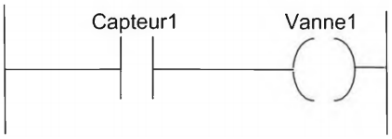
```
    IF VOLTS2 > 12 THEN
    SURTENSION := TRUE; RETURN;
    END_IF;
    IF VOLTS3 > 12 THEN
    SURTENSION := TRUE;
    END_IF;
END_FUNCTION_BLOCK;
```

Si le test des tensions 1, 2 et 3 montre que l'une d'elles est supérieure à 12, la sortie `SURTENSION` est fixée à `TRUE` et l'instruction `RETURN` est invoquée pour terminer l'exécution du bloc fonctionnel. Ailleurs dans le programme, lorsque `SURTENSION` est à `TRUE`, une certaine action sera réalisée.

VI.3.4- Comparaison avec le langage à contacts

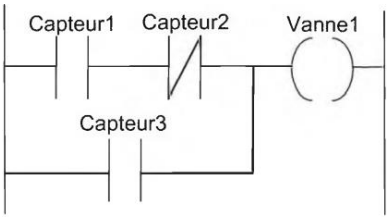
La figure ci-dessous montre un schéma à contacts simple et les expressions équivalentes en texte structuré :

```
Vanne1 := Capteur1;

IF Capteur1 THEN
     Vanne1 := 1;
END_IF;
```



Un autre schéma à contacts avec son équivalent sous forme de texte structuré sont illustrés comme suit :

```
Vanne1 := (Capteurl AND NOT Capteur2)
          OR Capteur3;

IF Capteur1 AND NOT Capteur2 THEN
     Vanne1 := 1;
ELSEIF Capteur3 THEN
     Vanne1 := 1;
END_IF;
```



VI.4- Relais (contacts) internes

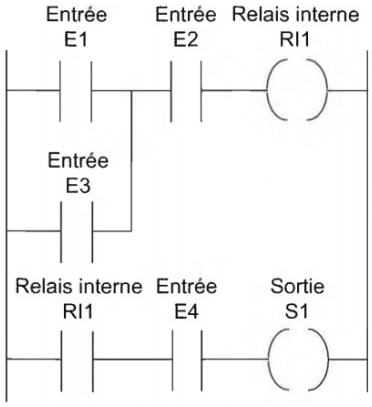
Dans les API, certains éléments sont utilisés pour conserver des données, c'est-à-dire des bits, et se comportent comme des relais, capables d'être allumés ou éteints et d'allumer ou d'éteindre d'autres dispositifs. Voilà donc l'origine du terme *relais interne*. Ces relais internes n'existent pas en tant que dispositif de commutation réels. II s'agit simplement de bits dans la mémoire de I'API qui se comportent comme des relais. Du point de vue programmation, ils peuvent être considérés de la même manière qu'un relais externe de sortie ou d'entrée.

Pour faire la différence entre des sorties de relais internes et des sorties de relais externes, leurs adresses sont de type diffèrent. Les fabricants ont tendance à employer des termes différents pour les relais internes et à exprimer leur adresse d'une autre manière. Par exemple, Mitsubishi emploie les termes *relais auxiliaires* ou *marqueurs* et la notation Ml00, Ml0l, … etc. Siemens utilise le terme *mémento* et la notation M0.0, M0.l, … etc (il existe d'autres

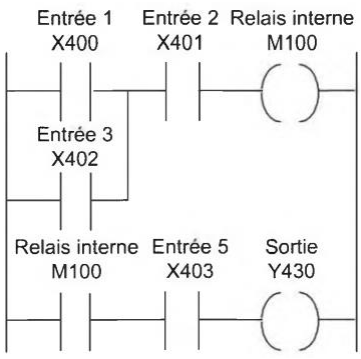
automates Siemens qui utilisent le terme *fanion* avec la notation F0.0, F0.l, …. etc). Télémécanique a choisi le terme *bit* et la notation B0, B1, … etc. Toshiba a opté pour *relais internes* et la notation R000, R001, … etc. Allen Bradley a retenu le terme *stockage binaire* et la notation B3/001, B3/002, … etc.

VI.4.1- Programmes avec plusieurs conditions d'entrée

Un exemple de levage automatique d'une barrière lorsqu'une personne s'en approche d'un côté ou de l'autre est considéré (voir la figure ci-dessous) :

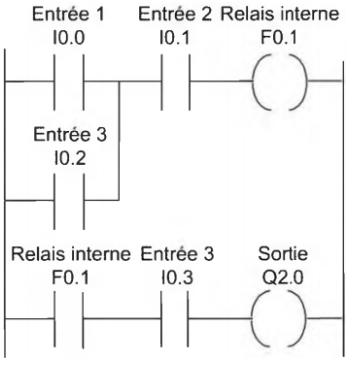


Les entrées El et E3 proviennent de capteurs photoélectriques qui détectent la présence d'une personne, quelle s'approche ou qu'elle s'éloigne de la barrière. El est activée pour l'un des côtes, E3, pour l'autre. L'interrupteur E2 permet au système d'être fermé. Ainsi, lorsque El ou E3, et E2, sont activées, il existe une sortie sur le relais interne RI1 et ses contacts se ferment. Si le capteur de fin de course E4 détecte que la barrière est fermée, il est alors activé et fermé. Le résultat est une sortie pour S1, qui représente un moteur qui lève la barrière. Si le capteur de fin de course E4 détecte que la barrière est déjà ouverte (i.e. S1 active), puisque la personne l'a traversée, il s'ouvre alors et la sortie S1 n'est plus activée. Un contrepoids se charge ensuite de fermer la barrière. Le relais interne a permis de lier deux parties du programme, l'une étant la détection de la présence d'une personne, l'autre, la détection de la barrière déjà levée ou abaissée. Les figures suivante montrent, respectivement, comment représenter le schéma à contacts précédent selon la notation de Mitsubishi et de Siemens.




```
LD      X400
OR      X402
AND     X401
OUT     M100
LD      M100
AND     X403
OUT     Y430
```

(a) Mitsubishi

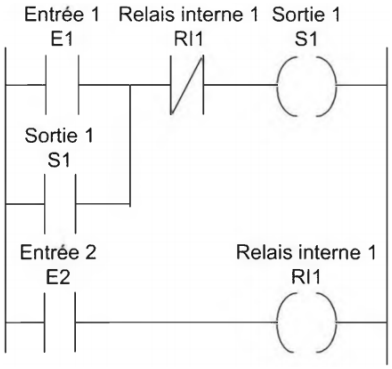



```
LD      I0.0
O       I0.2
A       I0.1
=       F0.1
LD      F0.1
A       I0.3
=       Q0.2
```

