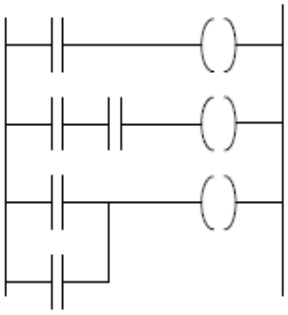
SEMAINE 4

LA PROGRAMMATION DES AUTOMATES

FICHE 20 : LES LANGAGES DE PROGRAMMATION

LE LANGAGE STL

Ladder



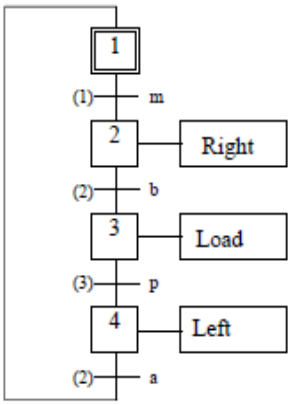
Structured Text (ST)

```
If %I1.0 THEN
  %Q2.1 := TRUE
ELSE
  %Q2.2 := FALSE
END_IF
```

Instruction List (IL)

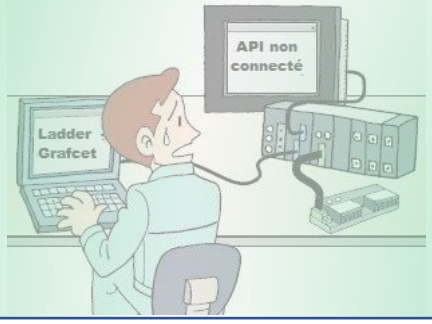
```
LD      %M12
AND     %I1.0
ANDN    %I1.1
OR      %M10
ST      %Q2.0
```

Grafcet



Automation & Sense

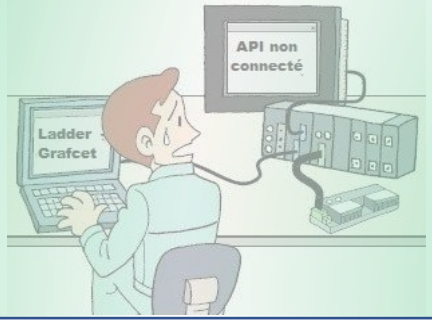
Novembre 2017 | www.automation-sense.com



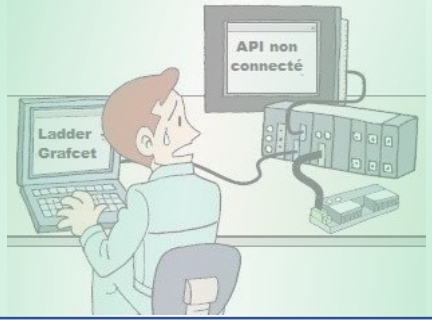
Objectifs :

Après la consultation de cette fiche, vous :

- Pourrez décrypter un programme conçu en langage STL (Statement List)
- Pourrez concevoir un programme en langage STL



SOMMAIRE



INTRODUCTION

LE **STL (Statement List)** est un langage de programmation bas niveau très proche du Hardware (matériel) : c'est comme l'assembleur en informatique dans lequel on manipule divers registres pendant la programmation.

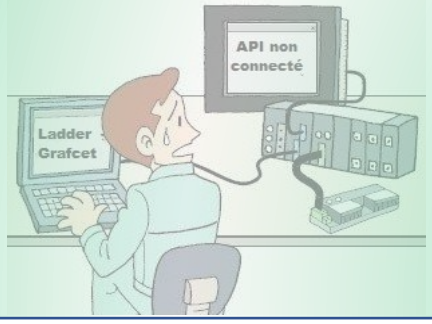
Le STL est une variante du langage **Instruction List (IL)** qui est un langage standard IEC.On retrouve le STL principalement au niveau des automates Siemens.En effet, en fonction de la marque de l'automate,vous ne retrouverez pas les mêmes instructions. Par exemple le langage IL utilisé pour les automates Omron est différent du langage IL que l'on retrouve chez les automates Siemens.

Comme le langage ST (Structured Text) que l'on a vu dans la fiche précédente, le langage STL est aussi composé d'une série d'instructions. Une instruction est un bout de code qui permet de réaliser une action bien définie.

