CHAPITRE IV

I. PRINCIPE D'UN SYSTÈME AUTOMATISÉ

Un système automatisé est un moyen d'assurer l'objectif primordial d'une entreprise et à la compétitivité de leurs produits. Il permet d'ajouter une valeur aux produits entrants.

I.1 Structure des systèmes automatisés de production (SAP) :

Tout système automatisé de production comporte 3 parties :

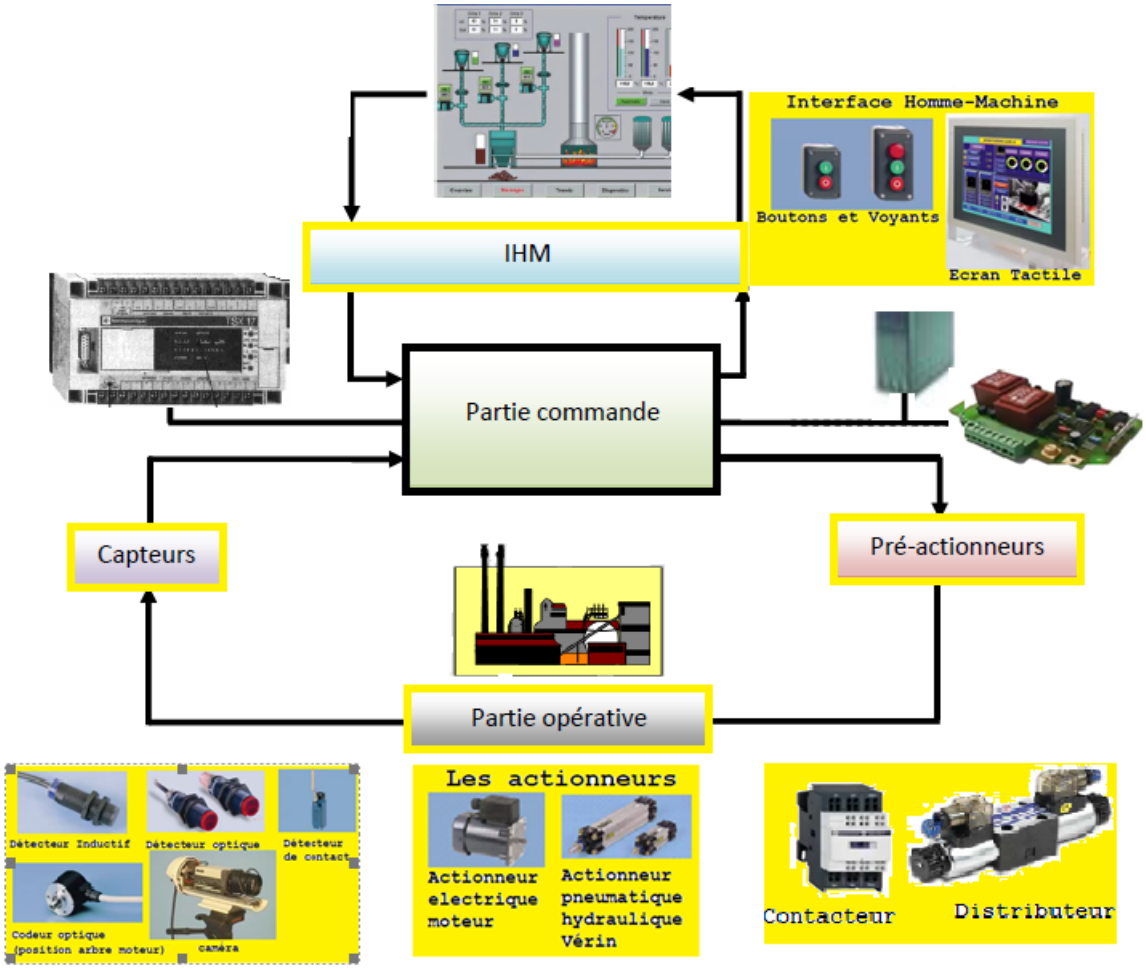


Figure 1 : Structure d'un système automatisé

- ✓ **Une partie opérative (P.O)** procédant au traitement des matières d'œuvre afin d'élaborer la valeur ajoutée ; c'est la partie mécanique du système qui effectue les opérations. Elle est constituée d'actionneurs (moteur électrique, vérins pneumatique et/ou hydraulique, …) en utilisant de l'énergie électrique, pneumatique, hydraulique...
- ✓ **Une partie interface (P.I)** : est la partie se trouvant entre les deux faces PO et PC Traduisant les ordres et les informations. elle permet la communication entre l'automate et l'opérateur Exemple : Retour de défauts, d'informations sur l'état de la machine (températures, vitesses), l'état actuel du processus (démarrage,

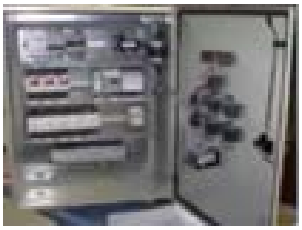
remplissage…) Envoi de consignes : marche, arrêt, consigne de vitesse, température pour un four…

- ✓ **Une partie commande (P.C.)** coordonnant la succession des actions sur la partie opérative avec la finalité d'obtenir cette valeur ajoutée. Elle comprend essentiellement les capteurs, les boutons de commande (bouton poussoir, bouton d'arrêt d'urgence), les préactionneurs…

I.2 technologies de la partie commande (PC)

La réalisation de la partie commande (PC) fait appel à diverses technologies dont les plus couramment utilisées sont :

- ✓ **Commande câblée :** les relais électromécaniques les relais statiques électroniques les relais pneumatiques, A partir d'une certaine, complexité, les relais électromécaniques et les relais statiques deviennent lourds à mettre en œuvre et le cout de l'automatisation est difficile à estimer, Peu souple, pas d'intégrations d'éléments communiquant

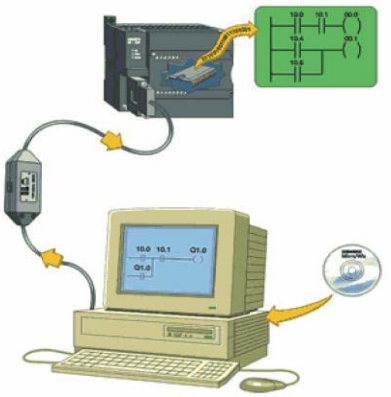


- ✓ **Microcontrôleur :** l'automate programmable c'est une carte électronique à base d'un microcontrôleur, L'automate programmable évite de faire appel à l'ordinateur qui, lui a souvent des performances trop élevées pour le problème à résoudre et demande un personnel spécialisé. Particulièrement bien adaptés aux problèmes de commande séquentielle et d'acquisition des données

- ✓ **Automate programmable industriel (ou API)**

  Les API autorisent la réalisation aisée d'automatismes comprenant de quelques di-zaines jusqu'à plusieurs milliers d'entrées/sorties. En plus en peut faire l'extension

sans changer le câblage de l'installation existante mais en changeons uniquement le programme d'exécution



I.3 Automate programmable industriel (ou API)

1.3.1 Définition

Un automate programmable industriel (ou API) est un dispositif électronique programmable destiné à automatiser des processus tels que la commande de machines au sein d'une usine et à piloter des robots industriels par exemple.