(b) Siemens

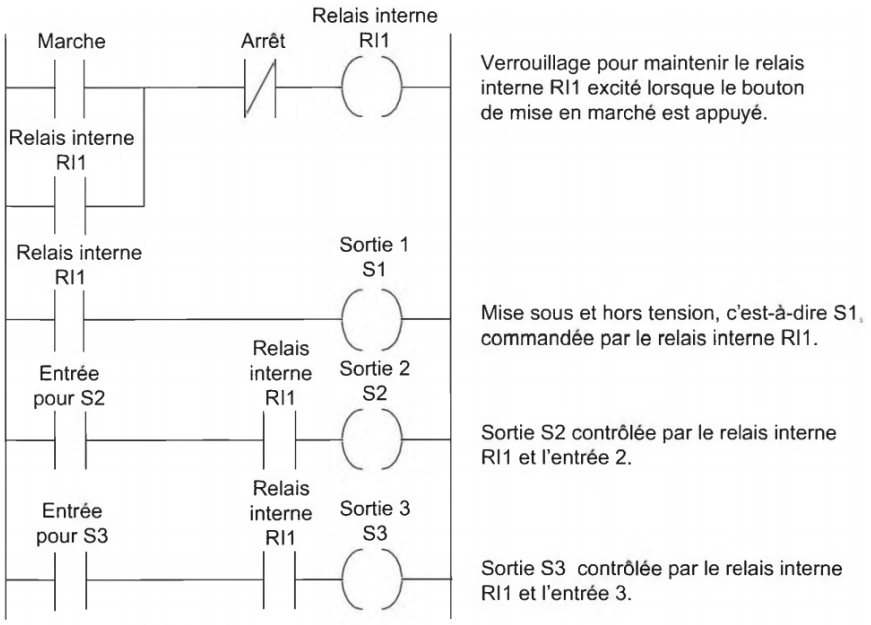
VI.4.2- Programmes à verrouillage

Les relais internes peuvent également être utilisés pour réinitialiser un circuit de verrouillage. La figure suivante montre un exemple de schéma à contacts qui met en œuvre ce type d'utilisation.



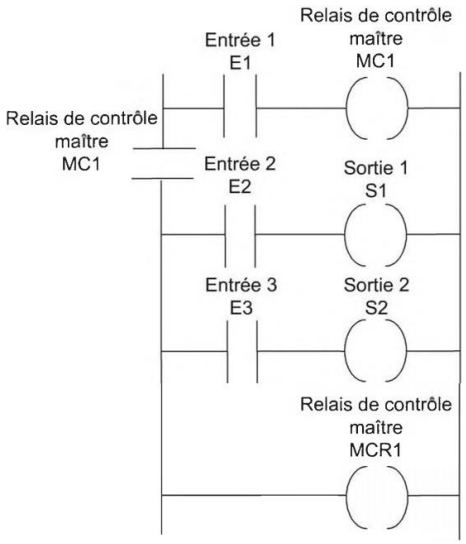
Lorsque les contacts de l'entrée E1 se ferment momentanément, une sortie est présente sur S1. Les contacts de S1 se ferment donc et la sortie est maintenue même lorsque l'entrée E1 disparaît. Lorsque l'entrée E2 est fermée, le relais interne RI1 est excité et ses contacts, normalement fermés, s'ouvrent. La sortie S1 est alors désactivée et déverrouillée.

Examinons un cas d'activation de plusieurs sorties, qui requiert des circuits de verrouillage, car une machine automatique est démarrée ou arrêtée à l'aide des boutons poussoirs. Un circuit à verrouillage applique ou coupe l'alimentation de la machine. Celle-ci possède plusieurs sorties, qui sont activées lorsque l'alimentation est appliquée, et désactivées lorsqu'elle est coupée. II est possible de concevoir un schéma à contacts qui se fonde sur des commandes verrouillées pour chacune de ces sorties. Toutefois, une méthode plus simple consiste à utiliser un relais interne comme illustré ci-dessous :



VI.4.3- Relais de contrôle maitre

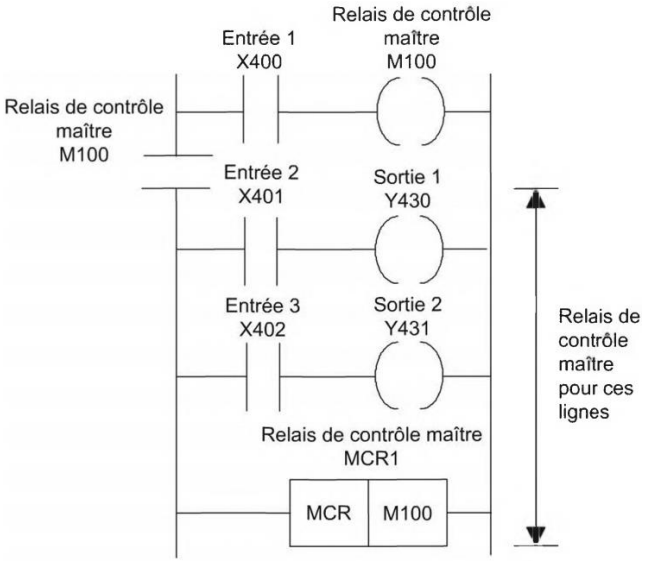
Lorsque le nombre de sorties à contrôler est important, il est parfois nécessaire que des sections entières d'un schéma à contacts soient activées ou désactivées en fonction de certaines conditions. Ce fonctionnement peut être obtenu en incluant les contacts du même relais interne dans chacune des lignes afin que sa commande les affecte toutes. Une autre solution se fonde sur le *relais de contrôle maitre* (MCR, *Master Control Relay).* La figure suivante illustre l'emploi d'un tel relais pour contrôler une section d'un schéma à contacts :



En l'absence d'entrée sur El, le relais interne de sortie MC1 n'est pas excité et ses contacts sont donc ouverts. Cela signifie que toutes les lignes placées entre celle où il est actionné et celle sur laquelle se trouve son relais de remise à zéro, ou un autre relais de contrôle maitre, sont désactivées. Dans ce cas, les lignes 2 et 3 sont donc désactivées. Lorsque les contacts de l'entrée El se ferment, le relais maitre MC1 est excité. Alors, toutes les lignes qui se trouvent entre sa ligne et celle sur laquelle se trouve MCRl sont activées. Par conséquent, les sorties S1 et S2 ne peuvent pas être activées tant que le relais de contrôle maitre n'est pas excité. Le relais MC1 affecte uniquement la section placée entre la ligne ou il est actionné et la ligne sur laquelle se trouve MCRl.

En utilisant des adresses au format Mitsubishi, le schéma à contacts équivalent ainsi que la liste d'instructions correspondante sont données comme suit :

```
LD      X400
OUT     M100
MC      M100
LD      X401
OUT     Y430
LD      X402
OUT     Y431
MCR     M100
```



VI.5- Sauts et appels

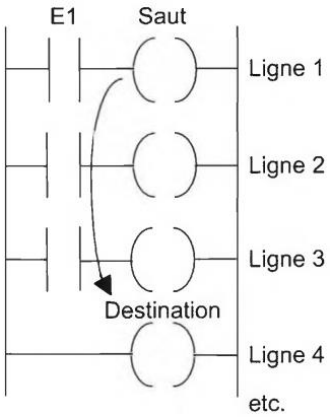
VI.5.1- Sauts

Les automates programmables industriels possèdent souvent une fonction de *saut conditionnel* que nous pouvons décrire de la manière suivante :

```
SI (une centaine condition se produit) ALORS
     exécuter ces instructions
SINON
     exécuter d'autres instructions
```

Grace à cette fonction, les programmes peuvent être conçus de manière telle que lorsque certaines conditions sont remplies, certains évènements se produisent, et lorsqu'elles ne sont pas remplies, d'autres évènements se produisent.

