Chapitre 4 Automate programmable industriel

4.1 Introduction

Depuis leurs apparitions, les automates programmables se sont répondus très rapidement dans l'industrie au point de représenter aujourd'hui plus de la moitié des équipements informatiques qui sont utilisées pour ce type d'application. Ce succès des API est dû en grande partie à leurs faible coût, à la facilité avec laquelle ils peuvent être mis en œuvre par du personnel non informaticien et à la leurs pilotage en ambiance industrielle et en temps réel des procédés logiques séquentiels et combinatoires.

Les API sont en générale conçus dès l'origine pour fonctionner en milieu industriel, avec toutes les protections nécessaires contre l'environnement, et avec les interfaces adaptées aux capteurs et aux actionneurs les plus courants.

4.2 Architecture d'un API

L'automate programmable est un système de traitement logique d'information dont le programme de fonctionnement est effectué à partir d'instructions établies en fonction du processus à réaliser.

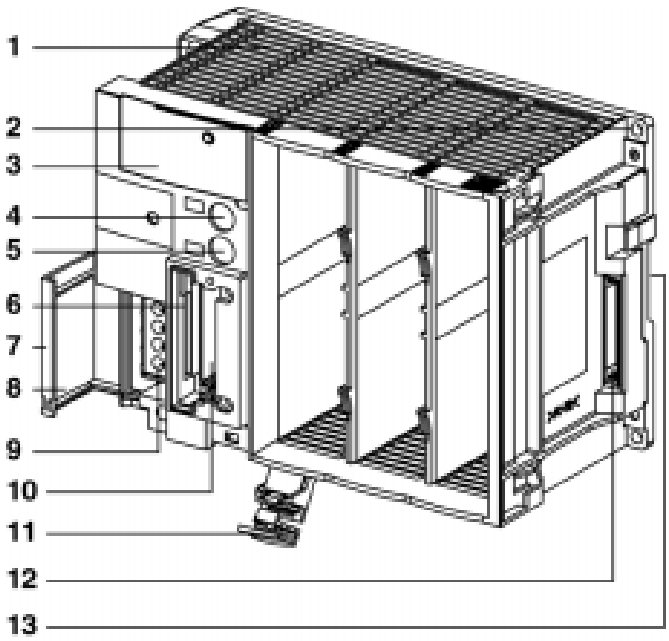


Figure 4-1- API TSX 37-21

1	Bac à 3 emplacements, intégrant l'alimentation, le processeur et sa mémoire de base.
2	Trou de fixation de l'automate.
3	Bloc de visualisation centralisée.
4	Prise terminal TER.
5	Prise de dialogue opérateur AUX.
6	Emplacement pour une carte d'extension mémoire. En l'absence de carte, cet emplacement est équipé d'un cache qu'il est obligatoire de maintenir en place; son extraction provoquant l'arrêt de l'automate.
7	Trappe d'accès aux bornes d'alimentation.
8	Etiquette à renseigner pour le changement de la pile.
9	Bornes d'alimentation.
10	Emplacement pour un coupleur de communication.
11	Trappe d'accès à la pile optionnelle et au commutateur de protection en écriture du système d'exploitation.
12	Connecteur de raccordement du mini-bac d'extension, protégé de base par un cache amovible.
13	Dispositif pour montage sur profilé DIN.

Le schéma simplifié de la structure interne d'un automate est donné par la figure 4.2. Comme pour un microcontrôleur ou un système à microprocesseur, on remarque la présence d'un bus de données sur lequel sont reliées la mémoire données, la mémoire programme et les périphériques (timers, compteurs, périphériques de communication). Cette ressemblance avec la structure du microcontrôleur est normale, comme le processeur étant au cœur de l'automate.