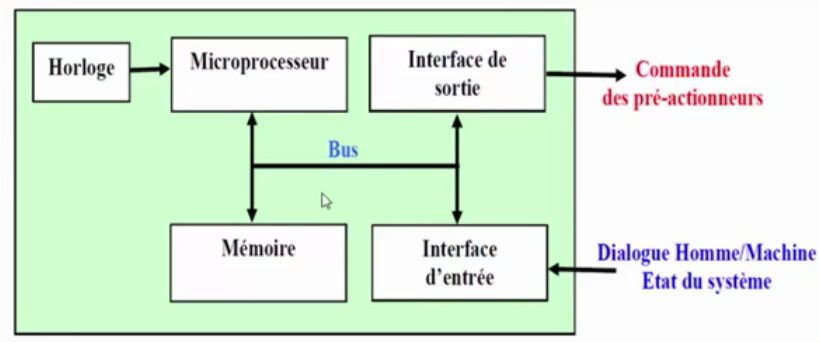
1.3.2 Les avantage

- Puissance-rapidité
- Autonomie-Facilité de maintenance
- Simplification du câblage
- Modification du programme facile par rapport à la logique câblée
- Fiabilité et énorme possibilité d'exploitation

I.3.3 Architecture matérielle d'un API

Un automate programmable industriel se compose :

- D'une unité de traitement (Microprocesseur + Mémoire)
- D'interfaces d'entrées et de sorties
- De modules de communication
- D'un module d'alimentation



1.3.4 Programmation d'API

Le même type d'automate peut être utilisé pour différentes applications, la différence s'effectue avec le programme installé dans celui-ci.
Pour réaliser ces programmes on utilise différents langages en fonction de l'automate, de l'utilisateur et du concepteur.

1.3.4.1 Les différents langages

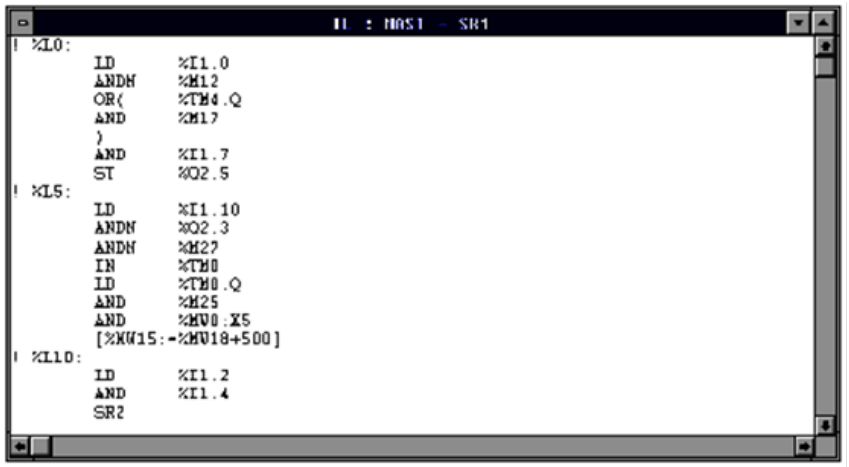
L'écriture d'un programme correspond à l'établissement du cycle d'un système automatique. Cette écriture peut s'effectuer à partir des 5 langages de programmation d'API représenté dans le tableau suivant

IL	ST	FDB	LD	SFC
Instruction List	Structured Text	Function Diagram Block	Ladder diagram	Sequential Function Chart,
Listes d'instructions	Texte structuré	Langage a blocs fonctionnels	Langage a contacts	langage inspiré du GRAFCET

Tableau 1 : les langages de programmation d'API

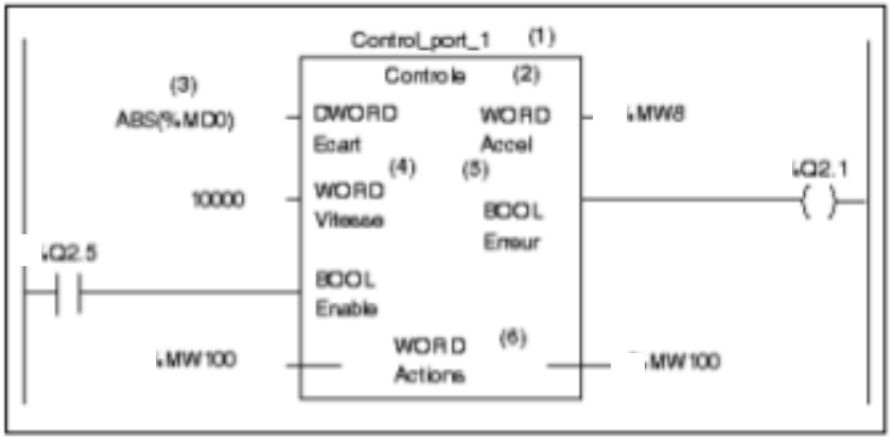
a) **Liste d'instructions ou IL** : ce langage textuel de bas niveau est un langage à une

b) instruction par ligne. Il peut être comparé au langage assembleur.

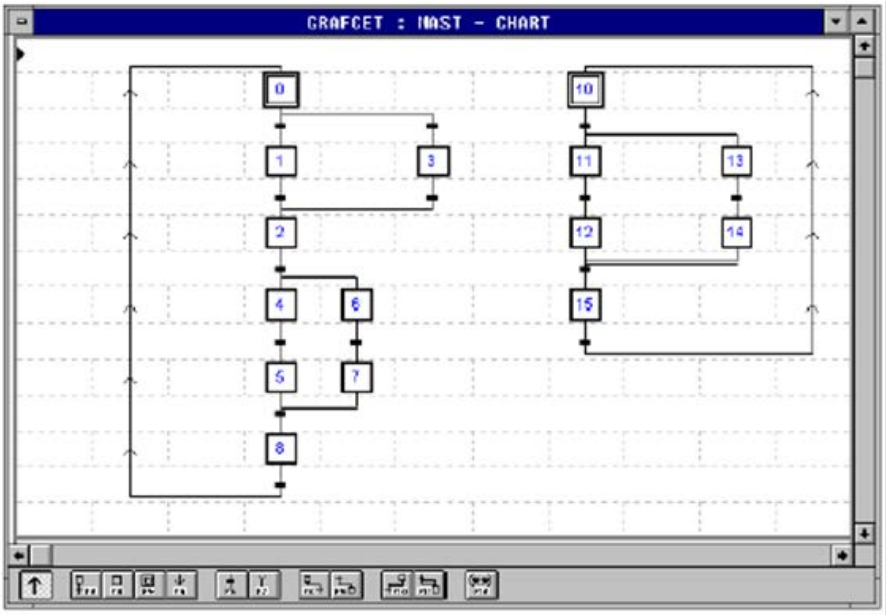


c) **Texte structuré ou ST :** ce langage est un langage textuel de haut niveau. Il permet la programmation de tout type d'algorithme plus ou moins complexe.

d) Schéma par blocs ou FBD : 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. Utilisé par les automaticiens.



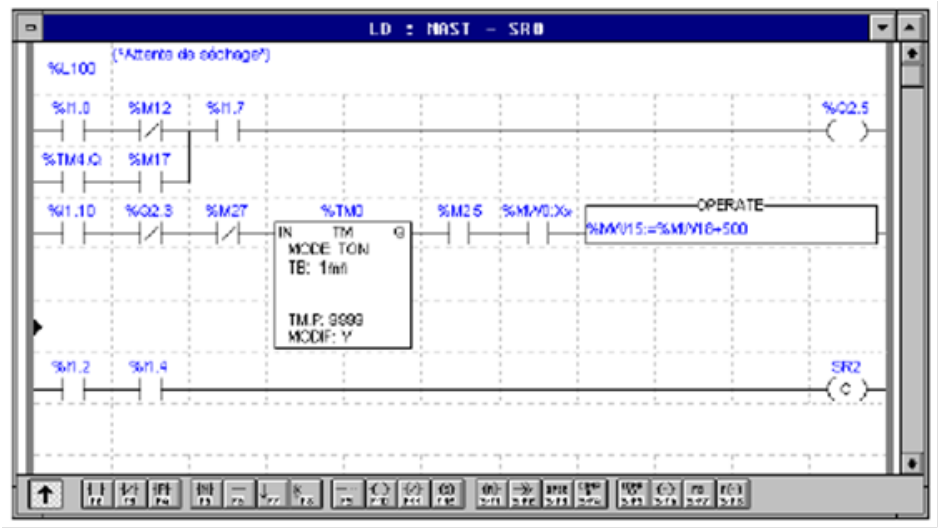
e) GRAFCET ou SFC : Ce langage de programmation de haut niveau permet la programmation aisée de tous les procédés séquentiels, représenter par une étape, une liaison orientée entre étape /transition et une transition entre deux étapes



