Chapitre II LANGAGES DE PROGRAMMATION : NORME IEC 1131-3

II.1 Introduction

II.1.1 Logique câblée et programmée

II.1.1.a. La logique câblée

L'automatisme est obtenu en reliant entre eux les différents constituants de base ou fonctions logiques par câblage. La logique câblée correspond donc à un traitement parallèle de l'information. Plusieurs constituants peuvent être sollicités simultanément.

Elle est étudiée et réalisée une fois pour toutes sur un schéma donné : Les fonctions sont réalisées par voie matérielle.

- ***Inconvénients***
  - ✓ Volume du contrôle proportionnel à la complexité du problème,
  - ✓ Des modifications de la commande impliquent des modifications de câblage,
  - ✓ Exige un grand nombre de composants et rend les montages encombrants,
  - ✓ La durée des études pour réaliser un montage donné (et donc pour le modifier le cas échéant) est longue
- ***Avantages***
  - ✓ Moins chère pour les cas simple et non complexe.

II.1.1.b. Logique programmée

Elle correspond à une démarche séquentielle, seule une opération élémentaire est exécutée à la fois, c'est un traitement série. Le schéma électrique est transcrit en une suite d'instruction qui constitue le programme.

- ***Inconvénients***
  - ✓ Coût de l'API est relativement cher,
  - ✓ Investissement non rentable si le montage est simple.

- ***Avantages***
  - ✓ En cas de modification des équations avec les mêmes accessoires, l'installation ne comporte aucune modification de câblage seul le jeu d'instructions est modifié,
  - ✓ Utilisation réduite de composants puisque ils sont remplacés directement les fonctions logiques désirées,
  - ✓ Un circuit ayant moins de composants sera habituellement moins coûteux à concevoir, réaliser et distribuer, et simplification de la maintenance,
  - ✓ La réduction du nombre de composants électroniques tend aussi à augmenter la fiabilité des circuits et à réduire la consommation énergétique,
  - ✓ L'automate simplifie grandement le schéma de la logique câblée prenant en compte tout ce qui est extérieur à la programmation, comme les voyants.
  - ✓ Réduire les coûts d'ingénierie
  - ✓ Réduire les coûts de maintenance

➢ ***Les limites :***

Le choix du type d'une logique pour résoudre un problème, dépend de plusieurs critères : Complexité ; coût ; évolutivité ; rapidité.

- ***Limite inférieure :***

Si la fonction à réaliser est trop simple, il est plus économique de conserver une logique câblée.

- ***Limite supérieure :***

Si le nombre d'unités à réaliser est très important, il est plus économique de la fabriquer en circuits intégrés à la demande ou en logique câblée pour des fonctions simples.

II.1.2 Besoin de la normalisation

Les besoins d'une normalisation des langages pour API s'exprime, pour les industriels, en termes de:

- ✓ Faciliter la formation des réalisateurs de configurations d'automates programmables,
- ✓ Obtenir un bon niveau de portabilité des programmes,
- ✓ Favoriser la création de bibliothèques de blocs fonctionnels fiables,
- ✓ Faciliter les configurations en réseau d'automates,
- ✓ Améliorer la qualité des applications (sûreté de fonctionnement, maintenabilité, extensibilité),
- ✓ Réutiliser les outils de configuration et de programmation,
- ✓ Produire des dossiers d'applications homogènes,
- ✓ Faciliter la maintenance du logiciel d'application.

II.1.3 Historique

Le CENELEC (Comité Européen de Normalisation Electrotechnique) a adopté en 1993 le texte (65B) de la CEI (Commission Electrotechnique Internationale) comme norme EN 61131 pour la commande des processus industriels. Cette norme traite des automates programmables en 5 parties :

- ✓ CEI 1131-1 définitions, , informations générales.
- ✓ CEI 1131-2 spécifications et essais matériels,
- ✓ CEI 1131-3 langages de programmations
- ✓ CEI 1131-4 documentations
- ✓ CEI 1131-5 communications

L'originalité des langages de programmation pour automates est qu'ils sont généralement imagés par rapport à l'expression de la commande des machines automatisées. Ils sont souvent graphiques et depuis 20 ans des formes de langages se sont dégagés et sont mise à disposition par les constructeurs d'API (liste d'instruction mnémonique, réseau à contacts, GRAFCET).

II.1.4 Norme ICE 1131

La norme CEI 1131 s'applique aux automates programmables industriels et à leurs périphériques. Les objectifs sont:

- ✓ ***Donner les définitions et identifier*** les principales caractéristiques permettant de sélectionner et utiliser les automates programmables et leurs périphériques associés.
- ✓ ***Déterminer les prescriptions minimales*** relatives aux caractéristiques fonctionnelles, aux conditions de service, aux caractéristiques constructives, à la sécurité générale ainsi qu'aux essais applicables aux automates programmables et à leurs périphériques.
- ✓ ***Définir pour les langages de programmation*** les principaux champs d'applications, les règles syntaxiques et sémantiques ainsi que des ensembles de base simples mais exhaustifs d'éléments de programmation.
- ✓ ***Fournir à l'utilisateur*** des informations générales didactiques et des recommandations quant à son application.

La partie 3 de cette norme, 1131-3, définit :

- ✓ ***Les langages de programmation,***
- ✓ ***Les modules logiciels ou unités d'organisation de programmes.***

➢ ***Remarque***

L'automaticien peut introduire son programme dans l'API de trois manières différentes :

- ✓ A l'aide de touches sur l'A.P.I. lui-même,
- ✓ Avec une console de programmation reliée par un câble spécifique à l'A.P.I,
- ✓ Avec un PC et un logiciel approprié, (exemple : Step 7 pout les API Siemens ou PL7 pour Schneider, millenium pour les API Crouet…) et le câble reliant le PC à l'API.

II.2 Langages de programmation

La norme IEC 1131-3 définit cinq langages qui peuvent être utilisés pour la programmation des automates programmables industriels. Ces cinq langages sont :

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

II.2.1 Langages graphiques

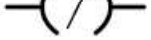
Ils permettent une transcription graphique aussi directe que possible des modèles afin de faciliter les tâches de programmation et de réduire les sources d'incertitudes. On distingue les trois langages suivants : Schéma à contact, Schéma par blocs et Diagramme fonctionnel de séquence.

II.2.1.a. Schéma à contacts ou relais, Ladder Diagram (LD)

Il utilise des symboles électriques qui assemblés forment le programme. Ce type de programmation à l'avantage de pouvoir être utilisé par du personnel électricien ou ayant une connaissance de la schématique électrique sans pour autant apprendre un langage spécifique. Il est adapté au traitement combinatoire (dédié à la programmation d'équations booléennes).

Le langage à relais est basé sur un symbolisme très proche de celui utilisé pour les schémas de câblage classiques. Les symboles les plus utilisés sont:

Désignation	**Schéma à contact**	**Fonction**	**Symbole API**
Contact à Ouverture		Passant à l'état de repos quand il n'est pas actionné.	
Contact à Fermeture		Ouvert à l'état de repos, continuité électrique non assurée	
Connexion horizontale		Relie des éléments en série	
Connexion verticale		Relie des éléments en parallèle	

Désignation	**Schéma à contact**	**Fonction**	**Symbole API**
Bobine directs		La sortie prend la valeur du résultat logique.	
Bobine inverse		La sortie prend la valeur inverse du résultat logique.	
Bobine d'enclenchement		Le bit interne est mis à **1** et garde cet état	
Bobine de déclenchement		Le bit interne est mis à **0** et garde cet état	

➢ ***Remarque***

- Un bit étant une mémoire interne logique prenant la valeur 0 ou 1.
- Une bobine d'enclenchement S « Set » et une bobine de déclenchement R « Reset » correspondent à un relais bistable.
- En plus des blocs fonctions logiques d'automatisme , il existe les blocs de temporisation de comptage….
-

II.2.1.b. Schéma par blocs Fonctionnel, ou function block diagram (FBD)

Le langage FBD permet de programmer graphiquement à l'aide de blocs, représentant des variables, des opérateurs ou des fonctions. Il permet d'une part la construction d'équations complexes à partir des opérateurs standards, de fonctions ou de blocs fonctionnels et d'autre part de manipuler tous les types de variables. Les blocs sont programmés (bibliothèque) ou programmables.

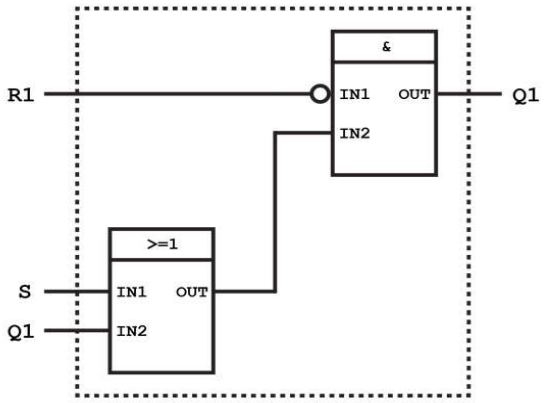


Fig II. 1 : Schéma de principe d'une fonction bloc.

