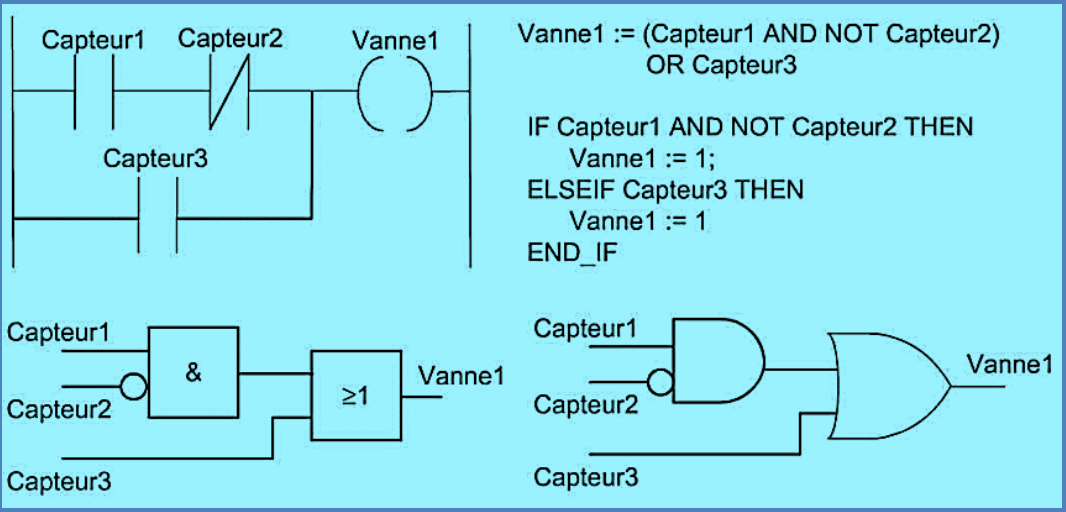
Programmation des API



Objectifs

- Comprendre le principe du langage à contact *Ladder*
- Comprendre le principe du diagramme à bloc fonctionnel *FBD*
- Comprendre la programmation en liste d'instruction *IL*
- Comprendre la programmation en texte structuré *SQL*

❑ ***Prérequis:***

- Auto 1 et 2
- CMSE 1, CMSE 2, CMSE 3

❑ ***public cible:***

- 3ième année Génie industriel et maintenance.

INTRODUCTION

Dans les systèmes à base de microprocesseur, les programmes sont chargés sous forme de *code machine,* c est-à-dire sous forme d'une suite de nombres binaires qui représentent les instructions. Pour faciliter l'écriture des programmes, il est possible d'utiliser un *langage assembleur,* qui se fonde sur des mnémoniques. Par exemple, le mnémonique LD désigne une opération de chargement des données indiquées après LD. Un programme informatique, appelé *assembleur,* convertit ensuite les mnémoniques en code machine. L'utilisation d'un *langage de haut niveau,* comme C, BASIC, Pascal, FORTRAN ou COBOL, facilite encore plus la programmation. Ces langages prédéfinissent des fonctions, qui sont représentées par des mots simples ou des symboles qui les décrivent. Par exemple, dans le langage C, le symbole & est utilisé pour l'opération ET logique.

- Cependant, l'utilisation de ces langages nécessite des connaissances en programmation, alors que les API sont destinés à des ingénieurs dont les compétences dans ce domaine peuvent être limitées. Par conséquent, le ***langage à contacts*** (LD, *Ladder Diagram)* a été développé pour l'écriture des programmes. Ils sont ensuite convertis en code machine par un logiciel et sont utilisés par le microprocesseur d'un API. Cette méthode d'écriture des programmes a été adoptée par la plupart des fabricants d'API, mais chacun a eu tendance à développer ses propres versions. C'est ainsi qu'une norme internationale a été établie pour le langage à contacts et, par voie de conséquence, pour toutes les méthodes de programmation employées avec les API.
- Cette norme, publiée en 1993 par la Commission électrotechnique internationale, est désignée sous la référence CEI 61131-3. Les *diagrammes de schémas fonctionnels* (FBD, *Function Block Diagram*) constituent une autre méthode de programmation.

Langage à contacts (Ladder)

Pour présenter le langage à contacts, prenons le schéma du circuit électrique illustré à la Figure 1a, qui permet de démarrer et d arrêter un moteur électrique. Nous pouvons redessiner ce schéma de manière différente, en utilisant deux lignes verticales pour représenter les sources d'alimentation et en insérant les autres éléments du circuit entre ces deux lignes (voir Figure 1b). Les deux versions placent un interrupteur en série avec un moteur, qui est alimenté lorsque l'interrupteur est fermé. La Figure 1b présente un schéma à contacts.

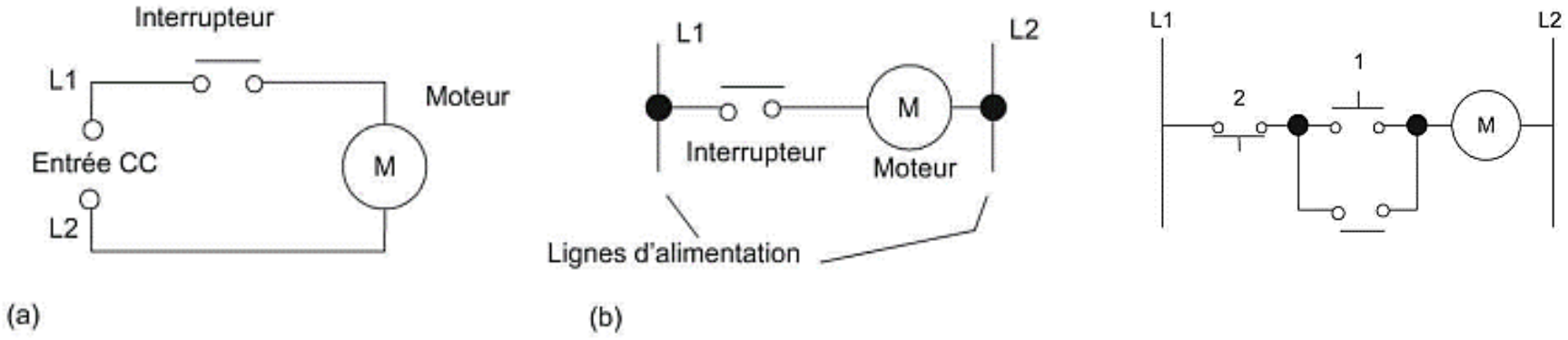


Figure 1 - Deux manières de dessiner le même circuit électrique.

Programmation

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. Pour le tracé d'un schéma à contacts, voici les conventions à respecter :

- **Les lignes verticales du schéma représentent les barres d'alimentation**, entre lesquelles des circuits sont connectés. Le flux du courant part de la ligne verticale de gauche et traverse une ligne horizontale.
- **Chaque ligne horizontale définit une opération du processus de commande**.
- **Chaque ligne doit commencer par une ou plusieurs entrées et doit se terminer par au moins une sortie.** Le terme entrée correspond à une action de commande, comme fermer les contacts d'un interrupteur utilisé en entrée; le terme sortie correspond à un dispositif, comme un moteur, connecté à la sortie d'un API. Les sorties ne sont pas affectées immédiatement au cours de l'analyse du programme. Les résultats des instructions sont placés en mémoire et toutes les sorties sont affectées simultanément à la fin de l'analyse du programme

- **Les dispositifs électriques sont représentés dans leur condition normale**. Ainsi, un interrupteur qui est normalement ouvert jusqu'à ce qu'un objet le ferme est représenté ouvert sur le schéma à contacts. Un interrupteur normalement fermé est représenté fermé.
- **Un même dispositif peut apparaître sur plusieurs lignes du schéma**. Par exemple, un relais peut commuter un ou plusieurs systèmes. Les mêmes lettres et/ou numéros sont utilisés comme libellés du dispositif dans chaque cas.
- **Les entrées et les sorties sont toutes identifiées par leur adresse**, dont le format dépend du fabricant de l'API. Il s agit de l'adresse de l'entrée ou de la sortie dans la mémoire de l'API

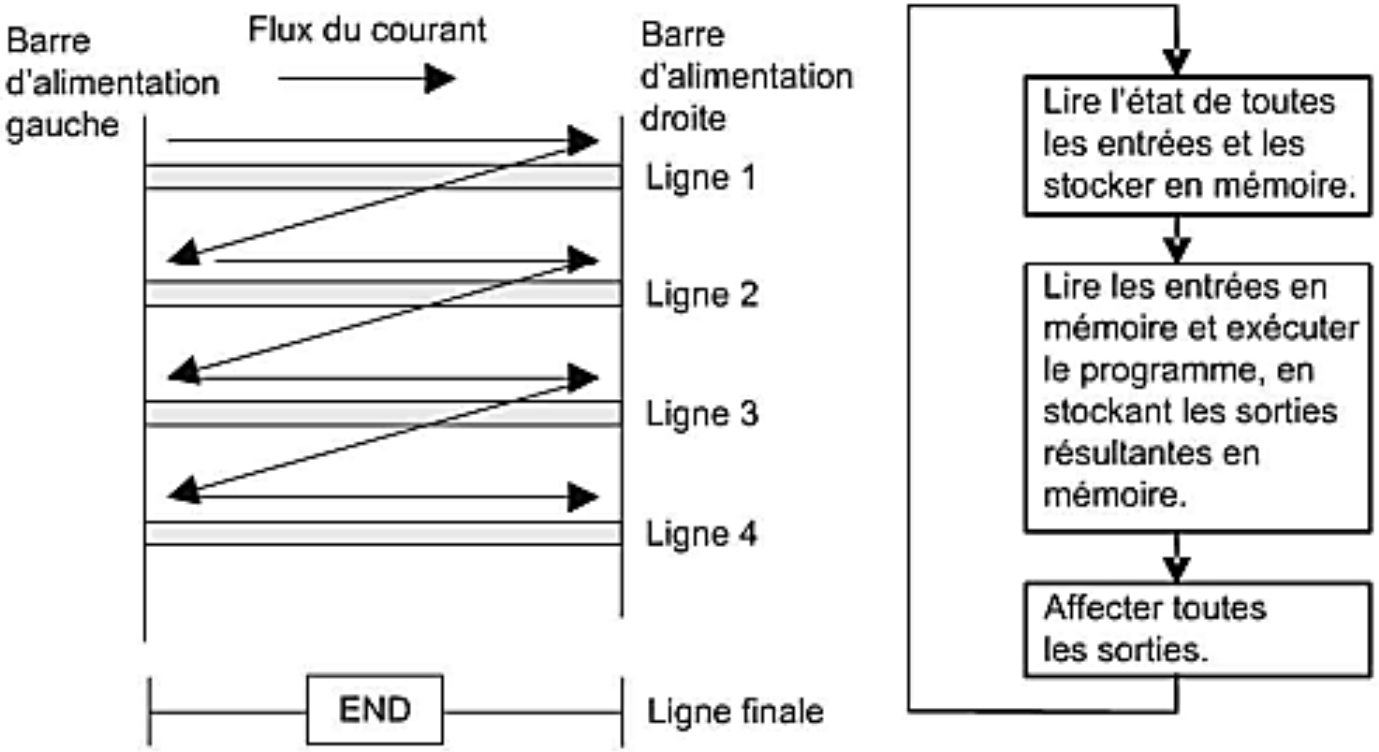


Figure 2 - Analyse du schéma à contacts.

La Figure 3 présente les symboles standard définis par la CEI 61131-3 et utilisés pour les dispositifs d'entrées et de sorties.

