# Initiation à PL7 (Micro/Junior/Pro)

# Initiation à PL7 (Micro/Junior/Pro)

## 1 - GENERALITES.

Ce logiciel est conforme à la norme IEC 1131-3.

Il permet entre autres, d'écrire des applications pour les automates TSX37 et les automates TSX57. La version PL7 PRO comporte quelques fonctionnalités supplémentaires par rapport à PL7 Micro et Junior. Dans les trois cas, plusieurs langages de programmation sont utilisables :

- Le **langage** graphique **à relais** LD (appelé aussi langage à contacts ou "ladder"), qui utilise des contacts, des bobines et des blocs (temporisations, compteurs, etc.). L'écriture de calculs numériques est possible dans des blocs opérations ;
- Le **langage Booléen** (IL) sous forme d'une liste d'instructions qui permet d'écrire des traitements logiques et numériques (ressemble à la programmation d'un micro-contrôleur) ;
- Le **langage littéral structuré** (ST) ressemble à un langage d'informatique évolué (Junior/Pro) ;
- Le **grafcet** .

Le programme réalisé peut avoir une structure monotâche ou multitâche.

- **Structure monotâche** : C'est la structure par défaut. Elle comporte seulement une tâche maître exécutée cycliquement (mode par défaut) ou périodiquement (période fixée par l'utilisateur) ;
- **Structure multitâche** : A utiliser quand le processus commandé comporte des priorités d'exécution différentes. Elle comprend la tâche maître, la tâche rapide (facultative) et une ou plusieurs tâches événementielles. La tâche rapide, exécutée périodiquement, permet d'effectuer des traitements courts avec une priorité plus élevée que pour la tâche maître. Lorsqu'elle est programmée, elle est automatiquement lancée par le système au démarrage. Elle peut être arrêtée et redémarrée par action sur un bit système. Les tâches événementielles ne sont pas liées à une période comme les tâches précédemment décrites. Leur exécution est déclenchée par un appel en provenance de certains modules. Ces tâches sont les plus prioritaires. Le traitement qu'elle doivent assurer est nécessairement court afin de ne pas perturber l'exécution des autres tâches.

Les tâches du programme sont elles même subdivisées en plusieurs parties appelées **sections**, et en **sous-programmes**. Les sous-programmes peuvent être appelés à partir de n'importe quelle section de la tâche à laquelle ils appartiennent ou depuis d'autres sous-programmes de la même tâche.

Les instructions de PL7-MICRO comprennent des **instructions de base** (celles que nous utiliserons en T.P.) et des **instructions avancées**.

## 2 - PROGRAMMATION.

Lors de la programmation, on peut appeler les variables par leur nom (ex : %Q2.5) mais il est possible de leur donner une dénomination appropriée, composée d'une chaîne de 32 caractères au maximum (ex : sortie_3). L'éditeur de variables permet d'établir cette liste d'assignation avec un commentaire pour chaque variable.

**Objets adressables.**

D'une manière générale, la lettre X signifie "booléen", W signifie "mot" et D signifie "double mot".
**Bits images des entrées : %IXn.i** (On peut omettre le X).
Le nombre n est le numéro de l'emplacement de la carte d'entrée dans le « fond de panier » de l'automate.

i est le numéro de la voie d'entrée.

exemples :

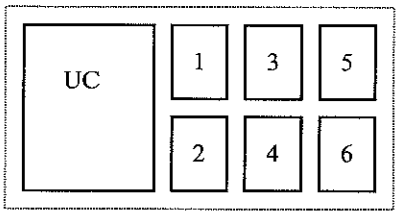
%IX1.3 voie numéro 3 de la carte d'entrée située à l'emplacement 1.
%IW1 Le mot de 16 bits image des 16 entrées de la carte située à l'emplacement 1.

**Bits images des sorties : %QXn.i** (On peut omettre le X).
Le nombre n est le numéro de l'emplacement de la carte de sortie dans le « fond de panier » de l'automate.
i est le numéro de la voie de sortie.

exemples :

%QX3.5 voie numéro 5 de la carte de sortie située à l'emplacement 3.

Les emplacements dans le bac des TSX37 ou 57 sont "demi-format". Leurs positions sont schématisées ainsi :




Les 16 entrées disponibles de l'automate TSX37 utilisé en TP seront : %I1.0, ..., %I1.15 et les 12 sorties seront : %Q2.0,..., %Q2.11 (la carte 16 entrées 12 sorties est au premier emplacement).

**Bits internes : %Mi** (de %M0 à %M255)

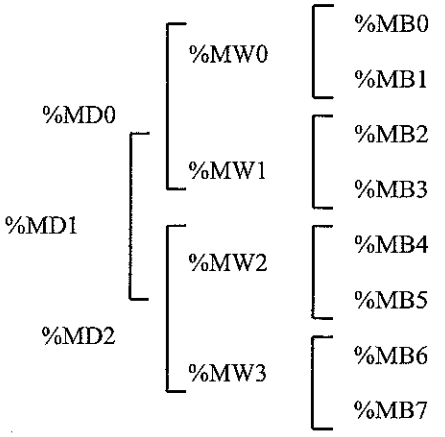
**Bits système : %Si** (de %S0 à %S127)

**Bits images des étapes de Grafcet** : %Xi (de %X0 à %X127)

**Mots :** Il existe des mots internes destinés à stocker des valeurs en cours de programme, des mots constants mémorisant des valeurs numériques ou des données alphanumériques, et des mots système.
Ces mots sont traités différemment lors d'une reprise secteur, selon la configuration de l'automate.

Exemples de mots :
%MB1 octet interne n°1
%MW3 mot interne n°3
%MD8 mot interne double n°8
%MF12 mot interne n°12, en format flottant (32 bits).
%KB1 octet constant N°1
%KW3 mot constant n°3
%SW15 mot système n°15

Les mots sont stockés dans une mémoire RAM différente de celle qui contient les bits (différence avec Siemens). Le nombre de mots utilisables varie selon la configuration mémoire choisie pour l'automate. La RAM est secourue en cas d'arrêt de l'alimentation.
Attention aux "recouvrements" qui peuvent avoir lieu en cas de programmation maladroite :




%MW0 : poids faibles de %MD0
%MW1 : poids forts de %MD0

%MWi se subdivise en deux octets appelés %MB2i et %MB2i+1 (poids faibles et poids forts).

**Bits extraits de mots** : %MWi:Xj
Exemple : %MW0:X12 est le bit n°12 du mot %MW0. Si le mot %MW0 a reçu un nom alphanumérique sur la liste d'assignation, par exemple "température", le bit %MW0:X12 s'appellera alors : "température:12".
(Ne pas confondre cette notation avec celle des tableaux – voir page 26).

# 3 - STRUCTURE D'UN PROGRAMME MONOTACHE.

Ici, une seule tâche appelée **tâche maître** (MAST). Le programme comprend un **traitement principal** (lui même subdivisé en sections) et des **sous-programmes**.

Chaque section est programmée comme une entité séparée. Elle peut être écrite en langage LD, IL , ST ou Grafcet. Avant de l'écrire, il faut préciser :
- son nom (24 caractères maximum) ;
- le langage dans lequel elle est programmée ;
- la tâche à laquelle elle appartient ;
- une condition d'exécution optionnelle (%M, %X, %S, %I, %Q ...) ;
- un commentaire facultatif (250 caractères) ;
- une protection facultative (en lecture, en lecture/écriture).

Une section est une entité autonome. Les étiquettes de repérage des lignes d'instruction ou des réseaux de langage à relais, sont propres à la section (pas de saut de programme possible vers une autre section).

Les sections sont exécutées dans leur ordre de programmation dans la fenêtre du navigateur.

Une section programmée en Grafcet comprend :

- le traitement préliminaire (PRL) écrit en LD, IL ou ST. Il est exécuté avant le Grafcet.
- le Grafcet (CHART). Dans les pages Grafcet, sont programmées les réceptivités associées aux transitions et les actions associées aux étapes.
- le traitement postérieur (POST) écrit en LD, IL ou ST. Il est exécuté après le Grafcet.

Les sous-programmes SRi (i de 0 à 253) se programment comme des entités séparées. Les appels aux sous-programmes s'effectuent dans PRL, dans POST, dans les actions associées aux étapes et même depuis d'autres sous-programmes (8 imbrications maximum).

**Le cycle d'exécution de l'automate :**

L'automate étant en "RUN", les opérations suivantes s'exécutent dans l'ordre, d'une façon cyclique :

1) Traitement interne (bits systèmes, horodateur, requêtes en provenance de la console de programmation).
2) acquisition des entrées.
3) traitement du programme.

4) mise à jour des sorties.

Ce type de fonctionnement correspond à l'exécution normale du cycle automate (choix par défaut). Le chien de garde (réglé à 250 ms par défaut) vérifie que la durée du cycle ne dépasse pas cette limite (sinon, arrêt immédiat de l'automate). Il existe aussi un autre mode de fonctionnement dit "périodique" . Les traitements ci-dessus ont alors lieu selon une périodicité choisie par l'utilisateur (1 à 255 ms). Il est possible de connaître la durée du temps de cycle automate (programmation avancée).

## 4 - STRUCTURE D'UN PROGRAMME MULTITACHE (quelques généralités).

Cette structure comprend :

La tâche maître (MAST) pouvant être cyclique ou périodique. Elle est structurée en sections et peut contenir une section écrite en Grafcet.

La tâche rapide (FAST) optionnelle et toujours périodique. Elle est structurée en sections mais ne peut pas contenir une section écrite en Grafcet.

Les traitements sur événement EVTi, lancés par le système dès l'apparition d'un événement sur des entrées définies à l'avance (une seule section dont le nom n'est pas modifiable).

La tâche rapide (FAST) est prioritaire sur la tâche maître et les traitements sur événement son prioritaires sur la tâche rapide.

Des entrées et sorties devront être affectées préalablement aux tâches maître et rapide. Des bits systèmes permettent de valider ou d'invalider le fonctionnement de ces différentes tâches (%S30, %S31, %S38).

# 5 - Le langage à relais LD (ou langage à contacts ou « ladder »).

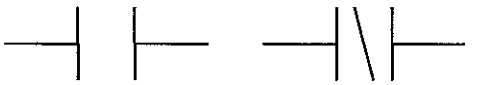
C'est le langage le plus connu car le plus ancien. Il se compose d'une suite de "réseaux" exécutés en séquence par l'automate. Le logiciel PL7-..... utilise un éditeur de langage à contact, organisé en pages de 7 lignes et 11 colonnes.

