CHAPITRE 4 : LA PROGRAMMATION DES API

I- 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.

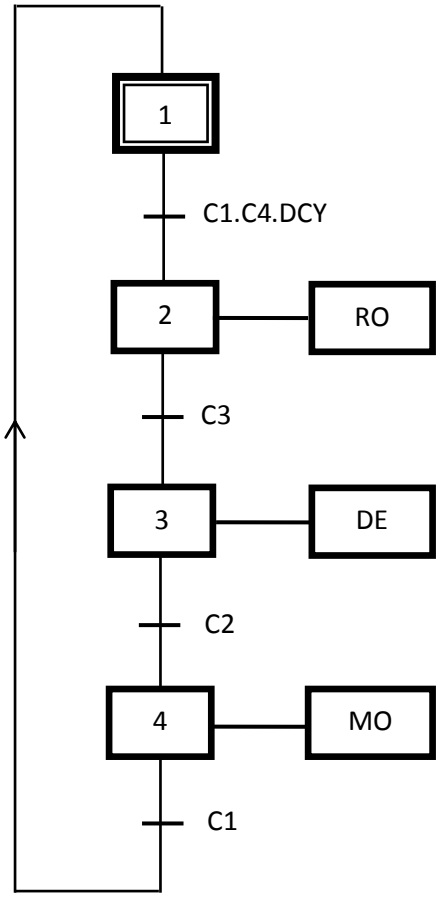
I-1 Mémoire d'étape :

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 :

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

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

ETAPE X
- Encl : Etat logique de l'Etape(s) précédente(s). Réceptivité
- RAZ : Etat logique de l'Etape(s) suivante(s)