- ***Les principales fonctions sont***
  - ✓ L'énonce RETURN (peut apparaître comme une sortie du diagramme, si liaison connectée prend l'état booléen TRUE, la fin du diagramme n'est pas interprétée.
  - ✓ Les étiquettes et les sauts conditionnels sont utilisés pour contrôler l'exécution du diagramme. Aucune connexion ne peut être réalisée à droite d'un symbole d'étiquette ou de saut.
  - ✓ Saut à une étiquette (le nom de l'étiquette est « LAB ») :
    - ▪ Si la liaison à gauche du symbole de saut prend l'état booléen TRUE, l'exécution du programme est déroutée après l'étiquette correspondante.
    - ▪ L'inversion booléenne est représentée par un petit cercle

II.2.1.c. Langage GRAFCET (Sequential Function Chart :SFC)

Connu sous aussi sous le nom de Diagramme fonctionnel de Séquence, il permet de représenter graphiquement et de façon structurée le fonctionnement d'un automatisme séquentiel. Il est dérivé du GRAFCET. Le procédé est représenté comme une suite connue d'étapes (états stables), reliées entre elles par des transitions, une condition booléenne est attachée à chaque transition. Les actions dans les étapes sont décrites avec les langages ST, IL, LD ou FBD.

- ***Les principales règles graphiques sont :***
  - ✓ Un programme SFC doit contenir au moins une étape initiale,
  - ✓ Une étape ne peut pas être suivie d'une autre étape,
  - ✓ Une transition ne peut pas être suivi d'une autre transition.
- ***Les composants de base (symboles graphiques) du graphique SFC sont :***
  - ✓ Etapes et étapes initiales,
  - ✓ Transitions,
  - ✓ Liaisons orientées,
  - ✓ Renvoi à une étape.

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

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

II.2.2 Langages textuels

Il s'agit de décrire le fonctionnement du processus par un programme sous forme un texte. Deux langage existent : Liste d'instruction et littéral structuré.

II.2.2.a. Liste d'instructions ou Instruction List (IL)

Le langage IL (instruction list), est un langage textuel de bas niveau. Il est particulièrement adapté aux applications de petite taille . C'est un langage "machine" qui permet l'écriture de traitements logiques et numériques. Il peut être comparé au langage assembleur. Très peu utilisé par les automaticiens.

L'opérateur indique le type d'opération à effectuer entre le résultat courant et l'opérande. Le résultat de l'opération est stocké à son tour dans le résultat courant.

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

II.2.2.b. Texte structuré ou Structured Text (ST)

Le **ST** est un langage textuel de haut niveau de type "informatique" permettant l'écriture structurée (algorithmes) de traitements logiques et numériques. Il est de même nature que le Pascal. Peu utilisé par les automaticiens.

C'est un langage qui peut être utiliser pour la programmation des actions dans les étapes et des conditions associées aux transitions du langage SFC.

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

- ***Les types d'énoncés standard sont :***
  - ✓ Assignation (variable **:=** expression;) ;
  - ✓ Appel de fonction ;
  - ✓ Appel de bloc fonctionnel ;
  - ✓ Enoncés de sélection (if, then, else, case) ;
  - ✓ Enoncés d'itération (for, while, repeat) ;
  - ✓ Enoncés de contrôle (return, exit) ;
  - ✓ Opérateurs booléens (not, and, or, xor) ;
  - ✓ Enoncés spéciaux pour le lien avec le langage sfc.

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

- ✓ Ne pas écrire plusieurs énoncés sur la même ligne ;
- ✓ Utiliser les tabulations pour indenter les structures de contrôle ;
- ✓ Insérer des commentaires.

Le langage liste d'instruction permet de transcrire sous forme de liste :

- ✓ Un schéma à contact ;
- ✓ Un logigramme, équations booléennes ;
- ✓ Un GRAFCET.
  - ✓ Il réalise aussi des fonctions d'automatisme telles que temporisation, comptage, pas à pas ...

Instructions de test : Instructions d'entrée	
Désignation	**Fonctions**
LD	Le résultat est égal à l'opérande (load : lire la valeur).
LDN	Le résultat est égal à l'inverse de l'opérande (contact ouverture).
AND	ET logique entre le résultat et précèdent et l'état de l'opérande.
ANDN	ET logique entre le résultat et précèdent et l'état inverse de l'opérande.
OR	OU logique entre le résultat et précèdent et l'état de l'opérande.
ORN	OU logique entre le résultat et précèdent et l'état inverse de l'opérande.
XOR, XORN	OU exclusif.

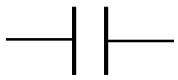
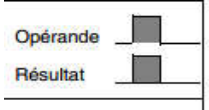
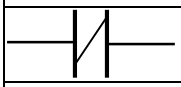
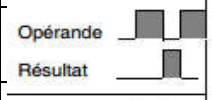
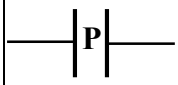
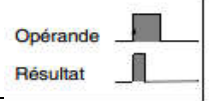
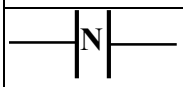
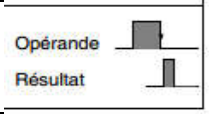
Instructions de test : Instructions d'action	
Désignation	**Fonctions**
ST	L'opérande associé prend la valeur de la zone de test.
STN	L'opérande associé prend la valeur inverse de la zone de test.
S	L'opérande associé est mis à 1 lorsque le résultat de la zone de test est à 1.
R	L'opérande associé est mis à 1 lorsque le résultat de la zone de test est à 1.

II.3 Instructions de base en langages : LD, IL et ST

Les instructions booléennes et les blocs fonctions ont des représentations différentes suivant le langage.

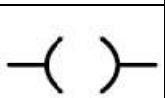
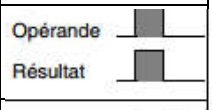
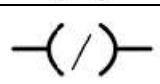
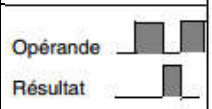
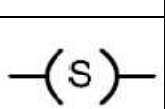
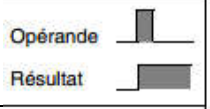
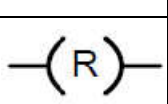
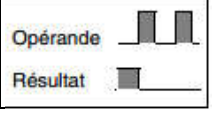
II.3.1 Instructions de chargement

Le tableau suivant décrit le rôle de chacune des instructions de chargement en langage LD, IL et ST.

Langage LD	Langage IL	Langage ST	Fonction	Chronogramme
	**LD**	**:=**	Contact à fermeture : résultat **1**, quand l'objet bit qui le pilote est à **1.**	
	**LDN**	**:=NOT**	Contact à ouverture : résultat **1**, quand l'objet bit qui le pilote est à **0.**	
	**LDR**	**:=RE**	Contact à front montant : détection du passage de **0 à 1**de l'objet bit qui le la mise à **1** du résultat s'effectue pendant 1 cycle.	
	**LDF**	**:= FE**	Contact à front descendant : détection, du passage de **1 à 0** de l'objet bit qui le la mise à **1** du résultat s'effectue pendant 1 cycle.	

II.3.2 Instructions d'affectation

Le tableau suivant décrit le rôle de chacune des instructions d'affectation en langage LD, IL et ST.

Langage LD	Langage IL	Langage ST	Fonction	Chronogramme
	**ST**	**:=**	aux bobines directes : l'objet bit associé prend la valeur du résultat de l'équation **.**	
	**STN**	**:=NOT**	aux bobines inverses : l'objet bit associé prend la valeur de l'inverse du résultat de l'équation.	
	**S**	**SET**	aux bobines à enclenchement : l'objet bit associé est mis à **1** lorsque le résultat de l'équation est à **1**.	
	**R**	**RESET**	aux bobines à déclenchement : l'objet bit associé est mis à **0** lorsque le résultat de l'équation est à **1**.	