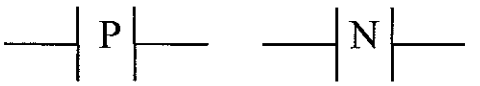
On y utilise des contacts (entrées), des bobines (sorties), des temporisateurs, des compteurs, des boites de calculs, etc.. Ces éléments graphiques sont reliés entre eux par des connexions horizontales et verticales.

Chaque réseau porte une étiquette qui à la syntaxe suivante : %Li (i, de 0 à 999).
Une étiquette ne peut être affectée qu'à un seul réseau au sein d'une même entité de programme. L'ordre des étiquettes est quelconque.

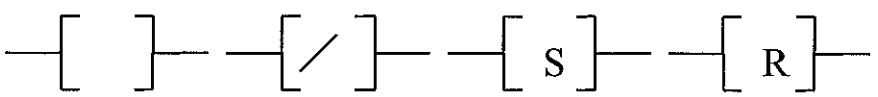
Les réseaux sont lus non pas dans l'ordre des étiquettes, mais dans l'ordre de leur programmation, et, de gauche à droite.

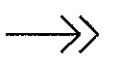
On appelle "zone de tests" la partie gauche du schéma (colonnes 1 à 10) où se trouvent les contacts, et on appelle "zone d'actions" la partie droite du schéma (colonne 11) où se trouvent les bobines.

Eléments de tests :  contacts simples droit et inverse.

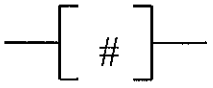
 contacts de front.

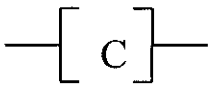
Les deux contacts ci-dessus détectent respectivement les fronts montants (P) et les fronts descendants (N). *Attention*, à leur fonctionnement :
- pour toutes les entrées T.O.R., un front est détecté lorsque l'état du bit a changé entre le cycle n-1 et le cycle n, en cours. *Ce front reste détecté pendant tout le cycle en cours.*
- pour un bit interne %Mi, la détection du front est indépendante du cycle de la tâche. Le front est détecté lorsque l'état de %Mi a changé entre deux lectures. *Ce front reste détecté* tant que ce bit %Mi *n'est pas scruté en zone action.*
- *ne jamais effectuer de SET ou RESET sur un objet dont on teste le front.*

 bobines normale, inverse, SET et RESET

 %Li branchement à un autre réseau amont ou aval.

Les sauts ne peuvent être effectués qu'au sein d'une même entité (programme principal, sous programme). Ils provoquent l'exécution du réseau dont l'étiquette est demandée et la non scrutation du programme situé entre la demande de saut et le réseau désigné.

 bobine "dièse" utilisée pour la programmation des réceptivités d'un grafcet.

 appel à un sous programme (CALL)

 se place à la fin d'un sous programme

Le langage à relais utilise aussi des blocs fonctions standards : temporisateurs, compteurs, monostables, blocs opérations (voir ci-dessous).

A chaque réseau, on peut associer un commentaire (222 caractères alphanumériques au maximum) encadré par des astérisques. Les commentaires sont mémorisés dans l'automate, et, à ce titre, ils consomment de la mémoire programme.

## 5-1 Bloc fonction TEMPORISATEUR.

**Nom** : %TMi (63 au maximum pour un TSX37, selon base de temps).

**Mode** : Au choix TON, TOF et TP.

**Base de temps** : TB (1 mn par défaut, 1s, 100ms, 10ms. 16 temporisateurs au maximum si la base de temps est 10 ms).

**Valeur courante** : %TMi.V mot qui croit de 0 à %TMi.P sur écoulement du temporisateur. Ce mot peut être lu, testé, mais non écrit par programme.

**Valeur de présélection** : %TMi.P mot qui peut être lu, testé et écrit par programme. Il est mis à la valeur 9999 par défaut. La durée ou le retard élaboré est %TMi.P x TB.

**Y/N** : réglage possible ou non, par la console.

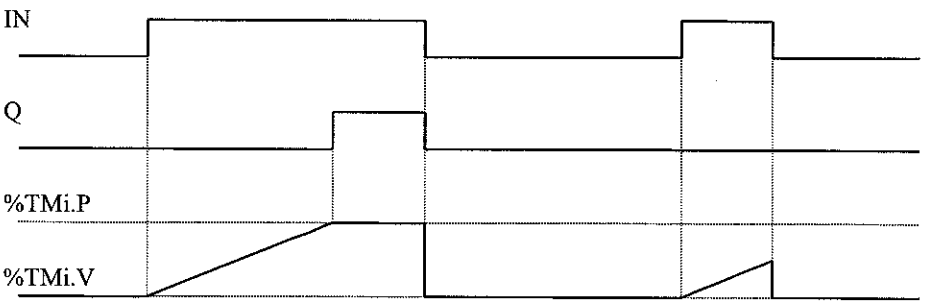
**IN** : entrée "armement" sur front montant (en TON ou TP) et sur front descendant (en TOF).

**Q** : sortie du temporisateur. Le bit associé est %TMi.Q.

### 5-1-1 Utilisation en mode "TON" :

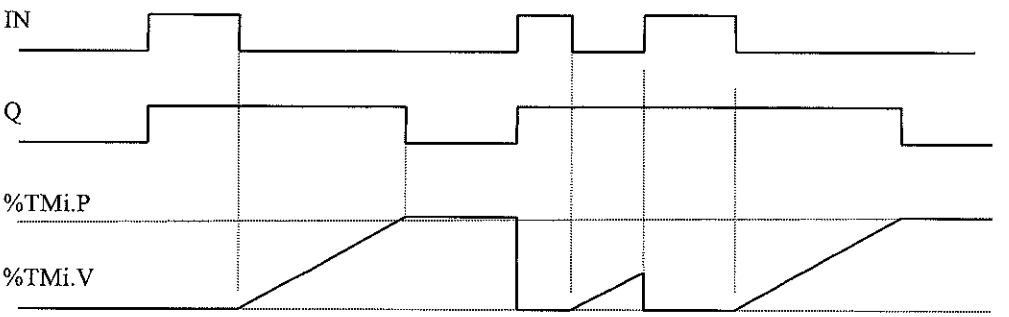
Lors d'un front montant sur l'entrée IN, le temporisateur est lancé. Sa valeur courante %TMi.V croit de 0 vers %TMi.P d'une unité, à chaque impulsion de la base de temps TB. Le bit de sortie %TMi.Q passe à l'état logique "1" dès que la valeur courante a atteint %TMi.P, puis reste à "1" tant que IN reste à "1".

Quand l'entrée IN est à "0", le temporisateur est arrêté. Si le temporisateur était en cours d'évolution, il est stoppé. %TMi .V prend la valeur 0.



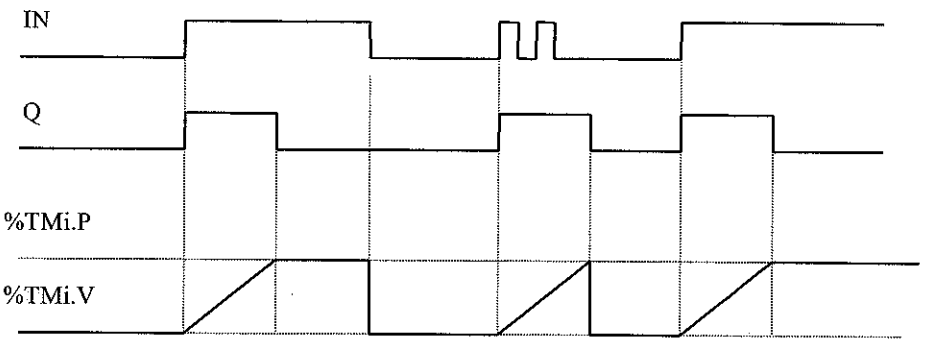

### 5-1-2 Utilisation en mode "TOF" :

La valeur courante %TMi.V prend la valeur 0 sur un front descendant de l'entrée IN, même si ce temporisateur est en cours d'évolution. Le temporisateur est lancé par le front descendant sur l'entrée IN. La valeur courante croît vers %TMi.P d'une unité à chaque impulsion de la base de temps TB. Le bit de sortie %TMi.Q passe à "1" dès qu'un front montant est détecté sur l'entrée IN et retombe à "0" quand la valeur courante atteint %TMi.P.




**5-1-3 Utilisation en mode "TP" (monostable) :**

Lors d'un front montant sur l'entrée IN, le temporisateur est lancé (s'il n'est pas déjà en cours d'évolution). %TMi.V croît de 0 vers TMi.P d'une unité, à chaque impulsion de la base de temps TB. Le bit de sortie %TMi.Q passe à "1" dès que le temporisateur est lancé et retombe à "0" quand la valeur courante a atteint %TMi.P. Quand l'entrée IN et la sortie %TMi.Q sont à "0", %TMi.V prend la valeur "0".




**Remarques générales :**

Une reprise "à froid" (%S0="1") remet la valeur courante à 0. Une reprise "à chaud" (%S1="1") ne modifie pas la valeur courante.

Le bit de sortie %TMi.Q ne doit être testé qu'une seule fois dans le programme.

## 5-2 Bloc fonction compteur-décompteur.

**Nom** : %Ci avec i de 0 à 31 pour un TSX 37.

**Valeur courante** : %Ci.V mot incrémenté ou décrémenté en fonction des entrées CU ou CD. Il peut être lu et testé mais non écrit par programme.

**Valeur de présélection** : %Ci.P 9999 par défaut. Ce mot peut être lu, testé et écrit par programme.

**Y/N** : présélection réglable ou non par console.

**R** : Si R = "1", %Ci.V = 0.

**S** : Si S = "1", %Ci.V = %Ci.P

**CU** : Incrémente %Ci.V sur front montant.

**CD** : Décrémente %Ci.V sur front montant.

**E** : Sortie "débordement" (*empty*). Le bit associé %Ci.E = "1" lorsque, en décomptage, %Ci.V passe de 0 à 9999 (Ce bit est mis à "1" au moment où Ci.V vaut 9999. Il est remis à "0" si le compteur continue de décompter).

**D** : Présélection atteinte. Le bit associé %Ci.D = "1" quand %Ci.V = %Ci.P

**F** : Sortie "débordement" (*full*). Le bit associé %Ci.F = "1" lorsque, en comptage, %Ci.V passe de 9999 à 0 (Ce bit est mis à "1" au moment où Ci.V vaut 0. Il est remis à "0" si le compteur continue de compter).

**Remarque** : Quand il y a débordement, en comptage ou en décomptage, le bit système %S18 passe à "1".

**Fonctionnement en comptage** : A l'apparition d'un front montant sur l'entrée de comptage CU, la valeur courante est incrémentée d'une unité. Lorsque cette valeur est égale à la valeur de présélection %Ci.P, le bit de sortie %Ci.D (présélection atteinte) associé à la sortie D, passe à l'état "1". Le bit de sortie %Ci.F (débordement comptage) passe à "1" lorsque %Ci.V passe de 9999 à 0. Il est remis à "0" si le compteur continue de compter.