X1 { Encl : X4.C1 ; RAZ : X2 }

$$X1 = X4.C1 + \overline{X2}\,.\,X1$$

X2 { Encl : X1.C1.C4.DCY ; RAZ : X3 }

$$X2 = X1.C1.C4.DCY + \overline{X3}\,.X2$$

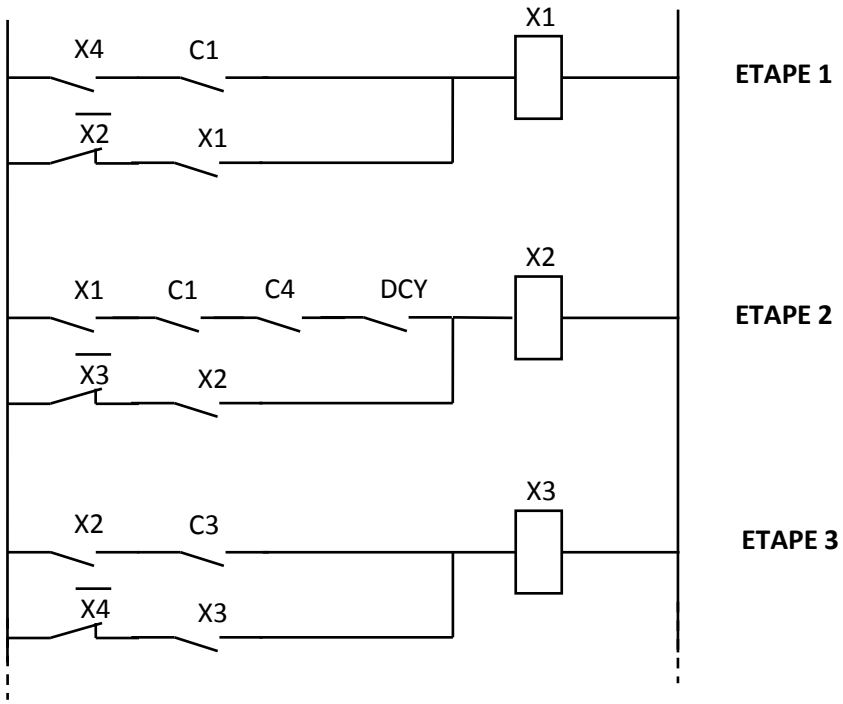
X3 { Encl : X2.C3 ; RAZ : X4 }

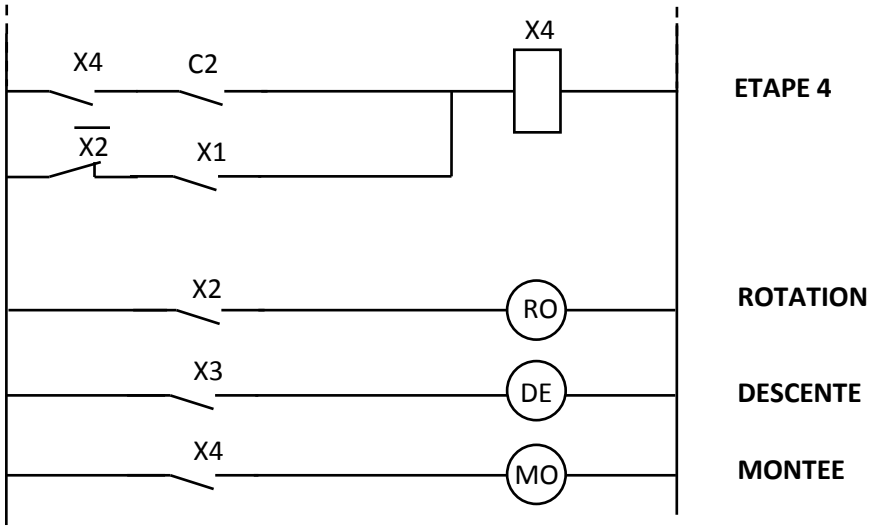
$$X3 = X2.C3 + \overline{X4}\,.\,X3$$

X4 { Encl : X3.C2 ; RAZ : X1 }

$$X4 = X3.C2 + \overline{X1}\,.\,X4$$

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





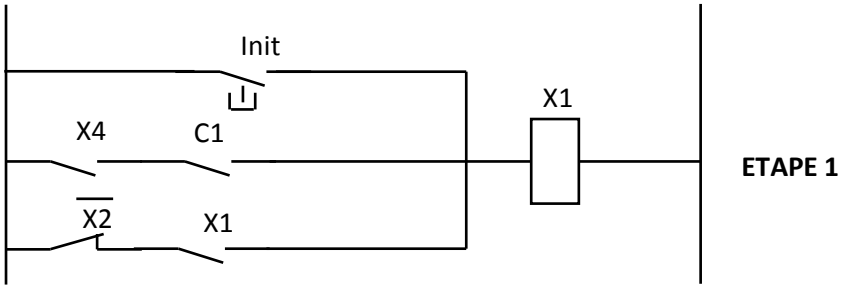
Pour établir la commande de chaque sortie, il suffit de considérer la ou les étapes durant lesquelles la sortie doit être enclenchée. Ainsi :

- 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

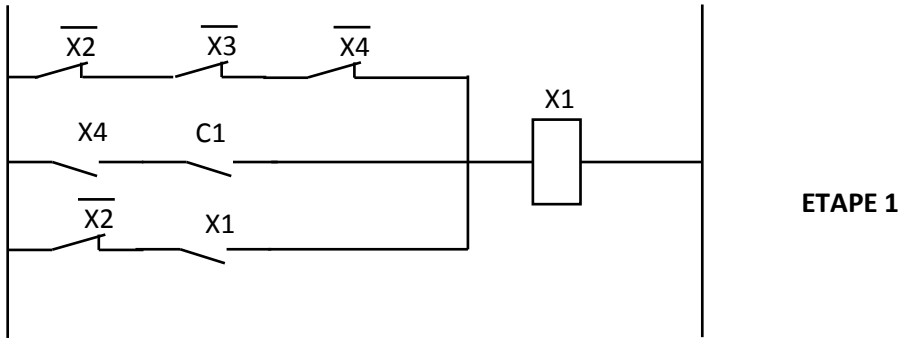
I-2 Initialisation de la séquence :

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- Langages de programmation des API:

Les langages de programmation des API sont de natures diverses étant donné la diversité, des utilisateurs pouvant les utiliser.

II-1 Le langage LADER (LD) :

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,…..).

C'est une suite de réseaux qui seront parcourus séquentiellement. 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 pour la fonction **ET**, en parallèle pour 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)

II-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 1 de type logique (Bool) affecté à la mémoire M0.1*
ETAPE 2	Bool	M0.2	
ETAPE 3	Bool	M0.3	
ETAPE 4	Bool	M0.4	
C1	Bool	I0.0	← *Le capteur C1 est de type logique et affecté à l'adresse 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	← *La sortie DE est de type logique et affecté à l'adresse Q0.0*
MO	Bool	Q0.2	

Tableau 1 : 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 et **2**, l'adresse

