Langage LADDER

Ladder ou **schéma à contacts** est un langage graphique pour programmer les Automates Programmables Industriels. Il ressemble aux schémas électriques, et est facilement compréhensible.

Principe

Un programme Ladder se lit de haut en bas et l'évaluation des valeurs se fait de gauche à droite. Les valeurs correspondent en fait, si on le compare à un schéma électrique, à la présence ou non d'un potentiel électrique à chaque nœud de connexion.

Le Ladder est basé sur le principe d'une alimentation en tension représentée par deux traits verticaux reliée horizontalement par des bobines, des contacts et des blocs fonctionnels, d'où le nom 'Ladder' (échelle).

Les composants du langage

Il existe 3 types d'élément de langage :

- les entrées (ou contact), qui permettent de lire la valeur d'une variable booléenne ;
- les sorties (ou bobines) qui permettent d'écrire la valeur d'une variable booléenne ;
- les blocs fonctionnels qui permettent de réaliser des fonctions avancées.

Les entrées (ou contacts)

Il existe deux types de contact :

- Le **contact normalement ouvert** (NO) (en: NO normally open) :

```
   X
--| |--
```

  Ce contact est fermé lorsque la variable booléenne associée (X ici) est vraie, sinon, il est ouvert.

- Le **contact normalement fermé** (NF) (en: NC normally closed) :

```
   X
--|/|--
```

  Ce contact est ouvert lorsque la variable booléenne associée (X ici) est vraie, sinon il est fermé.

Les sorties (ou bobines)

Il existe, de même que pour les contacts, deux types de bobines :

- la bobine normalement ouverte (NO) (en: NO normally open) :

```
   X
--( )--
```

  Si il existe un circuit fermé reliant cette bobine des deux côtés du potentiel, alors la variable booléenne associée (X ici) est mémorisée à 'vraie', sinon elle est mémorisée à 'fausse'.

- la bobine normalement fermée(NF) (en: NC normally closed) :

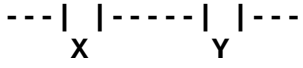
```
   X
--(/)--
```

  Si il existe un circuit fermé reliant cette bobine des deux côtés du potentiel, alors la variable booléenne associée (X ici) est mémorisée à 'fausse', sinon elle est mémorisée à 'vraie'.

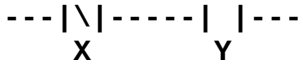
Réalisation de fonction logique

Comme dit précédemment, les fonctions logiques sont dérivées de leurs réalisations électriques. Donc chaque fonction logique (AND, OR, XOR, NAND, NOR, NOT) a une représentation qui correspond à son équivalent électrique.

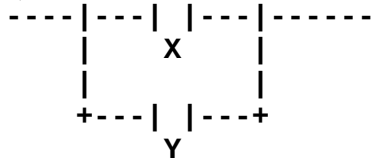
C'est-à-dire :



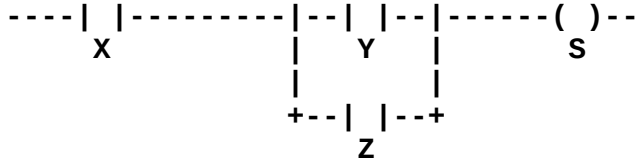
équivaut à X AND Y



équivaut à NOT(X) AND Y

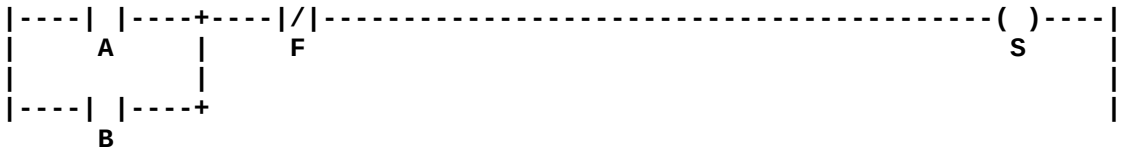


équivaut à X OR Y
Plus complexe :



équivaut à S = X.(Y+Z)

Exemple de lecture



Dans ce réseau, si A OU B est actionné ET si F n'est pas actionné, la sortie S est active; soit S = (A+B)./F

```
S := (A + B) . (/F) ;
```

Le signe "/F" signifie l'inversion de l'entrée "F", cela se prononce "F barre".