**Fonctionnement en décomptage** : A l'apparition d'un front montant sur l'entrée de décomptage CD, la valeur courante est décrémentée d'une unité. Le bit de sortie %Ci.E (débordement décomptage) passe à "1" lorsque %Ci.V passe de 0 à 9999. Il est remis à "0" si le compteur continue de décompter.

**Comptage/Décomptage** : Pour utiliser simultanément les fonctions de comptage et de décomptage, il faut commander les deux entrées CU et CD. Ces deux entrées sont scrutées successivement. Si elles sont à "1" simultanément, la valeur courante reste inchangée.

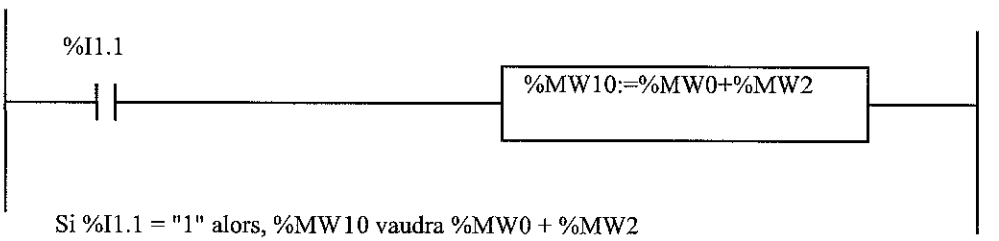
**Remise à zéro** : Dès la mise à "1" de R, la valeur courante %Ci.V est forcée à 0. Les sorties %Ci.E, %Ci.D et %Ci.F sont à "0". L'entrée R est prioritaire sur les autres.

**Présélection** : Si l'entrée S (présélection) est à "1" et l'entrée R à "0", la valeur courante %Ci.V prend la valeur %Ci.P, et le bit de sortie %Ci.D prend la valeur "1".

**Remarque** :
En cas de reprise "à froid" (%S0="1"), Ci.V=0, %Ci.E="0", %Ci.D="0", %Ci.F="0".
En cas de reprise "à chaud" (%S1="1") ou de passage en mode "STOP", pas d'incidence sur %Ci.V.

## 5-3 Blocs **OPERATIONS** ("OPERATE") et **COMPARAISONS** ("COMPARE").

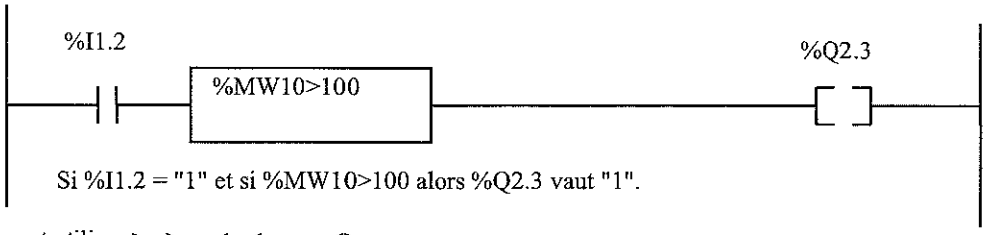
**Traitements numériques sur des nombres entiers :** (en zone opérations).




Si %I1.1 = "1" alors, %MW10 vaudra %MW0 + %MW2

On peut écrire des opérations de la forme : OP5:=(OP1+OP2)*OP3-OP4

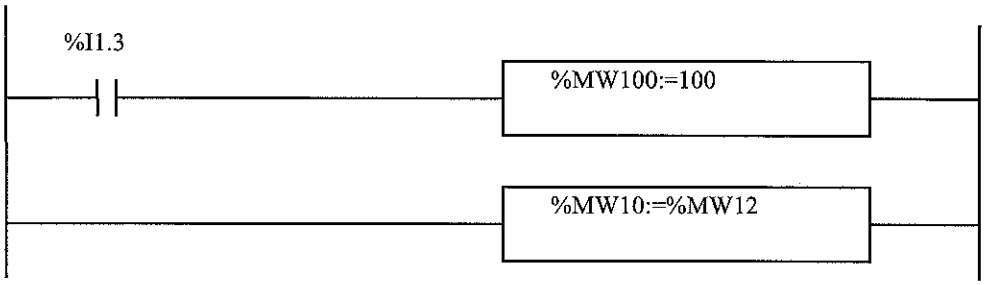
**Instructions de comparaison :** (en zone de tests).
(Il existe également un *bloc de comparaison* quelquefois plus simple d'utilisation).




Si %I1.2 = "1" et si %MW10>100 alors %Q2.3 vaut "1".

On peut utiliser > , >= , < , <= , = , <>

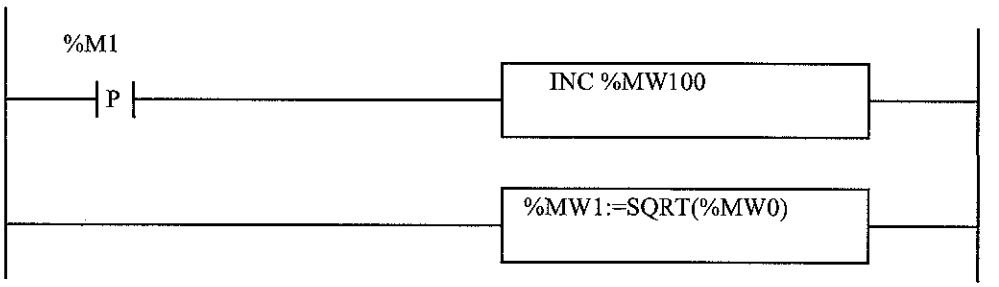
**Instructions d'affectation :** (en zone opérations).




Si %I1.3 = "1" alors %MW100 sera égal à 100 .
%MW10 sera égal à %MW12 .

(Les opérations d'affectations sont utilisables pour les mots simples, les mots doubles et les tableaux de bits).

**Instructions arithmétiques sur les entiers :** (en zone opérations).

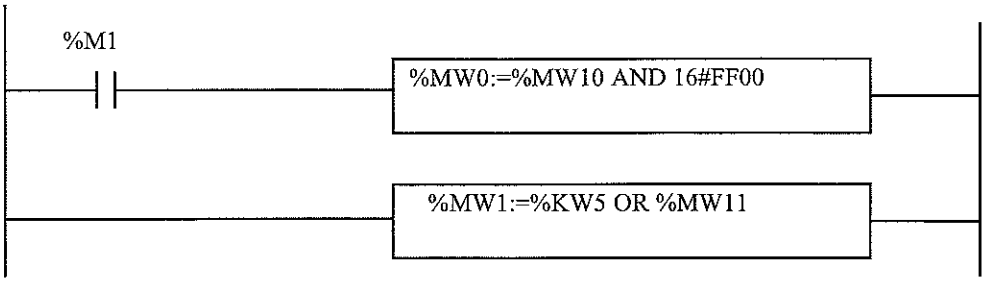



Incrémenter %MW100 sur un front montant de %M1.
%MW1 est la racine carrée de %MW0.

On peut utiliser : SQRT, INC, DEC, ABS, + , - , * (multiplication), / (division), et REM (reste de la division de deux opérandes).

Dans le cas où un résultat dépasse les limites :
-32768 ou +32767 pour un mot simple
-2147483648 ou +2147483647 pour un mot double
le bit système %S18 (*overflow*) est mis à "1". Le résultat est non significatif. La gestion de %S18 se fait par le programme utilisateur.

**Instructions logiques :** (en zone opérations).




Instructions utilisables : AND (ET bit à bit), OR (OU bit à bit), XOR (OU EXCLUSIF bit à bit), NOT (complément logique bit à bit).

# 6 - Le langage Grafcet.

L'éditeur de langage Grafcet de PL7-..... dispose de tous les symboles nécessaires : Etapes initiales et étapes simples, transitions, choix de séquences ("et" et ""ou"), renvois d'origine et renvois de destination, liaisons orientées.

Les objets spécifiques utilisés par le langage Grafcet :

%Xi : Etape n°i. (128 au maximum).

%Xi.T : Temps d'activité de l'étape n°i en dixièmes de secondes (de 0 à 9999 dixièmes de secondes). A la désactivation de l'étape, le contenu de Xi.T est figé. A l'activation de l'étape, Xi.T est remis à zéro puis incrémenté.

Le nombre maximal de transitions est 1024, dont, au plus, 24 simultanées.

Le Grafcet se programme sur 8 pages (numérotées de 0 à 7). Chaque page Grafcet est constituée de 14 lignes et 11 colonnes qui définissent 154 cellules. Dans chaque cellule, il est possible d'écrire un élément graphique.

La première ligne permet de saisir les renvois d'origine, et la dernière ligne, les renvois de destination. Les lignes paires (2 à 12) permettent d'écrire les étapes et les renvois de destination. Les lignes impaires (3 à 13) permettent d'écrire les transitions et les renvois d'origine.

Les étapes peuvent être numérotées dans un ordre quelconque. Une page peut contenir plusieurs graphes.

Commentaires : Dans une page Grafcet, il est possible de saisir un commentaire dans n'importe quelle cellule. Le texte du commentaire est encadré par deux astérisques et sa taille maximale est de 64 caractères. Ce commentaire consomme de la mémoire automate.

**Les actions associées aux étapes du Grafcet.**

A chaque étape peuvent être associées des actions programmables en langage à relais (également en IL et en ST). Ces actions ne sont exécutées que si l'étape à laquelle elles sont associée, est active. PL7 a la particularité de distinguer trois types d'actions :
- Les actions à l'activation ou actions exécutées quand l'étape associée passe de l'état inactif à l'état actif.
- Les actions à la désactivation ou actions exécutées quand l'étape associée passe de l'état actif à l'état inactif.
- Les actions continues ou actions exécutées quand l'étape associée est active.

Les actions à l'activation ou à la désactivation sont impulsionnelles et exécutées sur un seul tour de cycle automate. Elles sont faites par exemple, pour appeler un sous programme ou pour incrémenter un compteur, conditionnellement ou non.

Une action continue est mémorisée. *Les variables logiques qui ont été positionnées dans un certain état par cette action, le restent quand l'étape correspondante est désactivée.* Il convient donc de repositionner ces variables dans l'état convenable, soit au moment de la désactivation de la présente étape, soit au moment de l'activation de l'étape suivante.

Il est possible par exemple, de faire un "SET" d'une variable logique à l'activation d'une étape, et un "RESET" de la variable à la désactivation de cette même étape. Ceci peut être pratique. Attention, en cas de désactivation intempestive du Grafcet, le "RESET" n'aura pas lieu ... (Voir plus loin, les recommandations, au sujet du traitement postérieur).