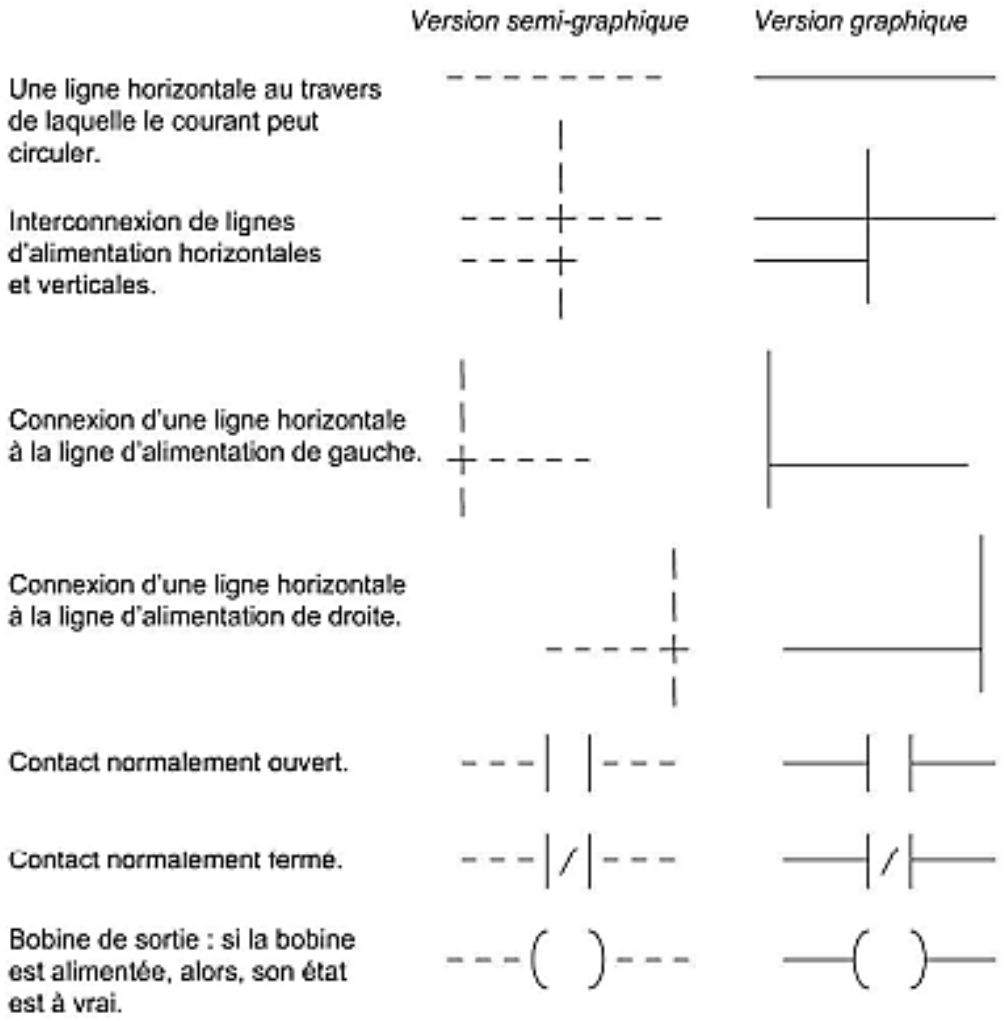


Figure 3 - Symboles de base.

Dans un schéma à contacts, le nom de la variable associée à chaque élément et son adresse sont ajoutés au symbole. Il est préférable de choisir un nom descriptif, par exemple *interrupteur de commande du moteur de la pompe*, à la place d un simple *entrée*, et *moteur de la pompe.,* à la place d'un simple *sortie.* La Figure 4 présente le schéma à contacts de la Figure 4a tel qu'il serait dessiné avec les adresses au format **(a) Mitsubishi**, **(b) Siemens**, **(c) Allen-Bradley** et **(d) Schneider**. La Figure 4a montre ainsi que cette ligne du schéma à contacts a une entrée à l adresse X400 et une sortie à l'adresse Y430. Lors de la connexion des entrées et des sorties, les dispositifs correspondants doivent être connectés aux ports possédant ces adresses.

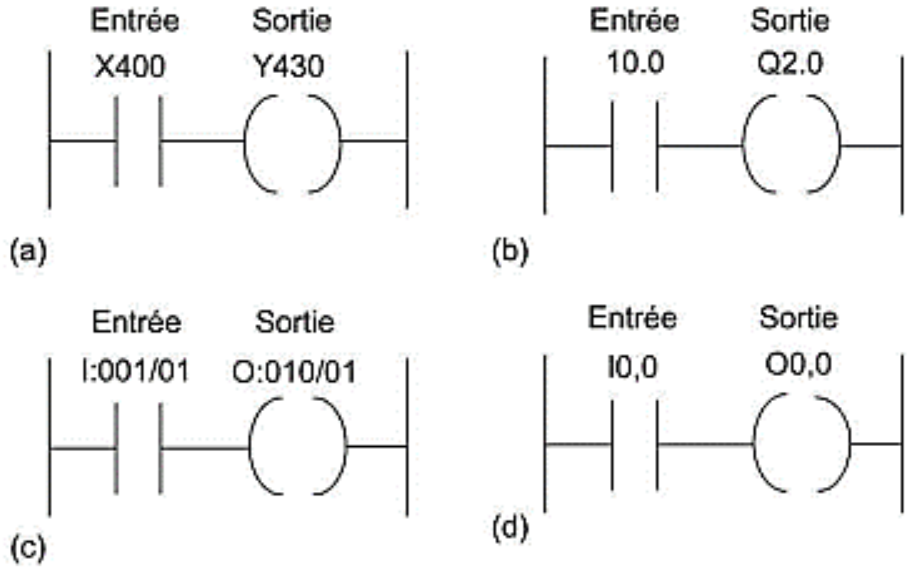


Figure 4 - Ajout des adresses : au format (a) Mitsubishi, (b) Siemens, (c) Allen Bradley et (d) Schneider.

FONCTIONS LOGIQUES

Dans de nombreux cas, les actions doivent être déclenchées lorsqu'une certaine combinaison de conditions est vérifiée.

• Fonction ET (AND)

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.

La Figure 5a montre un système à porte ET sur un schéma à contacts. Le schéma commence par **| |**, un ensemble de contacts normalement ouverts intitulé « **Entrée A »**, pour représenter l'interrupteur A. Il est placé en série avec un autre symbole **| |**, également un ensemble de contacts normalement ouverts, intitulé « « **Entrée B »**, pour représenter l'interrupteur B. La ligne se termine par **( )** pour représenter la sortie. Pour que la sortie soit active, les entrées A et B doivent se produire.

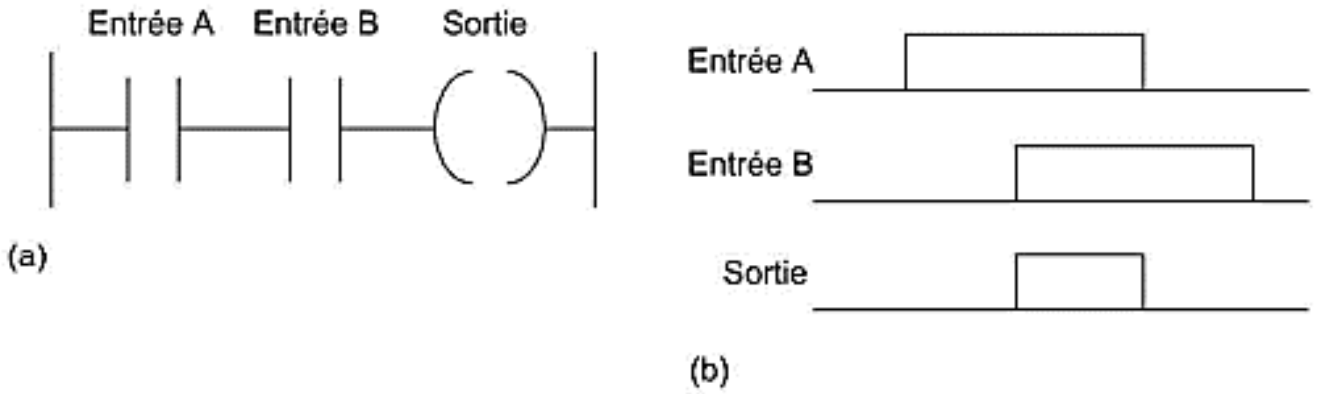


Figure 5 - Porte ET sur une ligne d'un schéma à contacts.

- **Fonction 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 activé 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.

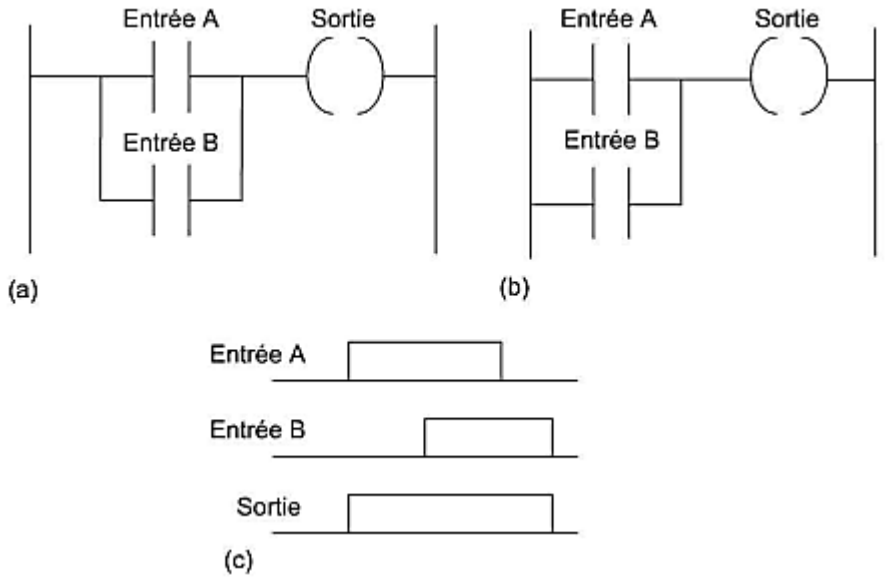


Figure 6 - Porte OU sur une ligne d'un schéma à contacts.

- **Fonction NON (NOT)**

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.

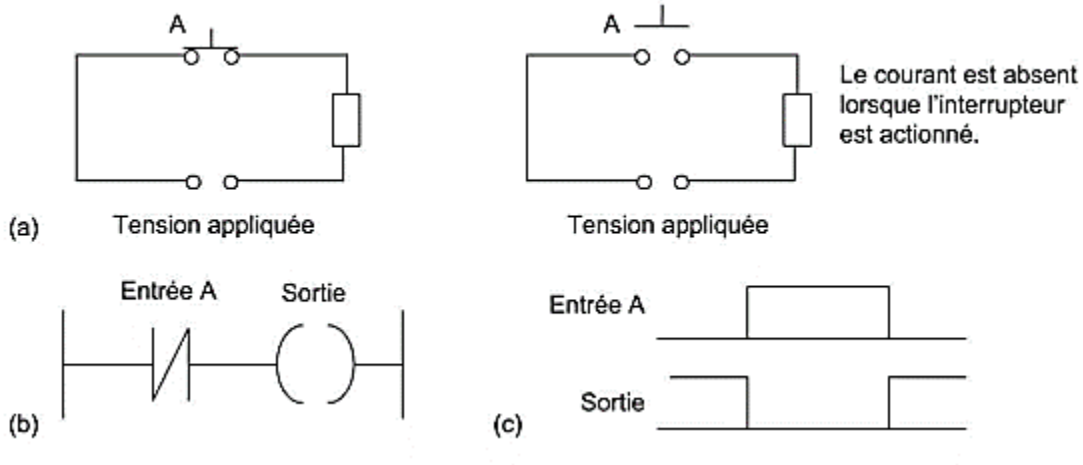


Figure 7 - (a) Un circuit NON, (b) une porte logique NON dans un schéma à contacts et (c) une sortie au niveau haut en cas d'absence d'entrée sur A.

- **Fonction NON-ET (NAND)**

La Figure 8 illustre un schéma à contacts qui donne une porte NON-ET.
Lorsque Entrée A et Entrée B sont toutes deux à 1, la sortie est à 0. Lorsque Entrée A et Entrée B sont à 0 ou que l'une est à 0 et l'autre à 1, la sortie est à 1.