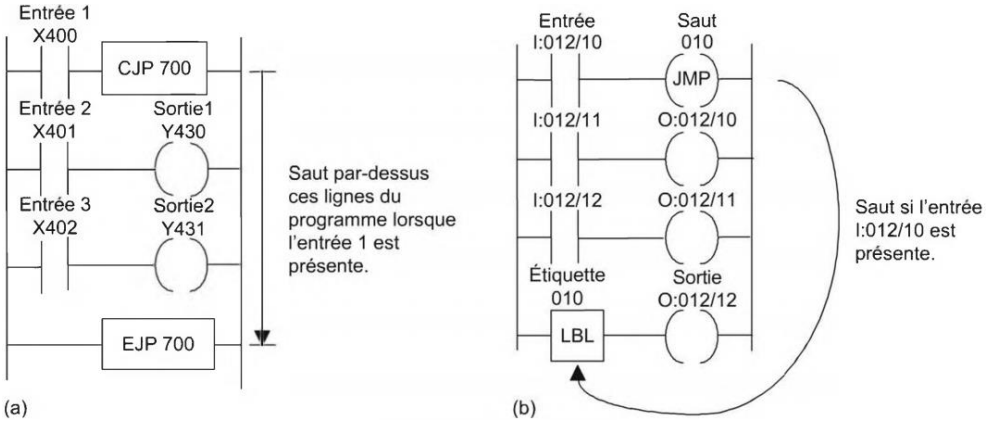
Ainsi, lorsque les conditions appropriées sont remplies, cette fonction permet de sauter par-dessus une partie d'un programme à contacts. La figure suivante illustre ce concept de manière générale.



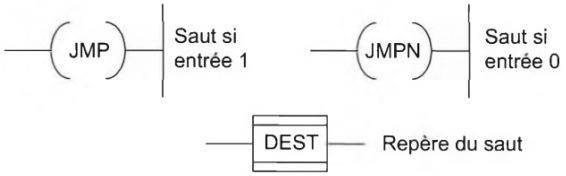
Lorsqu'il existe une entrée sur El, ses contacts se ferment et il existe une sortie sur le relais de saut. Cela conduit ensuite le programme à sauter jusqu'à la ligne dans laquelle se

trouve la destination du saut. Les lignes intermédiaires sont donc ignorées. Dans cet exemple, lorsqu'une entrée est présente sur El, le programme saute à la ligne 4 et poursuit son exécution sur les lignes 5, 6, … etc. Lorsqu'il n'y a pas d'entrée sur El, le relais de saut n'est pas active et le programme continue avec les lignes 2, 3, … etc.

La figure ci-dessous présente le programme à contacts précédent selon les notations de Mitsubishi (a) et d'Allen-Bradley (b). L'instruction de saut est indiquée par un saut conditionnel (CJP pour Mitsubishi et JMP pour Allen-Bradley) et la destination est signalée par une Fin de saut (EJP pour Mitsubishi et LBL pour Allen-Bradley) dont le numéro est identique à celui de l'instruction de début de saut.

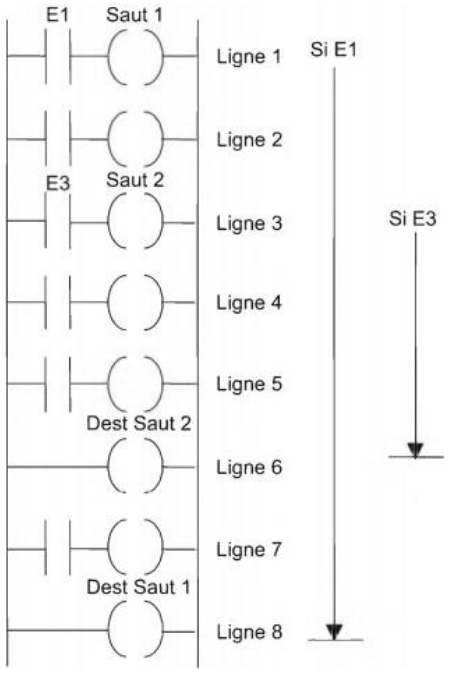


Dans un programme pour un API de Siemens, les sauts sont représentés conformément à la figure ci-dessous. L'instruction de saut JMP est exécutée si l'entrée est à 1, tandis que l'instruction de saut JMPN est exécutée si l'entrée est à 0. La destination de ces deux instructions est le repère DEST.



VI.5.1.1- Sauts imbriqués

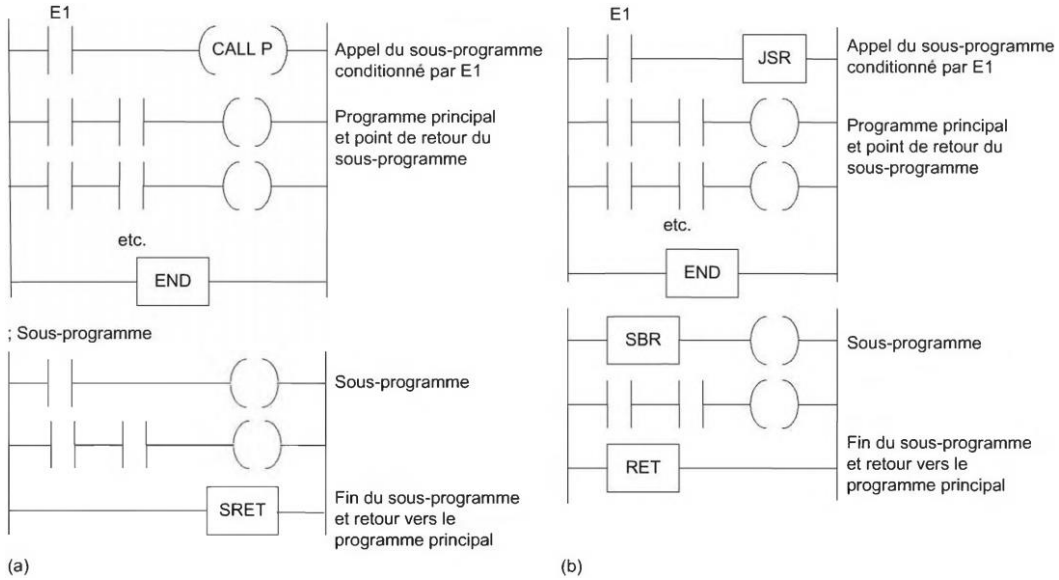
Les sauts imbriqués sont possibles, comme, par exemple, dans le cas suivant :



Si la condition du saut 1 est vérifiée, le programme saute à la ligne 8. Si la condition n'est pas remplie, le programme poursuit à la ligne 3. Si la condition du saut 2 est vérifiée, le programme passe à la ligne 6. Si elle ne l'est pas, le programme poursuit sur les lignes suivantes.