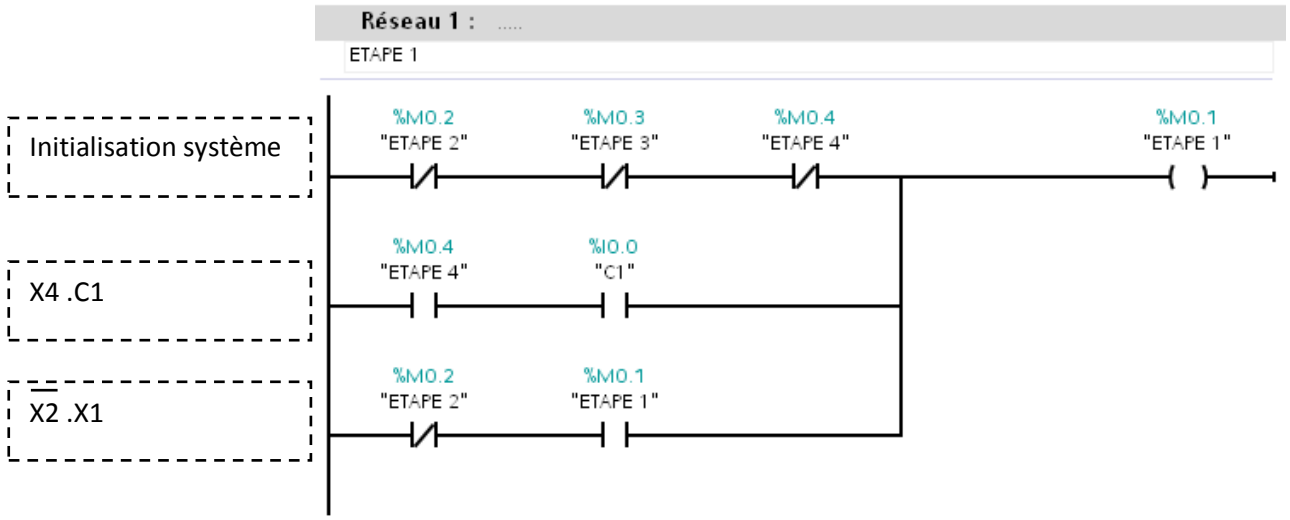
de bit. Les adresses d'octet et de bit sont toujours séparées par un point.

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

Exemple:

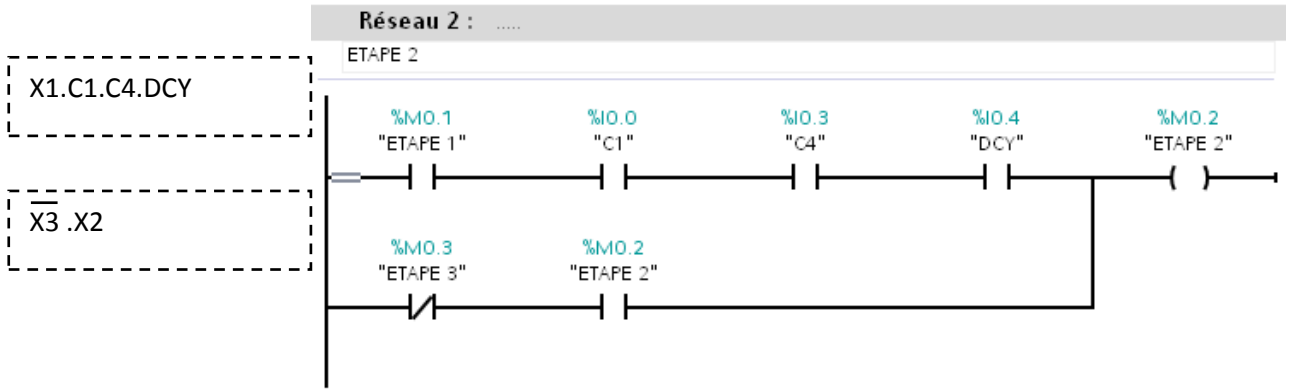
Dans l'exemple précédent et suivant la table mnémonique d'affectation le programme en LADER de la première étape est :

$$X1 = X4.C1 + \overline{X2}\,.\,X1$$



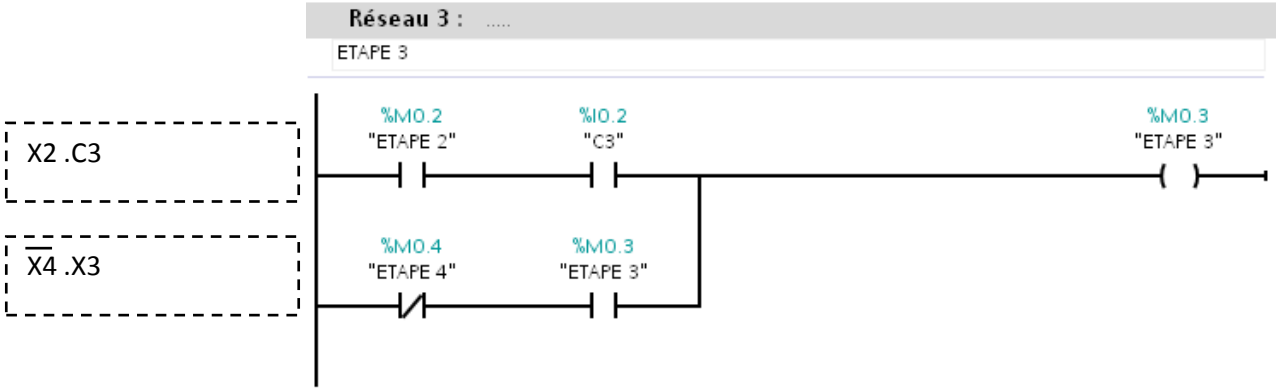
Et ainsi pour l'étape 2 est :