**Les réceptivités associées aux transitions.**

A chaque transition est associée une réceptivité qui est programmable en langage à relais (également en IL et en ST). Une réceptivité n'est scrutée que si la transition associée est valide.

En langage à relais, la zone de test comporte les conditions de validité de la réceptivité (contacts, entrées, sorties, blocs de comparaison). Dans la zone d'action, on utilise la bobine "dièse", toute autre bobine étant invalide dans ce cas.

**Organisation de la section Grafcet.**

Une section de programme écrit en Grafcet comporte trois traitements consécutifs :
- le **traitement préliminaire** (PRL) ;
- le **Grafcet proprement dit** (CHART) ;
- le **traitement postérieur** (POST).

La section Grafcet appartient toujours à la tâche MAST.

Le traitement préliminaire permet de traiter :
- les initialisations sur reprise secteur ou défaillance ;
- le prépositionnement du Grafcet ;
- la logique d'entrée.

Le traitement séquentiel (le Grafcet lui même) permet :
- de représenter l'ossature de l'application ;
- de donner accès au traitement des réceptivités ;
- de donner accès au traitement des actions associées aux étapes.

Le traitement postérieur permet de traiter :
- la surveillance et les sécurités indirectes spécifiques aux sorties ;
- la logique de sortie ;
- les actions associées aux étapes.

Les réseaux du traitement préliminaire et du traitement postérieur sont scrutés dans l'ordre de leur programmation et non pas dans l'ordre des étiquettes.

On voit ici que les actions associées aux étapes peuvent être programmées, soit dans le Grafcet proprement dit, soit dans le traitement postérieur. D'une façon générale, *il est recommandé de traiter ces actions dans le traitement postérieur* ce qui facilite la "visibilité" de ces actions lors de la phase de mise au point, et le respect de *l'unicité de la commande* (mais dans chaque entreprise, existent des habitudes de programmation à respecter...).

**Utilisation des bits de commande du Grafcet %S21, %S22 et %S23.**

**%S21 (initialisation des Grafcets).**

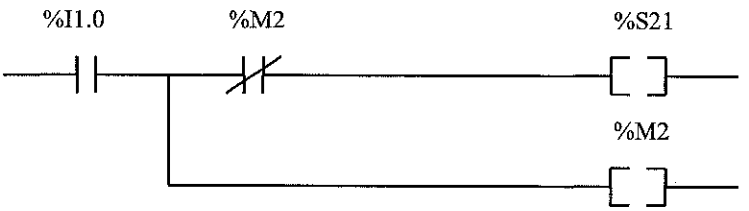
Le bit système %S21 est normalement à l'état logique "0". La mise à "1" de %S21 provoque la **désactivation des étapes actives et l'activation des étapes initiales**. %S21 est géré par le programme utilisateur (et éventuellement par la console).

Le bit %S21 est remis à l'état logique "0" automatiquement par le système, au début du traitement séquentiel, juste avant le traitement préliminaire. Le programme utilisateur utilisant %S21 doit donc être *obligatoirement placé dans le traitement préliminaire.*

Exemple 1 :

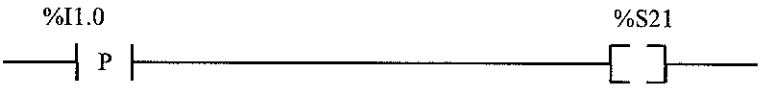

Les grafcets seront ici, maintenus à leur état initial pendant toute la durée d'activité de l'entrée %I1.4 (ex : Bouton poussoir "ré-initialisation générale").

Exemple 2 :




Un front montant de %I1.0, va faire passer %S21 à l'état "1" pendant le cycle (n) de l'automate. Les grafcets vont être initialisés au début du cycle (n+1) et %S21 sera remis à "0" automatiquement par le système. Les grafcets seront de nouveau, prêts à fonctionner au cycle suivant.

Exemple 3 :




Même fonctionnement apparent que pour l'exemple 2. (Attention aux restrictions d'utilisation du contact "P" et du contact "N").

**%S22 (désactivation).**

Normalement à l'état "0", la mise à l'état "1" de %S22 provoque la désactivation des étapes actives de l'ensemble du traitement séquentiel. %S22 est géré par le programme utilisateur.

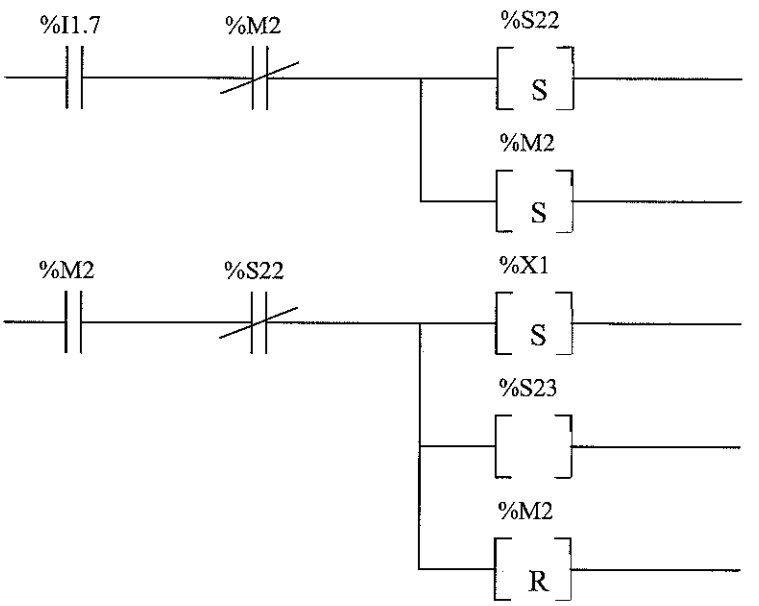
Le bit %S22 est remis à l'état logique "0" automatiquement par le système, au début du traitement séquentiel, juste avant le traitement préliminaire. Le programme utilisateur utilisant %S22 doit donc être *obligatoirement placé dans le traitement préliminaire.*

**%S23 (gel).**

Normalement à l'état "0", la mise à l'état "1" de %S23 provoque le maintien en l'état des Grafcets. Quelle que soit la valeur des réceptivités en aval des étapes actives, les Grafcets n'évoluent pas. Ce gel est maintenu tant que le bit %S23 vaut "1". Attention : Le temps Xi.T (durée de l'étape) *continue de s'écouler.*

Contrairement aux bits %S21 et %S22, le bit %S23 n'est pas remis à "0" automatiquement par le système, mais par le programme utilisateur. Le programme utilisateur utilisant %S23 doit être *obligatoirement placé dans le traitement préliminaire.*

Exemple 3 : On veut positionner un grafcet dans une situation donnée où l'étape X1, seule est active :




Le bit %M2 est à l'état logique "0". Lors du cycle (n), une action sur l'entrée %I1.7 provoque la mise à "1" de %S22 puis de %M2. La scrutation de la deuxième ligne est sans effet car %S22 est à l'état logique "1". Le Grafcet est alors désactivé.

Au début du cycle (n+1), %S22 est remis à "0" par le système. La scrutation de la première ligne est sans effet car %M2 est à l'état logique "1". La scrutation de la deuxième ligne provoque la mise à l'état logique "1" de l'étape %X1 et de %S23 puis la remise à l'état logique "0" de %M2. Le Grafcet est alors prépositionné à l'étape %X1, et prêt pour être scruté au cycle (n+2).

# 7 - Quelques autres bits système d'utilisation simple.

(Il existe une centaine de bits système et une centaine de mots système).

**%S4, %S5, %S6, %S7 :**

Ces bits système sont des "bases de temps" indépendantes du cycle de l'automate. %S4 "clignote" avec une période de 10 millisecondes (5 millisecondes à "0" puis 5 millisecondes à "1"). De la même manière, %S5 "clignote" avec une période de 100 millisecondes, %S6, avec une période de 1 seconde et %S7 avec une période de 1 minute.

**%S9 :**

Ce bit est normalement à "0". Il peut être mis à "1" par programme provoquant la mise en position de repli (par défaut à l'état "0") de toutes les sorties. Cette position de repli est modifiable au moment de la configuration matérielle de l'automate. Attention : Le processus continue normalement pendant ce temps.

**%S11 :**

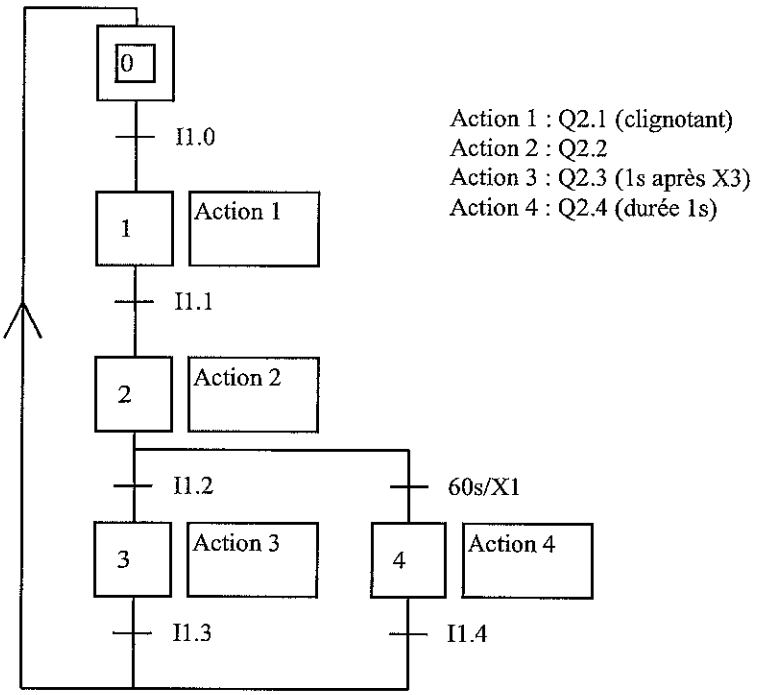
Normalement à "0", il passe à "1" en cas de dépassement de temps du chien de garde. L'automate est alors stoppé et le voyant "ERR" clignote.

**%S13 :**

Il est à "1" seulement pendant le premier cycle de l'automate, après le passage en "RUN" (utilisable pour des initialisations).

## 8 - Exemple simple.

Un processus est modélisé par le grafcet suivant : (CHART).