Ci-dessous une instruction :

L %IW 12

Une astuce pour apprendre facilement le langage STL est de traduire un programme Ladder en langage STL. Vous pouvez le faire avec le logiciel TIA Portal en changeant le langage du bloc à programmer au niveau de ses propriétés. C'est ce que nous allons faire sur cette fiche : nous allons apprendre par la pratique.



I) Les opérateurs logiques binaires

Les opérateurs logiques binaires permettent d'établir les relations entre les différentes variables logiques d'une fonction logique. Les principaux opérateurs logiques utilisés en langage STL sont :

- **AND** (ET logique)
- **OR** (OU logique)
- **Exclusive OR** (OU Exclusif)

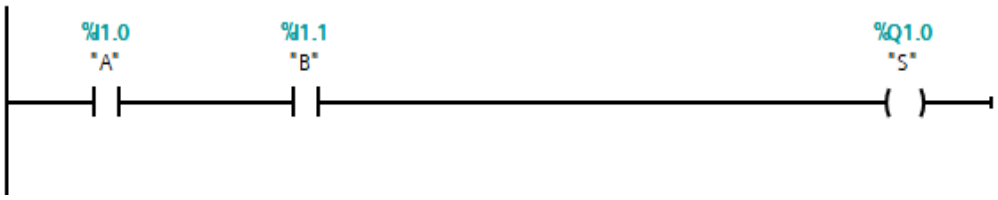
a) Le ET logique | Mnémonique A

Soit la fonction logique suivante :

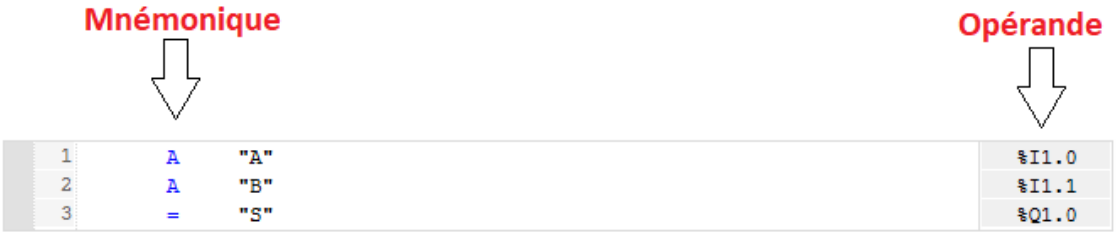
S = A . B

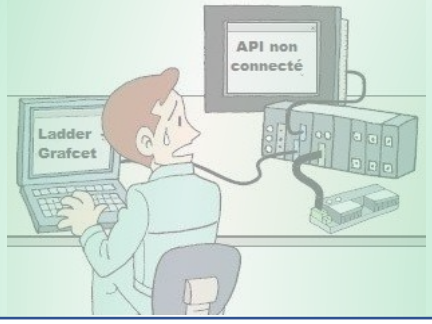
Ici l'opérateur logique utilisé est : le **« ET logique »** .

La sortie S est activée si le résultat de l'expression logique **A.B = 1** .Au niveau de l'automate, le résultat de l'expression A.B est stocké dans un bit appelé **RLO (Result of Logic Operation).** C'est ce RLO qui permet d'activer ou non la sortie S.



L'équivalent en langage STL du programme ci-dessus est :





Ici, les mnémoniques **A** correspondent à l'opérateur logique AND et le mnémonique **=** est un symbole d'affectation.

Le RLO **A.B** est donc affecté à la sortie **%Q1.0**.En d'autres termes cela signifie que la sortie S n'est active que si les entrées A et B sont simultanément actives

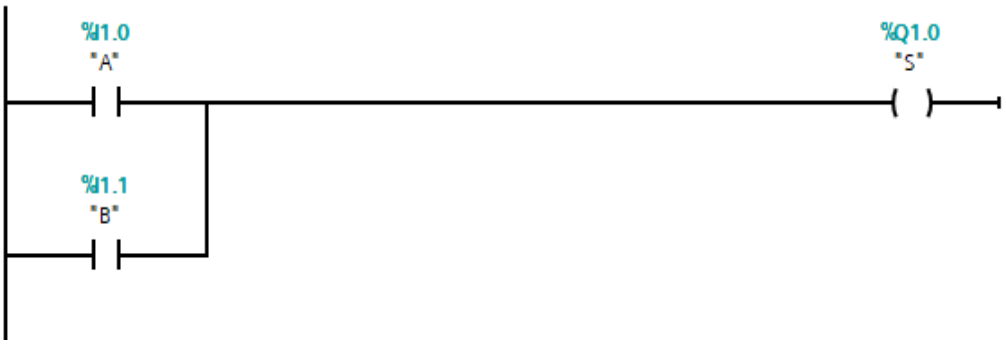
Vous pouvez maintenant vous poser la question suivante :