$$X2 = X1.C1.C4.DCY + \overline{X3}\,.X2$$

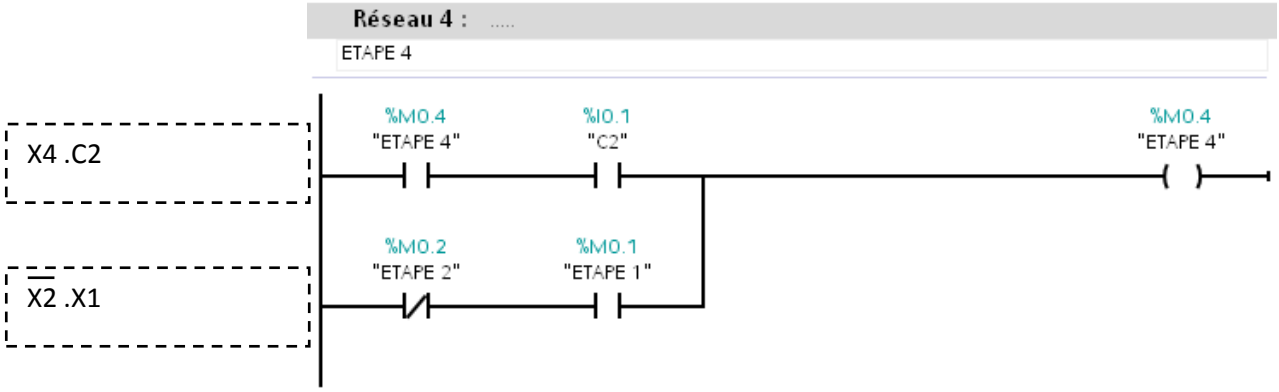


L'étape 3 :

$$X3 = X2.C3 + \overline{X4}\,.\,X3$$

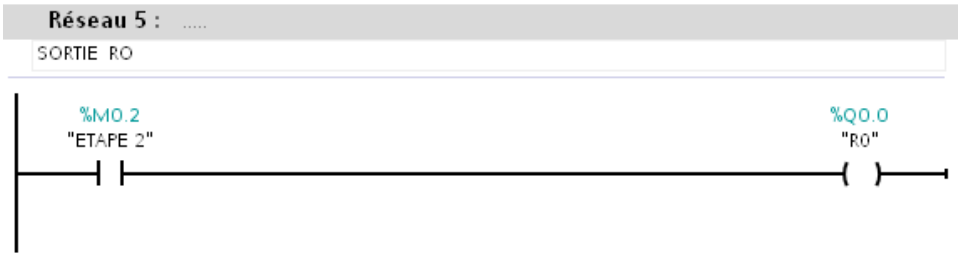


L'étape 4 :

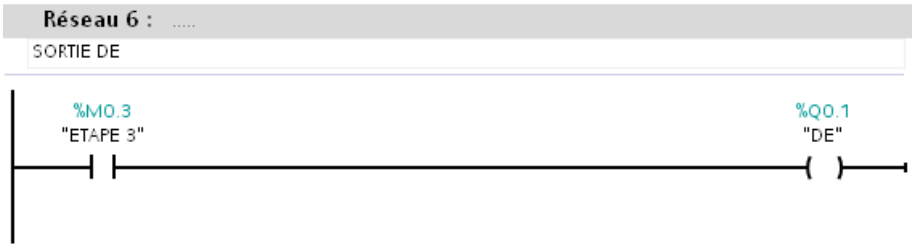


Pour la programmation des sorties

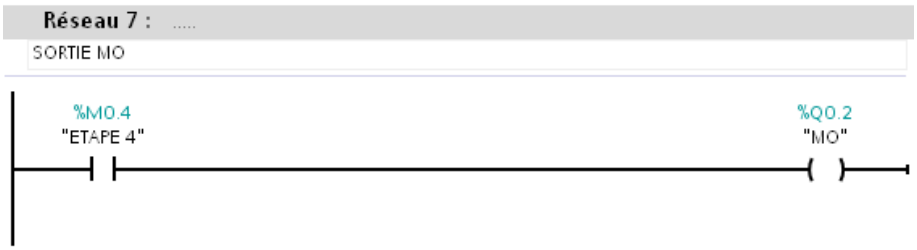
R0 : est actionné uniquement à l'étape 2