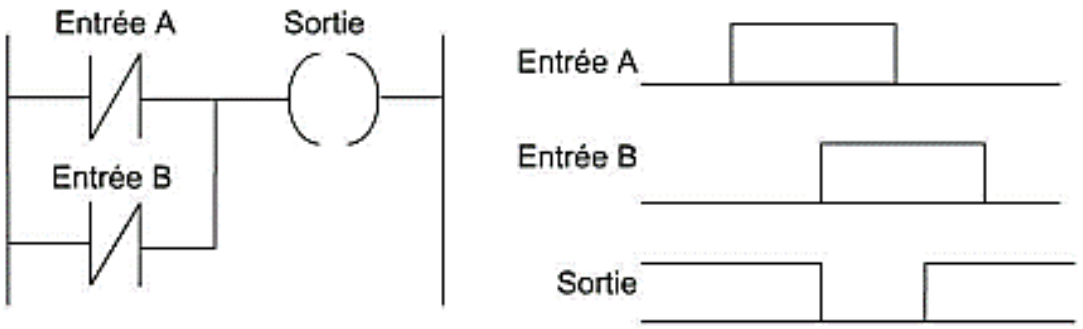


Figure 8 - Porte NON-ET sur une ligne d'un schéma à contacts.

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 activés.

- **Fonction NON-OU (NOR)**

La Figure 9 illustre un schéma à contacts qui donne une porte NON-OU.

Lorsque ni Entrée A ni Entrée B n'est activée, la sortie est à 1. Lorsque Entrée A ou Entrée B est à 1, la sortie est à 0.

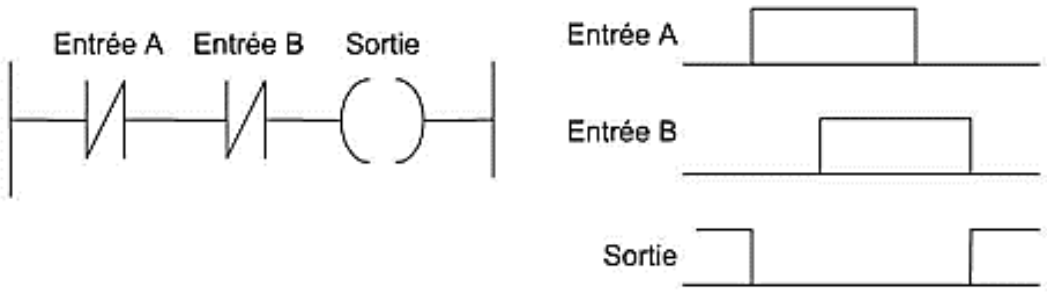


Figure 9 - Porte NON-OU sur une ligne d'un schéma à contacts.

- **OU exclusif (XOR)**

La Figure 10 montre un schéma à contacts qui représente une porte OU exclusif. Lorsque Entrée A et Entrée B ne sont pas activées, la sortie est absente. Lorsque seule Entrée A est activée, la branche supérieure produit une sortie à 1. Lorsque seule Entrée B est activée, la branche inférieure produit une sortie à 1. Lorsque Entrée A et Entrée B sont activées, il n'y a pas de sortie. Dans cet exemple de portes logiques, Entrée A et Entrée B possèdent deux ensembles de contacts dans les circuits, l'un étant normalement ouvert, l'autre, normalement fermé. Pour la programmation d'un API, chaque entrée est considérée comme disposant d'autant d'ensembles de contacts que nécessaire.

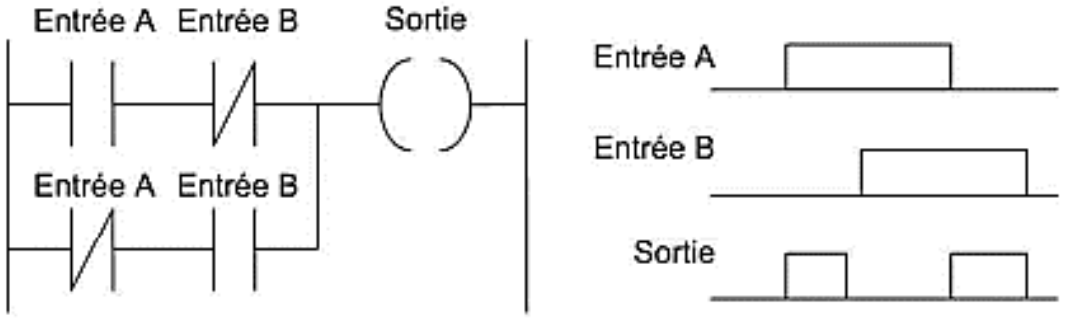


Figure 10 - Porte OU exclusif sur une ligne d'un schéma à contacts

Verrouillage

- Dans de nombreuses situations il est nécessaire de maintenir la sortie alimentée même lorsque l'entrée disparaît. 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. Il s'agit d un circuit de maintien car, après avoir été excité, il conserve cet état jusqu'à la réception d'une autre entrée.

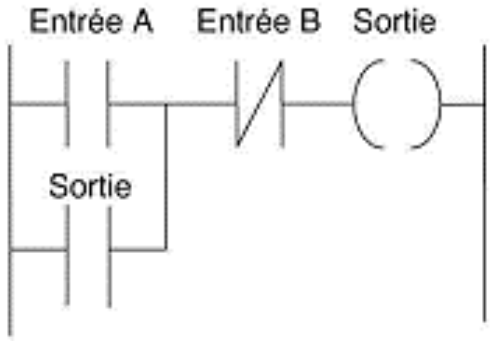


Figure 11 - Circuit à verrouillage.

Sorties multiples

- Dans un schéma à contacts, plusieurs sorties peuvent être connectées à un contact.
  La Figure 12 présente un tel schéma avec deux bobines de sortie. Lorsque le contact d'entrée se ferme, les deux bobines produisent des sorties.

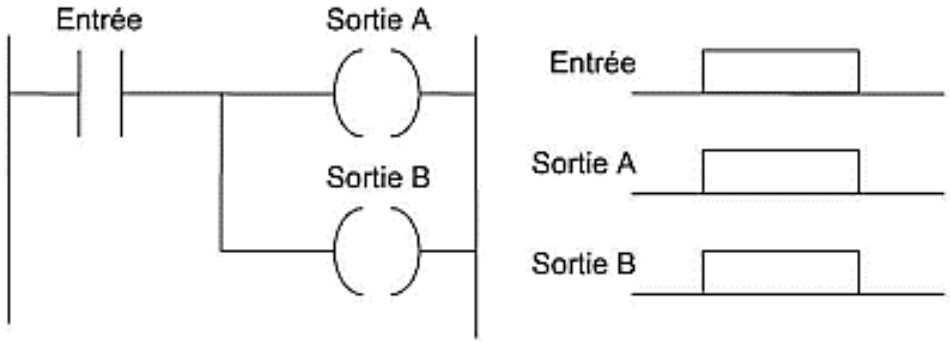


Figure 12- Ligne d'un schéma à contacts avec deux sorties.

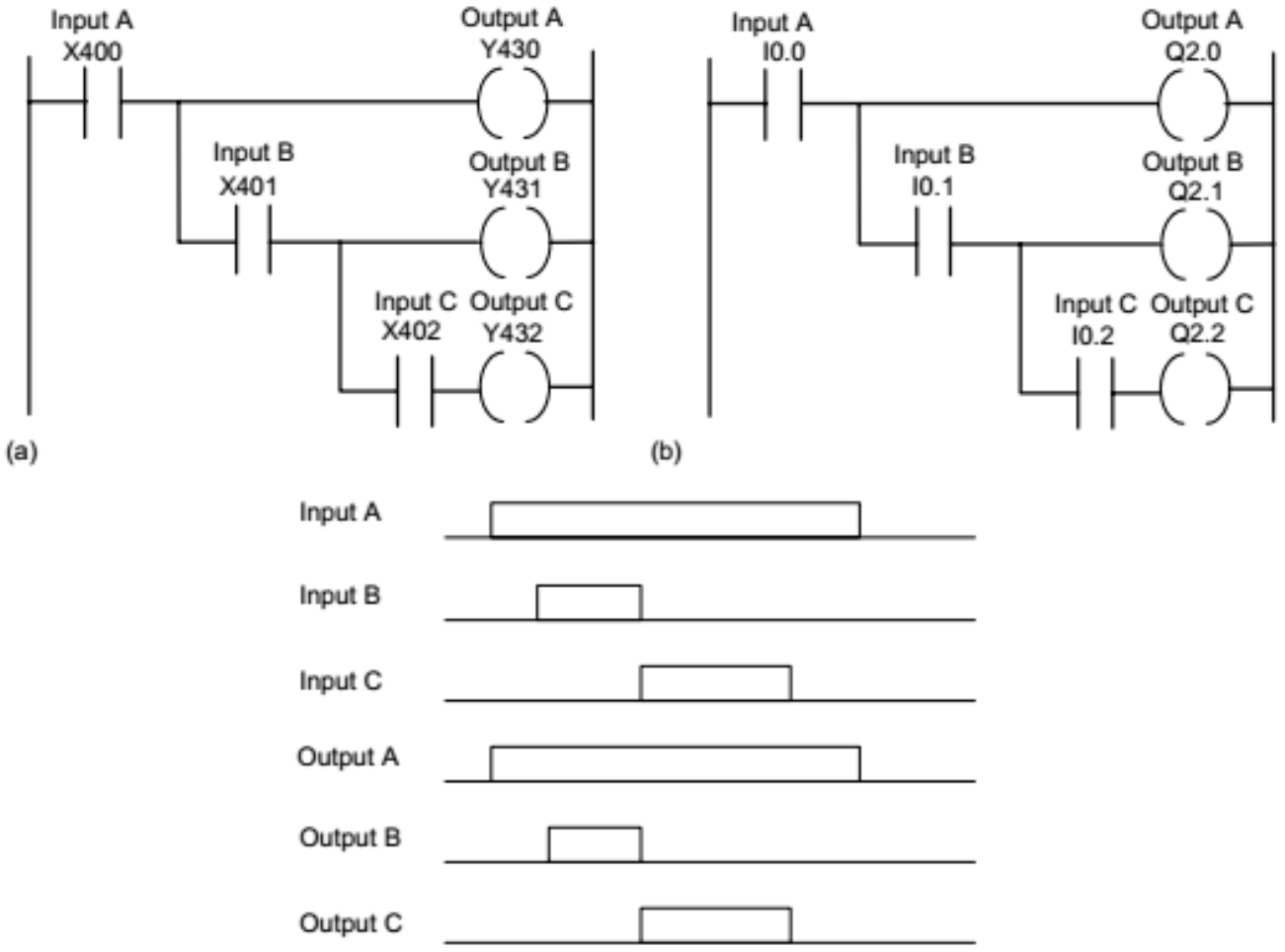


Figure 13- sorties ordonnées

Saisie des programmes

- Chaque ligne horizontale du schéma à contacts représente une instruction dans le programme exécuté par l'API. L'ensemble du schéma correspond au programme complet. Il existe plusieurs méthodes pour saisir le programme depuis un pupitre de programmation ou un ordinateur. Quelle que soit celle employée, le programme doit être placé dans la mémoire de l'API sous une forme compatible avec le microprocesseur. Cette forme est appelée *langage machine* et n'est rien d'autre qu'un code binaire, comme 0010100001110001.
- Les ordinateurs ou les pupitres de commande peuvent être utilisés pour tracer les schémas à contacts. Pour cela, il faut commencer par installer le logiciel approprié, par exemple: **RSLogix** de *Rockwell Automation Inc* pour les **API d'*Allen-Bradley***, **MELSOFT GX Developer** pour les API de **Mitsubishi** et **STEP 7/Tia Portal** pour les API de **Siemens**.

Diagrammes de schémas fonctionnels (FBD)

- Un *diagramme de schémas fonctionnels* (FBD, *Function Block Diagram)* est utilisé pour les programmes d'API décrits sous forme de blocs graphiques. Il 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 14 montre la représentation d'un bloc. Le nom de la fonction est écrit dans la boîte.