VI.5.2- Sous-programmes

Les *sous-programmes* sont de petits programmes qui réalisent des taches particulières et qui peuvent être appelés depuis des programmes plus importants. L'utilisation des sous programmes présente un avantage : ils peuvent être appelés de manière répétée pour réaliser des tâches spécifiques sans avoir à les récrire intégralement dans le programme principal. Par exemple, pour un programme Mitsubishi (voir figure (a)), nous pourrions avoir la situation suivante :

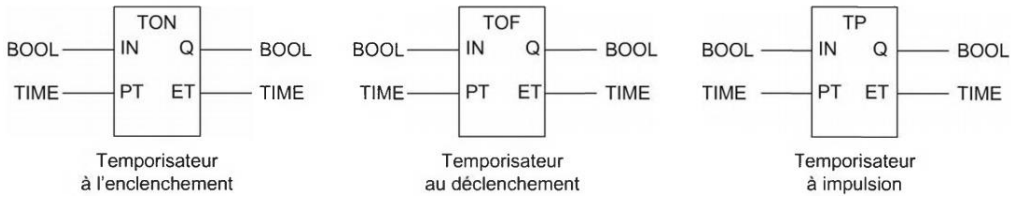


Lorsque l'entrée El se produit, le sous-programme P est appelé avec l'instruction CALL et exécuté. L'instruction SRET indique la fin du sous-programme et déclenche le retour au programme principal. Avec la syntaxe d'Allen-Bradley, les sous-programmes sont appelés à l'aide de l'instruction JSR. Le début du sous-programme est indiqué par SBR, tandis que sa fin et le retour au programme principal sont précisés par RET (voir figure (b)).

VI.6- Temporisateurs

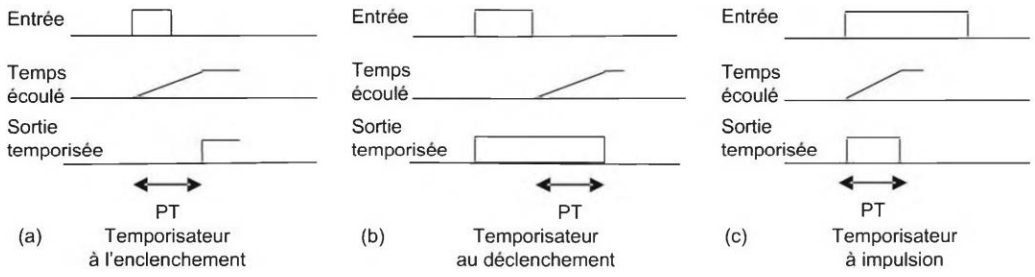
VI.6.1- Types de temporisateurs

Dans les API, vous trouverez plusieurs variantes de temporisateurs : à l'enclenchement (TON), au déclenchement (TOF) et à impulsion (TP) comme illustré ci-dessous :



Dans la norme CEI, IN est l'entrée booléenne, Q est la sortie booléenne, ET est la sortie de temps écoulé, PT est l'entrée qui précise le retard ou la durée de l'impulsion.

- Les *temporisateurs à l'enclenchement* (TON) s'activent après un certain retard (voir figure (a) ci-dessous). Lorsque l'entrée passe de 0 à 1, le temps écoulé ET commence à augmenter et, lorsqu'il atteint la durée fixée par l'entrée PT, la sortie passe à 1.
- Les *temporisateurs au déclenchement* (TOF) sont actifs pendant une durée fixée, avant de se désactiver (voir figure (b)). La temporisation débute lorsque le signal d'entrée passe de 1 à 0.
- Le *temporisateur à impulsion* (TP) représente un autre type de temporisateur, qui produit une sortie à 1 pendant une durée fixée (voir figure (c)). Elle débute lorsque l'entrée passe de 0 à 1, puis la sortie revient à 0 lorsque le temps indiqué par PT est écoulé.

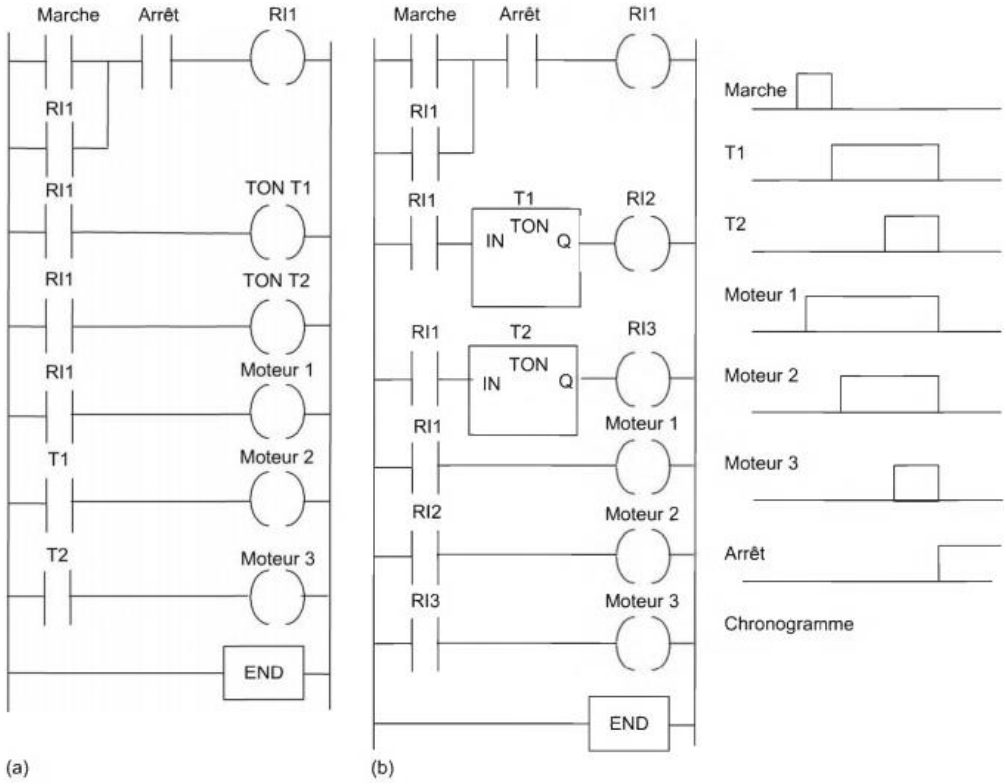


La temporisation définie est appelée *présélection* et correspond à un multiple de la base de temps. Les bases de temps les plus fréquentes sont 10 ms, 100 ms, 1 s, 10 s et 100 s. Ainsi, avec une présélection de 5 et une base de temps de 100 ms, la temporisation est de 500 ms.

VI.6.2- Temporisateurs à l'enclenchement

VI.6.2.1- Séquencement

Afin d'illustrer ce concept en utilisant des temporisateurs TON, examinons le cas de démarrage en séquence trois sorties, par exemple trois moteurs.