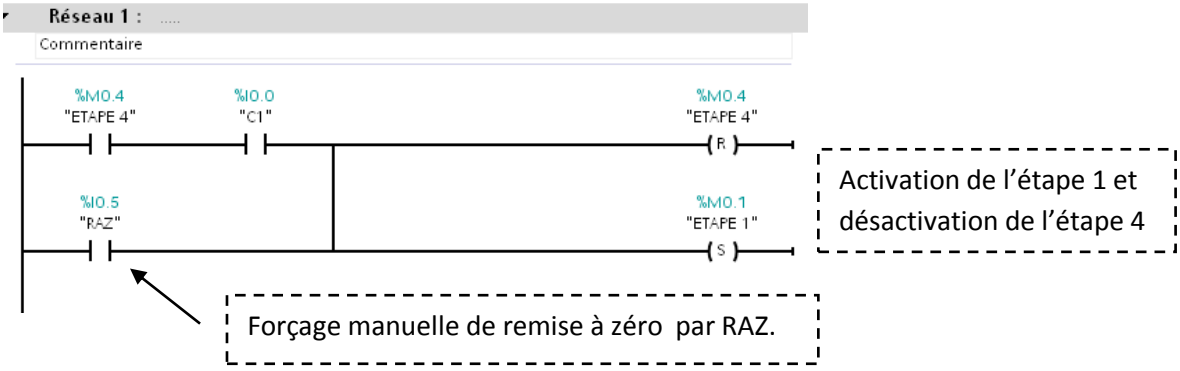
DE : est actionné uniquement à l'étape 3

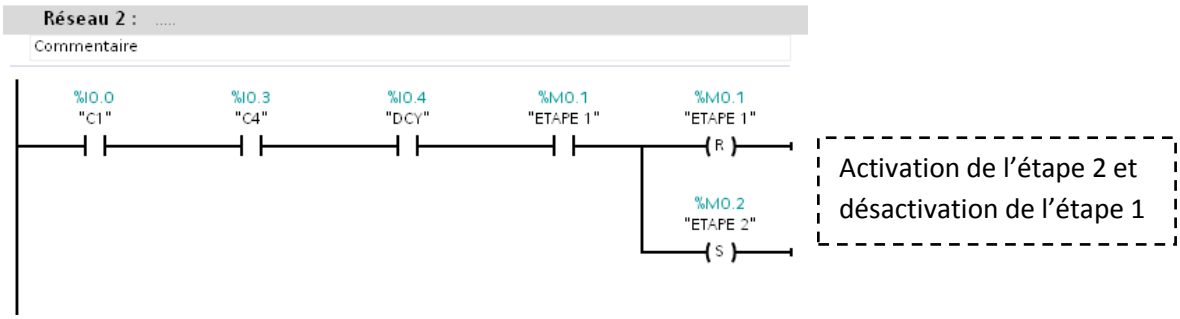


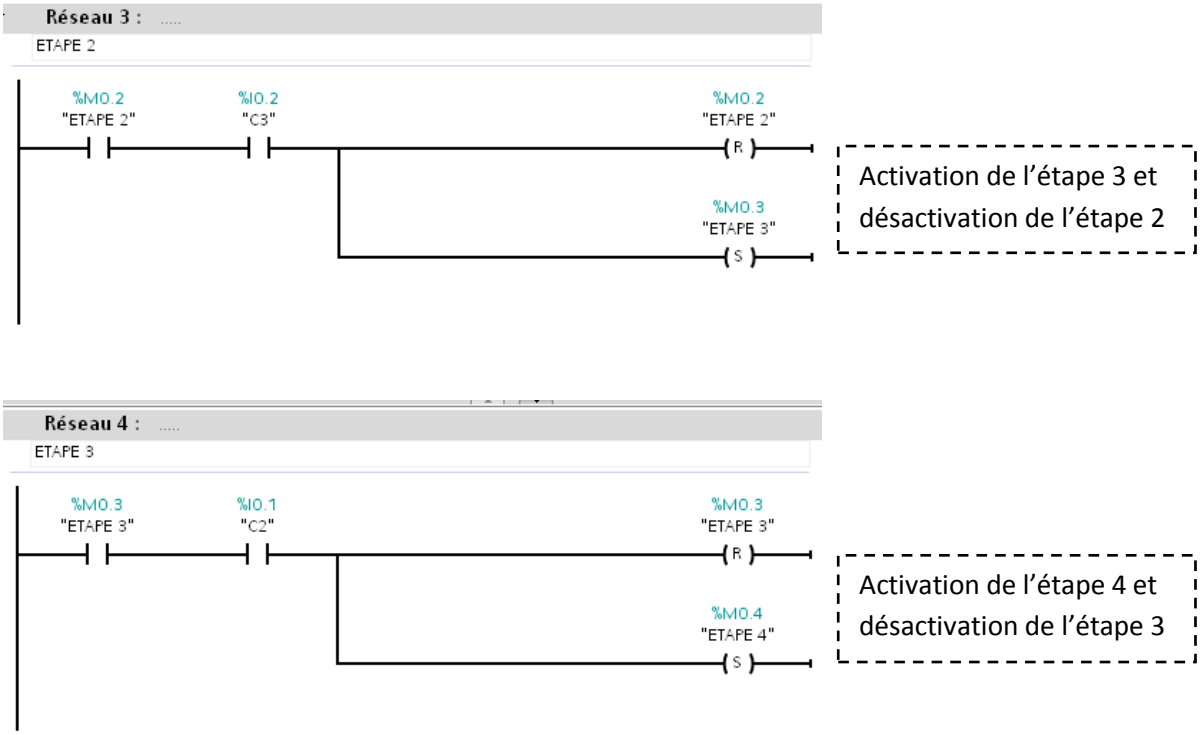
MO : est actionné uniquement à l'étape 4



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







II-2 Le langage LOG (Logigramme) :

Les opérations logiques servent à définir des conditions pour l'activation d'une sortie. Elles peuvent être créées dans le programme de l'API dans les langages de programmation Schéma des circuits **LADER** (**LD**) ou **Logigramme** (**LOG**).