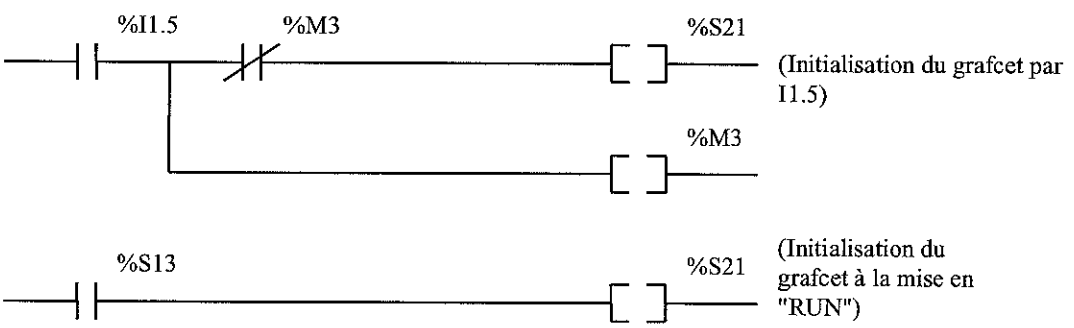



Les réceptivités sont liées à l'état logique des entrées de l'automate sauf une qui est vraie 60 secondes après l'activité de l'étape X1 (schéma ci-dessus).

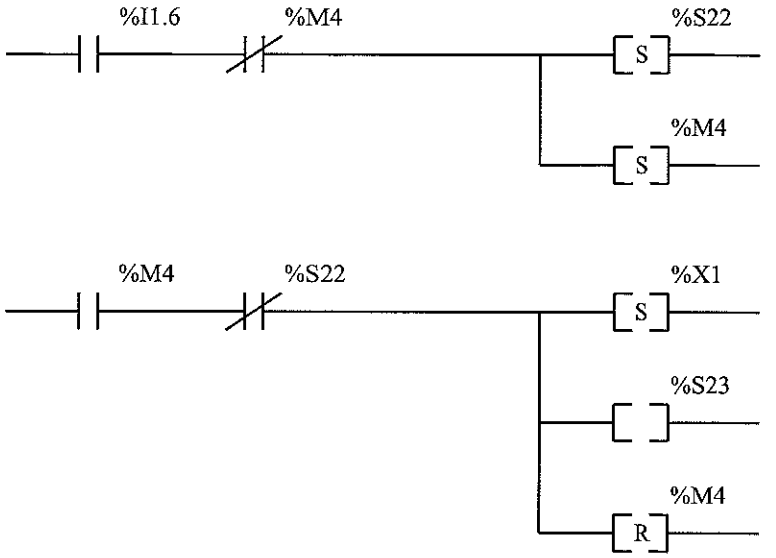
Les actions sont liées aux sorties de l'automate avec des conditions particulières :
L'action 1 sera "clignotante" (période : 1 seconde).
L'action 3 ne débutera qu'une seconde après l'activité de l'étape X3.
L'action 4 ne durera qu'une seconde.

Le grafcet devra s'initialiser automatiquement à la mise en "RUN" de l'automate mais également par la mise à "1" de l'entrée I1.5.
La mise à "1" de l'entrée I1.6 placera le grafcet dans la situation {X1}.

**PRL :**

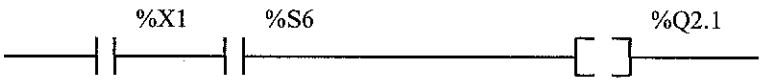


(suite de PRL) : %I1.6 positionne le grafcet en {X1}.

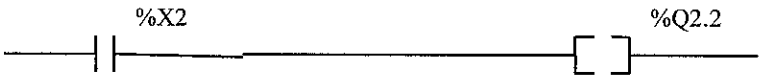



## POST :

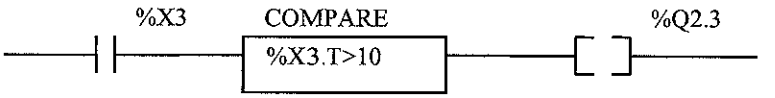
Clignotement de la sortie %Q2.1 :



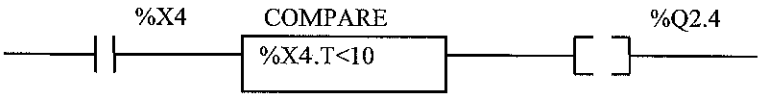

Action 2 :



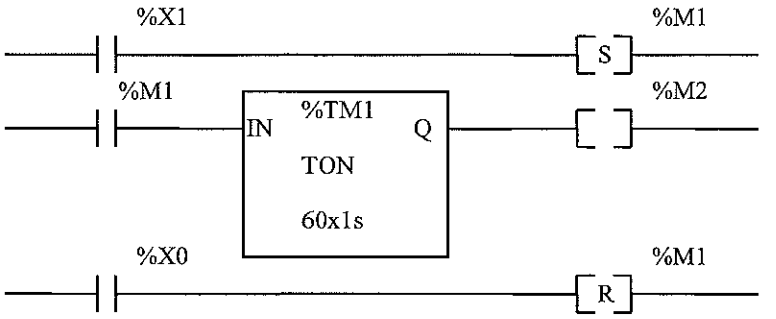

L'action 3 ne débute qu'une seconde après l'activation de X3 :




L'action 4 ne dure qu'une seconde :



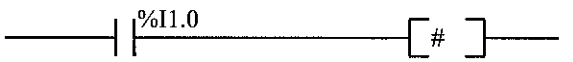
Commande de la réceptivité temporisée en amont de X4 :




L'activation de X1 va provoquer la mise à "1" de %M1.
%M1 va commander le temporisation %TM1 (durée 60 secondes) même si X1 est désactivée.
%TM1 va commander %M2 qui va servir à la programmation de la réceptivité précédant X4.
%M1 sera remis à "0" par X0.

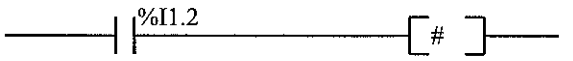
## Réceptivités : Elles seront programmées dans l'entité CHART.

entre X0 et X1 :



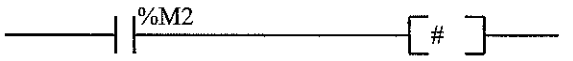

entre X1 et X2 :


entre X2 et X3 :

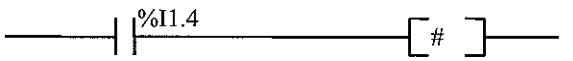



entre X3 et X0 :


entre X2 et X4 :




entre X4 et X0 :

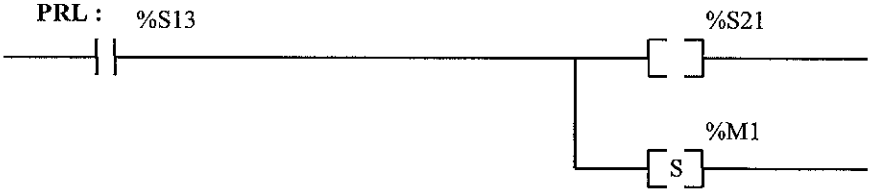


## 9 - Exemple d'application.

On réalise une guirlande avec cinq lampes L0, L1, L2, L3, L4. La séquence de fonctionnement est : L0 → L1 → L2 → L3 → L4 → L3 → L2 → L1 → L0 → L1 → L2 → L3 → L4 etc. Chaque lampe doit rester allumée pendant une seconde. Un intervalle d'extinction de une demi-seconde existe entre chaque lampe. Ces cinq lampes seront commandées par les cinq sorties Q2.0 ... Q2.4. L'entrée I1.0 sera la commande arrêt/ marche générale.

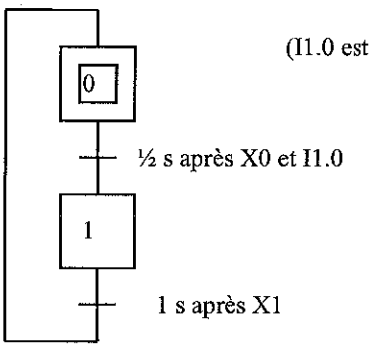
Il existe de nombreuses façon de traiter ce petit problème. Pour cet exemple, nous allons utiliser un grafcet à deux étapes qui va servir de "séquenceur". L'étape X0 sera active pendant une demi-seconde (extinction) et l'étape X1 sera active pendant une seconde (allumage). Ce grafcet sera initialisé au moment de la mise en "RUN" de l'automate.

**PRL :**



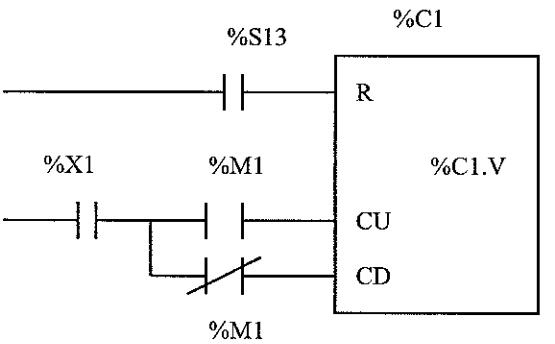

(%S13 est à "1" pendant le premier cycle automate après mise en "RUN").

**CHART :**




(I1.0 est la commande marche/arrêt).

