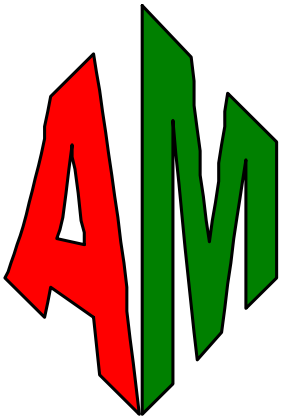


NOTICE DE PROGRAMMATION D'AUTOMATES SIEMENS S7 300 – S7 400

Ecrit par Alain Malvoisin

PROGRAMMATION

SIMATIC Manager

SOMMAIRE

I. / LES VARIABLES DE L'AUTOMATE

1.1 Types de variables

Zone E : Mémoire image des entrées sur bus locale ou bus de terrain tel que PROFIBUS
Zone A : Mémoire image des sorties sur bus locale ou bus de terrain tel que PROFIBUS
Zone M : Mémoire utilisateur
Zone L : Mémoire locale, associée à un module de programme
Zone P : Accès à la périphérie
Zone T : Mémoire des temporisations
Zone Z : Mémoire des compteurs
Zone DB : Mémoire utilisateur ou système structuré dans des blocs de données

Symbole SIMATIC	Définition	Symbole CEI
E	Bits d'entrée	I
EB	Octet d'entrée	IB
EW	Mot d'entrée	IW
ED	Double mot d'entrée	ID
A	Bits de sortie	Q
AB	Octet de sortie	QB
AW	Mot de sortie	QW
AD	Double mot de sortie	QD
M	Mémoires utilisateurs (variables auxiliaires)	M
MB	Octet mémoire	MB
MW	Mot mémoire	MW
MD	Double mot mémoire	MD
L	Bit dans la mémoire locale	L
LB	Octet dans la mémoire locale	LB
LW	Mot dans la mémoire locale	LW
LD	Double mot dans la mémoire locale	LD
PEB	Octet de périphérie d'entrée	
PAB	Octet de périphérie de sortie	
PEW	Mot de périphérie d'entrée	
PAW	Mot de périphérie de sortie	
PED	Double mot de périphérie d'entrée	
PAD	Double mot de périphérie de sortie	
T	Temporisation	T
Z	Compteur	C
DBX	Bit dans un bloc de donnée	
DBB	Octet dans un bloc de donnée	
DBW	Mot dans un bloc de donnée	
DBD	Double mot dans un bloc de donnée	

1.2 Adressage des variables

Les objets E, A, M, DB, PE et PA sont rangés dans des octets (8 bits), on peut accéder à un BIT, à un OCTET, à un MOT de 16 bits ou à un DOUBLE MOT (32 bits)

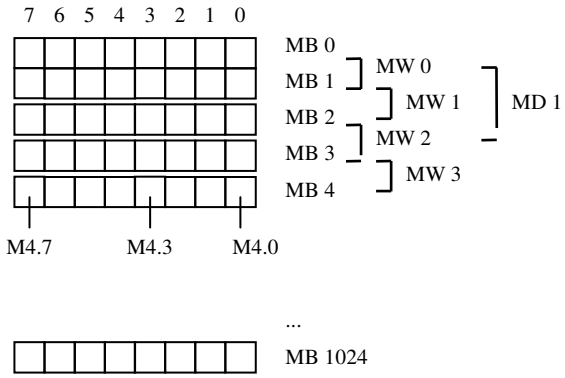
Exemples :

M4.3 correspond au bit 3 de l'octet 4, on peut le tester en combinaison ET, OU avec un autre bit, on peut lui affecter le résultat d'une combinaison, on peut le mettre à « 1 » ou à « 0 ».