Il existe de nombreuses opérations logiques pouvant être mises en œuvre dans des programmes API.

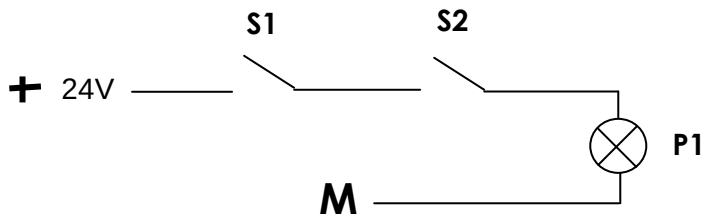
L'opération **ET** et l'opération **OU**, ainsi que la **NEGATION** d'une entrée sont les opérations les plus fréquemment utilisées et seront expliquées ici à l'appui d'un exemple.

II-2-1 Opération ET

Exemple d'une opération ET :

Une lampe doit s'allumer quand les deux interrupteurs sont fermés simultanément.

Schéma :

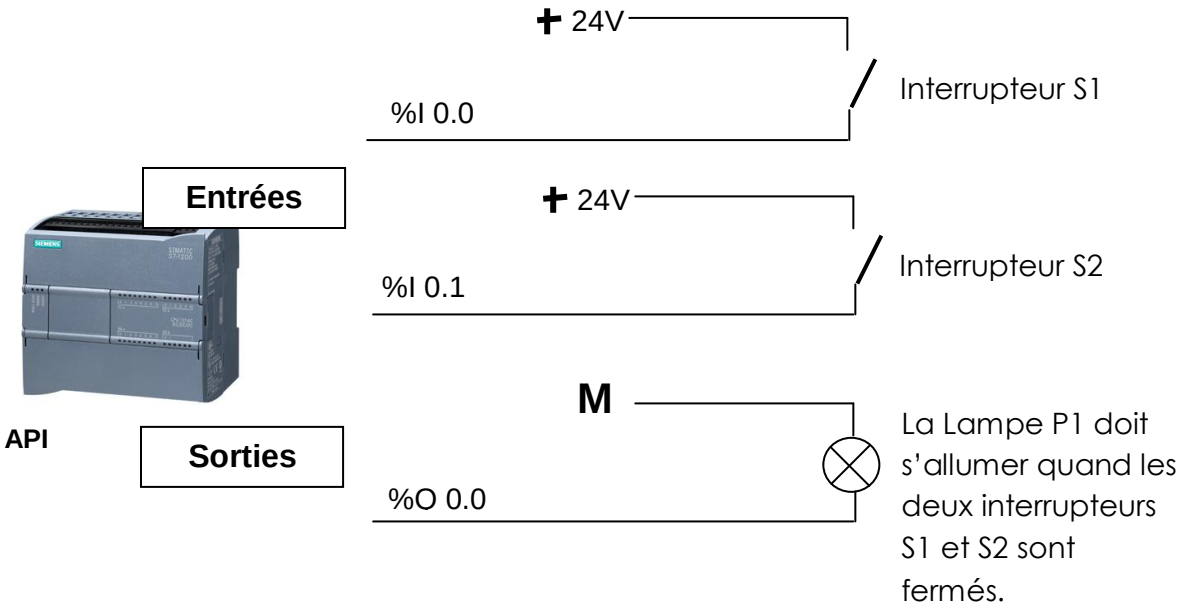


Explication :

La lampe s'allume uniquement quand les deux interrupteurs sont fermés.
C'est-à-dire, quand S1 **ET** S2 sont fermés, alors la lampe P1 est allumée.