Pourquoi a-t-on **A %I1.0 ?**

En effet, il n'y a pas de contact avant **%I1.0** donc normalement on ne devrait pas avoir A %I1.0.

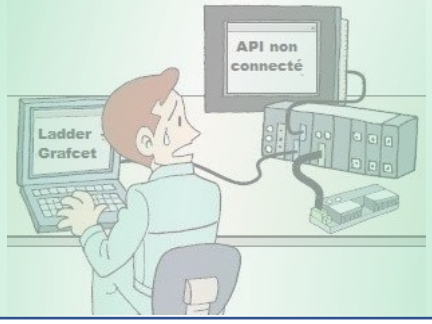
Explication : On a **A %I1.0** parce que c'est une norme de notation. Etant donné que A et B sont liés par l'opérateur logique AND, on utilise AND comme mnémonique pour les deux contacts. Si les deux contacts étaient liés par l'opérateur logique OU, on utiliserait celui-ci pour les deux contacts comme vous pouvez le constater ci-dessous pour le « OU logique ».

b) Le OU logique | Mnémonique O

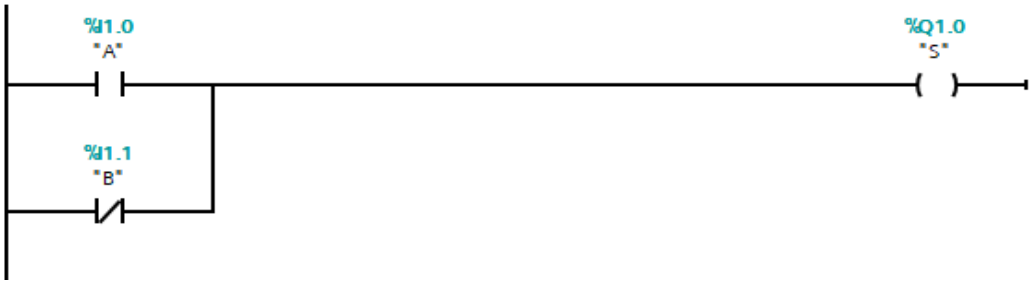


Traduction du Ladder ci-dessus en STL

```
1     O     "A"     %I1.0
2     O     "B"     %I1.1
3     =     "S"     %Q1.0
```



Un contact fermé se traduit tout simplement par une négation. On aura **ON** au lieu de **O** comme vous pouvez le constater sur le schéma ci-dessous.




```
1      O     "A"      %I1.0
2      ON    "B"      %I1.1
3      =     "S"      %Q1.0
```

c) Le OU Exclusif | Mnémonique X

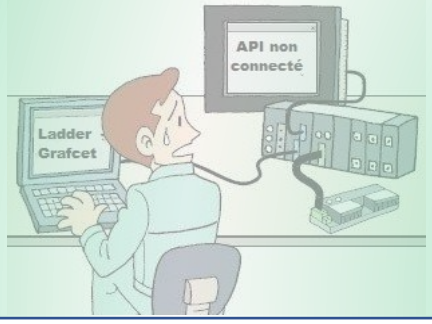
Avec le OU Exclusif, le RLO est à 1 quand les deux entrées sont différentes. Si les deux entrées sont les mêmes, le RLO sera à 0.

d) Le Non logique | Mnémonique NOT

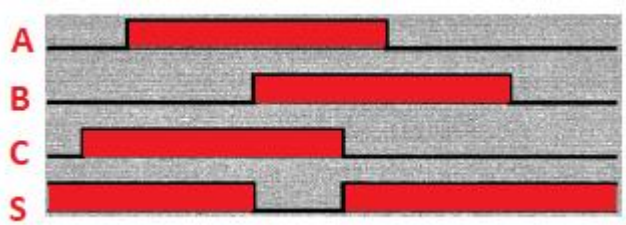
Il permet d'inverser le résultat d'un RLO (là où on avait 1 on aura 0 et là où on avait 0,on aura 1).