C’est ce langage que nous utiliserons pour programmer les automates TSX 37.

f) Le langage LADDLR

Diagramme à contacts, utilisé pour programmer des éléments combinatoires



II. LE GRAFCET

II.1principe du grafcet

Le GraFCET (Graphe Fonctionnel de Commande des Étapes et Transitions) est un mode de représentation et d'analyse d'un automatisme, particulièrement bien adapté aux systèmes à évolution séquentielle, c'est-à-dire décomposable en étapes. Il est dérivé du modèle mathématique des réseaux de Pétri

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

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

II.2 Les points de vue

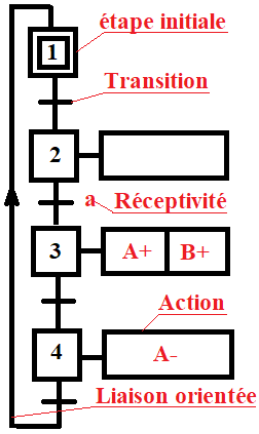
Le GRAFCET est employé à différents niveaux de l'étude d'un système. Il existe donc plusieurs types de GRAFCET selon le positionnement de cette étude.

On distingue les points de vue suivants :

- **Système :** Il définit les interactions entre le système et la matière d'œuvre.
- **Partie opérative :** Ce sont les actions des effecteurs ou les ordres émis aux actionneurs qui apparaissent sur ce grafcet.
- **Partie commande :** L'identification des signaux des capteurs est réalisée pour permettre l'écriture des réceptivités.

II.3 règles d'écriture du grafcet

II.3.1 structure graphique :



II.3.2 Normalisation :

- ✓ Les étapes initiales sont présentées par des carrés doubles 
- ✓ Les étapes sont représentées par des carrés 

- ✓ Les Liaisons Orientées de Haut en Bas ne sont pas fléchées |
- ✓ Les Liaison Orientées de Bas en Haut sont fléchées
- ✓ Les Transitions sont représentées par des segments Orthogonaux aux Liaisons orientées -|-
- ✓ Les Actions s'écrivent à droite des étapes 

- ✓ Les Réceptivités s'écrivent à droite des Transitions

II.3.3 Active / inactive :

Etape 2:Active

Etape 2:Inactive

II.3.4 transition :

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

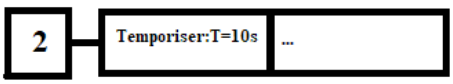
- ✓ **La réceptivité** est une fonction combinatoire d'informations telles que :
  - ➢ Etats de capteurs
  - ➢ Action de bouton-poussoir par l'opérateur.
  - ➢ Action d'un temporisateur, d'un compteur.
  - ➢ Etat actif ou inactif d'autres étapes
  - ➢ Comparaison d'une valeur analogique

- ✓ **Liaisons orientées**

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

II.3.5 Actions associées à l'étape

Une ou plusieurs actions élémentaires ou complexes peuvent être associées à une étape. Les actions associées à une étape traduisent ce qui doit être fait si l'étape est active.



N.B: On utilise toujours dans les actions les verbes à l'infinitif

II.4 Les règle d'évolutions

- ✓ **Règle n° :1 (Situation initiale)**

La situation initiale caractérise le comportement initial de la partie commande vis-à-vis de la partie opérative et correspond à l'étape active au début du fonctionnement

Elle traduit généralement un comportement de repos

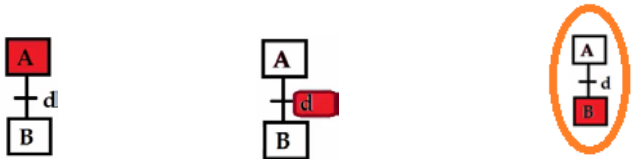
Le symbole est le double carré : 

- ✓ **Règle n° :2 (Franchissement d'une Transition)**

Le franchissement d'une transition s'effectue si :

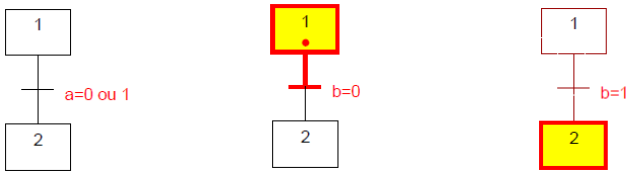
- ➢ l'étape précédente est active
- ➢ la réceptivité associée est vraie

Lorsque ces deux conditions sont réunies, la transition devient franchissable et est obligatoirement franchie



- ✓ **règle n° :3 (Évolution des étapes Actives)**

Le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.



Cas 1 **Cas 2** **Cas 3**

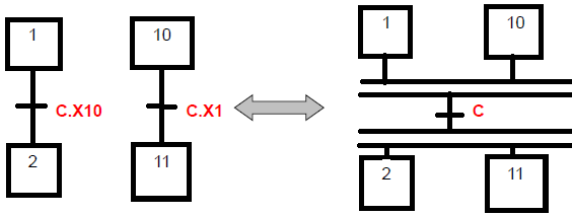
Cas 1 : La transition 1-2 est non validée, l'étape 2 étant inactive

Cas 2 : L'étape 1 étant active, la transition 1-2 est validée mais ne peut être franchie car la réceptivité n'est pas vraie : b=0.

Cas 3 : La transition 1-2 est franchie car la réceptivité est vraie : b=l. Dans ce cas l'étape 2 est activée et l'étape 1 est désactivée.

- ✓ **règle n° :4 (Évolutions simultanées)**

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

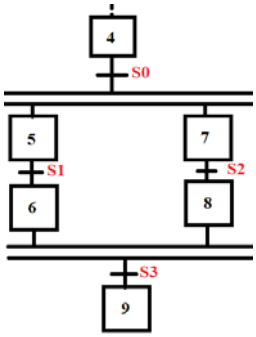


X_1 : Variable Booléenne correspondant à l'étape 1 :

- ➢ Si l'étape 1 est active $X_1 = 1$
- ➢ Si l'étape 1 est inactive $X_1 = 0$

- ✓ **règle n° :5 (Activation et désactivation simultanées)**

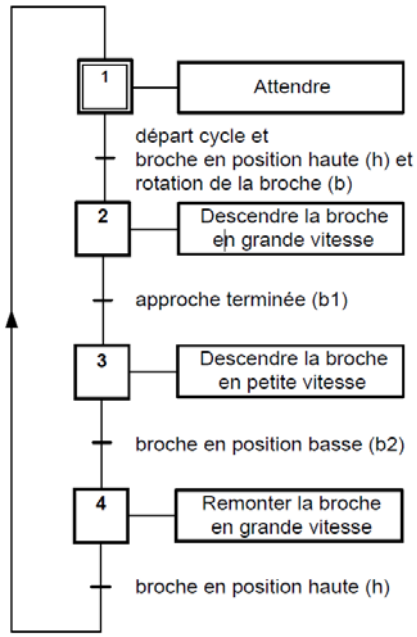
Si au cours du fonctionnement de l'automatisme une même étape doit être simultanément Activée et désactivée, elle reste activée



II.5 Séquences de base

II.5.1 Grafcet linéaire (séquence unique)

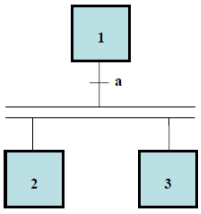
Le Grafcet linéaire ci-dessous, représente un cycle fonctionnel d'une perceuse, c'est une succession d'étape et de transitions.