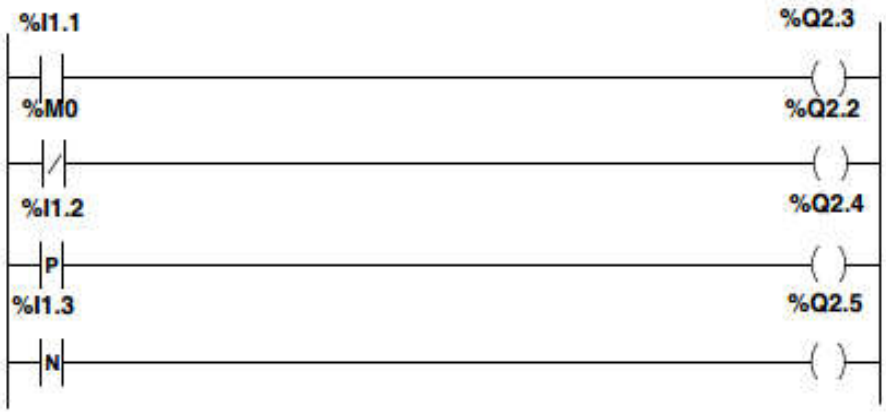
II.4 Exemples et problèmes de programmations

II.4.1 Exemples

II.4.1.a. Exemple 1 : Programmation en LD, IL et ST

Exemple en langage Ladder LD :

- Traduisez le schéma ci-dessous en langage liste d'instruction et texte structuré :



➢ ***Exemple en langage liste d'instructions IL***

N° de ligne	Instruction	Opérande
00	LD	% I1.1
01	ST	% Q2.3
02	LDN	% M0
03	ST	% Q2.2
04	LDR	% I1.2
05	ST	% Q2.4
06	LDF	% I1.3
07	ST	% Q2.5

➢ ***Exemple en langage littéral structuré ST***

% Q2.3 := % I1.1

% Q2.2 :=NOT % M0

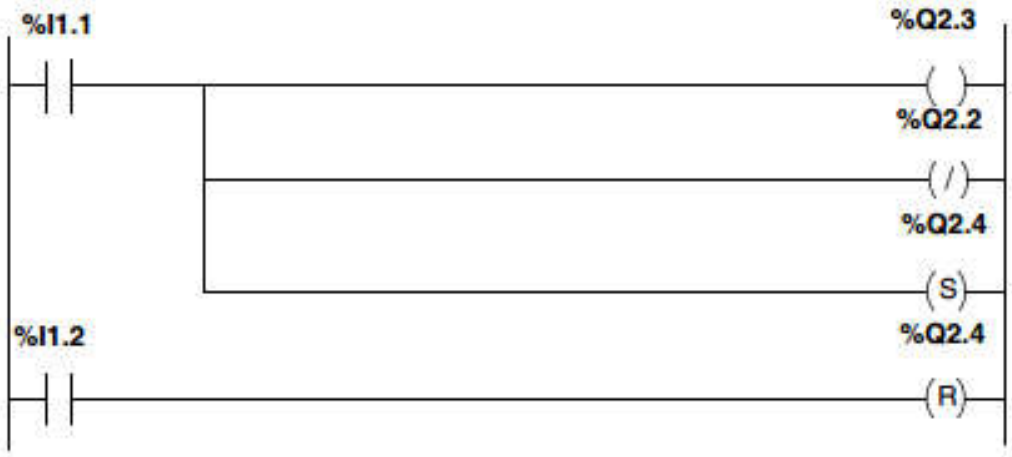
% Q2.4 := RE % I1.2

% Q2.5 := FE % I1.3

II.4.1.b. Exemple 2 : Programmation en LD, IL et ST

➢ ***Exemple en langage Ladder LD :***

- Traduisez le schéma ci-dessous en langage liste d'instruction (PL7) :



➢ ***Exemple en langage liste d'instructions IL***

N° de ligne	Instruction	Opérande
00	LD	% I1.1
01	ST	% Q2.3
02	STN	% Q2.2
03	S	% Q2.4
04	LD	% I1.2
05	R	% Q2.4

➢ ***Exemple en langage littéral structuré ST***

```
% Q2.3 := % I1.1;

% Q2.2 :=NOT   % I1.1;

IF % I1.1   THEN

     SET  % Q2.4
END_IF;

IF % I1.2   THEN

     RESET  % Q2.4;

END_IF;
```

II.4.1.c. Exemple 2 : Programmation de temporisation

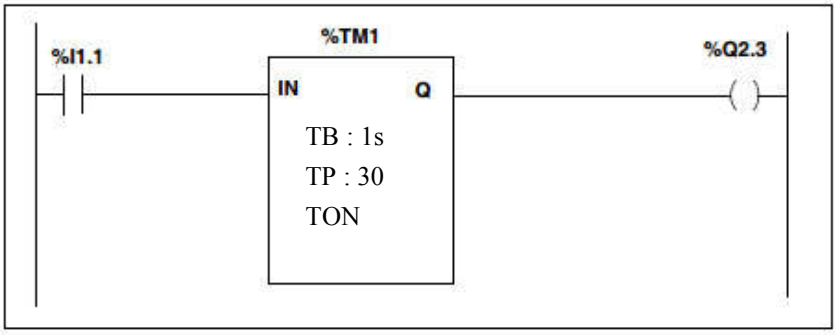
➢ ***Cahier des charges :***

La programmation des blocs fonction temporisateur est identique quel que soit le mode d'utilisation sélectionné.

Le choix du fonctionnement TMi , TON, TB et TP s'effectue dans l'éditeur de variables.

Mode	TON
TB	1s,
TP	30
%TMi	1

➢ ***Exemple en langage Ladder LD***



➢ ***Exemple en langage liste d'instructions IL***

N° de ligne	Instruction	Opérande
00	LD	% I1.1
01	IN	% TM1
02	LD	% TM1.Q
03	ST	% Q2.3

➢ ***Exemple en langage littéral structuré ST***

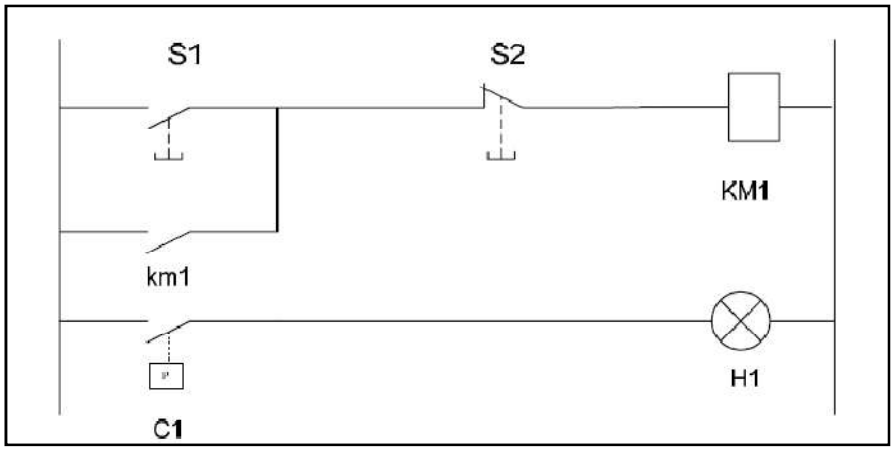
```
IF   RE  % I1.1   THEN
     START   % TM1;
ELSIF  FE % I1.1   THEN
DOWN  % TM1;
     END_IF;
% Q2.3 := % % TM1.Q
```

- L'instruction START %TMi, génère un front montant sur l'entrée IN du bloc temporisateur.
- L'instruction DOWN %TMi, génère un front descendant sur l'entrée IN du bloc temporisateur.

II.4.1.d. Exemple 4 : Commande d'une pompe

➢ ***Cahier des charges :***

- Traduisez le schéma ci-dessous en langage liste d'instruction (PL7) :



➢ ***Solution*** :

Soit le tableau des entrées et sortie avec l'adressage API :

Elément d'entrée	désignation	Adressage API	Elément de sortie	désignation	Adressage API
S1	Bouton d'arrêt	% I1.0	KM1	Contacteur de pompe	% Q2.0
S2	Bouton Marche	% I1.1	H1	Voyant de pression	% Q2.1
C1	Capteur de pression	% I1.2			

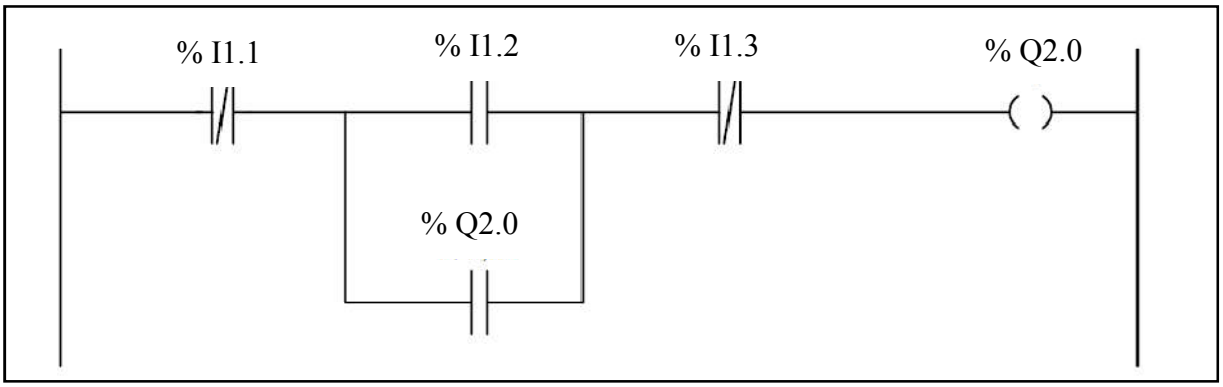
➢ ***Programme en langage IL du schéma de commande de la pompe***

N° de ligne	Instruction	Opérande	Commentaire
00	LD	% I1.0	Tester le bouton marche S1
01	OR	% Q2.0	Exécuter un OU avec KM1
02	AND	% I1.1	Exécuter un ET avec Le bouton marche S2
03	ST	% Q2.0	Activer la sortie du contacteur KM1 (Q2.0)
04	LD	% I1.2	Tester la capteur de pression C1
05	ST	% Q2.1	Activer la sortie voyant H1

II.4.1.e. Exemple 5 : Transcription du langage Ladder en langage IL

➢ ***Cahier des charges :***

- A partir du programme Ladder ci-dessous déterminer la liste des instructions.