A *A*
A *B*
A *C*
NOT
= *S*

Au niveau du programme ci-dessus,le mnémonique NOT permet d'inverser le RLO (A.B.C).



Chronogramme OU Exclusif

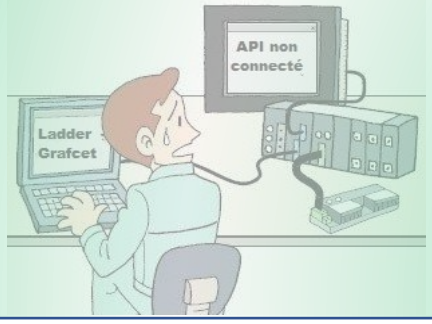


II) Les expressions logiques

Au niveau des équations logiques, on retrouve généralement plusieurs opérateurs logiques à la fois. Dans certains cas, un opérateur logique peut avoir la priorité sur un autre opérateur logique. C'est pourquoi on utilise les parenthèses pour regrouper certaines expressions logiques prioritaires au sein d'une équation logique.

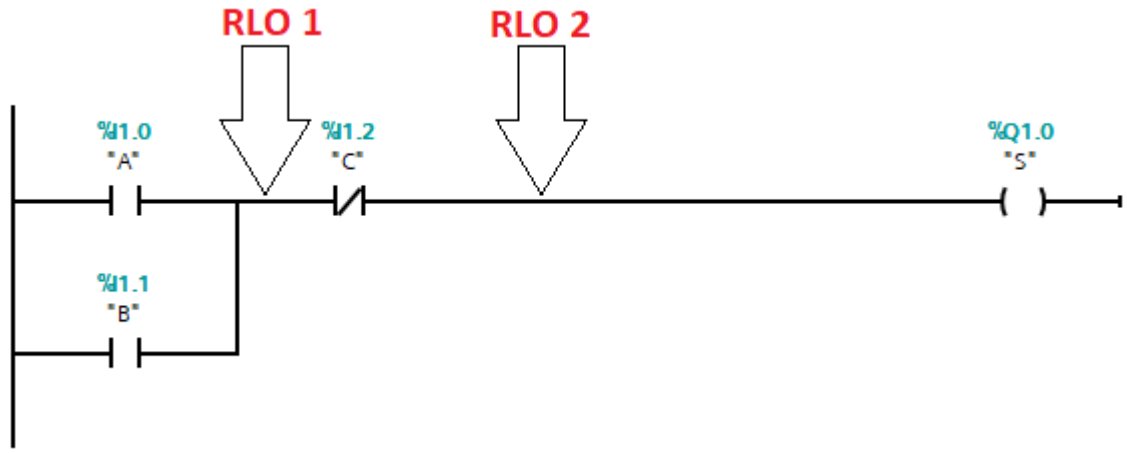
Ainsi, on pourra rencontrer les mnémoniques suivantes :

A(
O(
X(
AN
ON
XN
)



On va prendre un petit exemple afin de mieux comprendre le principe.

Dans le programme Ladder ci-dessous, l'évaluation de l'expression **(A + B)** donne le **RLO 1**, le RLO final (**RLO 2**) est obtenu en faisant (**RLO 1 . /C**)



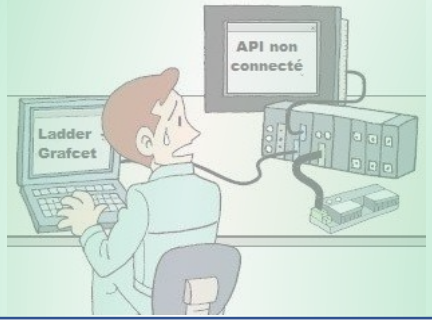
Ainsi, le programme STL du Ladder ci-dessus est la suivante :

```
1       A(
2       O       "A"                     %I1.0
3       O       "B"                     %I1.1
4       )
5       AN      "C"                     %I1.2
6       =       "S"                     %Q1.0
```

III) Les fonctions mémoires

a) L'affectation

Symbolisé par le mnémonique **=**, elle permet d'affecter un RLO à une variable logique.