II.5.2 Divergence et convergence

a) Divergence en ET (Structure Simultanée) :

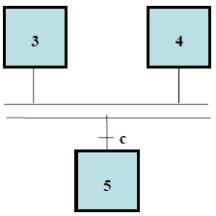
Quand la transition est franchissable, il suffit d'activer deux étapes au lieu d'une



Si 1 Active et si le Capteur (a) Activé, désactivation de 1 et Activation de 2 et 3 simultanément

b) La Convergence en ET (Structure Simultanée) :

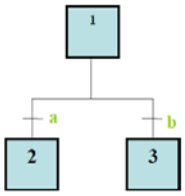
Quand les deux étapes (3 et 4) sont actives et la réceptivité c est vraie alors l'étape 5 devient active et les deux étapes (3 et 4) désactives



c) Divergence en OU (Structure Alternative) :

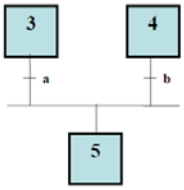
Quand l'étape 1 est active et la réceptivité a (ou b) est vraie, l'étape 2 (ou 3) devient active et l'étape 1 désactive.

NB : Il est possible que l'évolution devienne simultanée si les deux réceptivités sont vraies.



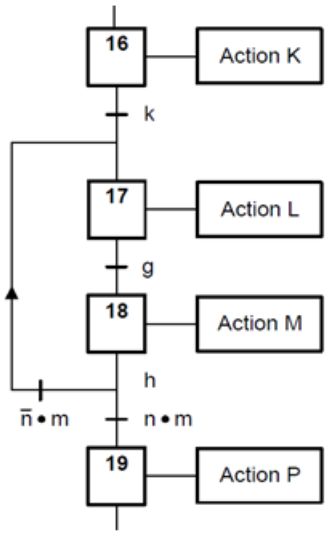
d) la Convergence en OU (Structure Alternative) :

Quand l'étape 3 (ou 4) est active ET la réceptivité a (ou b) est vraie alors l'étape 5 devient active et l'étape 3 (ou 4) désactive.

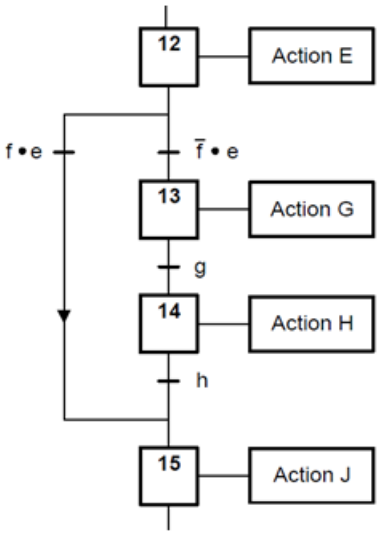


II.5.3 Sauts d'étapes et reprise de séquences

Le saut conditionnel est un aiguillage particulier permettant de sauter une ou plusieurs étapes lorsque les actions à réaliser deviennent utiles, tandis que la reprise de séquences permet au contraire de reprendre une ou plusieurs fois la même séquence tant qu'une condition fixée n'est pas obtenue**.**



Saut d'étapes

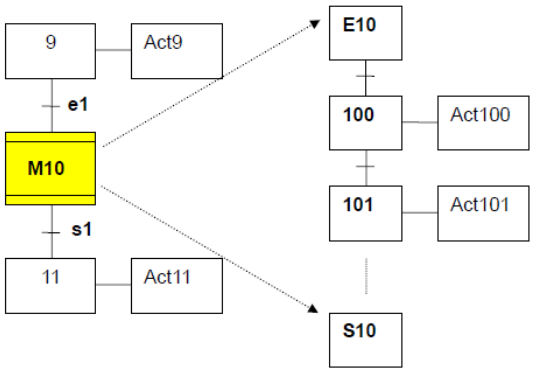


Reprise de séquence

II.5.4 Sous-programmes

a) macro-étape

Une macro-étape (ME) est la représentation unique d'un ensemble d'étapes et de transition nommé "Expansion d'étapes", la macro-étape se substitue à une étape du GRAFCET.



1) L'expansion de ME comporte une étape d'entrée et une étape de sortie repérées par E et S.

2) Tout franchissement de la transition amont de la macro-étape active l'étape E d'entrée de son Expansion

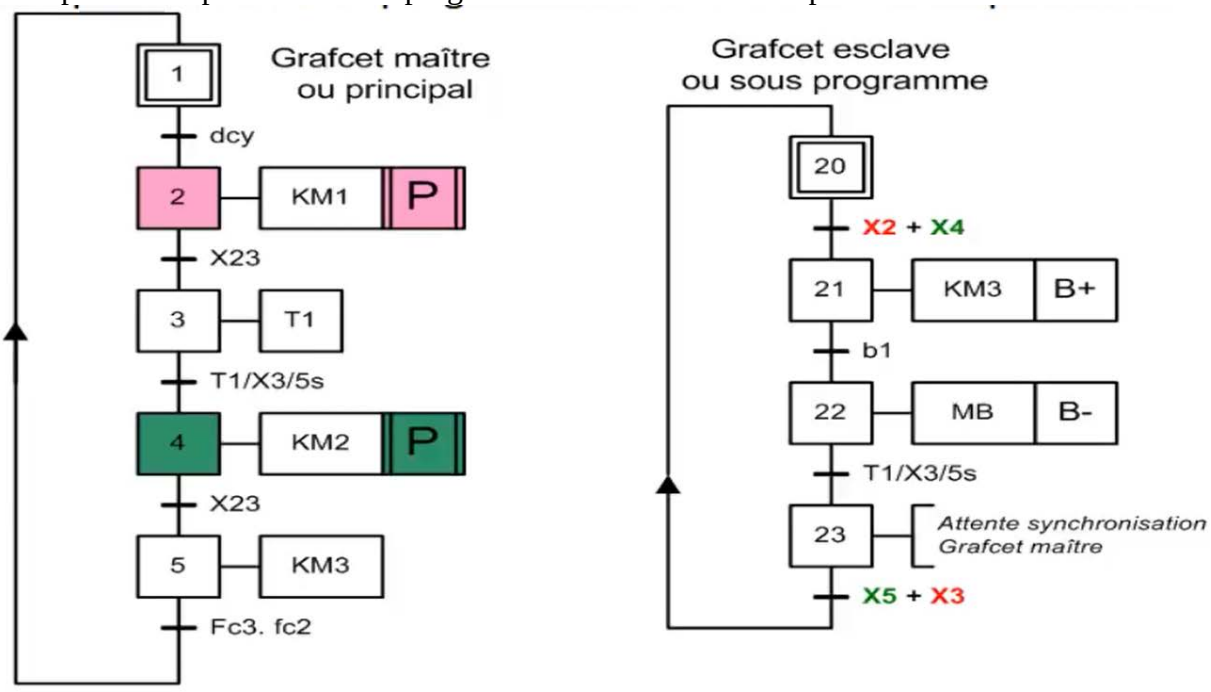
3) L'étape de sortie participe à la validation des transitions aval de la macro-étape.

4) La transition suivant la macro-étape n'est validée que lorsque la dernière étape de l'expansion de macro-étape est active.

NB : Il est préférable de ne pas associer d'actions aux étapes d'entrées et de sortie de la macro-étape

b) Graphe auxiliaire

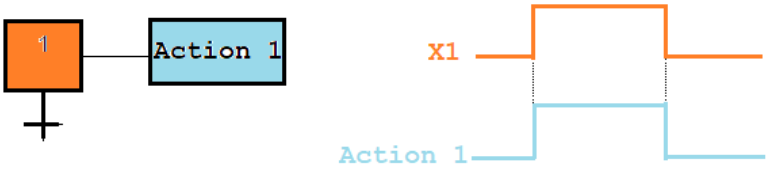
Le sous-programme est représenté dans la case action par un rectangle dont les côtés verticaux sont doublés. Nom tâche Le sous-programme peut être appelé à différents endroits du grafcet principal. Dans l'exemple, l'étape 2 OU 4 (X2 + X4) permet l'évolution du sous-programme P. L'étape 23 permet au grafcet principal de passer à l'étape suivante.
L'étape 5 OU 3 permet au sous-programme de revenir à son étape initiale.