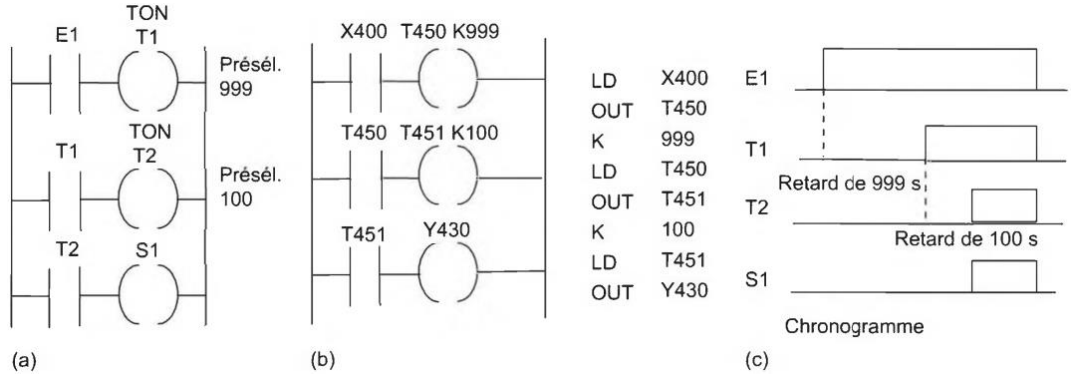


La figure ci-dessus montre deux solutions de ce problème. En (a), les temporisateurs sont programmés comme des bobines, tandis qu'en (b), ils sont programmes comme des retards. Un appui sur le bouton-poussoir marche provoque une sortie sur le relais interne RIl. Cela verrouille l'entrée de démarrage et déclenche les deux temporisateurs, Tl et T2, ainsi que le moteur 1. Lorsque la durée de présélection du temporisateur Tl est écoulée, ses contacts se ferment et le moteur 2 démarre. Lorsque la durée de présélection du temporisateur T2 est écoulée, ses contacts se ferment et le moteur 3 démarre. Les trois moteurs sont stoppés en appuyant sur le bouton-poussoir d'arrêt. Puisque nous présentons un programme complet, l'instruction de fin n'a pas été oubliée.

VI.6.2.2- Temporisateurs en cascade

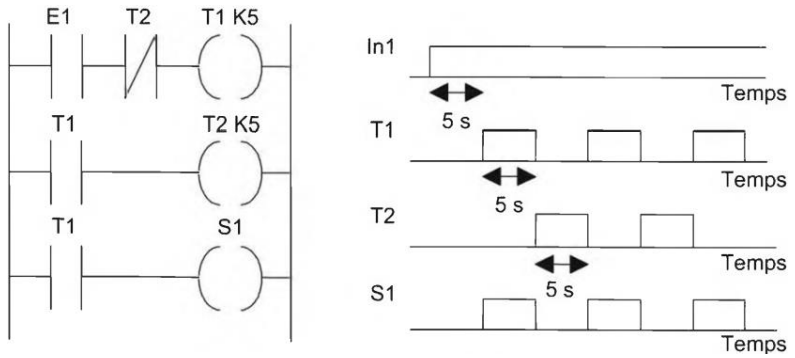
Des temporisateurs peuvent être reliés les uns aux autres (mis en *cascade)* pour obtenir un retard plus long que ne le permet un seul temporisateur. La figure (a) ci-dessous présente un schéma à contacts correspondant. Le temporisateur 1 peut être configuré avec un retard de 999 s. II est démarré par la présence de l'entrée E1. Lorsque 999 s se sont écoulées, les contacts du temporisateur 1 se ferment. Cela déclenche le temporisateur 2, dont la présélection est fixée à 100 s. Lorsque cette durée est écoulée, les contacts du temporisateur 2 se ferment et la sortie S1 est activée. Ainsi, la sortie se produit 1099 s après l'activation de

l'entrée El. La Figure (b) montre la version Mitsubishi de ce schéma à contacts avec des temporisateurs TON, ainsi que les instructions du programme correspondantes.



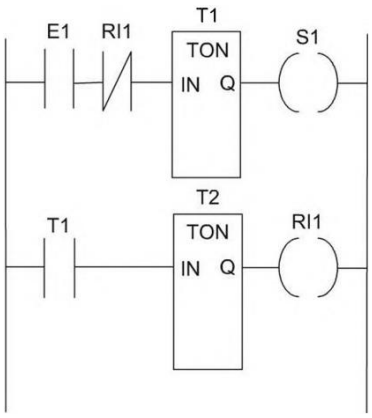
VI.6.2.3- Temporisateur cyclique marche-arrêt

L'utilisation des TON pour obtenir un *temporisateur cyclique marche-arrêt* peut être illustrée par l'exemple suivant :



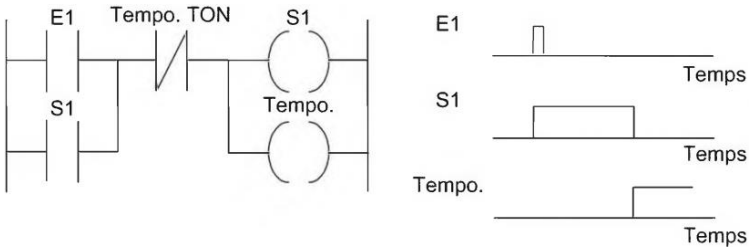
Le temporisateur est conçu pour activer une sortie pendant 5 s, puis la désactiver pendant 5 s, et le cycle se répète à chaque fois. Lorsque l'entrée El est active et que ses contacts se ferment, le temporisateur 1 démarre. II est configuré avec un retard de 5 s. Après 5 s, il active le temporisateur 2 et la sortie S1. Le temporisateur 2 est configuré avec un retard de 5 s. Après 5 s, les contacts du temporisateur 2, qui sont normalement fermés, s'ouvrent. Cela conduit à la désactivation du temporisateur 1 sur la première ligne, à l'ouverture de ses contacts sur la deuxième ligne et à la désactivation du temporisateur 2. Les contacts du temporisateur 2 reviennent donc à leur état normalement fermés et l'entrée sur El provoque la reprise du cycle.

Si on considère que le temporisateur est un retard et non pas une bobine (le cas, par exemple, d'un API de Siemens ou de Toshiba), le schéma à contacts correspondant à l'exemple précédent se donne comme suit :

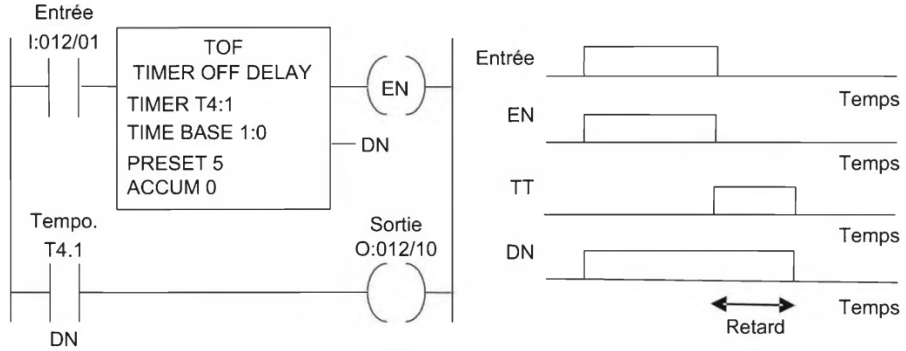


VI.6.3- Temporisateurs au déclenchement

La figure ci-dessous montre comment utiliser un temporisateur à l'enclenchement (TON) pour obtenir un *temporisateur au déclenchement* (TOF). Avec un tel schéma, lorsqu'une entrée momentanée est présente en El, la sortie S1 et le temporisateur sont activés. Puisque l'entrée est verrouillée par les contacts de S1, la sortie reste active. Lorsque la durée de présélection du temporisateur est écoulée, ses contacts normalement fermés s'ouvrent et désactivent la sortie. Par conséquent, la sortie est tout d'abord active et le reste jusqu'à ce que la durée de temporisation soit écoulée.