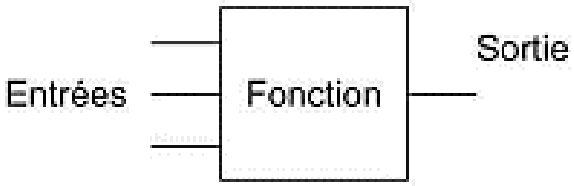


Figure 14 - Bloc fonctionnel

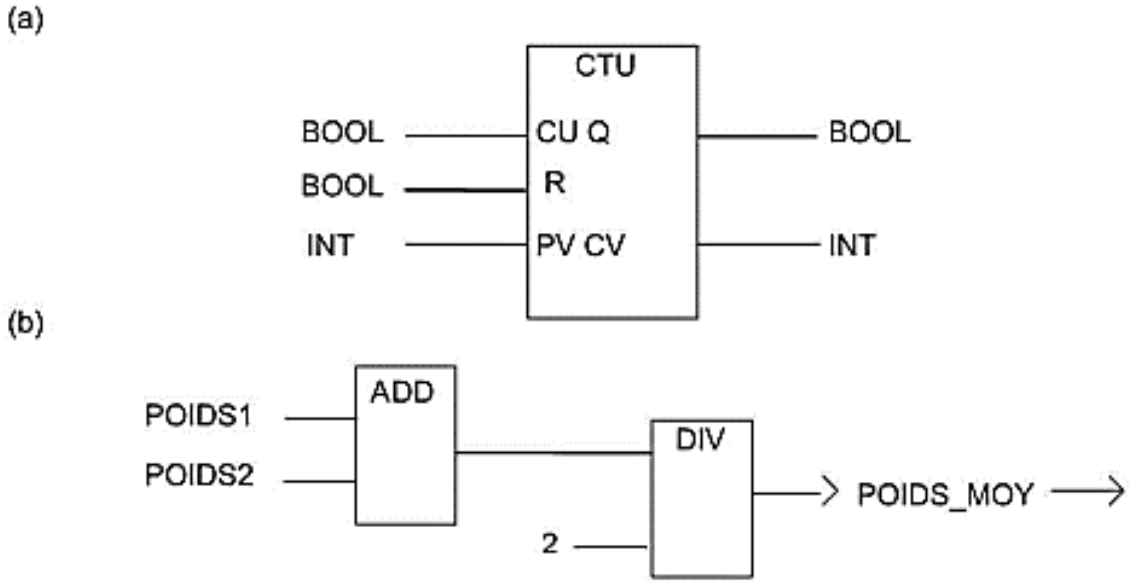


Figure 15 - Représentation d'un diagramme de schémas fonctionnels, (a) Un exemple de bloc fonctionnel standard : un compteur CTU qui produit une sortie sur Q lorsque le nombre d'impulsions sur l'entrée CU a atteint la valeur fixée par PV. À chaque impulsion d'entrée, la sortie CV est augmentée de 1. L'entrée R remet à zéro le compteur. Les libellés des entrées et des sorties indiquent le type du signal : BOOL pour booléen et INT pour entier. Consultez le Chapitre 10 pour l'utilisation d'un tel bloc, (b) Un exemple de bloc défini par l'utilisateur pour calculer la moyenne de deux poids. Il peut être réutilisé ensuite dans d'autres blocs fonctionnels.

Portes logiques

Les programmes impliquent souvent des portes logiques. Il existe deux formes de symbolisation des portes logiques, l'une provenant des États-Unis, l'autre étant un standard international (IEEE/ANSI) qui se fonde sur un rectangle dans lequel est écrit la fonction. Le 1 dans une boîte indique qu'il existe une sortie si l'entrée est à 1. La fonction OU est indiquée par >1 car la sortie est présente si une entrée est supérieure ou égale à 1. Une entrée inversée est représentée par un petit cercle sur l'entrée, tout comme une sortie inversée (voir Figure 16). La Figure 16 montre les symboles. Dans les diagrammes de schémas fonctionnels, les symboles IEEE/ANSI sont les plus souvent utilisés.

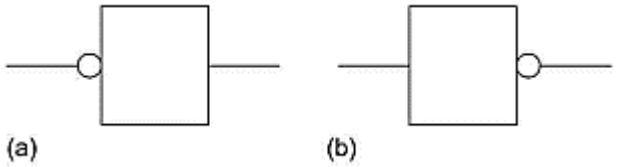


Figure 16 - (a) Entrée inversée et (b) sortie inversée.

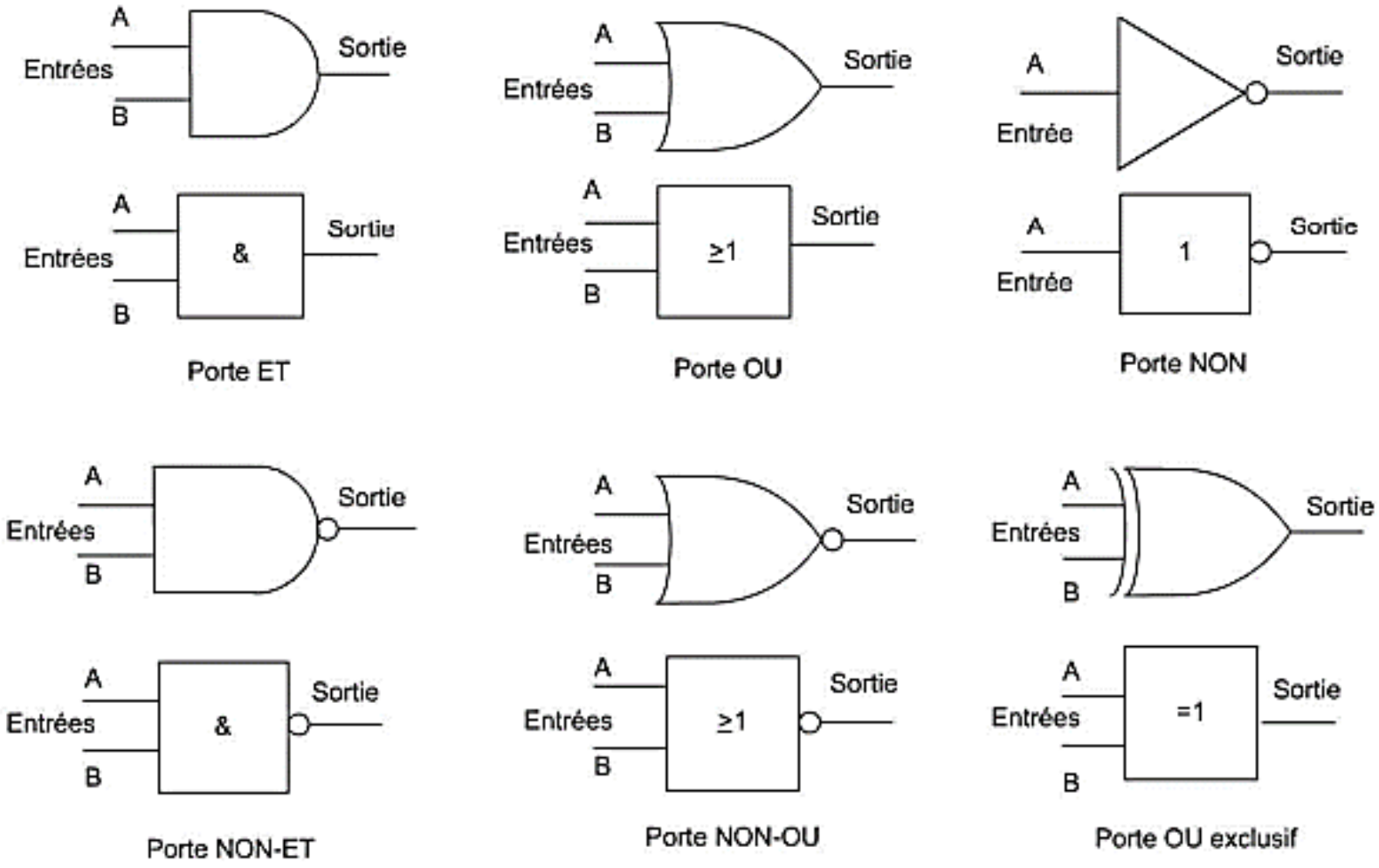


Figure 17- Symboles des portes logiques

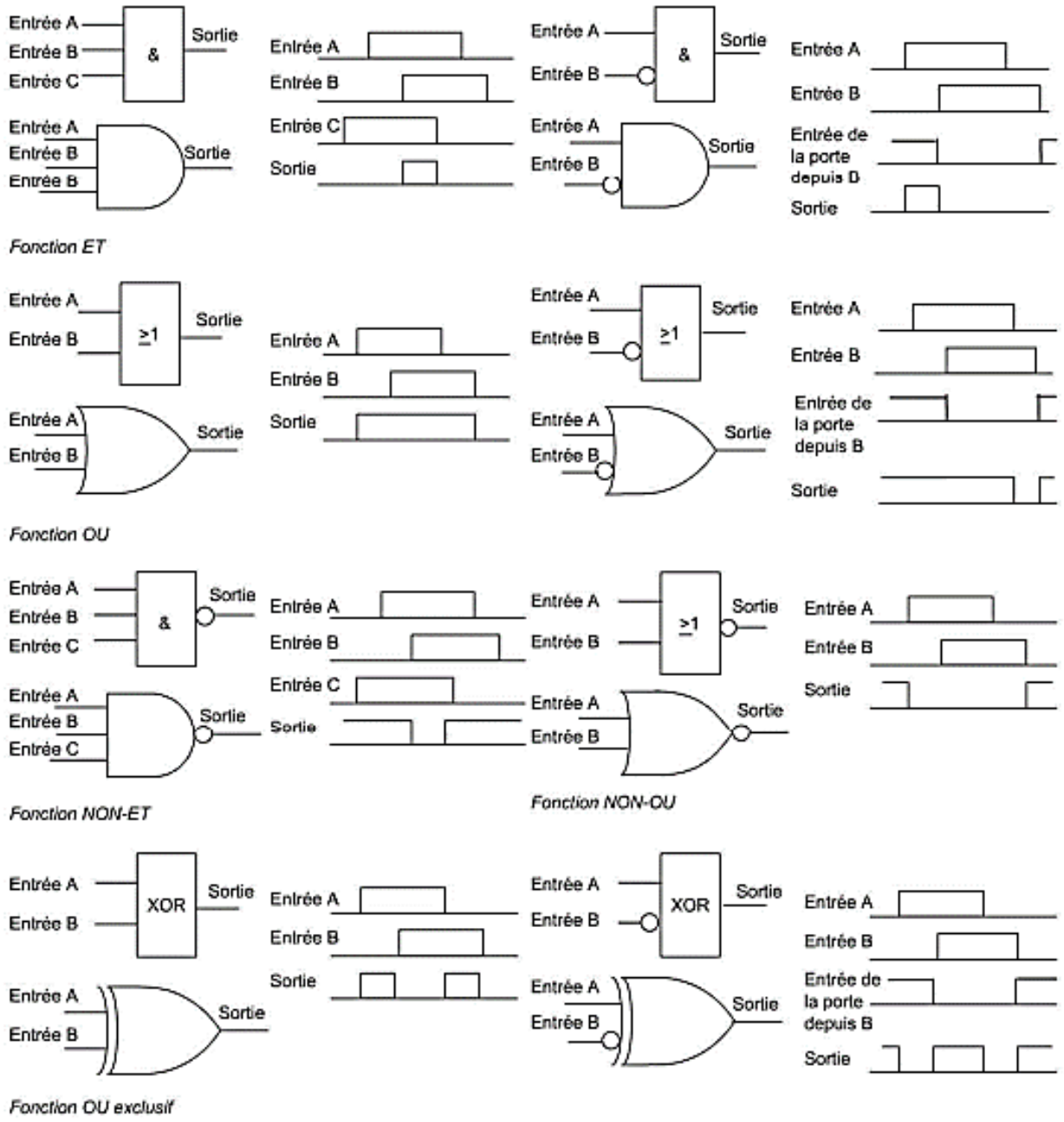


Figure 18 - Blocs fonctionnels des portes logiques.

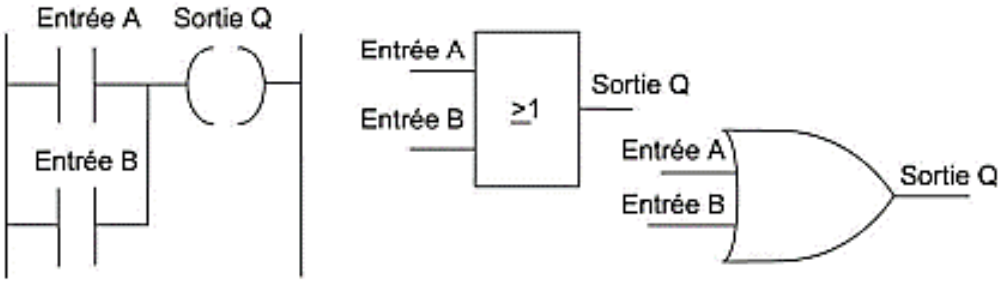


Figure 19 - Schéma à contacts et diagramme de schémas fonctionnels équivalent.