➢ ***Programme en langage IL***

N° de ligne	Instruction	Opérande	Commentaire
00	LDN	% I1.1	Lire l'entrée inverse
01	AND(	% I1.2	Exécuter un ET , On imbrique une parenthèse.
02	OR	% Q2.0	Exécuter un OU avec la ligne précédente
03	)		On ferme la parenthèse
04	ANDN	% I1.3	Exécuter un NON ET
05	ST	% Q2.0	Activer la sortie

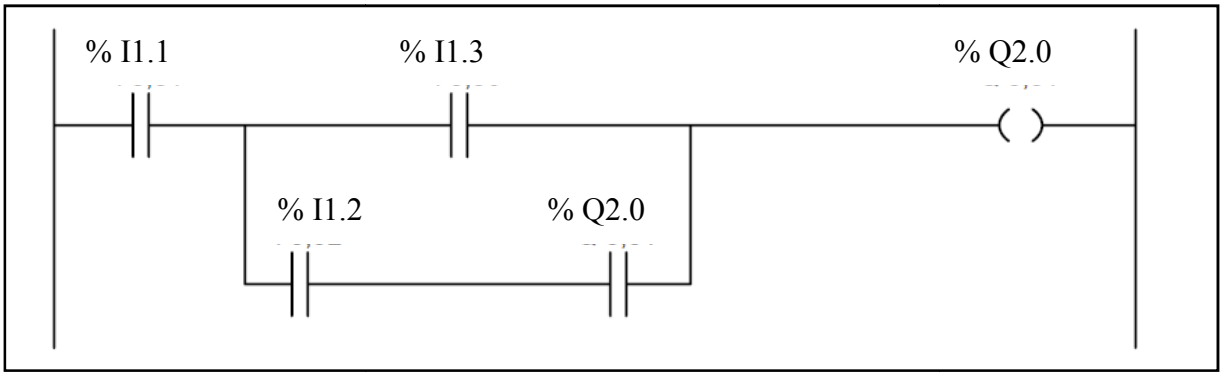
II.4.1.f. Exemple 6 : Transcription du langage IL vers le langage LD

➢ ***Cahier des charges :***

- A partir de la liste des instructions du programme Ladder ci-dessous déterminer.

N° de ligne	Instruction	Opérande	Commentaire
00	LD	% I1.1	Lire l'entrée
01	AND(	% I1.3	Exécuter un ET , On imbrique une parenthèse.
02	OR(	% I1.2	Exécuter un OU avec la ligne précédente
03	AND	% Q2.0	Exécuter un NON ET
04	)		On ferme la parenthèse
05	)		On ferme la parenthèse
06	ST	% Q2.0	Activer la sortie

➢ ***Schéma en langage LD de la transcription du programme IL .***



II.4.2 Problèmes

II.4.2.a. Problème 1 : Démarrage d'un moteur asynchrone

➢ ***Cahier des charges :***

On se proposer de programmé le démarrage d'un moteur asynchrone un seul sens de rotation, dont le circuit de commande est donné par la Figure (II. 2).

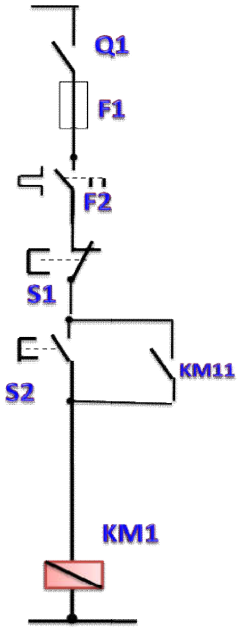


Fig II. 2 : Circuit de commande (logique câblée).

- ✓ Les entrées du système sont : S1 son adresse API est %I1.0
  S2 son adresse API est %I1.1
- ✓ La sortie du système est : KM1 son adresse API est %Q2.0Le contact auxiliaire KM11 d'auto maintien sera pris comme bit mémoire (image) de la sortie KM1, son adresse API est %Q2.0, Le fusible ainsi que le contact Q du sectionneur, le contact du relais thermique ne seront pas utilisés comme des entrées directes, mais ils seront câblés en série avec le commun de l'alimentation de l'interface de sortie.

- ***Programmation en langage à contact ou Ladder***

Le réseau à contact s'inscrit entre deux barres verticales représentant la tension d'alimentation
Exemple de schéma à contact programmable (sous millenium), Figure (II. 3).

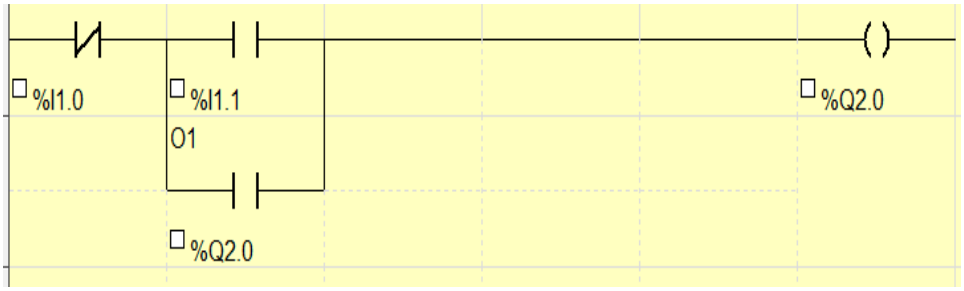


Fig II. 3 : Programme en langage LD du circuit de commande du moteur asynchrone.

- ***Programmation en langage liste d'instructions***

L'adresse ou le code opérande est précédé de %.

Exemple d'écriture **N° de ligne**	**Instruction**	**Opérande**	**Commentaire**
00	LDN	% I1.0	tester le bouton S1 (entrée d'adresse %I1.1)
01	AND(	% I1.1	Exécuter un ET avec Le bouton marche S2 (%I1.1) on imbrique une parenthèse .
02		% Q2.0	Exécuter un OU avec KM (mémoire Q2.0)
03			En ferme la parenthèse
04		% O 2.0	Activer la sortie du contacteur KM1 (Q2.0)

.

- ***Programmation en langage FBD***

Il est similaire aux circuits électroniques à base de portes logiques, Figure (II. 4).

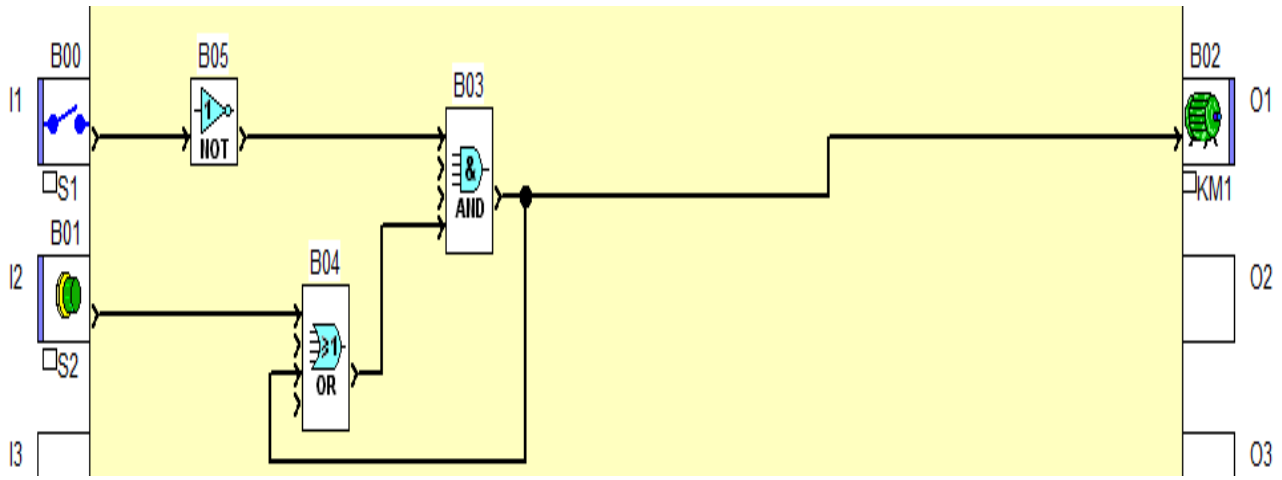


Fig II. 4 : Programme en langage FBD du circuit de commande du moteur asynchrone.