II.6 Actions associées aux étapes

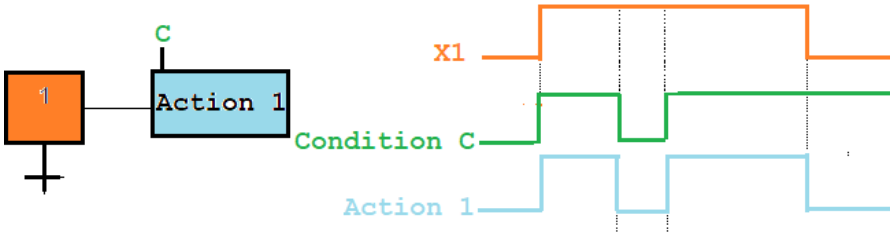
II.6.1 Action continue inconditionnelle

L'ordre d'action est émis de façon continue tant que l'étape à laquelle il est associé est active.



Action 1= X1

2.6.2 Action continue conditionnelle



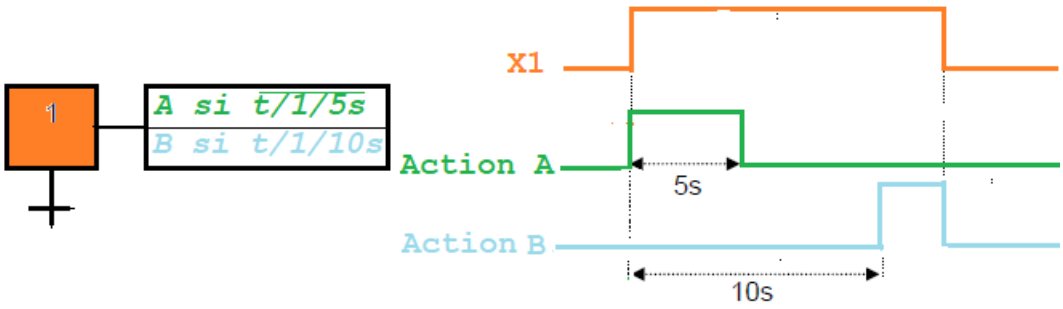
Action 1 = X1.C (Exemple C : sécurité)

II.6.3 Action temporisée

a) Action continue à durée Limitée :

Action A=t/1/5s

Action continue conditionnelle dont l'ordre sera maintenue pendant un certain temps à partir de l'activation de l'étape qui lui est associée.



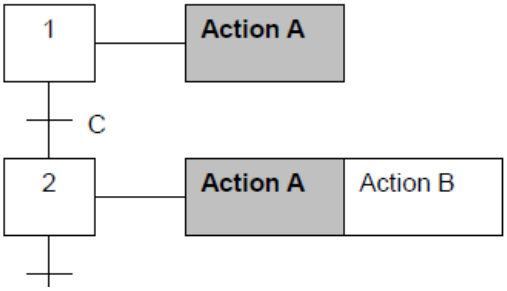
b) Action continue retardée :

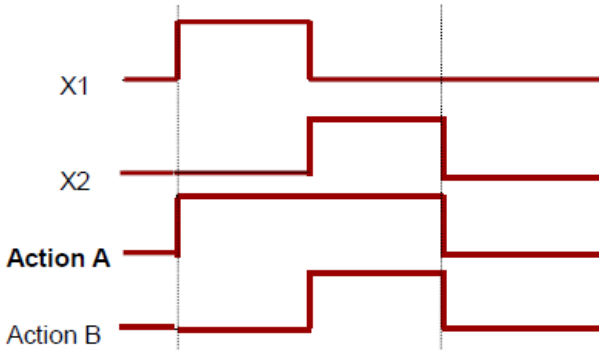
Action B=t/1/10s

Action continue conditionnelle dont la condition logique est une temporisation permettant de retarder l'action par rapport à l'activité de L'étape qui Lui est Associée.

II.6.4 Action maintenue

L'action A est maintenue dans les deux étapes 1 et 2





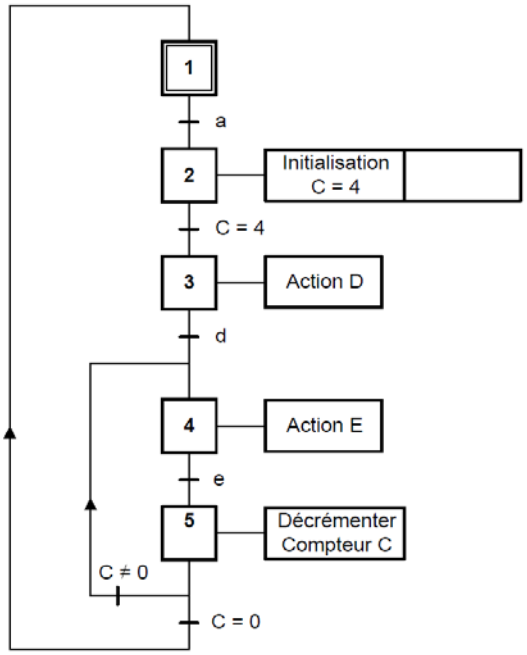
II.6.5 Ordre de mémorisation de l'action

Ces deux ordres, de mémorisation et d'effacement permettent d'élaborer l'action de sortie du composant :

- ✓ ordre de début d'action, noté : "action = 1" (set)
- ✓ ordre de fin d'action, noté : "action=0" (reset)

II.6.6 Décompteur

Il faut prévoir une séquence d'initialisation (ou de remise à zéro dans le cas d'un compteur). Après l'action, on établit une séquence de décrémentation du décompteur suivi d'une reprise de séquence en fonction de la valeur de celui-ci



II.7 Mise en équations des GRAFCET :

Malheureusement, ce ne sont pas tous les automates qui se programment en GRAFCET directement. Mais, généralement ils peuvent être programmés en « diagramme échelle » (ou LADDER).

Il faut donc pouvoir transformer le GRAFCET qui est la meilleure approche qui existe pour traiter les systèmes séquentiels en « diagramme échelle » qui est le langage le plus utilisé par les automates.

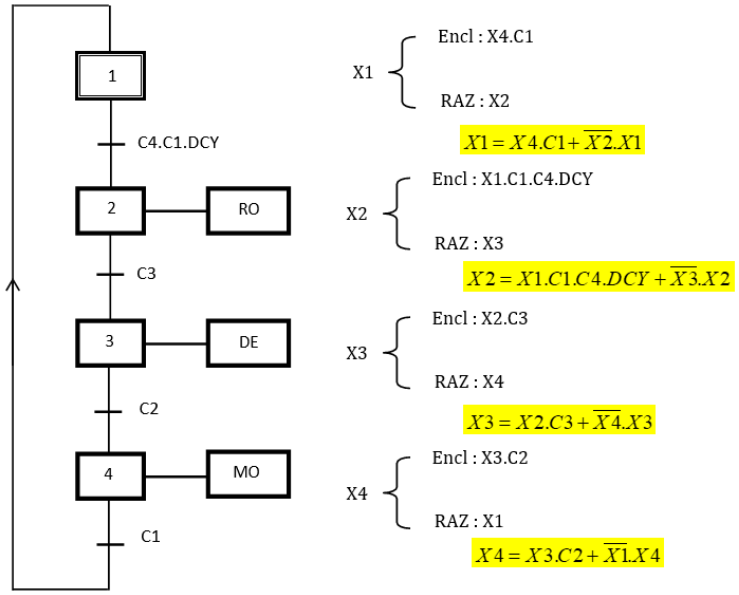
II.7.1 Mémoire d'étape :