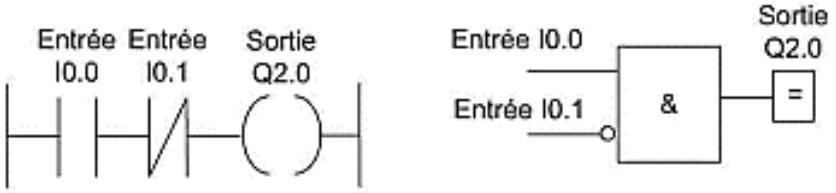


Figure 20 - Schéma à contacts et diagramme de schémas fonctionnels équivalent

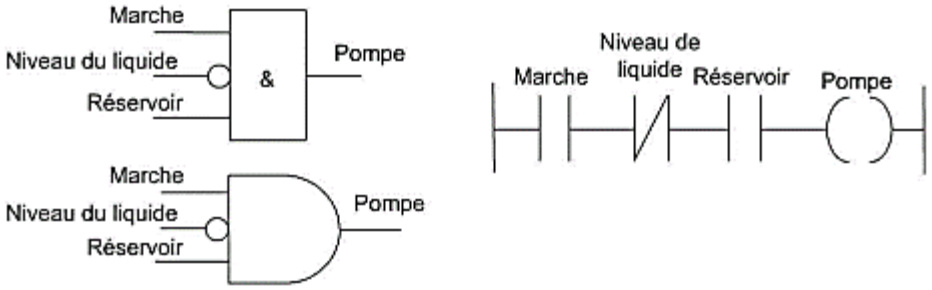


Figure 21- Mise en œuvre de la pompe.

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 de mise en 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 donnés à la Figure 22.

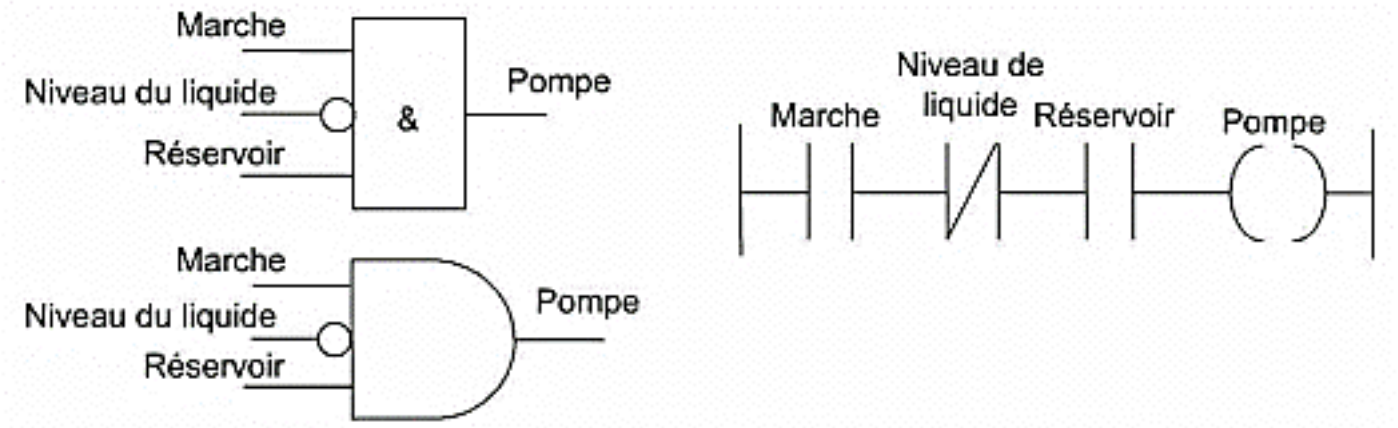


Figure 22- Mise en œuvre de la pompe.

Étudions un circuit logique avec de nombreuses entrées et sa représentation par une expression booléenne et une ligne d'un schéma à contacts (voir Figure 5.35). 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 suivante :

$$(A.B + C).\overline{D}.E.\overline{F} = Q$$

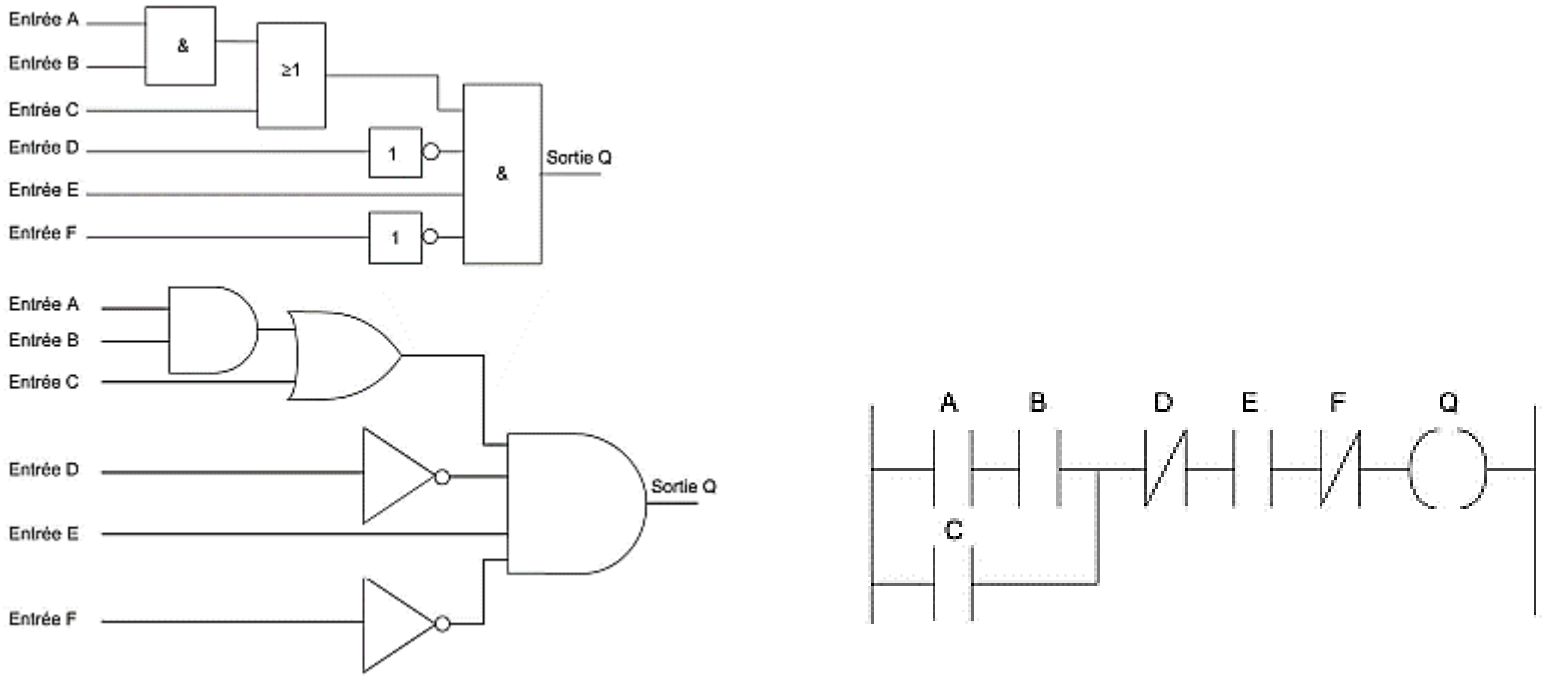


Figure 23 - Un circuit logique et son Schéma à contacts équivalent

Exemples de programmes

- **Exemple 1**

Un témoin lumineux doit être allumé lorsqu'une pompe est en marche et que la pression est satisfaisante, ou bien lorsque son interrupteur de test est fermé. Pour les entrées issues de la pompe et du capteur de pression, nous avons un cas de ET logique, puisque ces deux entrées doivent être actives pour avoir une sortie sur le témoin lumineux. Cependant, nous avons également un OU logique avec l'interrupteur de test, car il doit produire une sortie d'activation du témoin lumineux, que la combinaison ET produise ou non un signal. La Figure 24 révèle le diagramme de schémas fonctionnels et le schéma à contacts correspondants. Dans le schéma à contacts, vous remarquerez que nous indiquons à l'API la fin du programme à l'aide d'une instruction END ou RET.

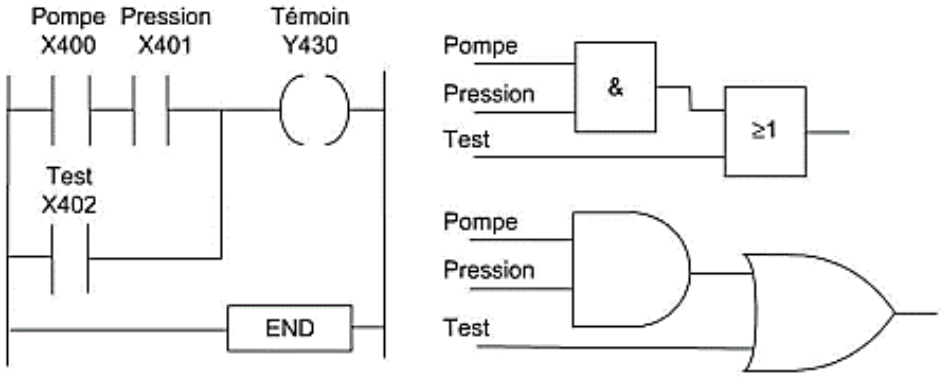


Figure 24 - Tâche pour le témoin lumineux.

- **Exemple 2**

Prenons un autre exemple, celui **d'une vanne qui doit être actionnée pour soulever une charge lorsqu'une pompe est en fonctionnement et lorsque l'interrupteur de levage ou lorsqu'un interrupteur indiquant que la charge n'est pas encore soulevée et quelle se trouve en bas de son guide de levage est actionné**. Nous avons un OU entre les deux interrupteurs et un ET entre les deux interrupteurs et la pompe. La Figure 25 propose un programme qui correspond à cette tâche.

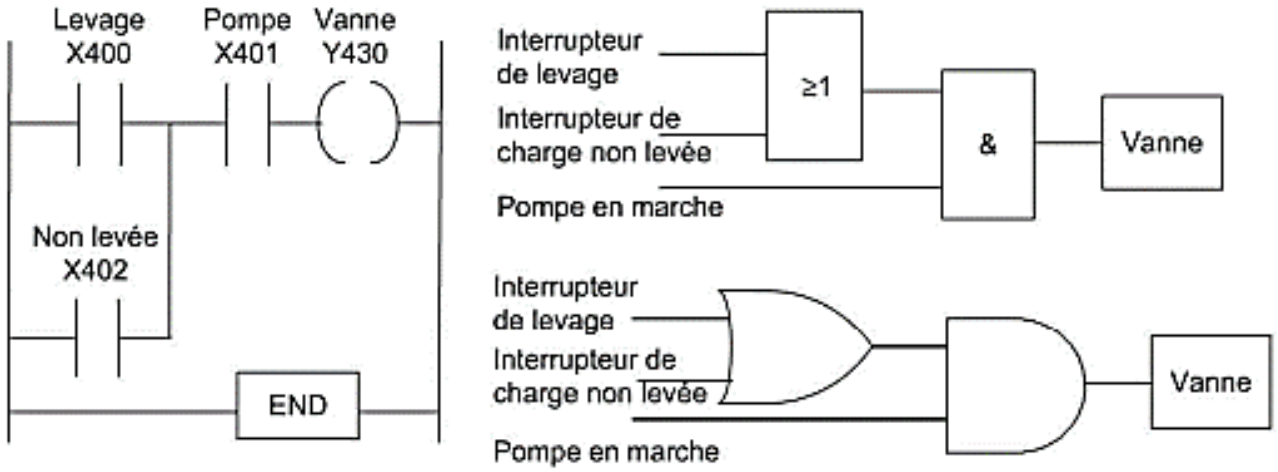


Figure 25 - Programme de commande d'une vanne.