II.4.2.b. Problème 2 : Etude et commande par API d'un poste de perçage

➢ ***Cahier des charges***

On se propose de programmé, sous langage SFC, la solution GRAFCET d'un système automatisé « Poste de perçage », Figure (II. 5)

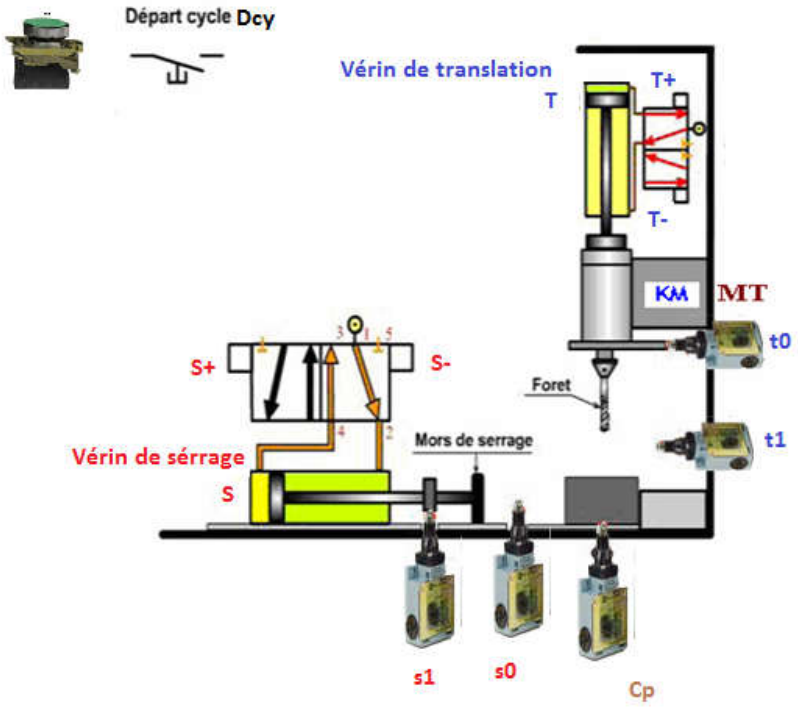


Fig II. 5 : Système de perçage automatique.

➢ ***Fonctionnement :***

L'appui sur le bouton Départ cycle (Dcy) lance le cycle suivant :

- ✓ Serrage de la pièce à percer ;
- ✓ Perçage de la pièce ;
- ✓ Desserrage de la pièce.

▪ ***Remarque :***

Le cycle ne peut être lancé que si la pièce à percer est présente, à l'arrêt les fins de courses **s0** et **t0** sont actionnés.

➢ ***Fonctionnement en tenant compte des solutions technologiques utilisées :***

- Le capteur (**Cp**) indique la présence de la pièce à percer;
- L'appui sur le bouton Départ cycle (**Dcy**) lance le cycle ;
- Le vérin de serrage (**S**) déplace la pièce pour la serrer; le capteur (**s1**) indique que la pièce
- est serrée ;
- Le moteur supportant le foret (**MT**) commence à tourner et le vérin (**T**) pousse le moteur
- vers le bas ;
- L'action sur le capteur (**t1**) indique que la pièce est percée, d'où la remonté du vérin (**P**) ;

- Quand le capteur (**t0**) est actionné, le moteur (**MT**) et le vérin (**T**) sont arrêtés , par contre Le vérin (**S**) retourne dans l'autre sens ;
- L'action sur le capteur (**s0**) indique que la pièce est desserrée, et le cycle est terminé (on revient à l'état initial).
- Afin d'améliorer le fonctionnement on doit prévoir le fonctionnement cycle par cycle (Cy/cy)et cycle continu ou automatique (AQ), ainsi qu'un bouton d'initialisation de l'étape initiale.

On demande en premier lieu le GRAFCET du point de commande et par la suite la programmation de la solution trouvée en langage SFC.

➢ ***Solution graphique par GRAFCET***

La solution GRAFCET proposée, Figure (II.6) :

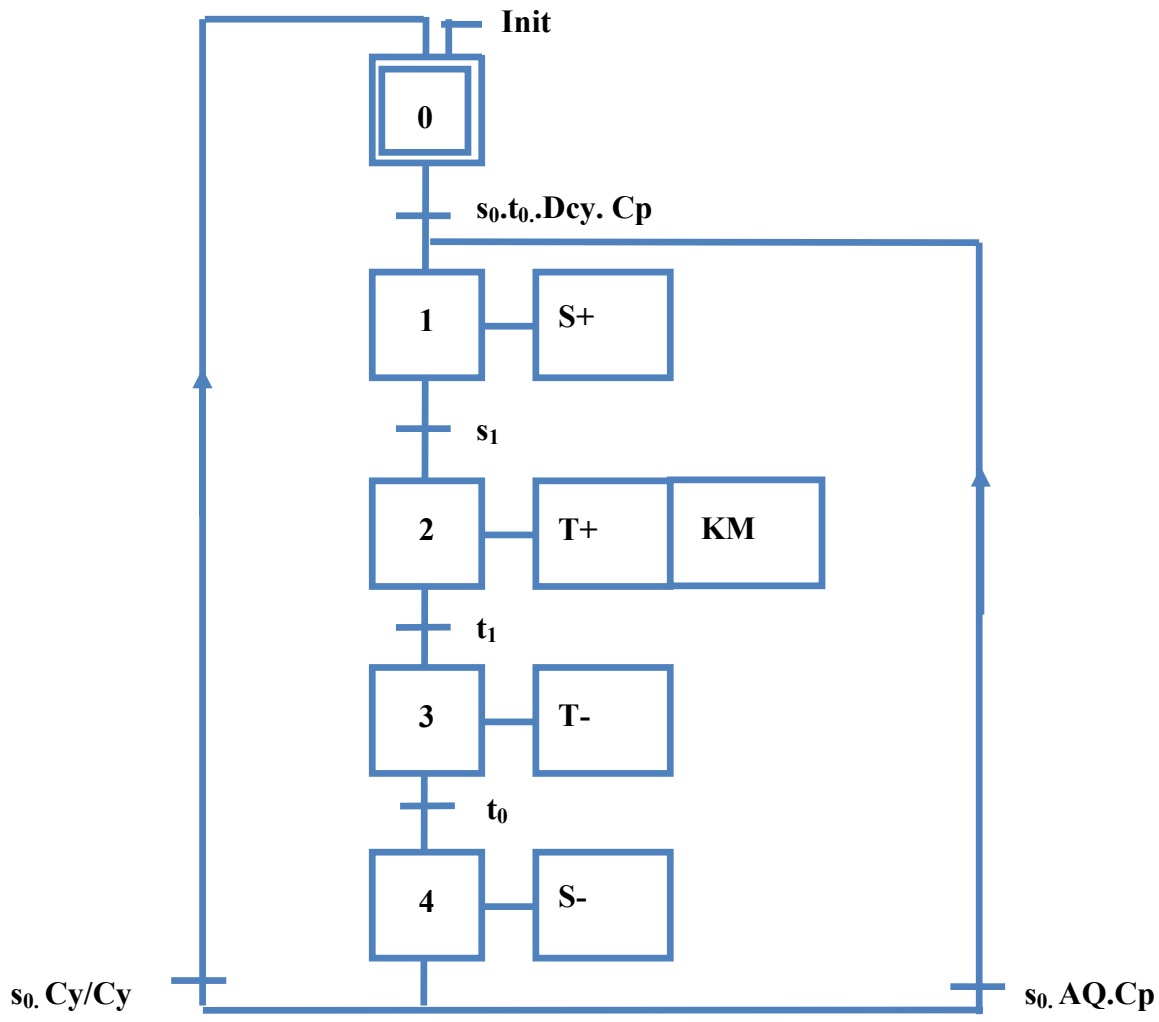


Fig II. 6 : Solution GRAFCET du système de perçage automatique.

- ***Programmation de la solution en langage SFC***

La Figure (II.7) illustre la programmation de la solution GRAFCET proposée en langage SFC sous le logiciel Millénium de API Crouzet,