Dans le langage STL, on peut avoir deux affectations qui se suivent.
Voir exemple ci-dessous :

```
A   I1
A   I2
=   Q1
=   Q2
```

b) Le SET/RESET

Le **SET** met la sortie à 1 quand le RLO est à 1
Le **RESET** met la sortie à 0 quand le RLO est à 1

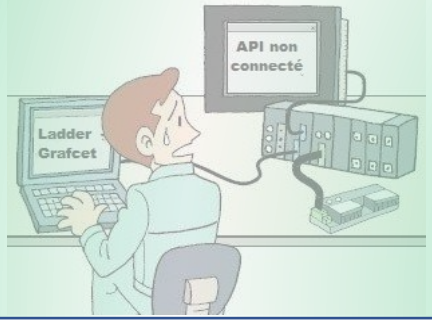
Les instructions SET et RESET ne sont exécutées que quand le RLO est à 1.

Exemple de programme utilisant SET et RESET :

```
A   I1
S   Q1
A   I2
R   Q1
```

IV) Les fonctions de transfert

Dans cette partie, nous allons étudier les fonctions du langage STL qui interagissent avec les accumulateurs (registres).Nous verrons principalement les fonctions **Load** et **Transfert**.



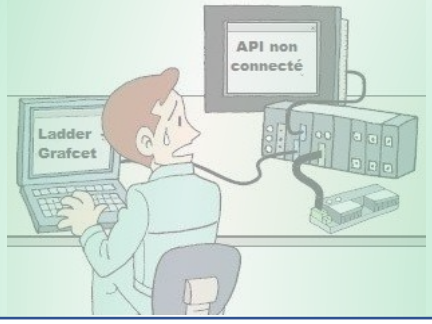
a) La fonction Load (chargement) | Mnémonique L

La fonction **Load** permet l'échange de données entre différentes zones mémoire. Pendant les échanges de données, celles-ci sont chargées dans des accumulateurs. Un accumulateur est un registre spécial qui se trouve au niveau du processeur de l'automate et qui sert de mémoire tampon. Quand l'échange des données se fait de la zone mémoire vers **l'accumulateur 1**, on appelle cela **chargement**. Par contre quand l'échange des données s'effectue de **l'accumulateur 1** vers la zone mémoire, on appelle cela **transfert**.

Les fonctions Load et Transfert sont très couramment utilisées, elles permettent par exemple d'initialiser des timers et des compteurs, de faire des conversions de données, des additions, des comparaisons, des décalages, de transférer des données de la mémoire de travail ou la mémoire système vers l'accumulateur 1 (cas de la fonction Load) etc…

Dans la plupart des CPU des automates Siemens, on retrouve deux accumulateurs nommés **accumulateur 1 (ACCU1)** et **accumulateur 2 (ACCU2)**.Sur les gammes d'API S7-400, on retrouve deux accumulateurs supplémentaires nommés **accumulateur 3** et **accumulateur 4**.Ces accumulateurs sont des registres de 32 bits qui jouent le rôle de registres tampons.

La fonction Load permet de charger des valeurs constantes, des variables et des adresses dans l'accumulateur 1.Les données contenues dans l'accumulateur 1 sont transférés vers l'accumulateur 2 par la suite.



Exemple de programme utilisant la fonction Load

```
L    +1200          // chargement de la valeur +1200 dans ACCU 1
L    -50            //chargement de la valeur -50 dans ACCU 1
L    IW16           // chargement IW16  dans ACCU1
L    capteur1       //chargement variable capteur1 dans ACCU1
L    MB1            //chargement momento MB1 dans ACCU 1
L    5.0            //chargement nombre Réel 5 dans ACCU 1
```

Il faut cependant signaler qu'on ne peut pas charger un pointeur de DB dans l'accumulateur vu qu'il dépasse 32 bits.

b) La fonction Transfert | Mnémonique T

Symbolisé par la mnémonique **T**, elle permet de transférer des données de l'accumulateur 1 vers les zones mémoires spécifiées. Les données transférées peuvent être un octet, un Word ou un Double Word.