- **Exemple 3**

Poursuivons nos exemples en examinant un système où **la sortie doit être absente lorsque aucun des quatre capteurs ne produit une sortie**. **Dans le cas contraire, la sortie doit être présente.** Pour écrire le programme qui correspond à ce système, nous pouvons utiliser des capteurs dont les contacts sont normalement fermés afin qu' une sortie soit présente. Lorsqu'une entrée est présente sur le capteur, les contacts s'ouvrent et la sortie disparaît. Nous avons ainsi un ET logique. La Figure 26 présente le diagramme de schémas fonctionnels et le schéma à contacts qui correspondent à ce système.

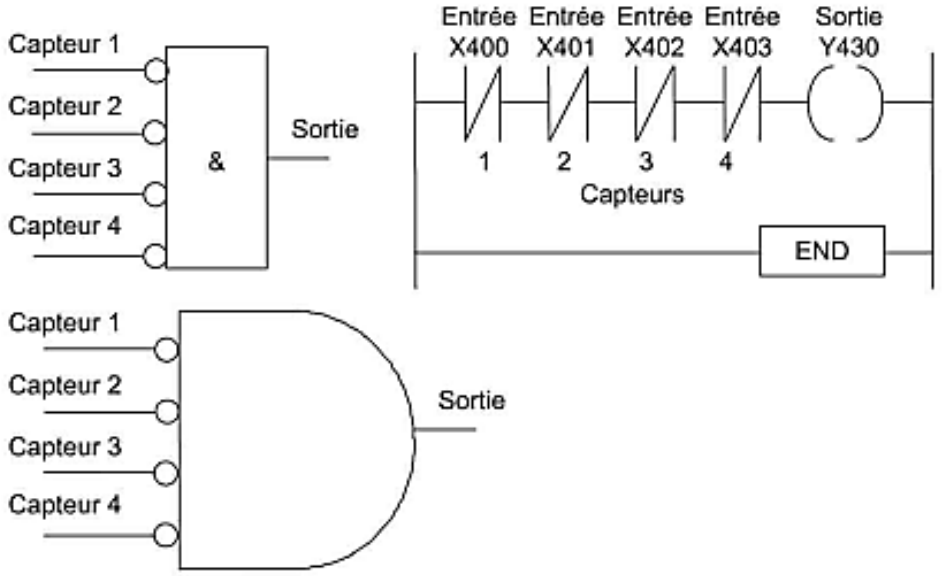


Figure 26 - Sortie arrêtée par activation d'un seul des quatre capteurs.

- **Exemple 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 Figure 27a). Dans le schéma à contacts, il est préférable que l'interrupteur d'arrêt soit programmé ouvert et d'utiliser un interrupteur d'arrêt normalement fermé et de l'actionner pour l'ouvrir. Un signal d'entrée ferme alors les contacts au démarrage du programme .

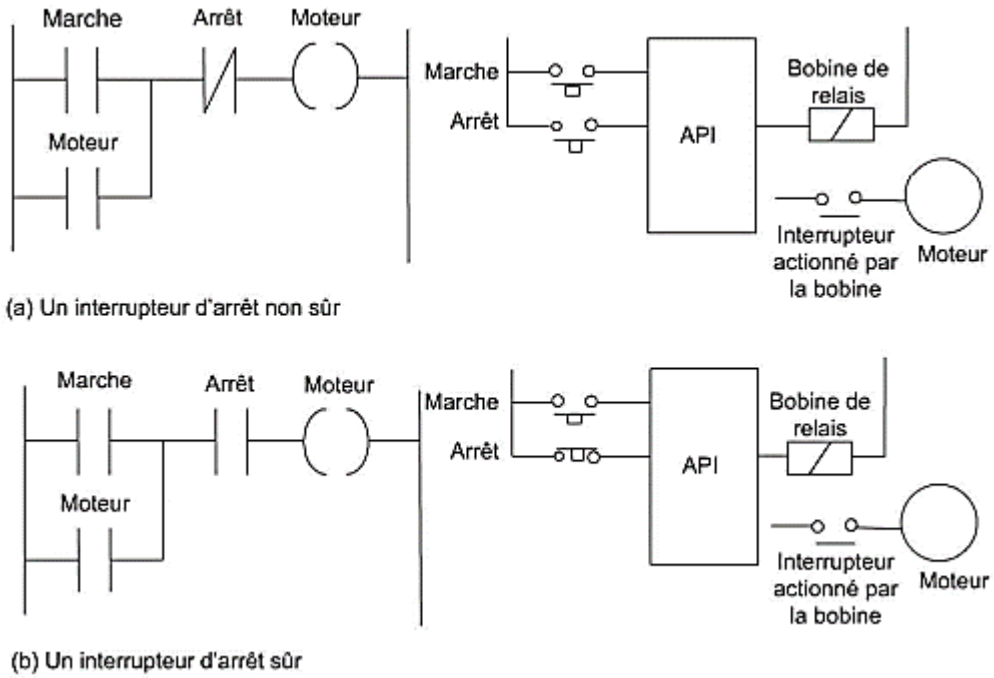


Figure 27 - Positionnement de l'interrupteur d'arrêt d'un moteur.

La Figure 28 montre o nous pouvons positionner de manière sûre un interrupteur d'arrêt d'urgence. S'il se trouve en entrée de l'API (voir Figure 28 a), un dysfonctionnement de l'API risque d'empêcher l'arrêt du moteur. En revanche, s'il se trouve en sortie, son activation arrête le moteur et déverrouille l'interrupteur de mise en marche si le montage correspond à celui de la Figure 28 b. Le moteur ne redémarrera pas tant que le bouton d'arrêt d'urgence sera relâché.
Vous devez toujours envisager la possibilité d'un dysfonctionnement de l'API. Les sorties doivent donc se placer dans un état de « sécurité intégrée » afin que les personnes travaillant dans l'usine ne risquent pas d'être blessées.

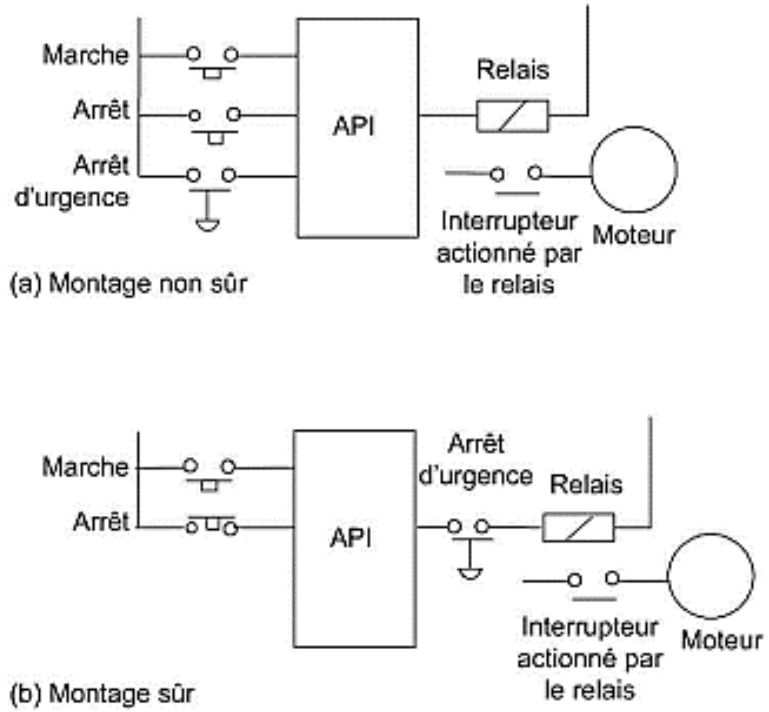


Figure 28- Positionnement d'un interrupteur d'arrêt d'urgence.

LISTE D'INSTRUCTION

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. Ainsi, nous pouvons avoir la ligne suivante :

LD A :Elle indique le chargement de l'opérande A ; LD est l'opérateur utilisé pour effectuer un chargement. 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. Voici une autre instruction :

OUT Q: Elle indique qu'une sortie doit être produite sur Q.

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 1 recense quelques codes utilisés par les fabricants et les instructions normalisées

Les listes d'instructions sont un **langage textuel de bas niveau simple** à implémenter et utilisé par nombre de fabricants d'API, principalement pour les modèles de petite et moyenne taille. Ce langage est particulièrement adapté aux programmes simples. Certains fabricants ne prennent pas en charge les listes d'instructions, mais proposent le texte structuré, un langage de plus haut niveau.

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

Tableau 1: Codes mnémoniques des instructions

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

```
LD      A           (*Charger A*)
AND     B           (*ET B*)
ST      Q           (*Stocker le résultat dans Q, c.-à-d. sortie sur 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 TAPI. LD A est ainsi une instruction qui charge A dans le registre mémoire. Il peut ensuite être rappelé pour d'autres opérations. La ligne suivante du programme effectue une opération booléenne ET entre A et B. La dernière ligne place le résultat dans Q, c'est-à-dire sur la sortie Q.

Il est possible d'utiliser des **libellés** pour identifier différents points d'entrée dans un programme. Nous le verrons plus loin, ils sont utiles pour sauter à des endroits d'un programme. Un libellé se place avant l'instruction et en est séparé par des deux-points :

```
POMPE_0K:   LD      C           (* Charger CD*)
```

Pour les opérateurs de la CEI 6113-3, l'ajout d'un N à la fin de leur mnémonique indique une valeur inversée. Par exemple :