**POST :** On utilise un compteur/décompteur %C1.

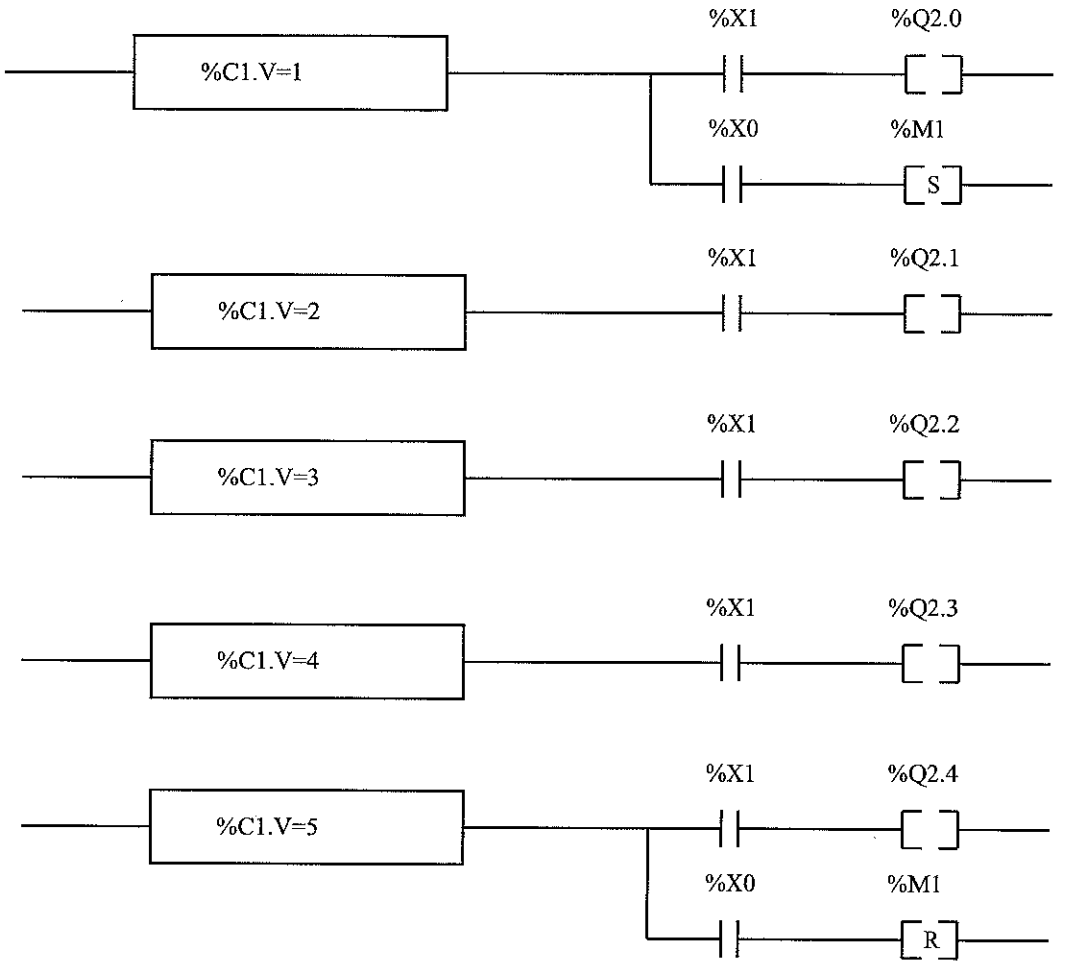



R : remise à zéro du compteur.
CU : "incrémentation".
CD : "décrémentation".

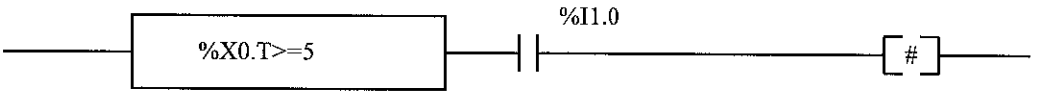
Comptage si %M1 = "1" et %X1 = "1"
Décomptage si %M1 = "0" et %X1 = "1".

(Suite du traitement postérieur page suivante).

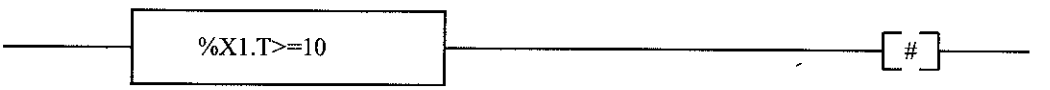
On utilise cinq blocs "COMPARE" :




Réceptivité entre X0 et X1 :




Réceptivité entre X1 et X0 :



# 10 - Le langage littéral structuré (ST).

C'est un langage évolué de type algorithmique plutôt adapté à la programmation de fonctions arithmétiques compliquées, à la manipulation de tableaux, à la gestion de messages, etc. ... Il est organisé en "phrases" elles mêmes composées d'instructions et de commentaires (256 caractères au maximum). Ces "phrases" peuvent comporter chacune, une étiquette (1000 étiquettes au maximum).

L'éditeur permet d'écrire ces "phrases" les unes à la suite des autres avec les possibilités de modifications, d'insertions, de copiage et de collage.

Le langage littéral structuré est très riche en instructions et en structures de contrôle. On peut citer quelques exemples de base :

**Instructions sur bits :**

:= application. Exemple : %Q2.0 := %M0 (le bit %Q2.0 prend la valeur de %M0)

**OR, AND, XOR, NOT, SET, RESET.**

**RE** front montant. Exemple : %Q2.0:=RE %M0 (le bit %Q2.0 devient "1" au front montant de %M0.

**FE** front descendant.

**Comparaisons numériques sur mots, doubles mots et flottants :**

< > <= >= = <>

Il existe également des instructions pour traiter les tableaux de bits, de mots, de doubles mots, l'arithmétique entière sur les mots et les doubles mots, l'arithmétique sur les flottants, les opérations logiques sur les mots et les doubles mots, les commandes d'exécution de programme et d'appels aux sous-programmes, l'horodateur, etc. .

Il est possible d'effectuer des conversions multiples (binaire, BCD, Gray), et des manipulations de chaînes de caractères.

**Structures de contrôle :**

**IF** condition **THEN**
actions;
**END_IF;**
.....
**IF** condition1 **THEN**
action1;
**ELSIF** condition2 **THEN**
action2;
**ELSE**
action3;
**END_IF;**
Le nombre de ELSIF est illimité, mais une seule partie ELSE au maximum.
.....

**WHILE** condition **DO**
action;
**END_WHILE;**

Plusieurs WHILE peuvent être imbriqués.
.....

**REPEAT**
action;
**UNTIL** condition **END_REPEAT;**

Plusieurs REPEAT peuvent être imbriqués.

.....

**FOR** indice := valeur initiale **TO** valeur finale **DO**
action;
**END_FOR;**

Plusieurs FOR peuvent être imbriqués.

.....

**EXIT** permet d'arrêter l'exécution d'une boucle WHILE, REPEAT ou FOR et de continuer sur l'instruction suivant le mot clef de fin de boucle.

.....

EXEMPLE : On "traduit" le grafcet de l'exemple simple (page 16) en langage littéral structuré :

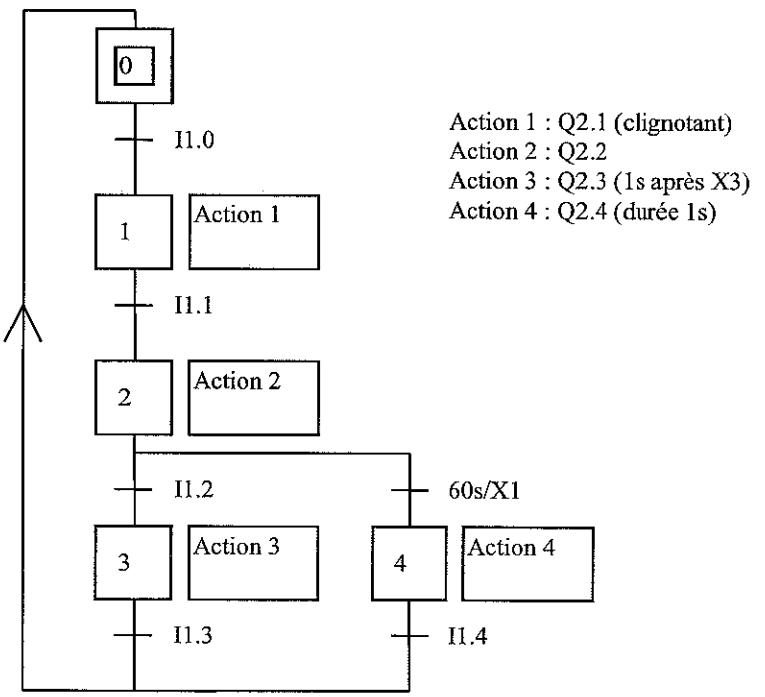
On renommera les variables pour plus de clarté :

%I1.0, %I1.2, %I1.3, %I1.4 s'appelleront : Entrée0, Entrée1, etc.

%Q2.0, %Q2.1, %Q2.2, %Q2.3 s'appelleront : Sortie0, Sortie1, etc.

%M, %M1, %M2, %M3, %M4 s'appelleront : Etape0, Etape1, etc.

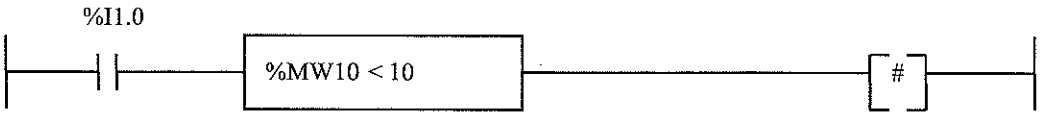
%TM1, %TM2, %TM3 s'appelleront : Tempo1, Tempo2, Tempo3.



```
(*initialisation du grafcet par %S13 et par l'entrée n°5*)
IF %S13 OR Entrée5 THEN
SET Etape0; RESET Etape1; RESET Etape2; RESET Etape3; RESET Etape4;
END_IF;

(*de l'état 0 à l'état 1*)
IF Etape0 AND Entrée0 THEN
SET Etape1; RESET Etape0;
END_IF;

(*de l'état 1 à l'état 2*)
IF Etape1 AND Entrée1 THEN
SET Etape2; RESET Etape1;
END_IF;

(*de l'état 2 à l'état 3 ou de l'état 2 à l'état 4*)
IF Etape2 AND Entrée2 AND NOT Tempo1.Q THEN
SET Etape3; RESET Etape2;
ELSIF Etape2 AND Tempo1.Q AND NOT Entrée2 THEN
SET Etape4; RESET Etape2;
END_IF;

(*de l'état 3 à l'état 0*)
IF Etape3 AND Entrée3 THEN
SET Etape0; RESET Etape3;
END_IF;

(*de l'état 4 à l'état 0*)
IF Etape4 AND Entrée4 THEN
SET Etape0; RESET Etape4;
END_IF;

(*temporisation n°1 de 60 secondes*)
IF RE Etape1 THEN
START Tempo1;
ELSIF Etape3 OR Etape0 THEN
DOWN Tempo1;
END_IF;

(*action 1*)
Sortie1 := Etape1 AND %S6;

(*action 2*)
Sortie2 := Etape2;

(*action 3*)
IF Etape3 THEN
START Tempo2;
ELSIF NOT Etape3 THEN
DOWN Tempo2;
END_IF;
Sortie3 := Tempo2.Q AND Etape3;

(*action 4*)
IF Etape4 THEN
START Tempo3;
ELSIF NOT Etape4 THEN
DOWN Tempo3;
END_IF;
Sortie4 := NOT Tempo3.Q AND Etape4;
```

(*fin du programme*)

**REMARQUE :** Quand on utilise le grafcet, on peut écrire les réceptivités en langage littéral structuré.
Exemple :




s'écrira alors :

%I1.0 **AND** %MW10 < 10

## 11 – Les blocs de fonction DFB (Derived Function Block).

PL7- PRO offre à l'utilisateur, la possibilité de créer ses propres blocs de fonction répondant aux spécificités de ses applications. Ils permettent de structurer une application. Ils seront utilisés dès qu'une séquence de programme se trouve présente à plusieurs reprises dans l'application. Dans les entreprises, ils servent à **standardiser** la programmation, ils sont alors considérés comme une sorte de « procédure » au sens de la certification, dans cette entreprise.