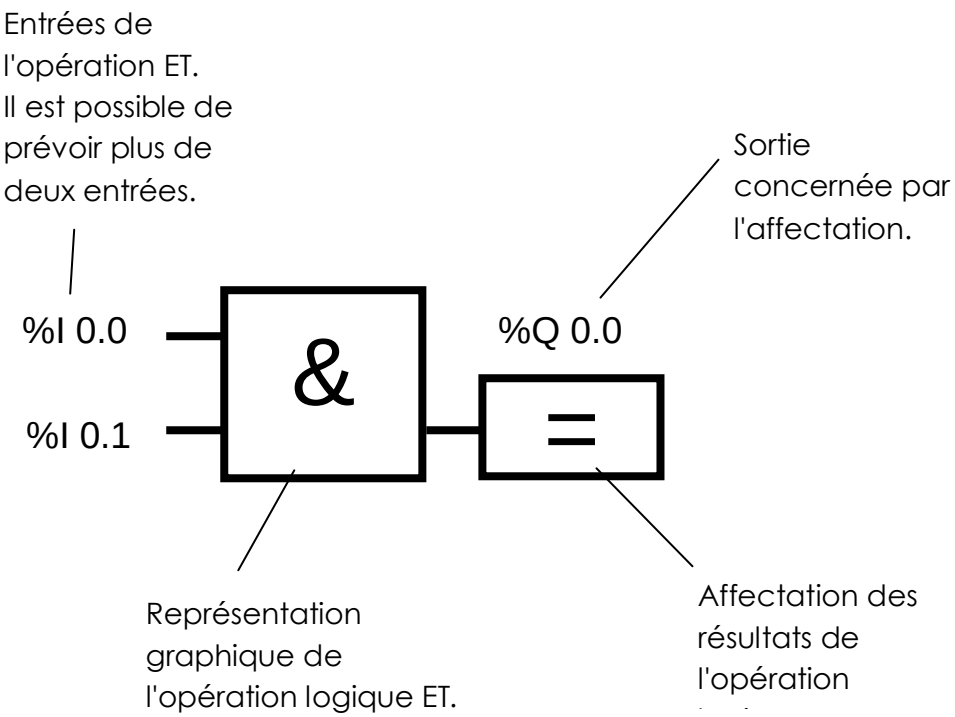
Câblage de l'API :

Pour appliquer cette opération au programme de l'API, les deux commutateurs doivent être connectés aux entrées de l'API. Ici, S1 est relié à l'entrée I 0.0 et S2 à l'entrée I 0.1.
De plus, la lampe P1 doit être connectée à une sortie, par exemple Q 0.0.



Opérateur ET dans LOG :

Dans le logigramme LOG, l'opérateur ET est programmé par le symbole ci-dessous et est représenté de la manière suivante :

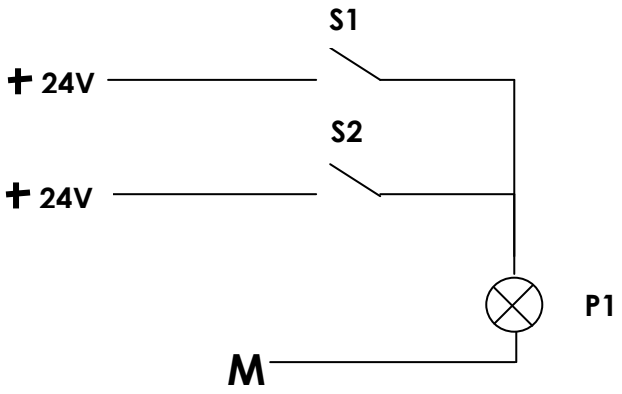


II-2-2 Opération OU

Exemple d'une opération OU :

Une lampe doit s'allumer si au moins un des deux interrupteurs est fermé.

Schéma :

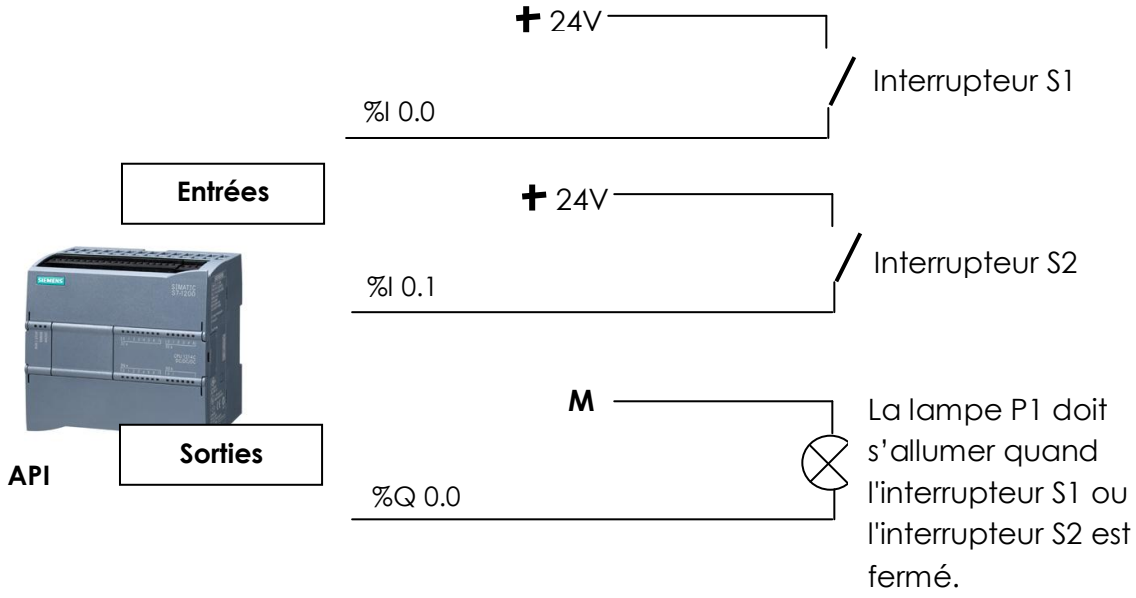


Explication :

La lampe s'allume à partir du moment où un des deux interrupteurs est fermé.
C'est-à-dire, quand S1 **OU** S2 est fermé, alors la lampe P1 est allumée.