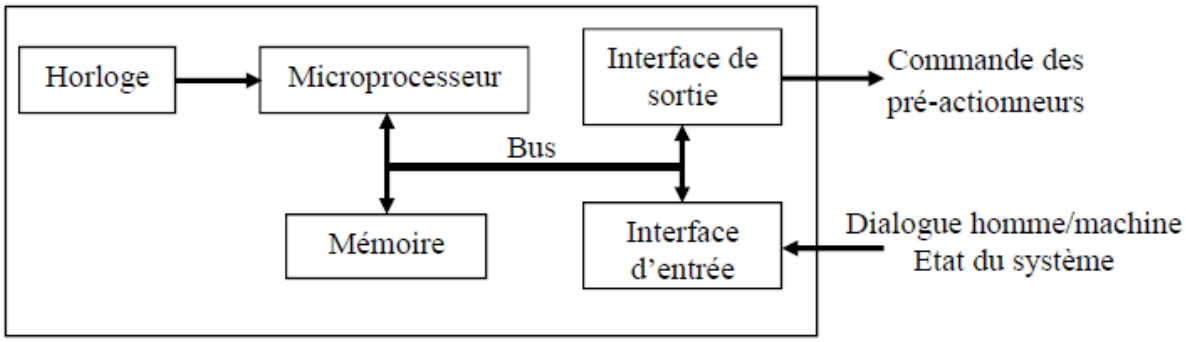


Figure 4-2- La structure interne d'un automate

Les API comportent essentiellement :

- Partie alimentation
- L'unité centrale
- La mémoire
- Interfaces d'entrées/sorties

4.2.1 L'alimentation

Ce module génère l'ensemble des tensions nécessaires au bon fonctionnement de l'automatisme.

L'alimentation intégrée dans l'API, fournit à partir des tensions usuelles des réseaux (380, 220, 110, 48, 24V) les tensions continues nécessaires au fonctionnement des circuits électroniques.

4.2.2 L'unité centrale

L’unité centrale est le regroupement du processeur et de la mémoire centrale. Elle commande l’interprétation et l’exécution des instructions programmes .Les instructions sont effectuées les unes après les autres, séquencée par une horloge.

Exemple: Si deux actions doivent être simultanées, l'API les traite successivement.

Le microprocesseur réalise toutes les fonctions arithmétique, logiques ET, OU, les fonctions de temporisation, de comptage, de calcul... à partir d'un programme contenu dans sa mémoire. Il est connecté aux autres éléments (mémoire et interface E/S) par des liaisons parallèles appelées ' BUS ' qui véhiculent les informations sous forme binaire.

Les Caractéristiques principales d’un microprocesseur sont :

- Vitesses de traitement : C'est la vitesse de l'UC pour exécuter 1 K-instructions logiques. (10 à 20 ms/Kmots).
- Temps de réponse : scrutation des entrées, vitesse de traitement et affectation des sorties.

4.2.3 La mémoire

Elle contient sous forme binaire des informations indispensables au bon fonctionnement d’un automate, et se compose de :

- **la mémoire programme** : où est stocké le langage programme, elle est en général figée c’est à dire accessible en lecture seulement : ROM (Read Only Memory) : mémoire morte. Il existe d’autres types de mémoires mortes : l’EPROM mémoires mortes reprogrammables effacement aux rayons ultra-violets et l’EEPROM mémoires mortes reprogrammables effacement électrique,

- **la mémoire des données** : dans laquelle on peut lire, écrire et effacer pendant le fonctionnement, c’est la RAM (Random Acces Memory), c’est une mémoire vive.

La Zone mémoire va permettre de :

- recevoir les informations issues des capteurs d’entrées

- recevoir les informations générées par le processeur et destinées à la commande des sorties (valeur des compteurs, des temporisations, …)
- recevoir et conserver le programme du processus

4.2.4 Interfaces d'entrées/sorties

Les entrées reçoivent des informations en provenance des éléments de détection (capteurs) et du pupitre opérateur.

Les sorties transmettent des informations aux pré-actionneurs (relais, électrovannes …) et aux éléments de signalisation (voyants) du pupitre.