Un bloc DFB permet d'accroître la lisibilité d'un programme. Il diminue le volume de code engendré (le code correspondant au DFB n'est chargé qu'une fois quel que soit le nombre d'appels à ce DFB). La simplicité apparente du programme obtenu ne doit pas faire oublier le temps de cycle automate qui peut devenir important lors de l'utilisation de plusieurs blocs DFB (dans cas d'un processus très rapide à gérer).

Les blocs DSB peuvent être « exportés » vers d'autres applications.

**Programmation :**

Les blocs DFB sont **utilisables** dans une section de programme écrite en langage à relais (LD) ou en langage littéral structuré (ST). Ils sont eux mêmes **programmables** en langage à relais ou en langage littéral structuré. Leur programmation comporte des restrictions. Ils peuvent être « protégés ».

Pour créer un bloc DFB, on réalise d'abord un modèle (appelé « type DFB »), puis on réalise une image de ce bloc appelée **instance** pour chaque utilisation dans l'application. Ensuite, on peut utiliser cette instance dans l'application.

Le modèle se compose :
- d'un nom (8 caractères maximum) ;
- de paramètres d'entrées, de sorties, d'entrées-sorties ;
- de variables publiques (connues dans toute l'application) ;
- de variables privées (connues dans le bloc seulement) ;
- d'un code (en LD ou ST) ;
- éventuellement d'un commentaire et d'une fiche de description.

Ceci se fait à l'aide de l'éditeur de blocs DFB.

Une fois le modèle réalisé, le programmeur définit une instance du bloc DFB lors de l'appel de la fonction dans l'éditeur de programme.

Cette instance du bloc DFB s'utilise ensuite comme un bloc fonction standard qui peut être placée dans les différentes tâches (sauf tâches événementielles) et sections de l'application.

Restrictions sur le nombre de variables utilisables :

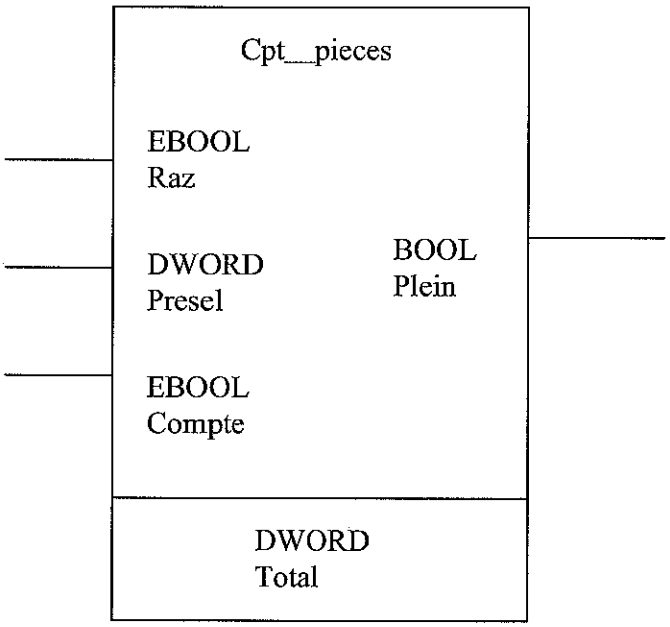
Nombre d'entrées + nombre d'entrées-sorties <= 15
Nombre de sorties + nombre d'entrées-sorties <= 15

100 variables publiques au maximum et 100 variables privées au maximum.

Ces variables sont du type BOOL (Booléen), EBOOL (Booléen mais on peut utiliser un front), REAL (réel), WORD (entier 16 bits), DWORD (entier 32 bits). On ne peut pas utiliser les objets d'entrée-sortie (%Ii.j, %Ql.m)

**Exemple simple** : Le modèle de bloc DFB ci-dessous s'appelle « Cpt_pieces », il a 3 entrées « Raz », « Presel » et « Compte », et une sortie « plein ». Il dispose d'une variable privée « Total ».

Il compte les impulsions sur l'entrée « Compte » et les totalise dans « Total ». Quand le contenu de « Total » atteint la valeur « Presel » alors la sortie « Plein » est mise à « 1 ».



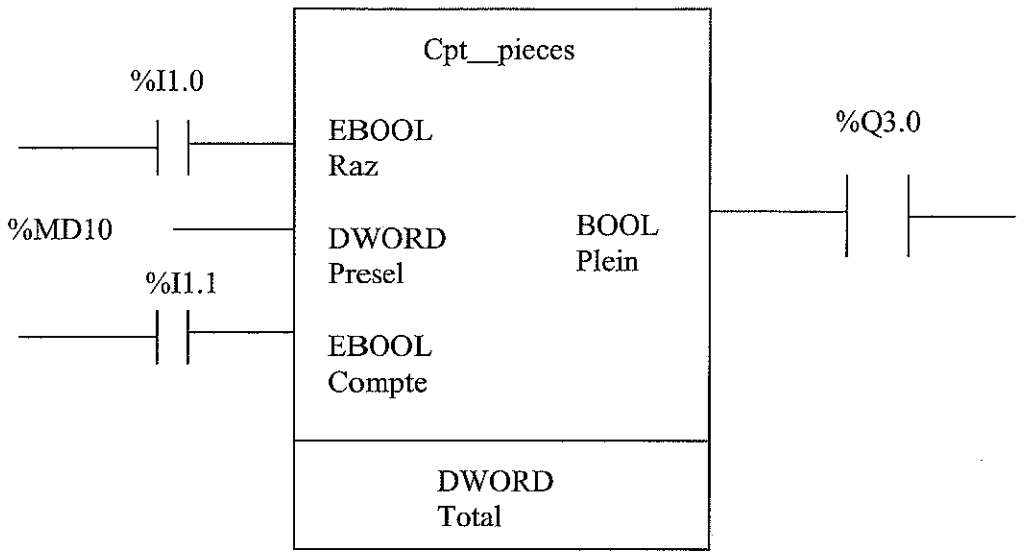

Code du bloc :

```
IF RE Raz THEN
Total:=0;
END_IF;
IF RE Compte THEN
Total:=Total+1
END_IF;
IF (Total>=Presel) THEN
SET Plein;
ELSE
RESET Plein;
END_IF;
```

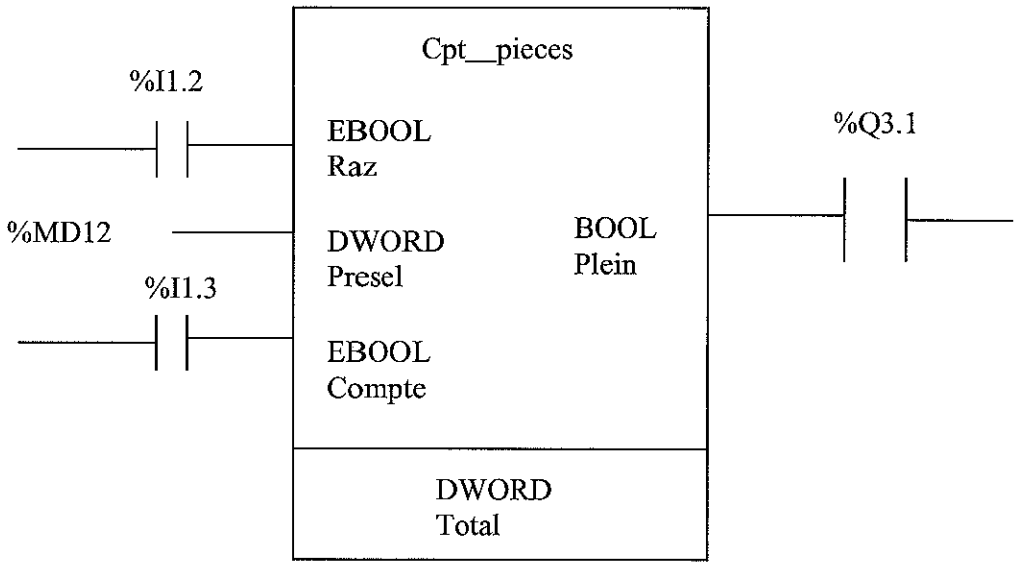
**Application exemple :** On va créer deux instances du bloc ci-dessus. L'une s'appelle « compte_vis » et l'autre, « compte_ecrou ».

(%MD10 et %MD12 sont placés par un double clic droit devant l'entrée correspondante).

Compte_vis




Compte_ecrou



## 12 - Utilisation de tableaux.

PL7 permet d'effectuer des opérations sur des tableaux de mots et de doubles mots.

Exemples :

**%KW10:5** est un tableau de 5 mots constants de 16 bits, commençant à %KW10 (%KW10, %KW11, %KW12, %KW13, %KW14).

**%MW100:20** tableau de 20 mots de 16 bits commençant à %MW100.

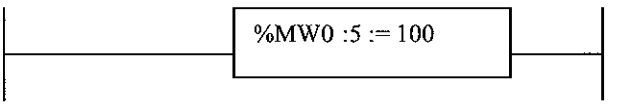
Règles :

Les opérations entre tableaux ne s'effectuent que sur des tableaux contenant des objets de même type.
Si, dans une opération, les tableaux sont de tailles différentes, le tableau résultat aura une taille correspondant au plus petit des deux tableaux opérandes.
Une opération sur deux tableaux s'effectue sur chaque élément de même rang des deux tableaux et le résultat est transféré dans l'élément de même rang du tableau résultat.
Il faudra être prudent au cours de la programmation, pour ne pas effectuer de recouvrements involontaires de tableaux (panne difficile à détecter).

Test :

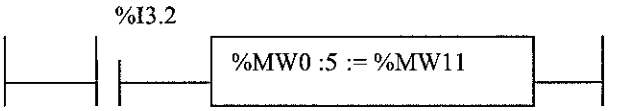
Si, lors d'une opération entre éléments, le bit système %S18 est positionné à « 1 », cela signifie que l'opération en cours est erronée, mais l'opération pour les éléments suivants est effectuée correctement.

Les exemples élémentaires suivants sont écrits en langage à relais (LD) et en langage littéral structuré (LS).



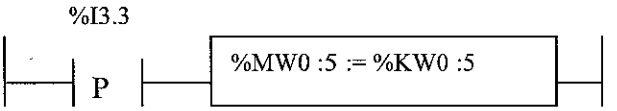

```
%MW0 :5 := 100 ;
```

(Les 5 mots %MW0 à %MW4 prendront la valeur 100)




```
IF %I3.2 THEN
%MW0 :5 := %MW11 ;
END_IF;
```

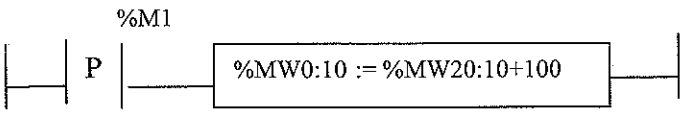
(Si %I3.2 vaut « 1 », les 5 mots %MW0 à %MW4 prendront la valeur contenue dans %MW11).