$X = Encl + \overline{RAZ}.X$

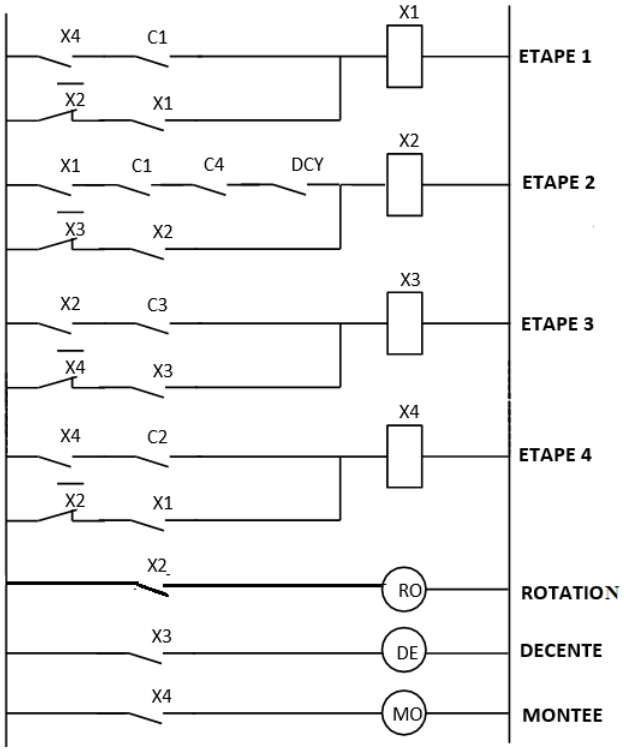
Afin de respecter les règles d'évolution du GRAFCET, chaque étape peut être matérialisée par une mémoire du type marche prioritaire possédant une structure de la forme :

Les termes d'enclenchement et de remise à zéro sont définis de la manière suivante :

$$ETAPE\ X \begin{cases} Encl : Etat\ \log ique\ de\ l'étape\ précédente.Réceptivité \\ RAZ : Etat\ \log ique\ de\ l'étape\ suivante \end{cases}$$



Les équations des mémoires étape déterminée précédemment nous donnent le schéma de câblage électrique suivant



Pour établir la commande de chaque sortie, il suffit de considérer la ou les étapes durant lesquelles la sortie doit être enclenchée. Ainsi :

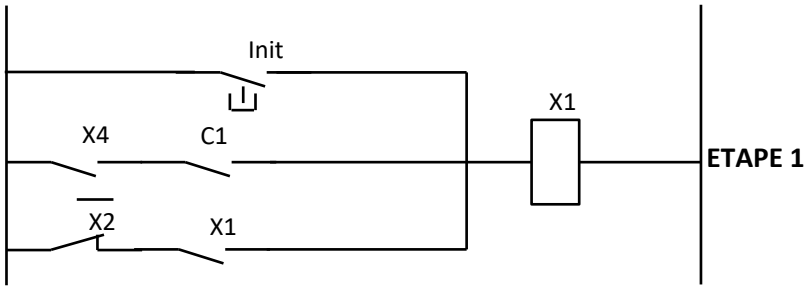
- ✓ La sortie RO a lieu durant l'ETAPE 2 d'où RO = X2
- ✓ La sortie DE a lieu durant l'ETAPE 3 d'où DE = X3
- ✓ La sortie MO a lieu durant l'ETAPE 4 d'où MO = X4

II.7.2 Initialisation de la séquence :

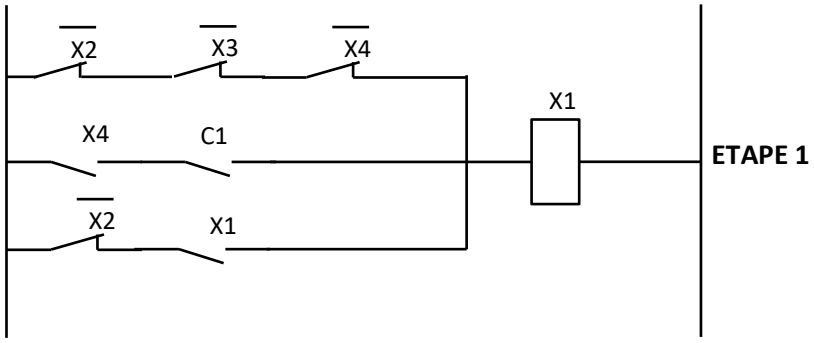
Nous remarquons sur le schéma précédent qu'à la mise sous tension, toutes les mémoires se trouvant ici à l'état repos, aucune évolution n'est possible.

Il est donc impératif d'initialiser la séquence en venant enclencher la mémoire X1 Matérialisant l'étape initiale de notre GRAFCET. Ceci est obtenu :

- ✓ Soit en utilisant un contact d'initialisation ou un contact de passage commandé lors de la mise sous tension de l'automatisme, comme le montre le schéma suivant :



- ✓ Soit en testant l'état repos de toutes les mémoires d'étape suivantes, pour venir alors systématiquement enclencher la mémoire X1, comme le montre le schéma suivant :



II.8 Grafcet et langage LADDER

Même si la forme est très différente, ces deux langages ont de nombreux points communs.

- ✓ tous deux décrivent un automatisme séquentiel sous forme graphique
- ✓ le fonctionnement est découpé en structures élémentaires que le Grafcet appelle étapes
- ✓ la progression d'une étape à l'autre se fait à la suite de la survenue d'un événement

III. LE LANGAGE LADER (LD)

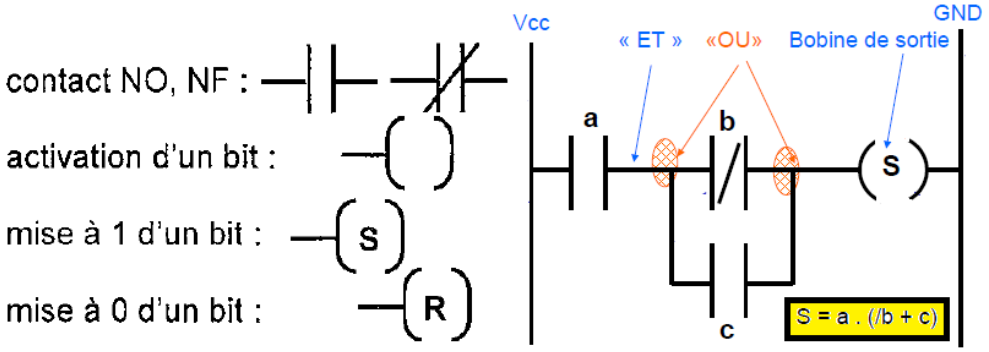
III.1 Description :

Le langage des API d'origine américaine utilise le symbolisme classique des schémas à relais accompagné de blocs graphiques préprogrammés pour réaliser des fonctions d'automatisme (calculs, temporisation, compteur,…..).

Le langage Ladder est une succession de "réseaux de contacts" véhiculant des informations logiques depuis les entrées vers les sorties. C'est une simple traduction des circuits de commande électriques (Commande câblée).

Les entrées sont représentées par des interrupteurs -| |- ou -|/|- si entrée inversée, les sorties par des bobines **-( )-** ou des bascules **-(S)- -(R)-.**Il y a également d'autre opérations :

- ✓ L'inverseur -|**NOT**|-,
- ✓ L'attente d'un front montant **-(P)-** ou descendant **-(N)-.**



Les sorties sont obligatoirement à droite du réseau On doit évidemment identifier nos **E/S**, soit directement par leur code (**Ia.b** / **Qa.b**), ou avec leur libellé en clair défini dans la table des mnémoniques. On relie les éléments en série par la fonction **ET**, en parallèle par le **OU**. On peut utiliser des bits internes (peuvent servir en bobines et interrupteurs), comme on utilise dans une calculatrice une mémoire pour stocker un résultat intermédiaire (**Ma.b**). On peut aussi introduire des éléments plus complexes, en particulier les opérations sur **bits** comme par exemple une bascule **SR** (priorité déclenchement), **RS** (priorité enclenchement), **POS** et **NEG** pour la détection de fronts... on trouvera d'autres fonctions utiles, les compteurs, les temporisateurs et le registre à décalage.