4.2.4.1 ***Interfaces d'entrées :***

Elles sont destinées à :

- Recevoir l'information en provenance des capteurs
- Traiter le signal en le mettant en forme, en éliminant les parasites et en isolant électriquement l'unité de commande de la partie opérative.

4.2.4.2 ***Interfaces de sorties :***

Elles sont destinées à :

- Commander les pré-actionneurs et éléments des signalisations du système
- Adapter les niveaux de tensions de l'unité de commande à celle de la partie opérative du système en garantissant une isolation galvanique entre ces dernières.

4.2.4.3 ***Terminaux de programmation et de réglage***

L'API doit permettre un dialogue avec :

- Le personnel d'étude et de réalisation pour réaliser la première mise en œuvre (Edition programme, Transfert, Sauvegarde...)

- Le personnel de mise au point et de maintenance pour réaliser des opérations sur le système (Forçage, Visualisation de l'état, Modification de paramètres temporisation, compteurs....)

Ce dialogue peut être réalisé par :

- Une Console : Elle sera utilisée sur site, elle comporte un clavier, un écran de visualisation et un langage de programmation.
- Un Micro-ordinateur avec un logiciel d'assistance à la programmation : Il sera utilisé hors site. Il comprend plusieurs modules pour permettre l'édition, l'archivage, la mise au point des applications.

4.3 Principe de fonctionnement

L'automate programmable reçoit les informations relatives à l'état du système, c'est la phase de détection, et puis commande les pré-actionneurs suivant le programme inscrit dans sa mémoire, c'est le traitement logique des informations.

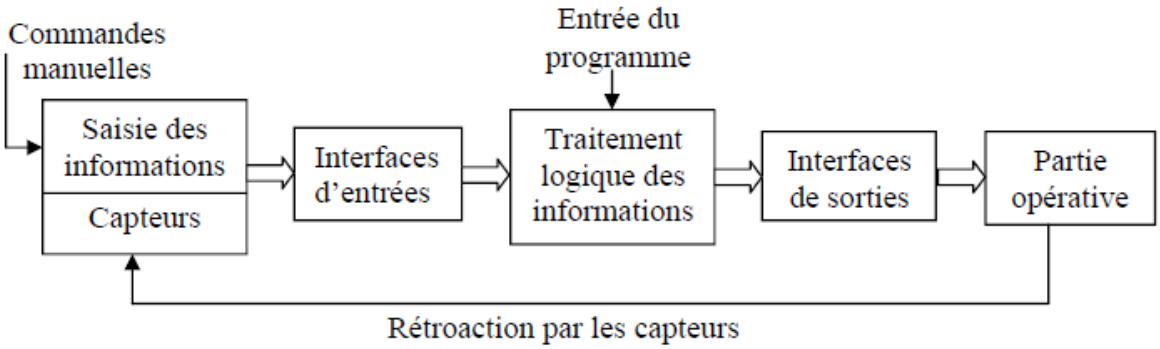


Figure 4-3- Principe de fonctionnement de l'API

4.3.1 La détection

Le capteur fournit au séquenceur, des comptes rendus sur l'état du système. Il convertit les informations physiques de la PO en grandeurs électriques exploitables par le séquenceur. Ils peuvent détecter des positions, des pressions, des températures, des vitesses, des accélérations, des niveaux, des présences, des états...

Au delà du simple capteur à contact et à commande mécanique, il existe un grand nombre de modèles afin de répondre aux multiples problèmes posés.

On distingue tout d'abord les capteurs par le type de signal qu'ils transmettent :

- Les capteurs analogiques traduisent des valeurs de positions, de pressions, de températures ... sous forme d'un signal (tension ou courant) évoluant constamment entre deux valeurs limites.
- Les capteurs numériques transmettent des valeurs numériques précisant des positions, des pressions,... pouvant être lus soit en parallèle sur plusieurs conducteurs ; soit en série sur un seul conducteur.

4.3.2 Traitement logique des informations

Le traitement a lieu en quatre phases.