MB4 correspond aux 8 bits de l'octet 4, on peut comparer sa valeur, on peut l'additionner, le soustraire, le multiplier, le diviser, on peut lui affecter le résultat d'une opération

MW2 correspond au 16 bits constitué par les octets 2 (poids fort) et 3 (poids faible)

MD1 correspond au 32 bits constitué des octets 1, 2, 3 et 4



1.3 Plages d'adressage

Bit	0.0 à 65535.7
Octet	0 à 65535
Mot	0 à 65534
Double mot	0 à 65532

II. / SAISIS DES MNEMONIQUES

Saisir les mnémoniques est très utile, il vaut mieux saisir un programme entièrement en symbole qu'en adressage absolu, c'est beaucoup plus lisible et compréhensible.

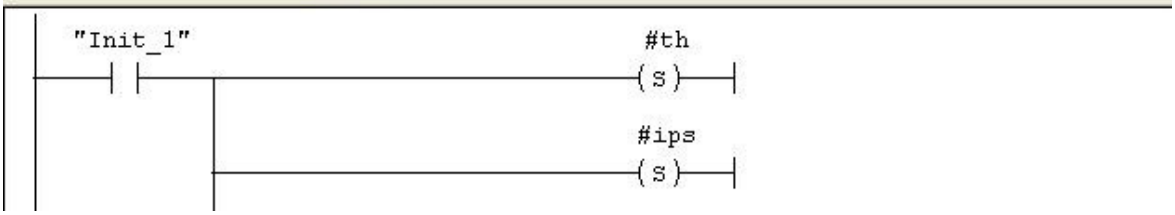
Il suffit d'aller dans la table des Mnémoniques et y entrer les différents éléments.

Le nom du symbole, sont adresse réel, sont type et sont commentaire

Voici un exemple

54		Capteurs	FC	13	FC	13	Module de gestion des capteur
55		Alarmes	FC	17	FC	17	Module de gestion des alarmes
56		Vanne	FC	25	FC	25	Module de gestion des vannes
57		Compteur	FC	26	FC	26	Module de gestion des compteurs
58		MOIS	MW	36	INT		Mois lu
59		JOUR	MW	34	INT		Jour lu
60		HEURE	MW	30	INT		Heure lu
61		Nombre	MW	10	INT		Compte les objet prédéfinis
62		MINUTE	MW	32	INT		Minute lu
63		tseconde	MW	5	INT		Tempo 1 seconde
64		SECONDE	MW	40	INT		Seconde lu
65		RET	MW	4	INT		Retour de fonctions
66		Tp_init	MW	2	INT		Temps d'initialisation
67		ANNEE	MW	38	INT		Année lu
68		SOMMAIRE	OB	1	OB	1	Appel des fonctions
69		SECONDES	OB	32	OB	32	Une fois par seconde
70		COLD_START	OB	102	OB	102	Demmarage à froid
71		Write_clock	SFC	0	SFC	0	Ecriture de l'heure
72		Read_clock	SFC	1	SFC	1	Lecture de l'heure
73		TessaiTP	T	2	TIMER		Temporisation un essai pour voir les Tp
74		TEssaiTOF	T	1	TIMER		Temporisation un essai pour voir
75		TEssai1	T	0	TIMER		Temporisation
76		Reserve_mot	MW	8	WORD		Mot résersé aux réserves

On peut accéder à la table des symboles depuis l'éditeur CONT/LIST/LOG



III. / INSTRUCTIONS SUR BITS

Tester des bits	
U	**Test un bit en combinaison ET**
O	**Test un bit en combinaison OU**
UN	**Test un bit en combinaison ET PAS**
ON	**Test un bit en combinaison OU PAS**
UN(	**Test un bit en combinaison ET PAS (**
ON(	**Test un bit en combinaison OU PAS (**
U(	**ET ouvrez la parenthèse**
O(	**OU ouvrez la parenthèse**
)	**Fermez la parenthèse**
X	**OU exclusif**
XN	**Identité**
X(	**Test d'un combinaison en exclusif (**
XN(	**Test d'une combinaison en identité (**
FN	**Front descendant du RLG**
FP	**Front montant du RLG**

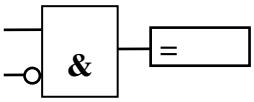
Ecrire des bits	
=	**Affecte le résultat logique d'un test**
S	**Mise à « 1 » (mémoire)**
R	**Mise à « 0 » (mémoire)**