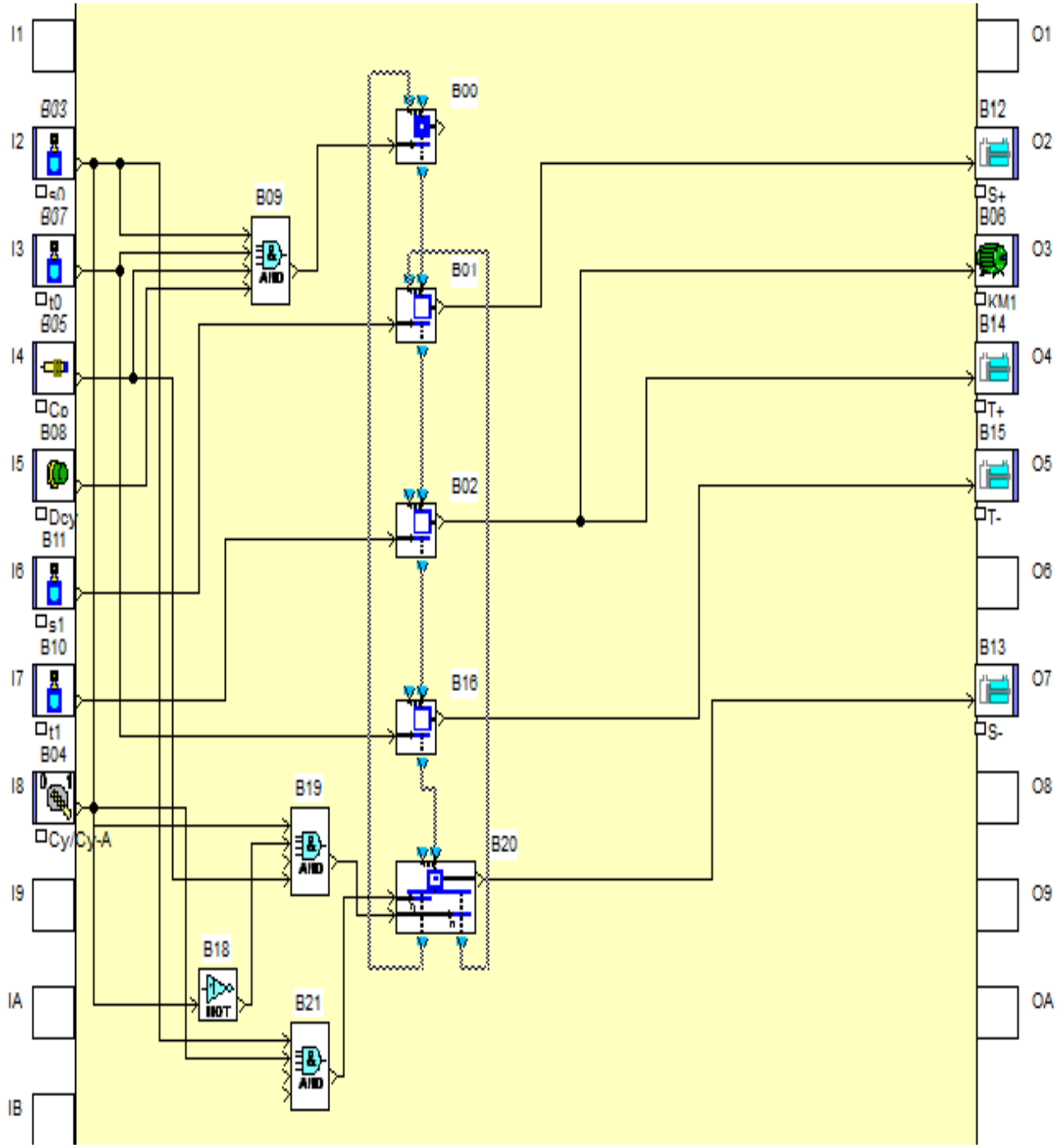
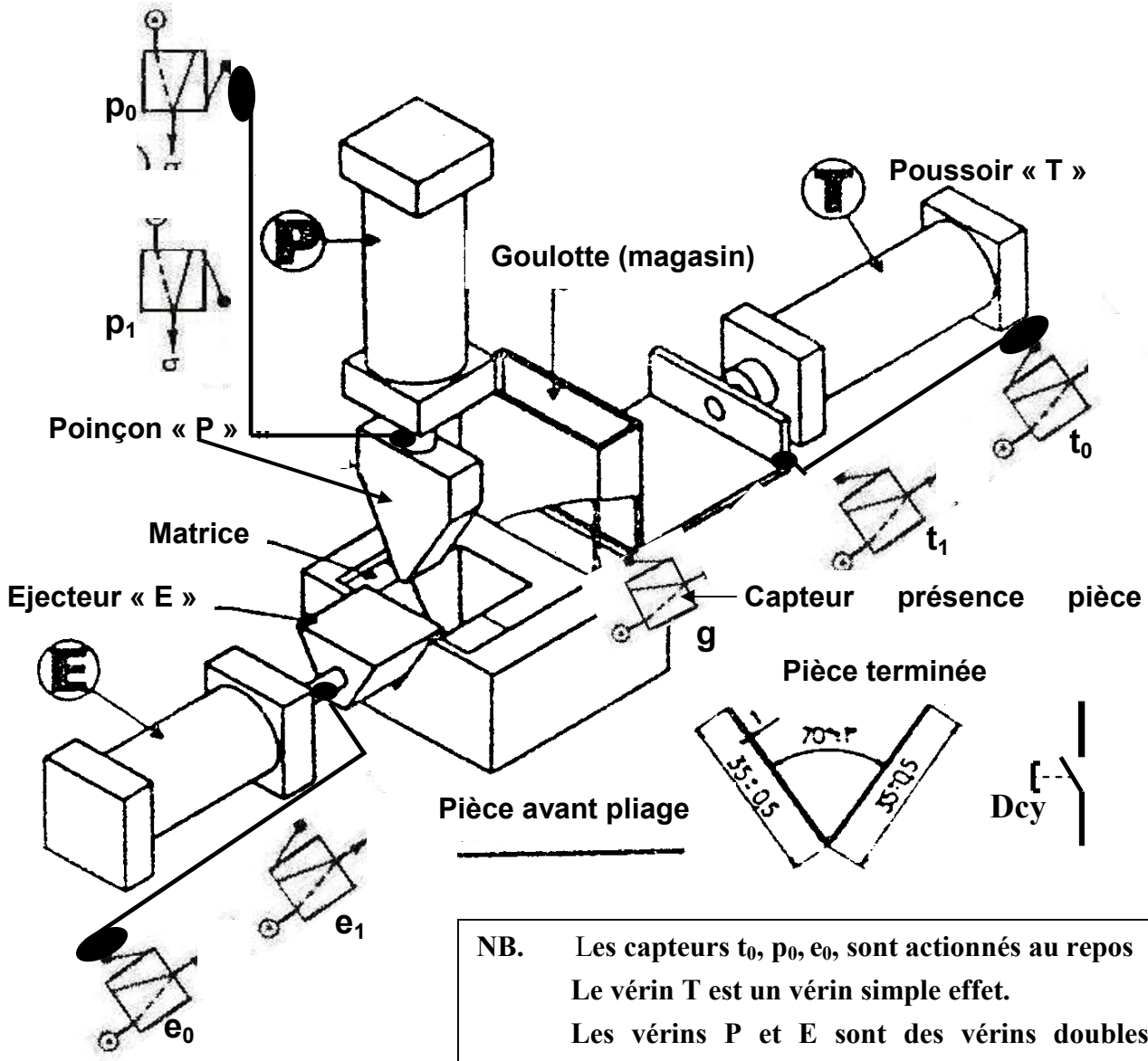


Fig II. 7 : Programmation en SFC de la Solution GRAFCET du système de perçage automatique.

II.4.2.c. Problème 3 : Etude et commande par API d'un appareil à plier les tôles

➢ ***Cahier des charges***



NB. Les capteurs t_0, p_0, e_0, sont actionnés au repos
Le vérin T est un vérin simple effet.
Les vérins P et E sont des vérins doubles effets.

Cet appareil sert à plier des tôles.

Des tôles non pliées sont empilées dans une goulotte (magasin). Un capteur **(g)**, détecte la présence de tôles dans la goulotte.

➢ ***Fonctionnement :***

Lorsque l'opérateur actionne le bouton poussoir **(Dcy)**, avec présence de pièce **(g),** Les actions suivantes se produisent :

- Sortie de la tige du vérin **T** (Le poussoir **(T)** pousse la tôle sous le poinçon).
- Action sur t_1, la tige du vérin **P** sort pour plier la tôle.

- Action $\mathbf{p_1}$, les tiges des vérins **P** et **T** rentrent.
- Action sur $\mathbf{t_0}$ et $\mathbf{p_0}$, la tige du vérin **E** sort pour éjecter la tôle pliée.
- Action sur $\mathbf{e_1}$, la tige du vérin **E** rentre.
- Action sur $\mathbf{e_0}$, le cycle se termine.