- ***Phase1 : Gestion du système*** (autocontrôle de l'automate)

- ***Phase2 : Acquisition des entrées:*** Durant cette phase, les signaux appliqués à l'interface d'entrée de l'API sont copiés en mémoire dans des emplacements accessibles au programme et qui correspondent aux variables d'entrée. Les variables d'entrée sont uniquement accessibles en lecture. Leurs valeurs resteront ainsi inchangées lors des deux phases suivantes du cycle. En d'autres termes, au moment de l'acquisition des entrées, l'API "prend une photo" de l'environnement physique.

- ***Phase 3 : traitement des données:*** c'est une phase d'exécution du programme et de calcul des valeurs de sorties à partir des valeurs stockées en mémoire dans la phase précédente, les résultats des calculs sont ensuite à leur tour stockés en mémoire. Elle se fait selon trois instructions :

    - Lecture du programme (située dans la RAM programme) par l'unité de traitement,
    - Lecture des variables (RAM données).

- Traitement et écriture des variables dans la RAM données.

- ***Phase 4 : Émission des ordres:*** Lecture des variables de sorties dans la RAM données et transfert vers le module des sorties. Les variables de sorties sont affectées à l'interface de sortie pour pouvoir être appliquées aux pré-actionneurs.

4.3.3 Dialogue d'exploitation

En période d'exploitation, des dialogues homme machine sont nécessaires :

- Pour la conduite de la machine,
- Pour des réglages,
- Pour des dépannages.

Les constituants prévus pour ces dialogues comportent :

- Des constituants implantés tels que boutons, voyants, pupitres configurables…
- Des terminaux d'exploitation amovibles, pour réglages et démarrages.

4.4 Jeu d'instructions

Le processeur peut exécuter un certain nombre d'opérations logiques; l'ensemble des instructions booléennes des instructions complémentaires de gestion de programme (saut, mémorisation, adressage ...) constitue un jeu d'instructions.

Chaque automate possède ses propres jeux d'instructions. Mais par contre, les constructeurs proposent tous une interface logicielle de programmation répondant à la norme CEI1131-3. Cette norme définit cinq langages de programmation utilisables, qui sont :

- Les langages graphiques :
  - **LD** : **L**adder **D**iagram (Diagrammes échelle)

- **FBD** : **F**unction **B**lock **D**iagram (Logigrammes)
- **SFC** : **S**equential **F**unction **C**hart (Grafcet)

- Les langages textuels :
  - **IL** : **I**nstruction **L**ist (Liste d'instructions).
  - **ST**: **S**tructured **T**ext (Texte structuré).

4.4.1 Langage à contact (LD)

Le langage à relais (Ladder Diagram) est basé sur un symbolisme très proche de celui utilisé pour les schémas de câblage classiques. Les symboles les plus utilisés sont donnés au tableau suivant :

Désignation	***Symbole***	***Fonctions***
Contact à fermeture		Contact passant quand l'objet bit qui le pilote est à l'état 1.
Contact à Ouverture		Contact passant quand l'objet bit qui le pilote est à l'état 0.
Contact à détection de front montant	P	Front montant : détection du passage de 0 à 1 de l'objet bit qui le pilote.
Contact à détection de front descendant	N	Front descendant : détection du passage de 1 à 0 de l'objet bit qui le pilote.
Bobine directe	( )	L'objet bit associé prend la valeur du résultat de la zone test.
Bobine Inverse	(/)	L'objet bit associé prend la valeur inverse du résultat de la zone test.
Bobine d'enclenchement	(S)	L'objet bit associé est mis à 1 lorsque le résultat de la zone test est à 1.

Bobine de déclenchement	—(R)—	L'objet bit associé est mis à 0 lorsque le résultat de la zone test est à 1.
Bobine dièse	—(#)—	Proposée en langage Grafcet, utilisée lors de dièse la programmation des réceptivités associées aux transitions provoque le passage à l'étape suivante.