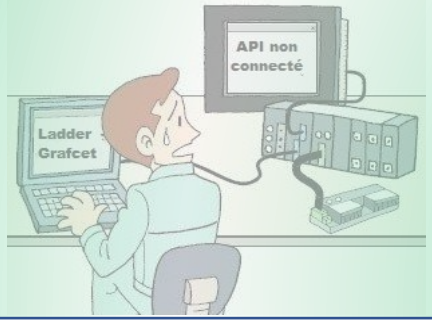
```
T    MB1            //transfert du contenu de ACCU1 vers MB1
T    IW16           //transfert du contenu de ACCU1 vers IW16
T    QB1            //transfert du contenu de ACCU1 vers QB1
```

V) Les fonctions mathématiques

a) Les fonctions de comparaison

Elles permettent de comparer des INT,DINT et REAL. Pour comparer deux valeurs, la syntaxe suivante est utilisée :

```
L    valeur1
L    valeur2
Opérateur de comparaison
=    résultat
```



Lors de la comparaison, **valeur1** est chargé dans l'ACCU1, lorsque **valeur2** est chargée à son tour, le contenu de l'ACCU1 est décalé dans l'ACCU2.Par la suite les deux valeurs contenues dans les deux accumulateurs sont comparées en utilisant l'opérateur de comparaison en fonction des valeurs à comparer.

	Comparaison de types de données		
Type de données	**INT**	**DINT**	**REAL**
Egal à	==I	==D	==R
Différent de	<>I	<>D	<>R
Supérieur à	>I	>D	>R
Inférieur à	<I	<D	<R
Supérieur ou égal à	>=I	>=D	>=R
Inférieur ou égal à	<=I	<=D	<=R

Le résultat de la fonction de comparaison est une valeur booléenne qui pourra être affectée à une adresse booléenne.

Exemple :

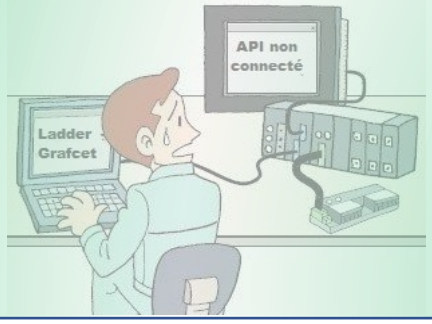
L MW92

L 120

==I

R M 99.0

Ici **M99.0** sera reseté si la valeur contenue dans **MW92** est égale à **120**,sinon rien ne se passe.



b) Les fonctions arithmétiques

Les fonctions arithmétiques permettent d'effectuer des opérations arithmétiques sur les valeurs contenues dans les accumulateurs 1 et 2.Le résultat de ces opérations sont transférés par la suite au niveau de l'accumulateur 1.La syntaxe suivante est utilisée :

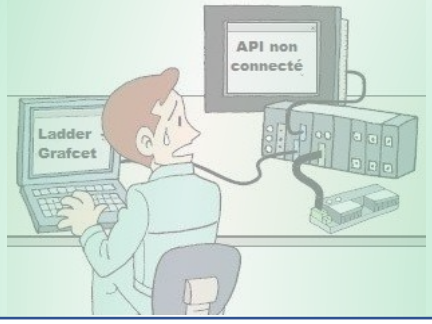
L valeur1
L valeur2
Opérateur arithmétique
T résultat

	INT	**DWORD**	**REAL**
Addition	+I	+D	+R
Soustraction	-I	-D	-R
Multiplication	*I	*D	*R
Division	/I	/D	/R
Modulo	-	MOD	-

Exemple 1 :

L MW100
L 250
/I
T MW102

La valeur contenue dans MW100 est divisée par 250.Le résultat est transféré vers MW102



Exemple 2 :

L valeur1
+100
T resultat

On ajoute 100 à **valeur1**.Le résultat est transféré dans la variable **resultat**.

Exemple 3 :

L valeur1
INC 5
T resultat

La variable **valeur1** est incrémentée de 5 puis transférée dans **resultat**

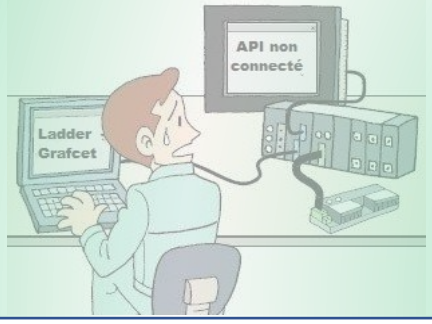
Exemple 4 :

L valeur1
DEC 5
T resultat

La variable **valeur1** est décrémentée de 5 puis transférée dans **resultat**

VI) Les fonctions de conversion

ITD convertit un INT en DINT
ITB convertit un INT en BCD
DTB convertit un DINT en BCD
DTR convertit un DINT en REAL



Exemple :

L MW120
ITB
T MW122

VII) Les fonctions de décalage

Elles permettent de décaler le contenu de l'accumulateur 1 bit par bit vers la gauche ou vers la droite. Le contenu de l'ACCU 1 doit être un Word ou un DWORD.