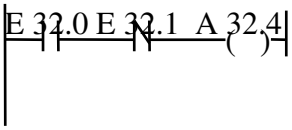
Exemple :

Représentation		
LISTe	**LOGigramme**	**CONTact**

Fonction ET

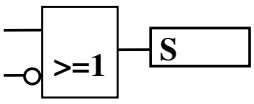
```
U     E 32.0
UN    E 32.1
=     A 32.4
```





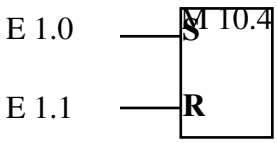
Fonction OU avec mise en mémoire

```
:O    M 25.4
:ON   E 32.3
:S    M 25.5
```



Fonction Mémoire (bascule RS)

```
:U    E 1.0
:S    M 10.4
:U    E 1.1
:R    M 10.4
```



:NOP 0 —— Cette instruction ne fait rien, elle permet la représentation graphique
:BE —— Cette instruction indique la fin d'un module

IV. / LES FLAGS

Les flags sont des bits vrais ou faux positionnés par certaines instructions dont dépendent d'autres instructions.

ETAT, 1er ?, RLG, RB, OU, BI1, BI0, OV, OS

4.1 L'état logique

C'est l'état de la variable binaire interrogée, la variable est à « 1 » ou à « 0 »

4.2 La première interrogation

Ce flag indique le début d'une opération booléenne, lorsqu'on écrit plusieurs équations booléennes les unes derrières les autres, chaque équation doit débuter par une première interrogation, sinon on combinerait l'équation en cours avec le résultat de l'équation précédente.

Pour avoir une première interrogation, il faut « limiter » l'équation précédente par une instruction.

EXEMPLE :

```
:U    E 1.2
:UN   E 1.3
:U    M 10.4
:=    A 4.5        Cette instruction limite l'équation ( le RLG )

:U    E 1.5        C'est une première interrogation
:U    E 1.6
:S    M 20.2       Cette instruction limite le RLG
```

Les instructions qui limitent le RLG sont:

=	Assignation
S	Mise à « 1 »
R	Mise à « 0 »
SE, SV, SI, SA, SS	Démarrage d'une temporisation
ZV,ZR	Comptage ou décomptage d'un compteur

☛ Au démarrage de L'OB1 la première interrogation est à « 1 »,
On peut donc commencer une équation booléenne.

4.3 Le RLG Résultat LoGique

Ce bit mémorise le résultat d'un test et permet de réaliser une combinaison avec l'instruction suivante.

Exemples :

	1er ?	ETAT	RLG	
:U E 1.4	1	1	1	
:UN E 1.6	0	0	1	RLG = RLG précédant ET PAS ETAT
:= A 4.5	0	1	1	A 4.5 prend la valeur du RLG
:O E 1.5	1	0	0	C'est une 1er interrogation donc on ne combine pas avec le RLG précédent
:ON E 1.6	0	0	1	RLG = RLG précédent OU PAS ETAT
:SE T 5	0	1	1	Démarre la tempo T5 si RLG = 1
	ACCU 1	ACCU 2	RLG	
:L MW 10	0030	?	1	Les accus sont représentés en HEXA
:L LF +50	0032	0030	1	
:>F			0	L'ACCU2 n'est pas > à l'ACCU1
:= A 4.7			0	A 4.7 prend la valeur du RLG

4.4 Instructions sur Résultat Logique

SAVE	Sauvegarde le Résultat logique dans le bit RB, pour tester le RB on utilise l'instruction : U BIE
SET	Mise à 1 inconditionnel du RLG
CLR	Mise à 0 inconditionnel du RLG
NOT	Complément du RLG

4.5 Le OU

Ce flag permet de mémoriser le résultat de la combinaison précédente pour la combiner en OU avec la combinaison suivante.