Tableau 1 : Les éléments graphiques des contacts

4.4.2 Langage liste d'instructions

Une section écrite en langage liste d'instructions se compose d'une suite d'instruction exécutées séquentiellement par l'automate.

La figure suivante présente un programme liste d'instruction PL7.

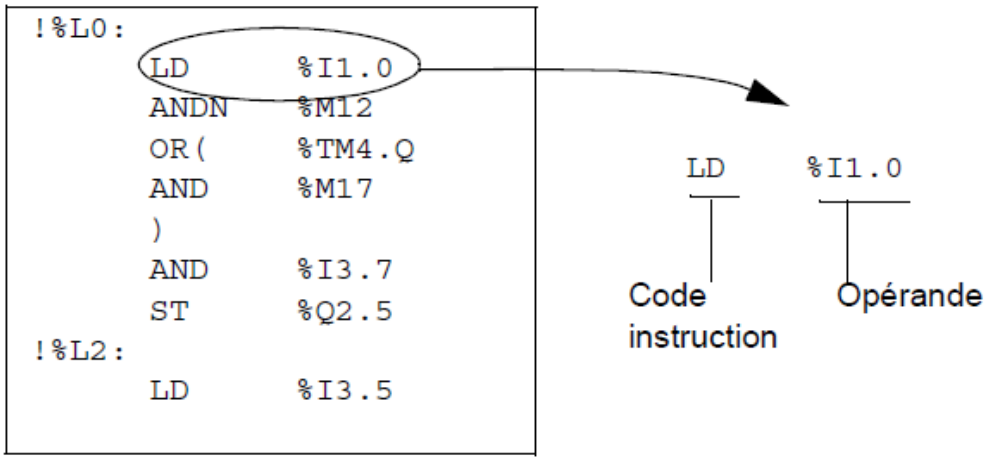


Figure 4-4 Exemple programme IL

Une instruction IL est composée de :

- **<u>Code instruction :</u>** Le code instruction détermine l'opération à exécuter. Il existe 2 types de codes instructions :
  - **test**, dans laquelle figurent les conditions nécessaires à une action (ex : `LD`, `AND`, `OR`...),
  - **action**, qui sanctionne le résultat consécutif à un enchaînement de test. (ex: `ST`, `STN`, `R`, ...).

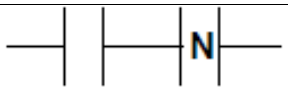
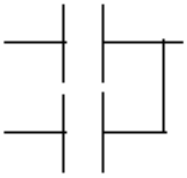
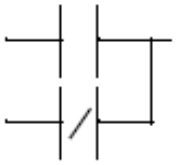
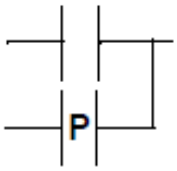
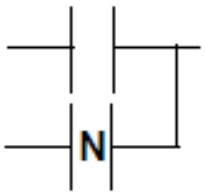
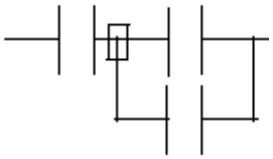
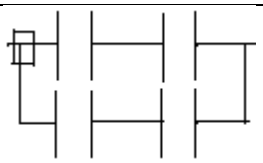
- **<u>Opérande :</u>** Une instruction agit sur un opérande. Cet opérande peut être :
    - une entrée/sortie de l'automate (boutons-poussoirs, détecteurs, relais, voyants...),
    - une fonction d'automatisme (temporisateurs, compteurs...),
    - une opération arithmétique et logique ou une opération de transfert,
    - une variable interne de l'automate

Le langage liste d'instructions comporte des instructions :

- de test
- d'action
- sur bloc fonction
- Numériques

Les deux tableaux suivant décrivent les instructions de test et d'action.