Dans certains API, les temporisateurs à l'enclenchement et les temporisateurs au déclenchement sont intégrés. Dans ce cas, il est inutile d'utiliser un TON pour obtenir un TOF. La figure suivante illustre cette fonction pour un API d'Allen-Bradley.

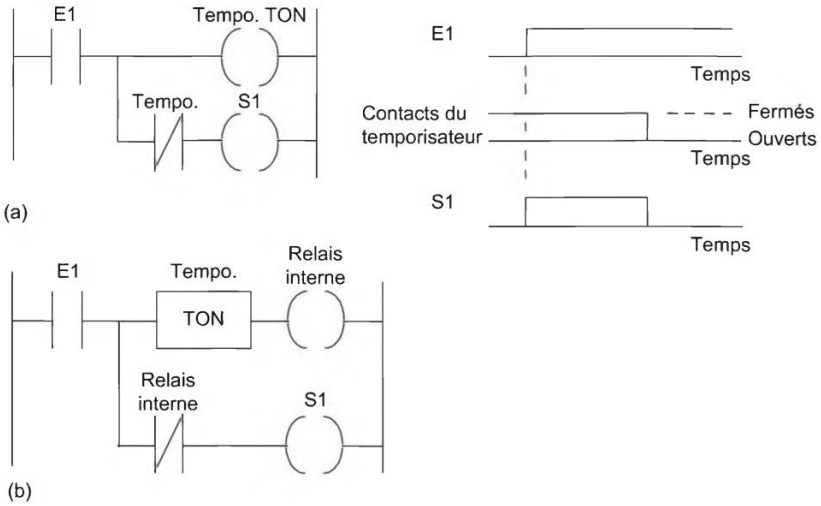


(EN : activation de temporisation, TT : temporisation en cours, DN : fin de la temporisation)

VI.6.4- Temporisateurs à impulsion

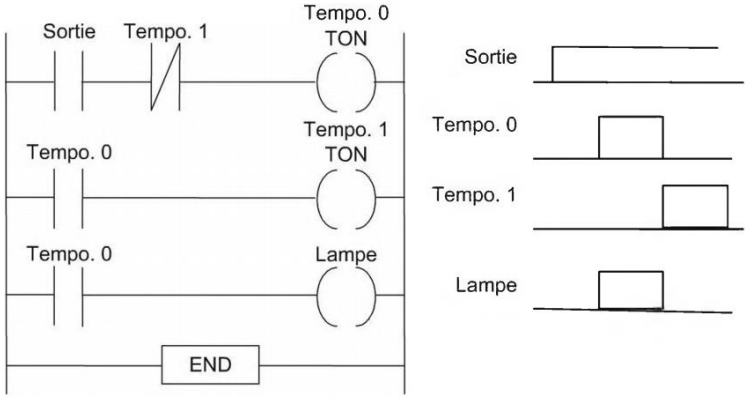
Les *temporisateurs à impulsion* produisent une sortie de durée fixée suite à l'activation d'une entrée. La figure ci-dessous montre le schéma à contacts d'un système qui produit une

sortie sur S1 pendant une durée déterminée lorsque l'entrée El est présente ; le temporisateur est vu comme une bobine. II existe deux sorties pour l'entrée E1. En cas d'entrée sur El, la sortie S1 est activée et le temporisateur démarre. Lorsque la durée présélectionnée est écoulée, les contacts du temporisateur se ferment, ce qui désactive la sortie. Par conséquent, la sortie reste active pendant la durée définie par le temporisateur. Le schéma à contacts équivalent basé sur un temporisateur qui retarde l'arrivée d'un signal sur la sortie, est celui de la figure (b).



VI.6.5- Exemples de programmes

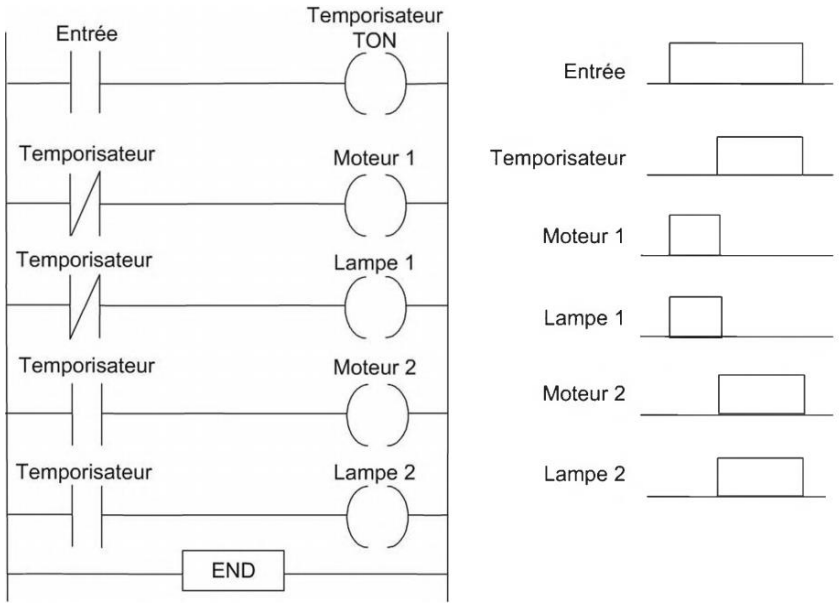
Examinons un programme qui permet de faire clignoter un témoin lumineux pendant qu'une sortie est activée (voir figure ci-dessous).



Les temporisateurs 0 et 1 sont configurés à 1 s. Lorsque la sortie est présente, le temporisateur 0 démarre et s'active après 1 s. Ses contacts se ferment alors, ce qui démarre le temporisateur 1. II devient actif après 1 s, ce qui désactive le temporisateur 0. La lampe est allumée uniquement lorsque le temporisateur 0 est actif. Nous disposons ainsi d'un programme qui fait clignoter le témoin lumineux tant qu'une sortie est présente.

La figure suivante illustre une autre utilisation d'un temporisateur. II s'agit d'un programme qui initialement active le moteur 1 et son témoin lumineux Ll, puis, après une

durée fixée par le temporisateur, arrête le moteur 1 et le témoin 1, et active le moteur 2 et le témoin lumineux 2.



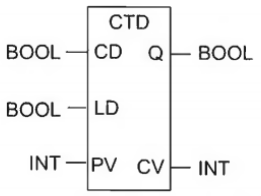
VI.7- Compteurs

VI.7.1- Types des compteurs

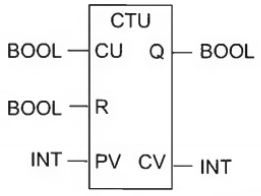
Un compteur est initialisé à une valeur de présélection et, lorsque ce nombre d'impulsions d'entrée a été reçu, il actionne ses contacts. Les contacts normalement ouverts sont fermés, les contacts normalement fermés sont ouverts. II existe deux types de compteurs : les compteurs dégressifs et les compteurs progressifs.