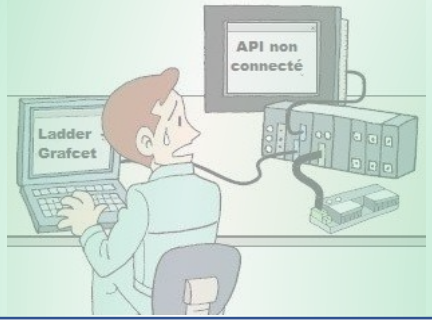
SLW : décalage vers la gauche d'un Word
SRW : décalage vers la droite d'un Word
SLD : décalage vers la gauche d'un DWORD
SRD : décalage vers la droite d'un DWORD

VIII) Les fonctions de saut

Les sauts permettent de sauter certaines parties d'un programme pour aller exécuter une autre partie. Il existe plusieurs types de sauts qui sont symbolisés par les fonctions suivantes :

JU label : saut inconditionnel
JC label : saut si RLO = 1
JCN label : saut si RLO = 0
LOOP label : saut en boucle
Etc…

Ici label correspond à l'endroit où le programme doit aller si la condition de saut est remplie. Un label est une chaine pouvant contenir jusqu'à 4 caractères alphanumériques incluant l'underscore. Un label ne doit pas débuter par un



chiffre. Suivi de **« deux points » ,**il sert à indiquer la ligne à exécuter si la condition est remplie.

Exemple :

L MW100
L 50
>I
JC GR50
Programme à exécuter si condition non remplie
GR50 : Programme à exécuter si condition remplie

IX) La gestion des blocs de données

OPN bloc : permet d'ouvrir le bloc DB spécifié
UC bloc : permet d'appeler le bloc FB ou FC spécifié

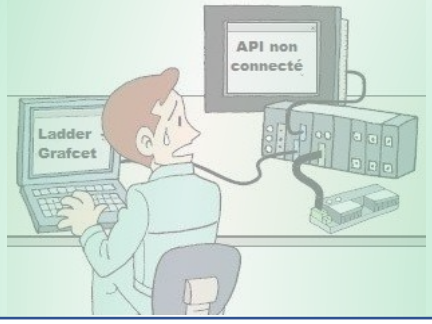
Exemple :

L MW0
OPN DB10
T DBW[MD10]

X) Les registres d'adresse

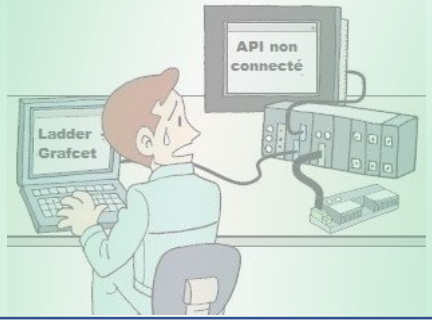
Outre les accumulateurs **ACCU 1** et **ACCU 2**, il existe deux registres de 32 bits (**AR1 et AR2**) qui permettent de stocker des pointeurs d'adresse utilisés lors des adressages indirects.

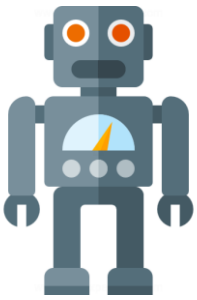
LAR1 : charge AR1 avec le contenu de l'ACCU1
LAR1 P#M100.0 : charge AR1 avec un pointeur de constante P#M100.0
LAR1 MD24 : charge AR1 avec le pointeur MD24
LAR1 AR2 : charge AR1 avec le contenu de AR2



TAR1 : transfert le contenu de AR1 dans ACCU1

+AR1 : ajoute le contenu de **ACCU1** à **AR1** et enregistre le résultat dans **AR1**





Dans cette fiche, vous avez pu découvrir le langage Statement List (STL)

Vous pourrez désormais concevoir avec un peu de pratique des programmes en STL sur TIA Portal ou Simatic Manager Step7