Désignation	***Graphisme équivalent***	***Fonctions***
LD		Le résultat booléen est égal à l'état de l'opérande.
LDN		Le résultat booléen est égal à l'état inverse de l'opérande.
LDR	P	Le résultat booléen passe à 1 à la détection du passage de 0 à 1 de l'opérande (front montant).
LDF	N	Le résultat booléen passe à 1 à la détection du passage de 1 à 0 de l'opérande (front descendant).
AND		Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente et l'état de l'opérande.
ANDN		Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente et l'état inverse de l'opérande.

ANDR		Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente, et la détection d'un front montant de l'opérande (1=front montant).
ANDF		Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente, et la détection d'un front descendant de l'opérande (1=front descendant).
OR		Le résultat booléen est égal au Ou logique entre le résultat booléen de l'instruction précédente et l'état de l'opérande.
ORN		Le résultat booléen est égal au Ou logique entre le résultat booléen de l'instruction précédente et l'état inverse de l'opérande.
ORR		Le résultat booléen est égal au Ou logique entre le résultat booléen de l'instruction précédente, et la détection d'un front montant de l'opérande (1=front montant).
ORF		résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente et l'état de l'opérande. et la détection d'un front descendant de l'opérande (1=front descendant).
AND(		Et logique (8 niveaux de parenthèses)
OR(		Ou logique (8 niveaux de parenthèses)
XOR, XORN,	-	Ou exclusif

XORR, XORF		
MPS **MRD** **MPP**		Aiguillage vers des bobines. MPS (Memory PuSh) MRD (Memory ReaD) MPP (Memory PoP)
N	-	Négation

Tableau 2- Instruction de test

Désignation	***Graphisme équivalent***	***Fonctions***
ST		L'opérande associé prend la valeur du résultat de la zone test.
STN		L'opérande associé prend la valeur inverse du résultat de la zone test.
S		L'opérande associé est mis à 1 lorsque le résultat de la zone test est à 1.
R		L'opérande associé est mis à 0 lorsque le résultat de la zone test est à 1.
JMP	-	Permet un branchement inconditionnel à une phrase étiquetée, amont ou aval.
JMPC	-	Permet un branchement conditionné à un résultat booléen à 1, à une phrase étiquetée amont ou aval.
JMPCN	-	Permet un branchement conditionné à un résultat booléen à 0, à

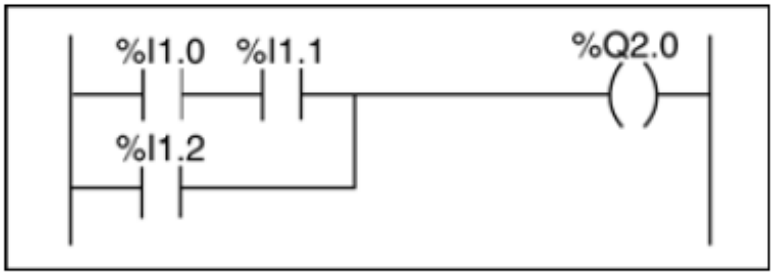
		une phrase étiquetée amont ou aval.
SRn	-	Branchement en début de sous-programme.
RET	-	Retour de sous-programme.
RETC	-	Retour de sous-programme conditionné à un résultat booléen à 1.
RETCN	-	Retour de sous-programme conditionné à un résultat booléen à 0.
END	-	Fin de programme.
ENDC	-	Fin de programme conditionné à un résultat booléen à 1.
ENDCN	-	Fin de programme conditionné à un résultat booléen à 0.

Tableau 3 Instructions d'action

4.4.3 Exercice d'application :

Traduire les programmes de langage LD en langage IL :

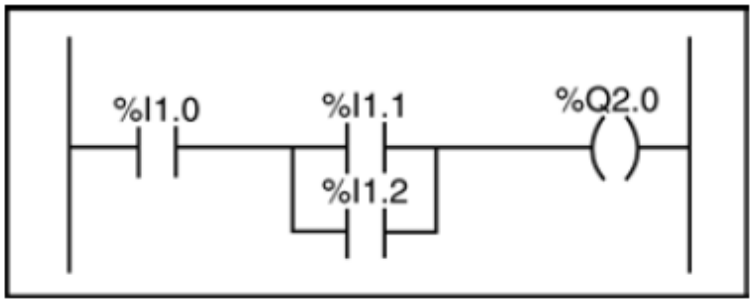
1.