➢ ***Travail demandé :***

a) Complétez le tableau des entrée et sorties avec les adresse API correspondantes, Donnez le GRAFCET du point commande et du point de vue automate;

b) Ecrire les équations de sorties (préactionneurs),;

c) Programmez le GRAFCET (chart) , les réceptivités et les actions (programmation postérieur), (Remarque pas de programmation préliminaire) ,

d) Complétez le schéma de câblage de l'automate.

a) Tableau Affectation des entrées / sorties

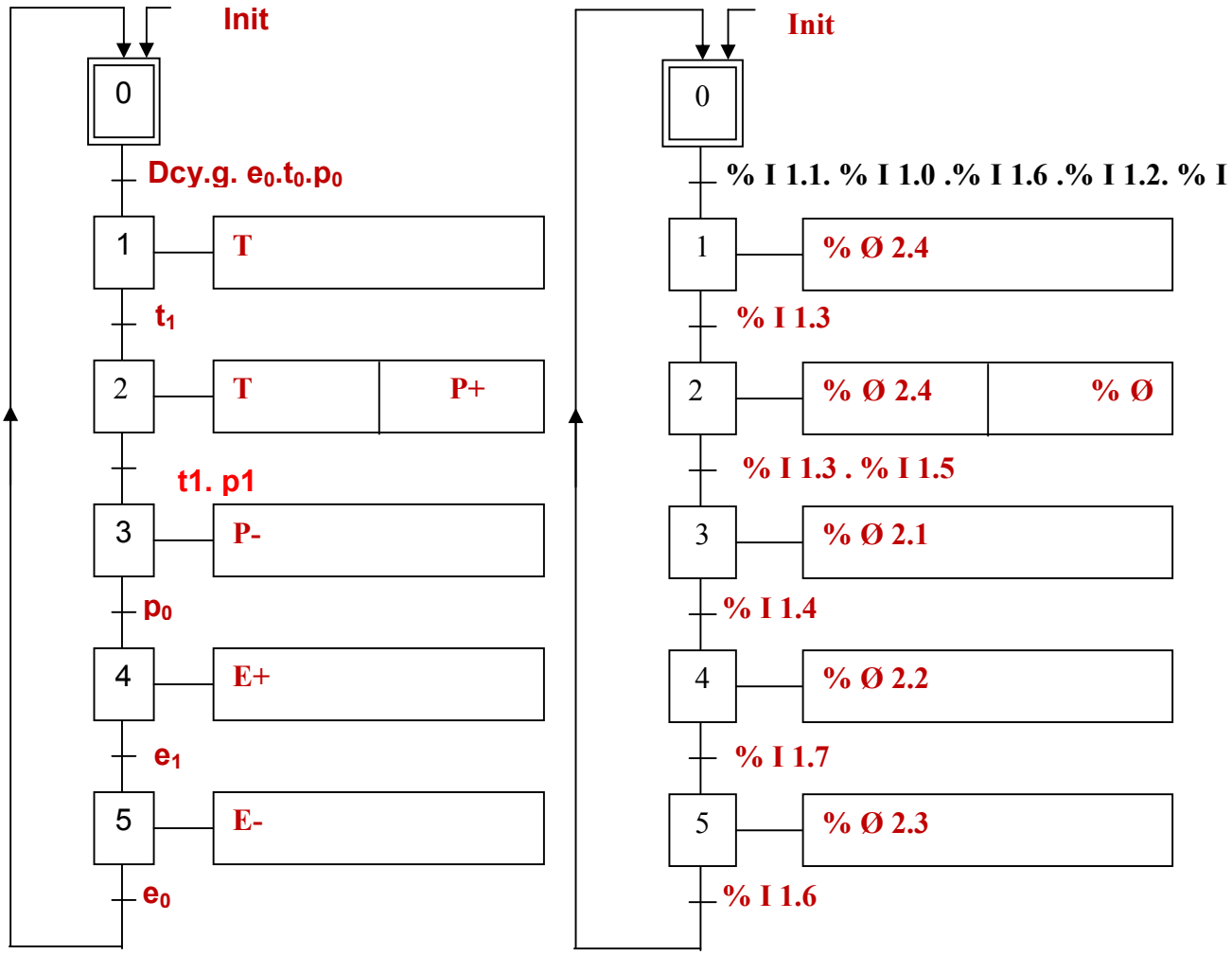
Désignation	Repère	Affectation adresse API	
Présence pièce	**g**		Boutons Poussoirs et capteurs
Bouton Poussoir départ cycle	**Dcy**		
Fin de course rentrée de tige vérin T	$\mathbf{t_0}$		
Fin de course sortie tige vérin T	$\mathbf{t_1}$		
Fin de course rentrée tige vérin P	$\mathbf{p_0}$		
Fin de course sortie tige vérin P	$\mathbf{p_1}$		
Fin de course rentrée tige vérin E	$\mathbf{e_0}$		
Fin de course sortie tige vérin E	$\mathbf{e_1}$		
P : Vérin de pliage de la tôle	**P+**		Pré actionneurs
	P-		
E : Vérin éjecteur de la tôle pliée	**E+**		
	E-		
T : Vérin de translation de la tôle	**T**		

➢ ***Solution graphique par GRAFCET***

a) Tableau d'adressage des entrées et sorties de l'API.

Désignation	Repère	Affectation adresse API	
Présence pièce	g	% I 1.0	Boutons Poussoirs et capteurs
Bouton Poussoir départ cycle	Dcy	% I 1.1	
Fin de course rentrée de tige vérin T	t_0	% I 1.2	
Fin de course sortie tige vérin T	t_1	% I 1.3	
Fin de course rentrée tige vérin P	p_0	% I 1.4	
Fin de course sortie tige vérin P	p_1	% I 1.5	
Fin de course rentrée tige vérin E	e_0	% I 1.6	
Fin de course sortie tige vérin E	e_1	% I 1.7	
P : Vérin de pliage de la tôle	P+	%Q 2.0	Pré actionneurs
	P-	%Q 2.1	
E : Vérin éjecteur de la tôle pliée	E+	%Q 2.2	
	E-	%Q 2.3	
T : Vérin de translation de la tôle	T	%Q 2.4	

b) GRAFCET point de vue partie commande (PC) : c) GRAFCET point de vue API



d) Ecrire les équations des sorties :

P+ .= %X2 **P-** = %X3

E+ .= %X4 **E-** = %X5

T .= %X1 + %X2

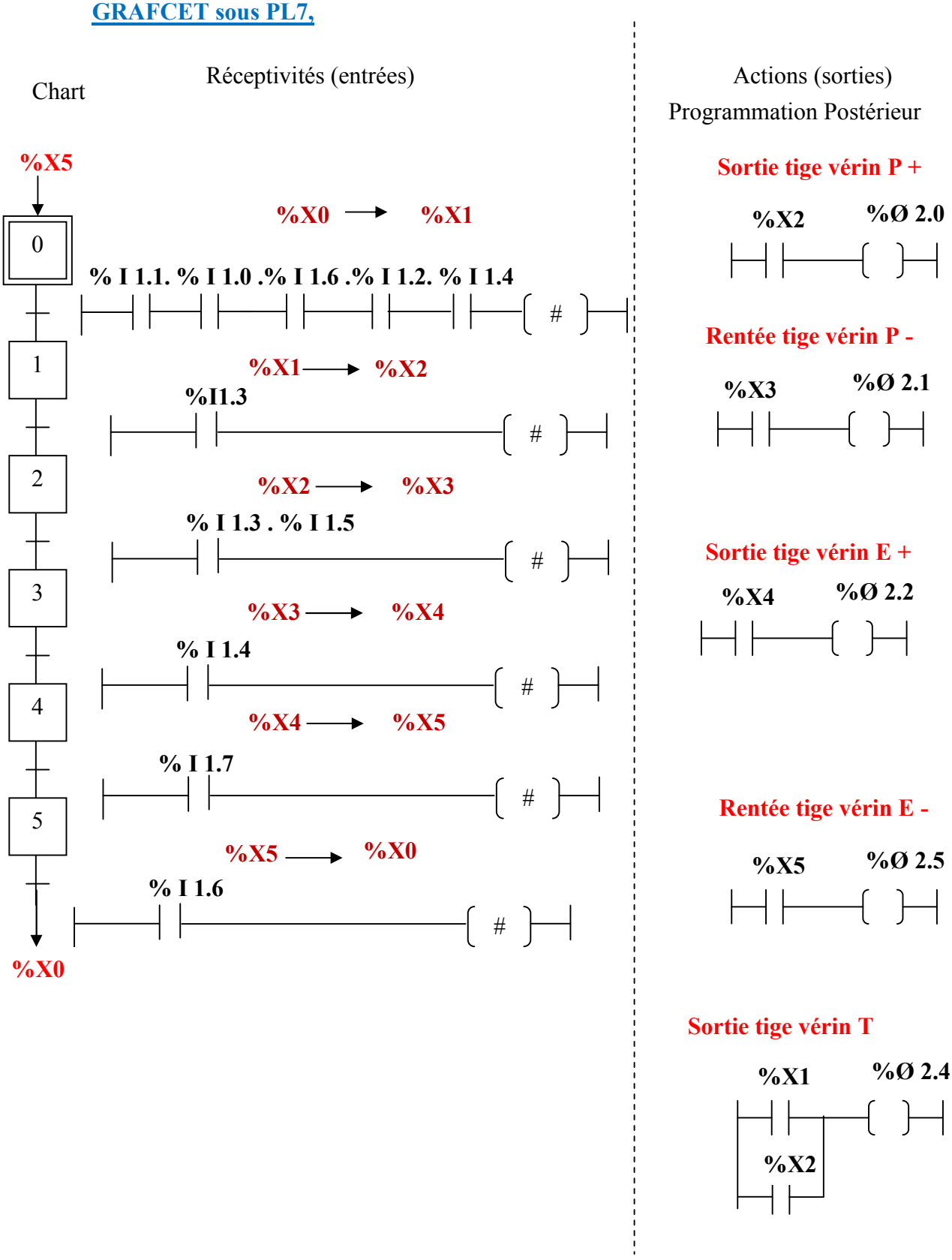
On peut aussi écrire directement les sortie selon leurs adresse API