```
LD      A       (*Charger A*)
ANDN    B       (*ET NON B*)
```

Ainsi, l'opérateur ANDN inverse la valeur des contacts et réalise un ET avec le résultat.

- # 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 ». Il peut s'agir de LD, ou peut-être A ou L, pour indiquer que la ligne commence par des contacts ouverts, ou de LDI, ou peut-être LDN, LD NOT, AN 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. Il 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 pour la porte ET illustrée à la Figure 29.

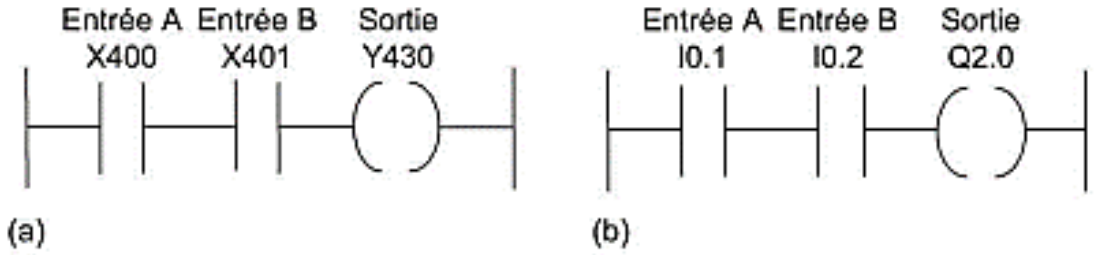
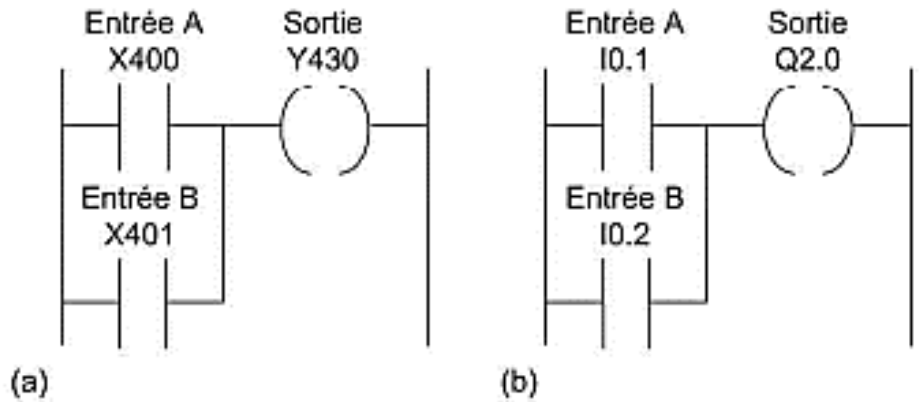


Figure 29 - Porte ET: (a) Mitsubishi et (b) Siemens en Ladder et IL

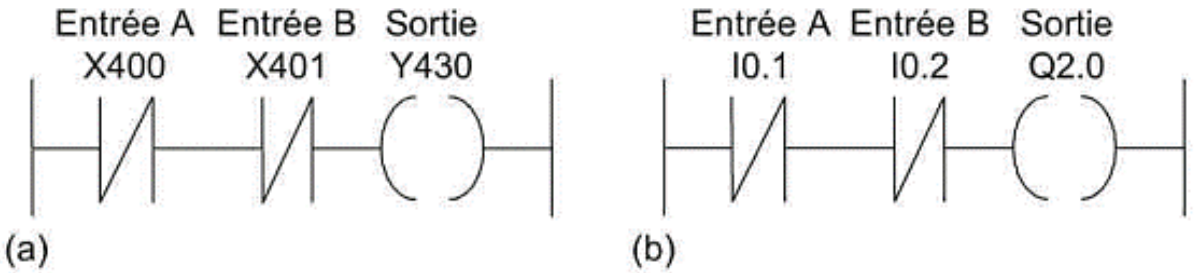
- Prenons un autre exemple, celui d'une porte OU.




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

Figure 30 - Porte OU : (a) Mitsubishi et (b) Siemens en Ladder et IL

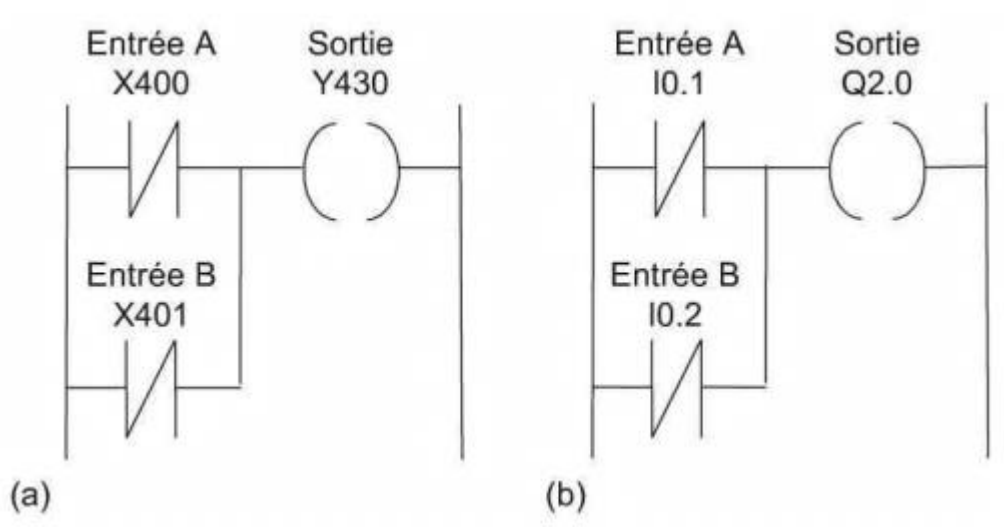
La Figure 31 présente le schéma à contacts d'une porte NON-OU dans la notation de Mitsubishi.




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

Figure 31 - Porte NOR : (a) Mitsubishi et (b) Siemens en Ladder et IL

- La Figure 32a illustre une porte NON-ET avec la notation de Mitsubishi.

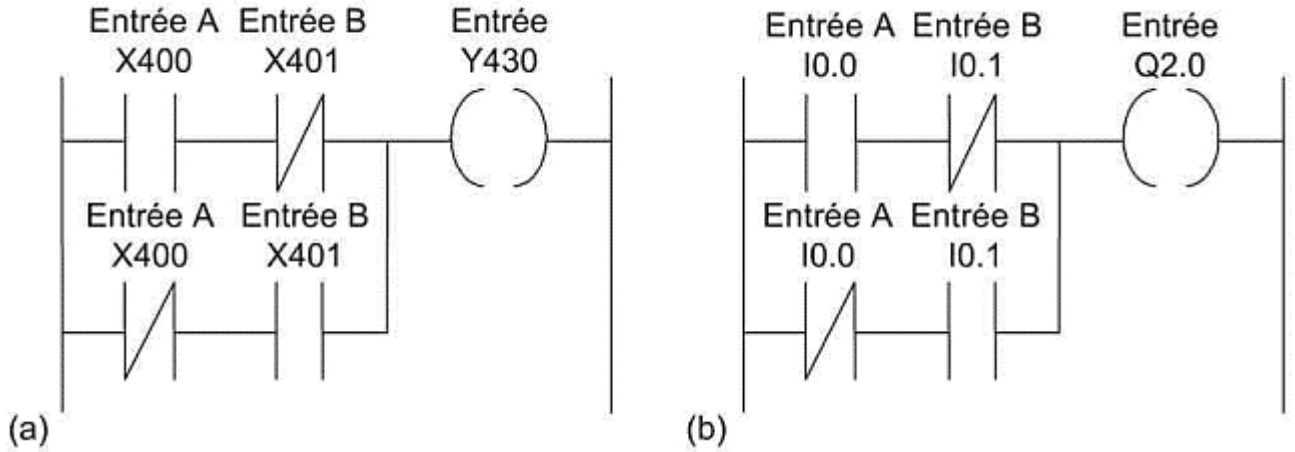



```
LDI     X400        AN      I0.1
ORI     X401        ON      I0.2
OUT     Y430        =       Q2.0
```

Figure 32 - Porte NAND : (a) Mitsubishi et (b) Siemens

Codage des embranchements

La porte OU exclusif de la Figure 33 comprend deux branches en parallèle avec un ET dans chacune d'elles.




```
LD      X400
ANI     X401
LDI     X400
AND     X401
ORB
OUT     Y430
```

Step	Instruction	
0	A(	
1	A	I0.0
2	AN	I0.1
3	)	
4	O(	
5	AN	I0.0
6	A	I0.1
7	)	
8	=	Q2.0

Figure 33 - Porte XOR

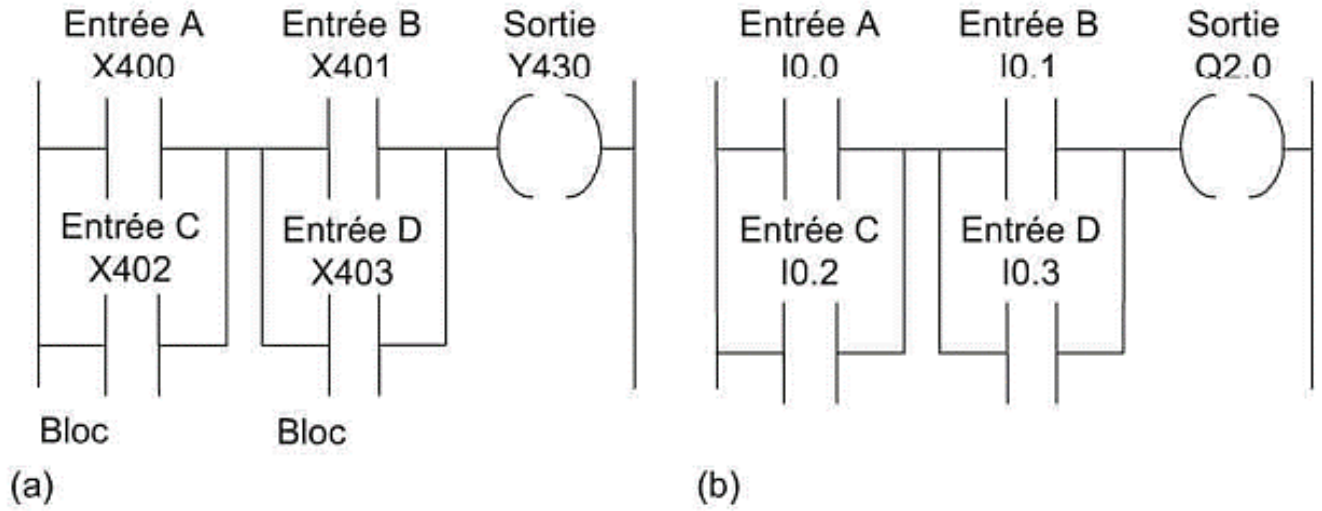
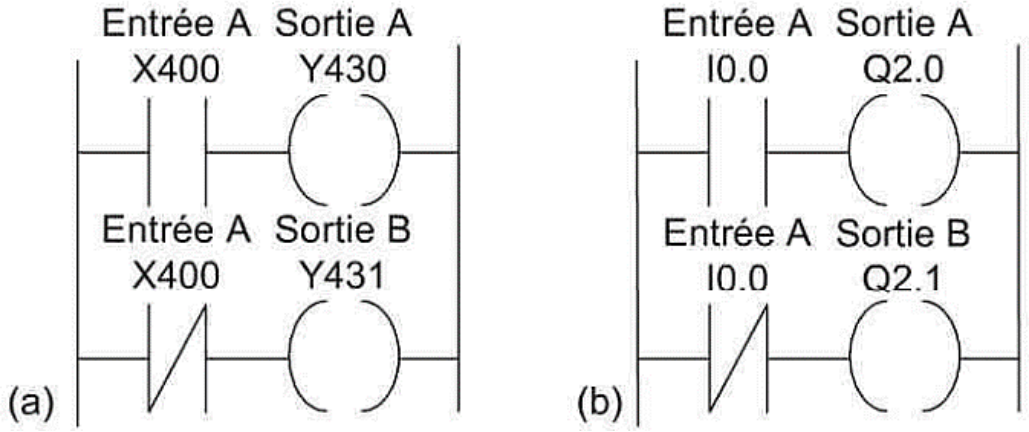


Figure 34- ET entre deux branches : (a) Mitsubishi et (b) Siemens

Étape	Instruction	
0	A(	
1	A	I0.0
2	O	I0.2
3	)	
4	A(	
5	A	I0.1
6	O	I0.3
7	)	
8	=	Q2.0

Plusieurs lignes horizontales

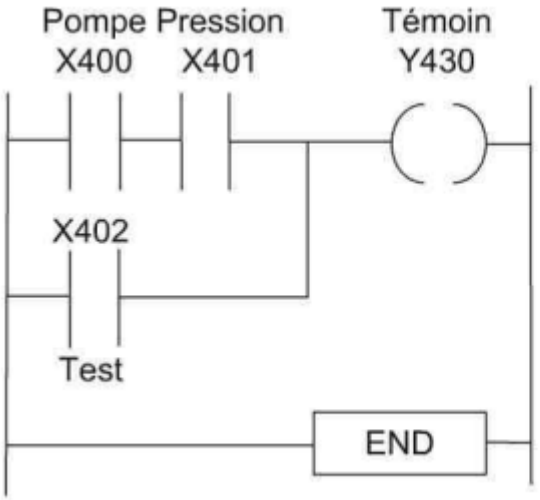



```
A       I0.0
=       Q2.0
AN      I0.0
=       Q2.1
```

Figure 35- Circuit à bascule : (a) Mitsubishi et (b) Siemens

Exemples de programmes

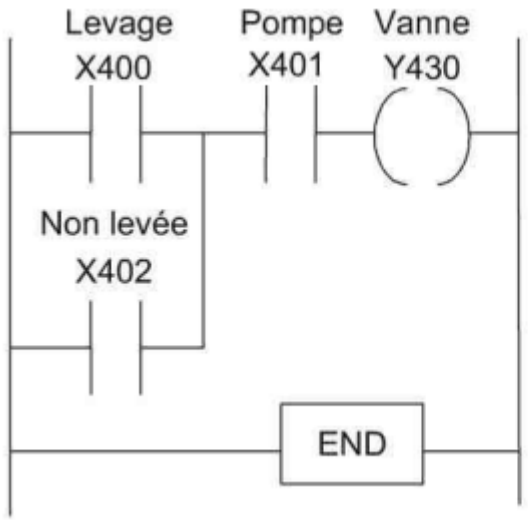
Un témoin lumineux doit être allumé lorsqu'une pompe est en marche et que la pression est satisfaisante, ou bien lorsque l'interrupteur de test est fermé. La Figure 36 montre le schéma à contacts et la liste d'instructions correspondante.




```
LD      X400
AND     X401
LD      X402
ORB
OUT     Y430
END
```

Figure 36 - Lampe témoin

Une vanne doit être actionnée pour soulever une charge lorsqu'une pompe est en fonctionnement et lorsque l'interrupteur de levage ou lorsqu'un interrupteur indiquant que la charge n'est pas encore soulevée et quelle se trouve en bas de son guide de levage est actionné. La Figure 37 présente le schéma à contacts et la liste d'instructions correspondante.