Solution :

LD %I1.0
AND %I1.1
OR %I1.2
ST %Q2.0

2.



Solution :

LD %I1.0

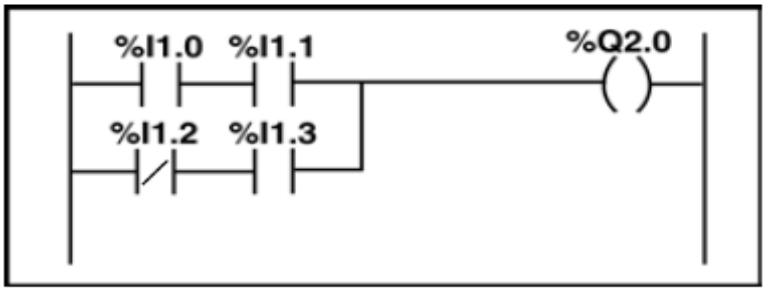
AND (%I1.1

OR %I1.2

)

ST %Q2.0

3.



Solution

LD %I1.0

AND %I1.1

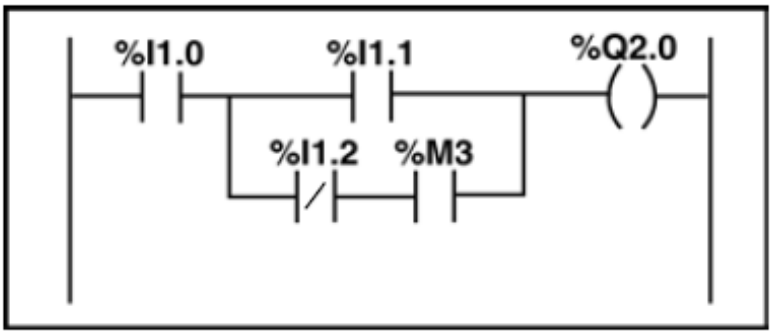
OR(N %I1.2

AND %I1.3

)

ST %Q2.0

4.



LD %I1.0

AND(%I1.1

OR(N %I1.2

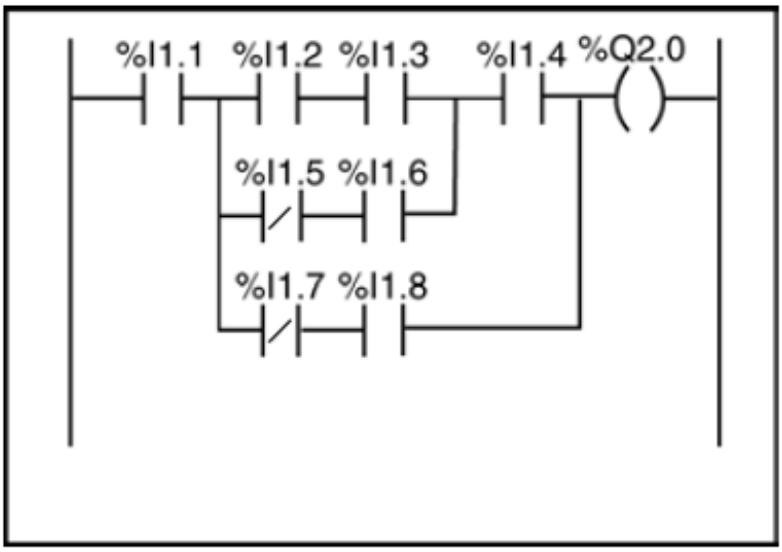
AND %M3

)

)