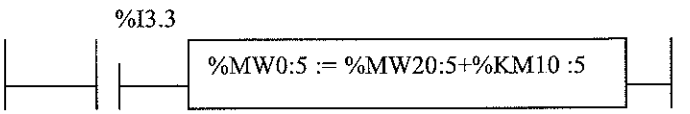

```
IF RE %I3.3 THEN
%MW0 :5 := %KW0:5 ;
END_IF;
```

(Lors du front montant de %I3.3, le tableau %KW0 :5 va se recopier dans le tableau %MW0 :5)

**Instructions arithmétiques** sur les tableaux (addition : +, soustraction : -, multiplication : *, division : /) :




```
IF RE %M1 THEN
%MW0 :10 := %MW20:10+100 ;
END_IF;
```




```
IF %I3.3 THEN
%MW0 :5 := %MW20:5+%KM10:5 ;
END_IF;
```

**Instructions logiques** à deux opérandes (et, ou et ou exclusif, bit à bit : AND, OR, XOR) et à un seul opérande (complément logique bit à bit : NOT).

Il existe de nombreuses instructions de programmation avancée, associées aux tableaux (somme des éléments, comparaison élément par élément, recherche d' un élément dans un tableau, recherche de mini et maxi, nombre d'occurrences d'une valeur, décalages circulaires droite et gauche, fonctions de tri, etc.).

## 13 - Utilisation simple de l'horodateur

(il existe d'autres modes d'exploitation de l'horodateur, utilisant les mots %SW49 à %SW59 et %SD18).

Les paramètres dates, heures et durées, utilisent des formats spéciaux définis par la norme IEC1131-3.

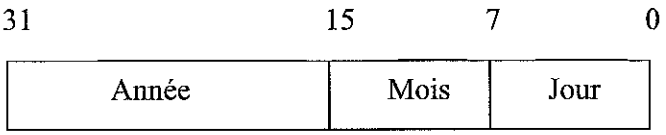
**Durée (format TIME).**

Les durées sont exprimées en dixièmes de secondes, par exemple : 3674.3 pour 1 heure, 1 minute, 14 secondes et 3 dixièmes de seconde.

La valeur est codée sur 32 bits (un mot double) entre 0 et 4294967295 (plus de 4971 jours).

**Date (format DATE).**

Ce format code l'année, le mois et le jour sous la forme AAAA-MM-JJ (exemple : 2004-06-25). Cette valeur est codée sur 32 bits (un double mot) en trois champs :

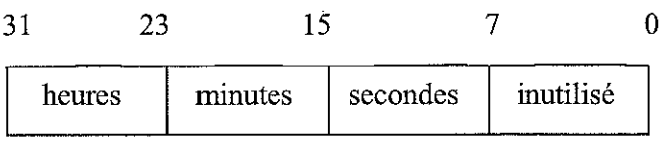



L'année utilise 4 digits codés en BCD (16 bits), le mois et je jour, 2 digits codés en BCD de 1990-01-01 à 2099-12-31. Exemple, en hexadécimal :

| 20h | 04h | 06h | 25h |
|---|---|---|---|

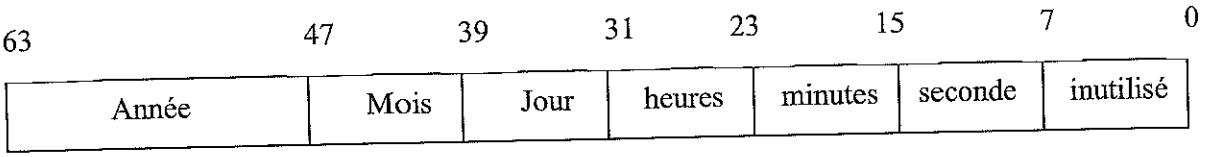
**Heure du jour (format TOD : « TIME_OF_DAY »).**

Ce format code les heures minutes et secondes sous la forme HH :MM :SS (exemple : 23 :12 :34). Cette valeur est codée en BCD sur 32 bits (un double mot) en trois champs :



**Date et heure (format DT : « DATE_AND_TIME »).**

Ce format code l'année, le mois, le jour, l'heure, les minutes et les secondes, sous la forme : AAAA-MM-JJ-HH :MM :SS (exemple : 2004-06-25-23 :12 :34). Cette valeur est codée en BCD sur 64 bits (un tableau de quatre mots de 16 bits).




**Test :**

Le bit système %S17 est positionné à « 1 » dans les cas suivants :

- Résultat d'une opération hors de l'intervalle de valeurs, autorisé ;
- Un paramètre d'entrée n'est pas cohérent avec le format souhaité (DATE, DT ou TOD) ;
- Opération sur le format TOD entraînant un changement de jour ;
- Conflit d'accès à l'horodateur.

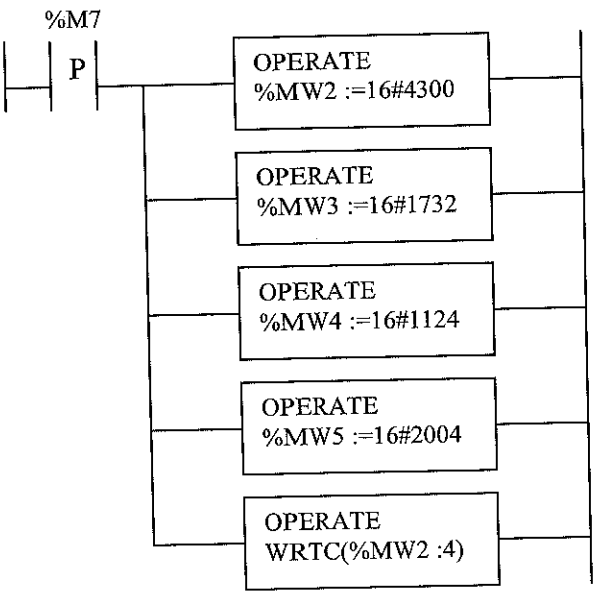
**Lecture de l'horodateur.** (en langage à relais et en langage littéral structuré)


```
IF %M6 THEN
RRTC(%MW2 :4) ;
END_IF;
```

La fonction RRTC va écrire au format DT dans le tableau %MW2 :4.

**Mise à jour de l'horodateur.**

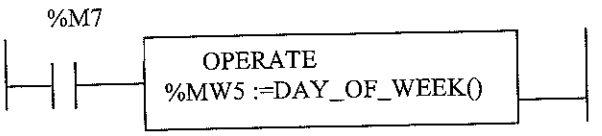
On veut écrire 24 novembre 2004, 17 heures 32 minutes 43 secondes, au moment du front montant de %M7 :




```
IF RE %M7 THEN
%MW2:=16#4300;
%MW3:=16#1732;
%MW4:=16#1124;
%MW5:=16#2004;
WRTC(%MW2:4);
END_IF;
```

**Lecture du jour de la semaine.**

Cette fonction place dans un mot de 16 bits, le jour de la semaine codé de 1 à 7 (1 : lundi ; 7 : dimanche).




```
IF %M7 THEN
%MW5:=DAY_OF_WEEK();
END_IF;
```

**Fonction « programmateur horaire ».**

Cette fonction permet de commander des actions à des horaires et dates, prédéfinies ou calculées.
Elle positionne à « 1 » le paramètre logique de sortie, si la date et l'heure fournies par l'horloge automate au moment de l'appel de la fonction, appartiennent à la période programmée dans les paramètres d'entrée.

**SCHEDULE (DDEB, DFIN, SEM, HDEB, HFIN, SORTIE)**

DDEB : date de début ; DFIN : date de fin ; SEM : semaine ; HDEB : heure de début ; HFIN : heure de fin ; SORTIE : paramètre logique de sortie (« 0 » ou « 1 »).

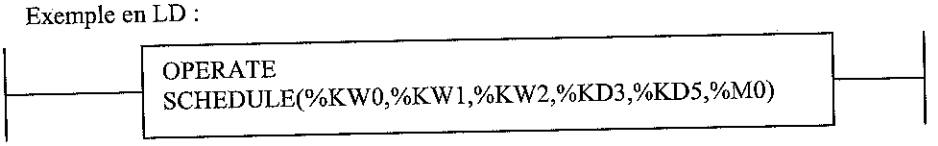
DDEB et DFIN sont codés en BCD entre 01-01 et 12-31 (un mot de 16 bits).

SEM utilise les 7 bits de poids faible d'un mot de 16 bits :

| | | | |
|---|---|---|---|
| Bit 6 : lundi | Bit 5 : mardi | Bit 4 : mercredi | Bit 3 : jeudi |
| Bit 2 : vendredi | Bit 1 : samedi | Bit 0 : dimanche | |

HDEB et HFIN, double mot codé en BCD (type TOD), entre 00 :00 :00 et 23 :59 :59.

Exemple en LD :




Autre exemple en LS : deux plages horaires non continues.

```
SCHEDULE ( 16#0501,16#1031,2#0000000001111100,16#08300000,16#12000000,%M0);
SCHEDULE ( 16#0501,16#1031,2#0000000001111100,16#14000000,16#18000000,%M1);
%Q0.0 := %M0 OR %M1;
```

1er SCHEDULE : date de début 1er mai ; date de fin : 31 octobre ; lundi à vendredi ; de 8h30 à 12h.
2ème SCHEDULE : du 1er mai au 31 octobre ; lundi à vendredi ; de 14h à 18h.
%Q0.0 vaudra « 1 » au cours de ces deux plages horaires.

La plage définie par DDEB et DFIN peut être « à cheval » sur deux années consécutives.
La plage définie par HDEB et HFIN peut être « à cheval » sur deux jours consécutifs.
Le 29 février sera ignoré lors des années non bissextiles.
Si la programmation de ces plages est erronée, %S17 sera mis à « 1 ».

**Remarque** : Le traitement de la fonction SCHEDULE allonge notablement le temps de cycle. Si la précision n'est pas nécessaire, on pourra cadancer l'appel à cette fonction, avec %S6 (1s) ou %S7 (1mn).

D' autres **fonctions avancées** existent :
- lire la date du dernier arrêt de l'automate (avec un code d'erreur) ;
- ajouter ou retirer une durée à une date ou à une heure du jour ;
- calculer l'écart entre deux dates ou entre deux horaires ;
- convertir des dates en chaînes de caractères ;
- etc