On peut également utiliser des fonctions plus complexes (calculs sur mots par exemple)

III.2 Adressage des entrées/sorties

La déclaration d'une entrée ou sortie donnée à l'intérieur d'un programme s'appelle l'adressage. Les entrées et sorties des API sont la plupart du temps regroupées en groupes de huit sur des modules d'entrées ou de sorties numériques. Cette unité de huit est appelée

octet. Chaque groupe reçoit un numéro que l'on appelle l'**adresse d'octet**.

Afin de permettre l'adressage d'une entrée ou sortie à l'intérieur d'un octet, chaque octet est divisé en huit **bits**. Ces derniers sont numérotés de 0 à 7.

On obtient ainsi l'**adresse du bit.**

L'API représenté ici a les octets d'entrée 0 et 1 ainsi que les octets de sortie 0 et 1.

Nom	Type de données	Adresse
ETAPE 1	Bool	M0.1
ETAPE 2	Bool	M0.2
ETAPE 3	Bool	M0.3
ETAPE 4	Bool	M0.4
C1	Bool	I0.0
C2	Bool	I0.1
C3	Bool	I0.2
C4	Bool	I0.3
DCY	Bool	I0.4
RO	Bool	Q0.0
DE	Bool	Q0.1
MO	Bool	Q0.2

__Etape 1__ de type logique __(Bool)__ affecté à la mémoire __M0.1__

Le capteur __C1__ est de type logique et affecté à l'adresse __I0.0__

La sortie __DE__ est de type logique et affecté à l'adresse __Q0.1__

Tableau 2 : table de variables mnémoniques

Par exemple

Pour adresser la 5ème entrée du **DCY** en partant de la gauche, on définit l'adresse suivante :

I0.4

I indique une adresse de type entrée,

0, l'adresse d'octet et

4, l'adresse de bit.

Les adresses d'octet et de bit sont toujours séparées par un point.

Pour adresser la 3ème sortie, par exemple, on définit l'adresse suivante :

Q0.2 :

Q indique une adresse de type Sortie,

0, l'adresse d'octet

2, l'adresse de bit.

Les adresses d'octet et de bit sont toujours séparées par un point.

Remarque : L'adresse du bit de la dixième sortie est un **1** car la numérotation commence à zéro.

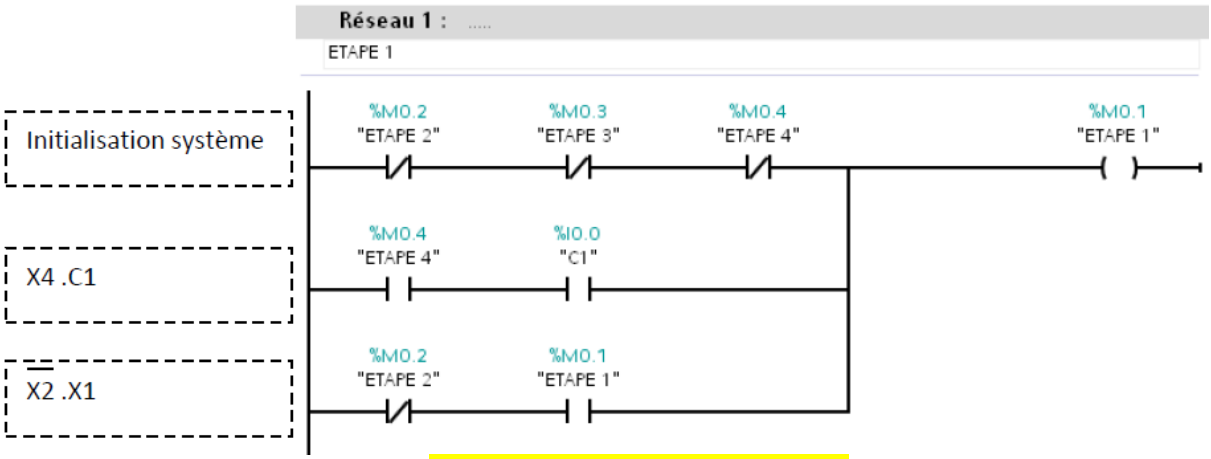
III.3 Exemple

Dans l'exemple précédant et suivant la table mnémonique d'affectation le programme en LADER

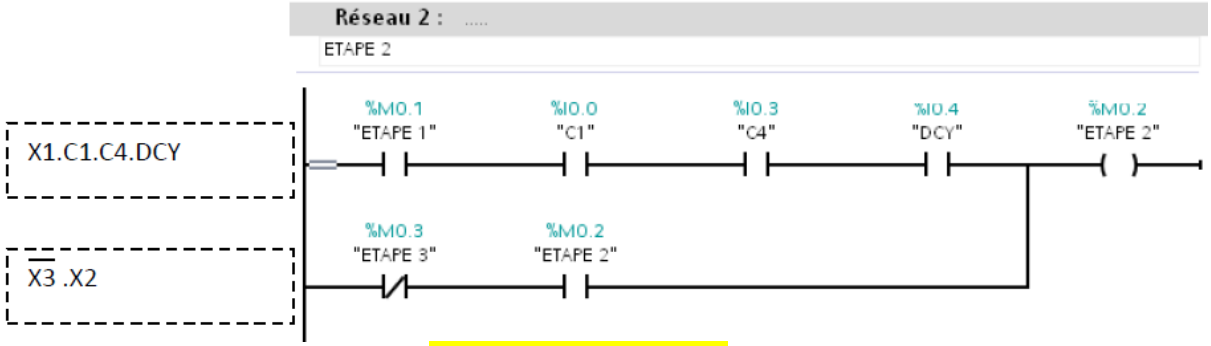
a) Sans bobines Set/ Reset

- **Pour la programmation des étapes**

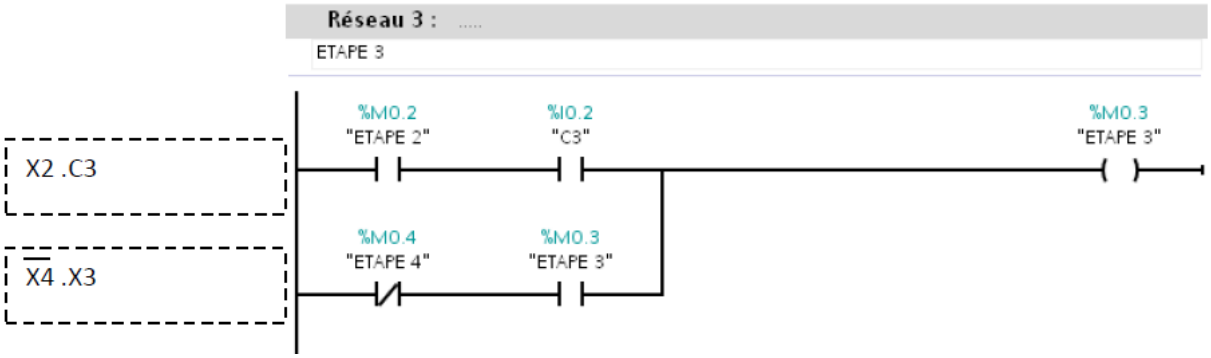
L'étape 1 : $X1 = X4.C1 + \overline{X2}.X1$