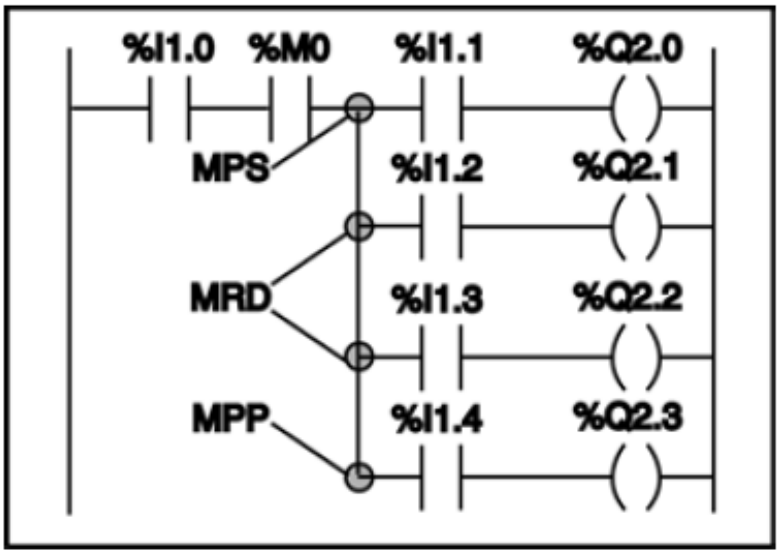
ST %Q2.0

5.



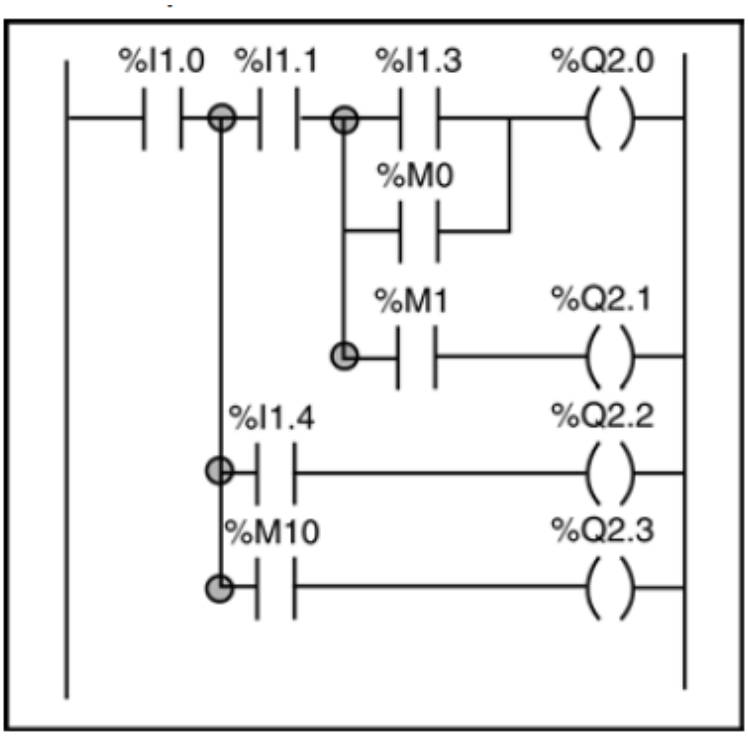
LD %I1.1
AND(%I1.2
AND %I1.3
OR(N %I1.5
AND %I1.6
)
AND %I1.4
OR (N %I1.7
AND %I1.8
)
)
ST %Q2.0

6.



LD %I1.0
AND %M0
MPS
AND %I1.1
ST %Q2.0
MRD
AND %I1.2
ST %Q2.1
MRD
AND %I1.3
ST %Q2.2
MPP
AND %I1.4
ST %Q2.3

7.



LD %I1.0
MPS
AND %I1.1
MPS
AND(%I1.3
OR %M0
)
ST %Q2.0
MPP
AND %M1
ST %Q2.1
MRD
AND %I1.4
ST %Q2.2
MPP
AND %M10
ST %Q2.3