- Un *compteur dégressif* ou compteur décrémental (CTD), décompte à partir de la valeur présélectionnée jusqu'à zéro. Autrement dit, des évènements sont soustraits de la valeur fixée. Lorsque le compteur atteint zéro, ses contacts changent d'état. La majorité des API disposent de compteurs dégressifs.
- Un *compteur progressif* ou compteur incrémental (CTU), compte à partir de zéro jusqu'à la valeur présélectionnée. Autrement dit, des évènements sont ajoutés jusqu'à ce que le cumul atteigne la présélection. Lorsque le compteur atteint la valeur présélectionnée, ses contacts changent d'état.
- Certains API offrent dans une même fonction une possibilité de comptage progressif et dégressif (CTUD).

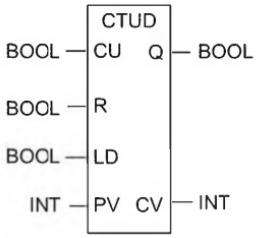
La figure suivante présente les symboles des compteurs tels que définis par la CEI.



(a) Les impulsions sur CD sont comptées. Le compteur débute à la valeur initiale PV. Lorsqu'il arrive à 0, Q est mis à 1 et le comptage s'arrête. Une entrée sur LD remet Q à 0.



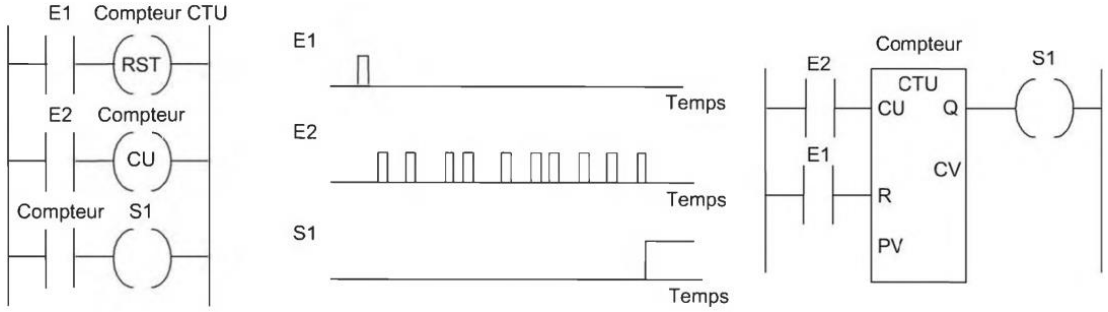
(b) Les impulsions sur CU sont comptées. Le compteur débute à 0. Lorsqu'il arrive à la valeur PV, Q est mis à 1 et le comptage s'arrête. Une entrée sur R remet Q à 0.



(c) Le compteur progressif-dégressif possède les deux entrées CU et CD. Il peut servir à compter jusqu'à une entrée et à décompter jusqu'à l'autre.

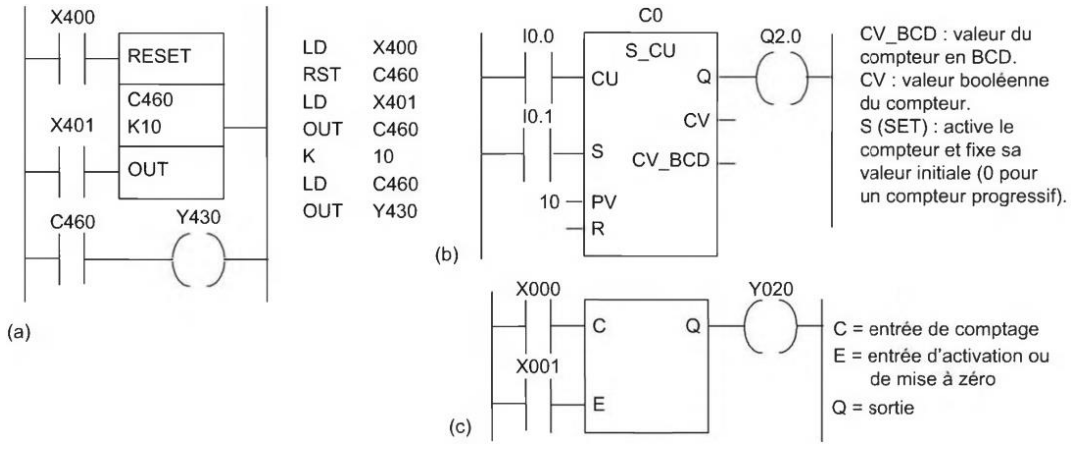
VI.7.2- Programmation

La figure ci-dessous présente un circuit de comptage de base selon la norme CEI.

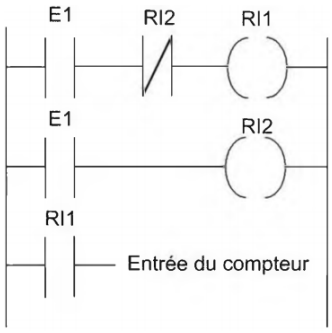


Lors d'un front montant sur l'entrée El, le compteur est mis à zéro. Lors d'un front montant sur l'entrée E2, le compteur se déclenche. S'il est configuré pour, par exemple, dix impulsions, alors, après dix impulsions sur l'entrée E2, c'est-à-dire dix transitions de 0 à 1, les contacts du compteur se ferment et la sortie S1 est activée. Si, au cours du comptage, une entrée est présente sur El, le compteur est réinitialisé et le cycle recommence pour dix impulsions.

Le programme précèdent est illustré dans la figure suivante pour les API de Mitsubishi (a), de Siemens (b) et de Toshiba (c).



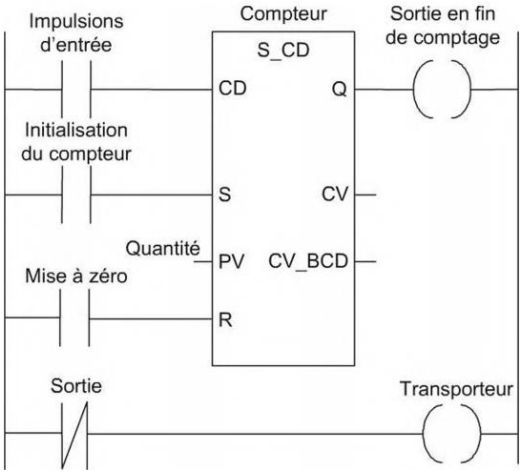
Pour garantir que les impulsions d'entrée d'un compteur soient de courte durée, vous pouvez utiliser le schéma à contacts suivant :