```
LD      X400
OR      X402
AND     X401
OUT     Y430
END
```

Figure 37 - Programme d'activation de la vanne.

GRAPHES DE FONCTION SÉQUENTIELLE (GRAFCET)

Le terme *graphe de fonction séquentielle* (SFC, *Sequential Function Chart)* correspond à une représentation graphique du fonctionnement d'un système afin de révéler l'enchaînement des événements qui conduisent à ce fonctionnement.

Les graphes de fonction séquentielle représentent une technique graphique puissante pour décrire le comportement séquentiel d'un programme. Les langages graphiques sont utilisés depuis plusieurs années, notamment le GRALCET. La norme CEI 61131-1, qui concerne les graphes de fonction séquentielle, emprunte énormément au GRAFCET.

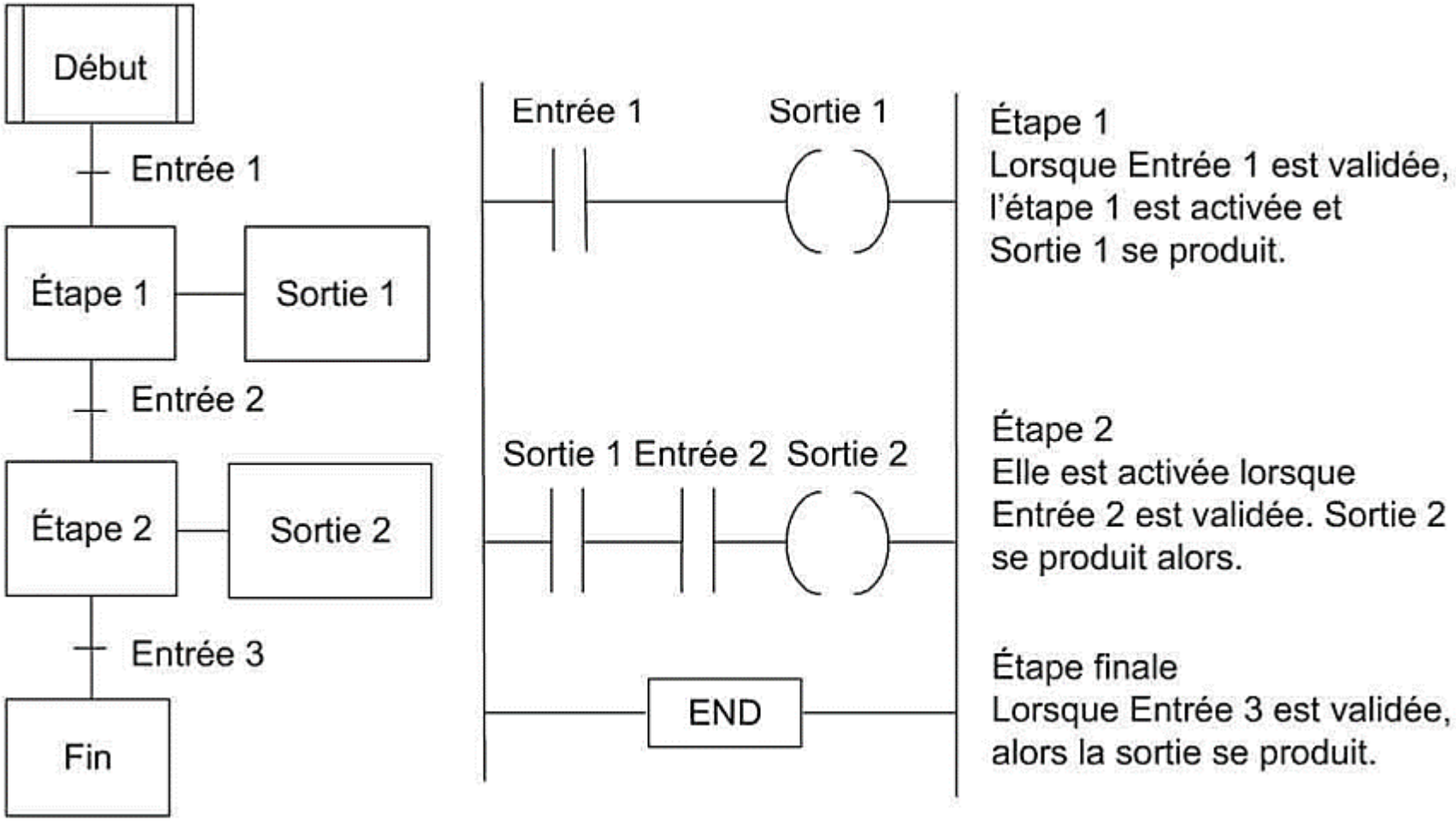


Figure 38 - Graphe de fonction séquentielle et schéma à contacts équivalent.

TEXTE STRUCTURÉ

Le texte structuré (*SQL Structured Query Language*) 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 ;
```

Cette instruction indique qu'une lampe, la variable, doit voir sa « valeur » modifiée, c'est-à-dire être allumée ou éteinte, lorsque l'interrupteur A change sa « valeur », c'est-à-dire marche ou arrêt.
Voici le format général d'une instruction d'affectation :

```
X : = Y ;
```

Y représente une expression qui produit une nouvelle valeur pour la variable X, tandis que := est le symbole d affectation. La variable conserve la valeur qui lui a été affectée jusqu'à ce qu'une autre affectation la modifie. 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.

Opérateur	Description	Précédence
(...)	Expression parenthésée	La plus élevée
Function(...)	Liste des paramètres d'une fonction	
**	Élévation à une puissance	
-, NOT	Négation, NON booléen	
*, /, MOD	Multiplication, division, modulo	
+, -	Addition, soustraction	
<, >, <=, >=	Inférieur à, supérieur à, inférieur ou égal à, supérieur ou égal à	
=, <>	Égalité, inégalité	
AND, &	ET booléen	
XOR	Ou exclusif booléen	
OR	OU booléen	La plus basse

Tableau 2 Opérateurs utilisés dans un texte structuré

Le Tableau 2 recense quelques opérateurs, comme les opérateurs OR et AND des instructions précédentes, utilisés dans les programmes en texte structuré et leur priorité relative lors de l'évaluation d'une expression. 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 ce programme, l expression (3 - **EntreeC)** est évaluée avant que sa valeur ne soit utilisée comme diviseur. La seconde partie de l'expression de **SortieQ** devient donc 4/(3- 2) = 4. Puisque la division est prioritaire sur l'addition, la première partie de l'expression de **Sorti eQ,** 6/3, est évaluée avant l'addition. Autrement dit, la valeur de **SortieQ** est 2 + 4 = 6.

Les identités des variables représentées directement commencent par le caractère % et sont suivies d'un code sur une ou deux lettres, qui indique si l'emplacement mémoire est associé à une entrée, une sortie ou à la mémoire interne, et s'il correspond à des bits, des octets ou des mots. Par exemple :

```
%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** (huit bits), **W pour un mot** (seize bits), **D pour un mot double** (trente-deux bits) ou **L pour un mot long** (soixante-quatre 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*)
```

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
```

Dans le cas suivant, 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, c'est-à-dire à 1 :

```
IF temp_fluidel OR temp_fluide2 THEN
```

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 ;
```

Notez que la fin de l instruction IF doit être indiquée. 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;
```

Dans ce cas, 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) 0F
   0 . . . 40:   Interrupteur_four := Marche ;
   40 . . . 100: Interrupteur_four := Arret ;
ELSE
   Interrupteur_four := Arret ;
END_CASE ;
```

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 ;
```

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

a 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;
```

Écriture des programmes

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

```
TYPE Moteur: (Arrête , 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: B00L ; (*Le défaut de fonctionnement de la
   pompe*)
END_VAR ; (*est une variable bool éenne. *)

VAR_OUT (*Sorties*)
   Vitesse_Moteur: REAL ;
END_VAR ;
```

```
VAR_IN
   Valeur: INT; (*La valeur est un entier*)
END_VAR ;
VAR
   Entreel AT %IX100; (*Entreel correspond au bit mémoire de
l entrée 100*)
END_VAR ;
```

Des valeurs initiales doivent être données aux variables :

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

Ce n'est qu'alors que vous pouvez ajouter les instructions
Le code suivant est un exemple de bloc fonctionnel qui peut apparaître 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.

Comparaison avec le langage à contacts

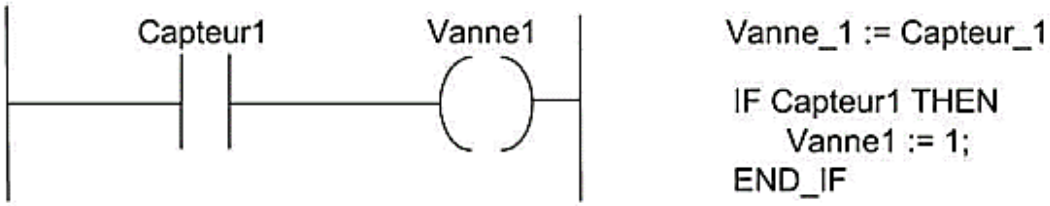


Figure 39 - Ligne d'un schéma à contacts et deux versions équivalentes en texte structuré.

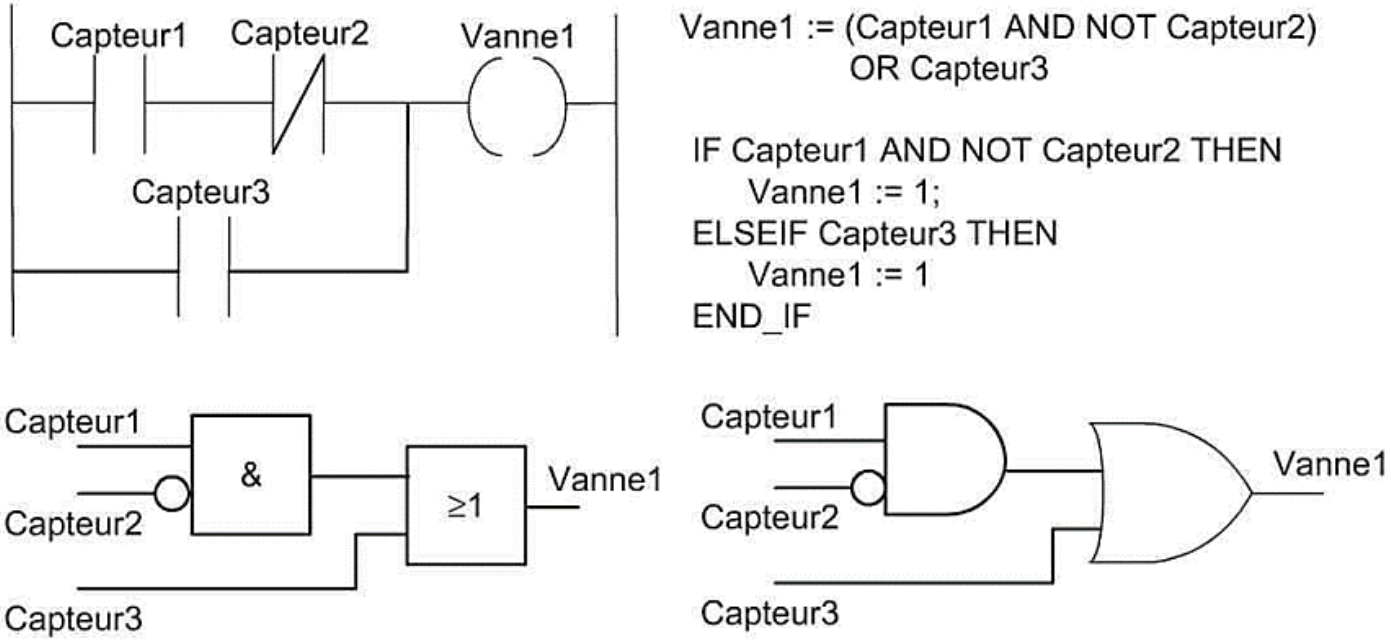


Figure 40 - Ligne d'un schéma à contacts, le diagramme de schémas fonctionnels équivalent et deux versions équivalentes en texte structuré.

Références bibliographiques

- https://www.youtube.com/user/ParsicAutomation/featured
- Automate Programmable Industriel, 2ième édition, DUNOD,
- SIEMENS TIA PORTAL13_help