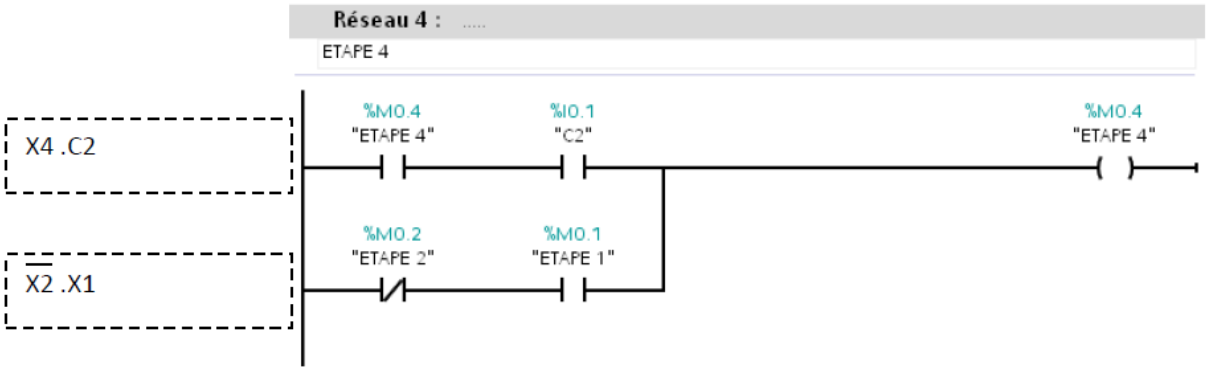
L'étape 2 : $X2 = X1.C1.C4.DCY + \overline{X3}.X2$



L'étape 3 : $X3 = X2.C3 + \overline{X4}.X3$



L'étape 4 : $X4 = X4.C2 + \overline{X2}.X1$



- **Pour la programmation des sorties**

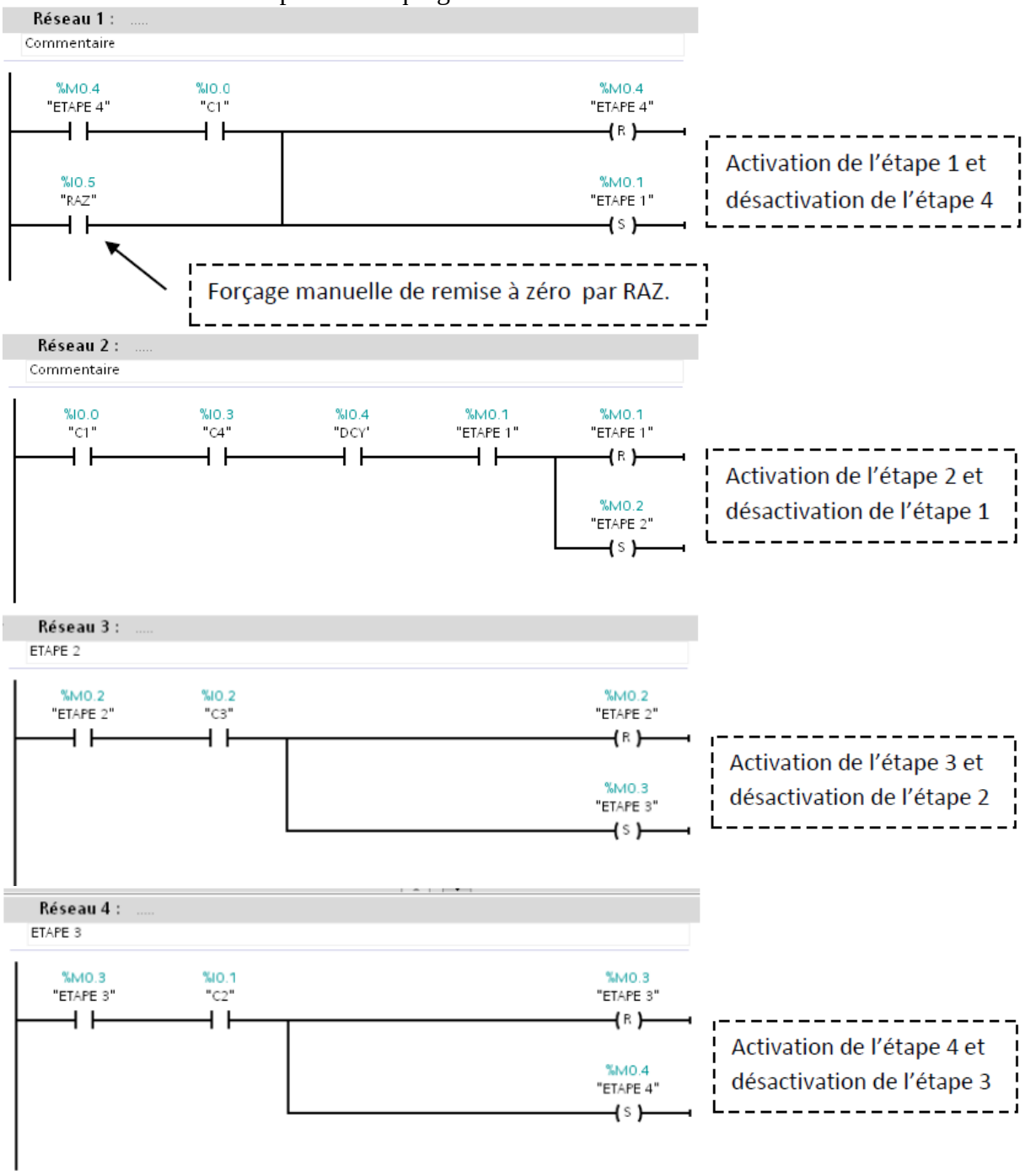
Sortie R0 : est actionné uniquement à l'étape 2

Sortie DE : est actionné uniquement à l'étape 3

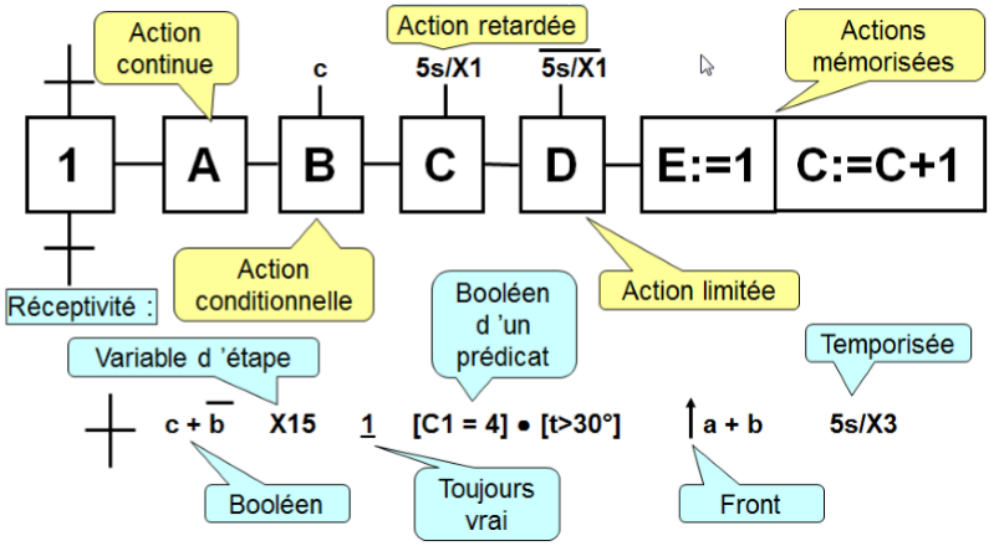
MO : est actionné uniquement à l'étape 4

b) Avec bobines Set/ Reset

Le programme peut être simplifié si en utilisant les bobines **Set/ Reset** ou les bascules **SR** ou **RS** et en tenant compte des cinq règles du GRAFCET.



Annexe


Action continue
Action retardée
Actions mémorisées
c
5s/X1
5s/X1
1
A
B
C
D
E:=1
C:=C+1
Réceptivité :
Action conditionnelle
Booléen d 'un prédicat
Action limitée
Variable d 'étape
Temporisée
c + b
X15
1
[C1 = 4] • [t>30°]
a + b
5s/X3
Booléen
Toujours vrai
Front