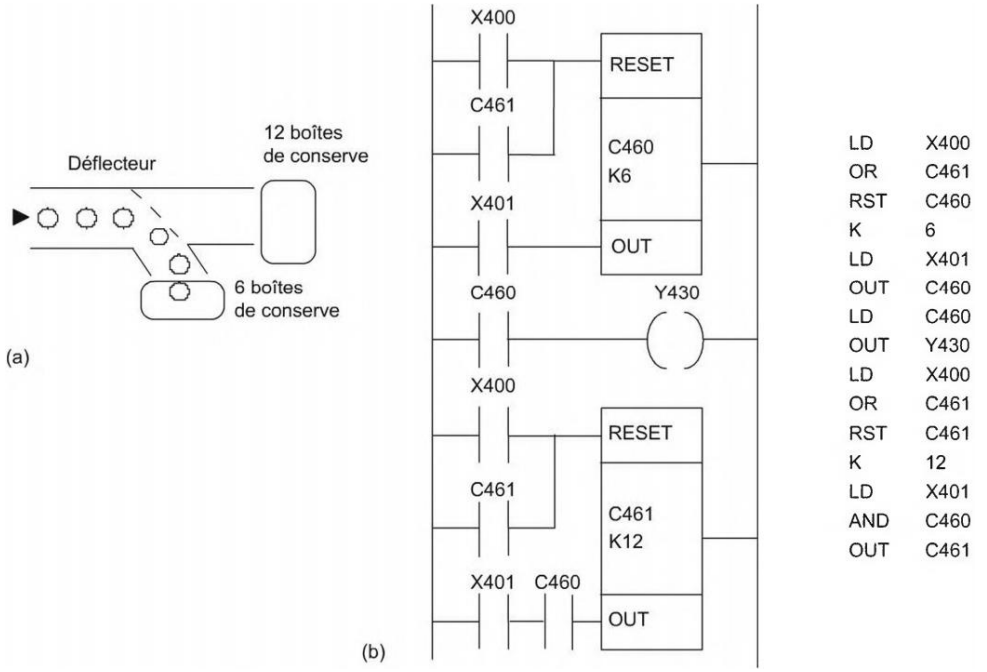
Lorsque l'entrée sur El est présente, le relais interne RIl est activé. Lors de l'analyse de la ligne suivante, c'est-à-dire un court moment après, le relais interne RI2 est activé. Lorsque RI2 est active, il désactive l'entrée sur RIl. Ainsi, RIl produit une impulsion de courte durée, qui sert ensuite d'entrée à un compteur.

VI.7.2.1- Mise en application d'un compteur

Afin d'illustrer l'utilisation d'un compteur, examinons le traitement des articles convoyés par une bande transporteuse. Le passage d'un article en un point particulier est enregistré par l'interruption d'un faisceau lumineux sur une cellule photoélectrique. Après un nombre défini, un signal est envoyé pour informer que ce compte à été atteint et la bande transporteuse est arrêtée. La figure ci-dessous présente les éléments de base d'un programme Siemens qui met en œuvre ce fonctionnement. Un signal de mise à zéro provoque la réinitialisation du compteur et la reprise du comptage. Un signal d'initialisation permet d'activer le compteur.



Comme autre illustration de l'emploi d'un compteur, étudions la commande d'une machine qui doit diriger six boites de conserve vers un chemin pour leur emballage dans un carton et douze boites vers un autre chemin pour leur emballage dans un autre carton (voir figure (a)). Un déflecteur est commandé par un capteur photoélectrique, qui produit une sortie chaque fois qu'une boite passe devant lui. Les impulsions produites par le capteur doivent donc être comptées et utilisées pour commander le déflecteur. La figure (b) présente le schéma à contacts qui peut être utilisé, avec la notation de Mitsubishi.

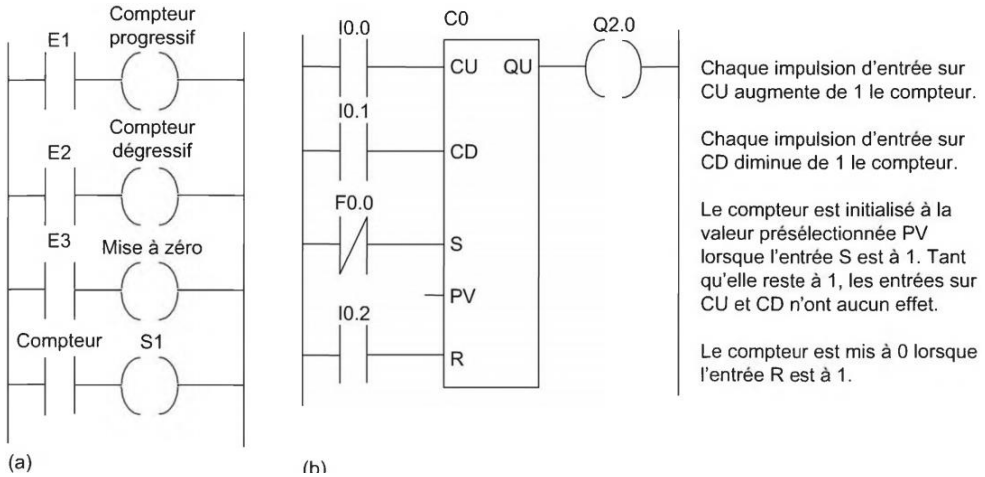


Une impulsion d'entrée sur X400 met à zéro les deux compteurs. L'entrée sur X400 pourrait être un bouton-poussoir qui permet de démarrer la bande transporteuse, L'entrée comptée est X401. Elle peut provenir d'un capteur photoélectrique qui détecte la présence des boites de conserve passant sur la bande transporteuse. C460 commence à compter après la fermeture momentanée de X400. Lorsque C460 a compté six articles, il ferme ses contacts et produit une sortie sur Y430. II peut s'agir d'un solénoïde qui active un déflecteur afin de dévier les articles vers un carton ou un autre. Le déflecteur peut ainsi se trouver dans une position qui permet de dévier les six premières boites de conserve vers le carton de six boites. Ensuite, le déflecteur est déplacé pour que les boites se dirigent vers le carton de douze boites. Lorsque C460 arrête de compter, il ferme ses contacts et permet à C461 de débuter son travail. C461 compte douze impulsions sur X401, puis ferme ses contacts. Cela remet à zéro les deux compteurs et le processus se répète.

VI.7.3- Compteurs progressifs-dégressifs

Etudions par exemple le problème de comptage des articles qui arrivent sur une bande transporteuse et qui en sortent, ou bien le comptage des voitures qui pénètrent dans un parking et qui en sortent. Une sortie doit être activée lorsque le nombre d'articles ou de voitures qui entrent est supérieur au nombre d'articles ou de voitures qui sortent. Autrement dit, le nombre d'articles ou de voitures a atteint une valeur de « saturation ». La sortie peut commander l'allumage d'un panneau « Complet ». Supposons que nous utilisions le compteur progressif pour les articles qui entrent et le compteur dégressif pour ceux qui sortent. La figure ci-

dessous montre la forme de base d'un schéma à contacts avec un API de Siemens, ainsi que la liste d'instructions correspondante pour une telle application.



Lorsqu'un article arrive, il produit une impulsion sur l'entrée El, ce qui incrémente le compteur. Ainsi, chaque article qui arrive augmente de 1 la valeur cumulée. Lorsqu'un article sort, il produit une entrée sur E2, ce qui diminue de 1 le nombre. Ainsi, chaque article qui sort diminue de 1 la valeur cumulée. Lorsque la valeur cumulée atteint la valeur de présélection, la sortie S1 est activée. La valeur de présélection est chargée via F0.0, qui est un relais interne.