% Ø 2.0 .= %X2 **% Ø 2.1** = %X3

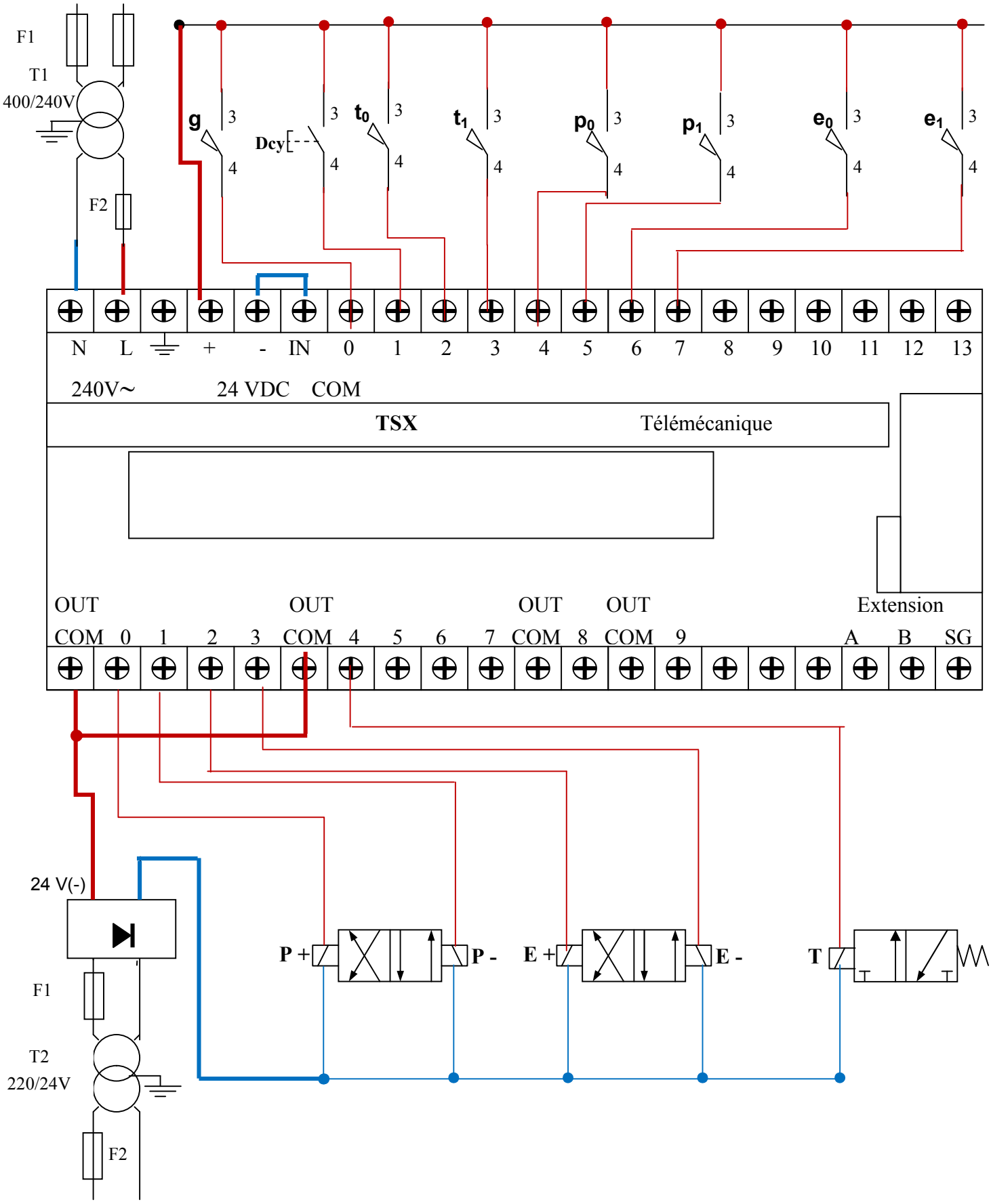
% Ø 2.2 = %X4 **% Ø 2.3** = %X5

% Ø 2.4 = %X1 + %X2

e) Document réponse programmation de réceptivités et action en langage GRAFCET sous PL7,



e) Schéma de câblage de l'API en logique positive



II.5 Annexe 1 : Adressage des Objets Bits et Mots

PRESENTATION GENERALE

QU'EST QU'UN OBJET

Un objet est une entité pouvant être manipulée par programme, ce peut être une image d'entrée, un élément d'un temporisateur, un élément du système, un élément de communication etc.

REPRESENTATION DES OBJETS

Les objets sont représentés par le symbole « % » suivit d'une ou deux lettres précisant leurs type puis d'une lettre précisant leur format (bits, octet, mots , double, réel etc.)

Exemples :	**%IW**	% : objet	**I**	: Type image d'entrée	**W**	: Format mot
	%MB	% : objet	**M**	: Type mémoire interne	**B**	: Format octet
	%Q ou %Qx	% : objet	**Q**	: Type Image de sortie	**X**	: Format bit

LISTE DES DIFFERENTS TYPES D'OBJETS

♦ OBJETS D'ENTREES	**%I**	*Images des entrées process*
♦ OBJETS DE SORTIES	**%Q**	*Images des sorties process*
♦ VARIABLES INTERNES	**%M**	*Mémoire utilisateur*
♦ VARIABLES GRACETS	**%X**	
♦ CONSTANTES	**%K**	*Mémoire constante ou de configurations*
♦ VARIABLES SYSTEME	**%S**	*Etats ou actions sur le système*
♦ VARIABLES RESEAUX (FIPWAY)	**%N**	*Mots communs échangés automatiquement*

Les types des objets des blocs fonctions prédéfinis (Pas de précision de format)

♦ TEMPORISATEURS	**%TM**	*Repos, travail*
♦ TEMPORISATEURS SERIE 7	**%T**	*Compatible série 7*
♦ MONOSTABLES	**%MN**	*Monostables « retriggerables »*
♦ COMPTEURS	**%C**	*Comptage, décomptage*
♦ REGISTRES	**%R**	*Pile FIFO ou LIFO*
♦ PROGRAMMATEURS tambour	**%DR**	Programmateurs cycliques à

Les types DFB (Pas de précision de format)

LES FORMATS DES OBJETS

BITS	**X** ou rien	0 - 1
OCTETS	**B**	Code ASCII uniquement
MOTS	**W**	16 bits signé ou pas
MOTS DOUBLES	**D**	32 bits signé
FLOTTANT	**F**	-3.402824E+38 et -1.175494E-38 et 1.175494E-38 et 3.402824E+38

15/77

LES ELEMENTS DES OBJETS

La plupart des objets sont constitués de plusieurs éléments ou sous éléments, par exemple, un compteur contient une valeur courante, une valeur de présélection, deux bits de dépassements et un bit de présélection atteinte

Les cartes d'entrées TOR disposent des bits représentant les états physiques des capteurs, mais aussi d'un bit de défaut du module et un bit de défaut par voie, certaines cartes disposent aussi de mots ou de tableaux de mots représentant leurs états ou leurs configurations.

On accède à un élément d'un objet en mettant un point derrière le numéro de l'objet puis l'identificateur de l'élément (un numéro ou un symbole réservé), puis si l'élément contient d'autre sous éléments on ajoute un point suivi de l'Identificateur etc.

Exemples :	**%I0.1**	Module à l'emplacement **0**, la voie **1**
	%I0.1.ERR	Module à l'emplacement **0**, la voie**1 défaut module**
	%C5.V	Le compteur **5**, la valeur courante
	%X3	L'étape **3**
	%X3.T	L'étape **3**, sont temps d'activité
	%IW0.12.2	Module à l'emplacement **0**, voie **12**, mot **2**

OBJETS DES CARTES

Pour identifier un objet d'une carte, il faut préciser son numéro de module puis le numéro de voie.
Si plusieurs objets sont associés à une voie, il faudra préciser le numéro d'objet. (sauf pour l'objet N°0)

Exemples:	**%IW0.12.2**	Mot **2** de la voie **12** du module **0**
	%IW0.12	Mot **0** de la voie **12** du module **0**

LES OBJETS STRUCTURES

(bits extraits de mots, tableaux de bits, tableaux de mots)

BIT EXTRAIT DE MOTS

mot : Xi		i : numéro du bit
Exemples :	**%MW5 :X4**	Le mot interne **5**, le bit **4**
	%IW0.12.2 :X3	Le module **0**, la voie **12**, le registre **2**, le bit **3**

TABLEAUX DE BITS OU CHAINES DE BITS

bit de départ : nombre		nombre compris entre 1 et 32
Exemples :	**%I0.0 :16**	Module **0**, voie **0**, **16** bits (voie 0 à 15)
	%M5 :32	Bit **5**, **32** bits (%M5 à %M36)

TABLEAU DE MOTS

mot de départ : nombre		nombre compris entre 1 et maximum mémoire
Exemples :	**%MW10 :50**	Mot interne **10**, **50** mots (%MW10 à %MW59)
	%KW25 :100	Mot constant **25**, **100** mots (%KW25 à %KW124)
	%MD20 :10	Mot double **20**, **10** mots (%MD20 a % MD38)