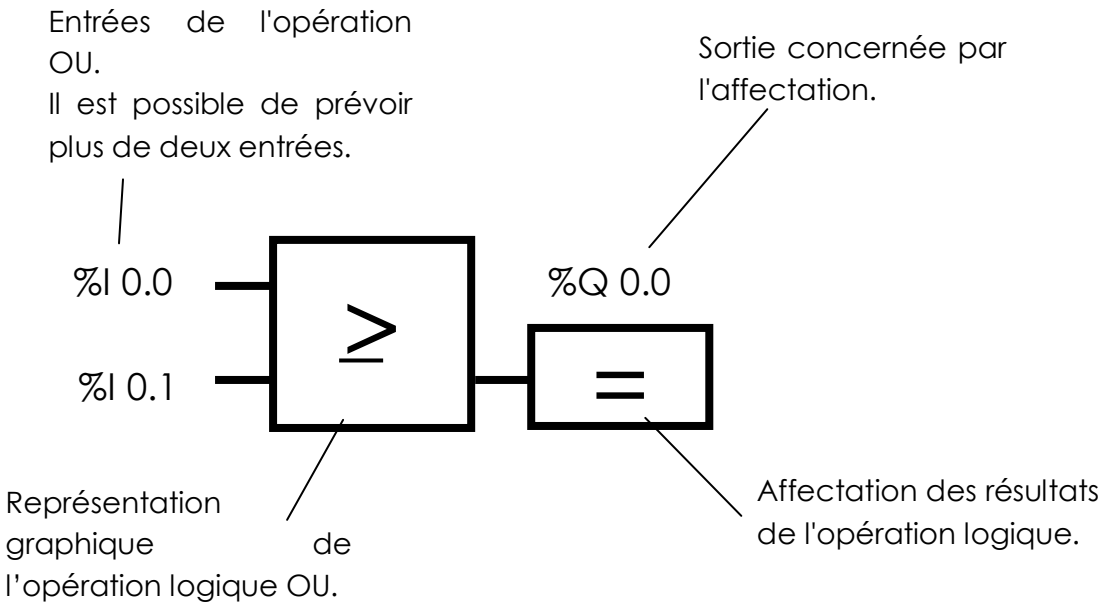
Câblage de l'API :

Pour appliquer cette opération au programme de l'API, les deux commutateurs doivent être connectés aux entrées de l'API. Ici, S1 est relié à l'entrée E 0.0 et S2 à l'entrée E 0.1. De plus, la lampe P1 doit être connectée à une sortie, par exemple A 0.0.



Opérateur OU dans LOG

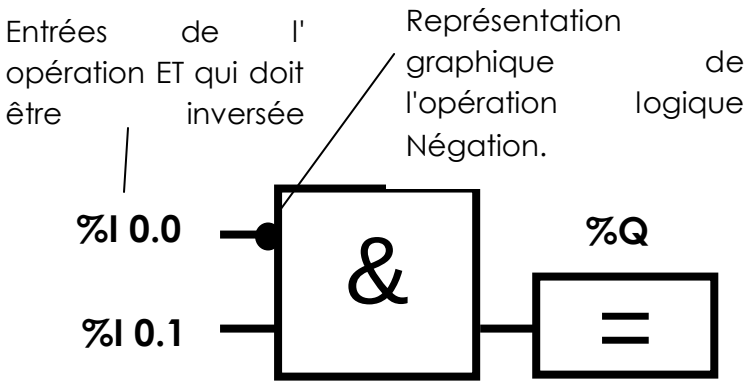
Dans le logigramme LOG, l'opérateur OU est programmé par le symbole ci-dessous et est représenté de la manière suivante :



II-2-3 Négation

Il est souvent nécessaire dans les opérations logiques d'interroger l'état d'un contact pour savoir : **dans le cas d'un contact à fermeture si celui-ci n'a pas été activé**, ou **dans le cas d'un contact à ouverture s'il a été activé**, et donc pour savoir si la tension est appliquée à la sortie ou non.
Ceci peut être réalisé par la programmation d'une **négation** à l'entrée de l'opération ET ou OU.

Dans le logigramme LOG, la négation de l'entrée (ou inversion) sur un opérateur ET est programmé de la façon suivante :



Ceci signifie qu'une tension est appliquée à la sortie %Q 0.0 uniquement si %I 0.0 est à 0 et %I 0.1 est à 1.