Exemple :

		1er?	ETAT	RLG	OU	
:U	E 1.5	1	1	1	0	
:UN	E 1.6	0	0	1	0	
:O		0	-	1	1	On mémorise le RLG
:UN	E 1.5	0	1	1	1	Le RLG se combine avec le OU
:U	E 1.6	0	0	1	1	
:=	A 4.1	0	1	1	0	« = » ferme l'équation, le OU retombe à « 0 »

4.6 Les bits de debordements

OV : Overflow, ce bit passe à 1 suite à une opération si le résultat déborde (sur 16 ou 32 bits), il peut être testé directement (ex : U OV) ou utilisé pour un saut conditionnel (SPO), le bit OV passe à 0 automatiquement au cycle suivant

OS : Overflow mémorisé, ce bit reste mémorisé jusqu'à la prochaine opération.

4.7 LES FLAGS BI1 et BI0

Ces bits donnent des informations sur le résultat des opérations

BI1	BI0	Description
1	0	>0
0	1	<0
0	0	=0
1	1	OU

On peut tester ces bits par les instructions

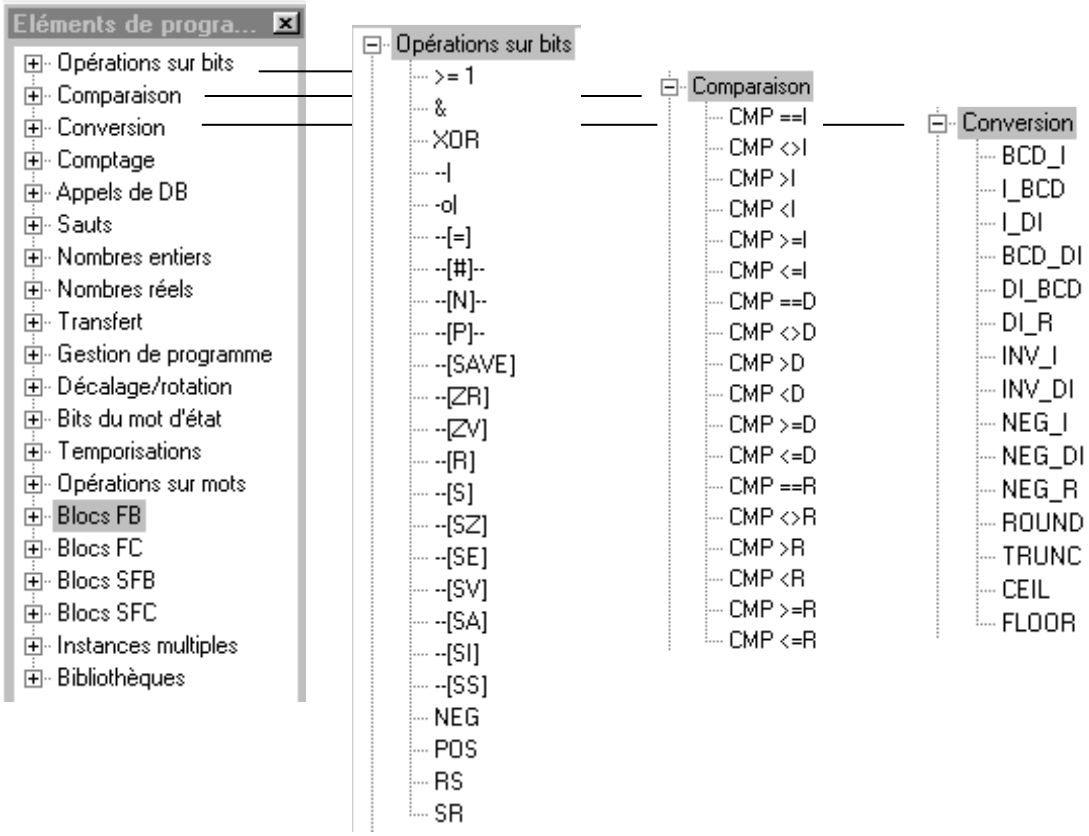
U	>0	Résultat positif
U	<0	Résultat négatif
U	==0	Résultat nul
U	<>0	Résultat non nul
U	>=0	Résultat positif ou nul
U	<=0	Résultat négatif ou nul

Ou par des instructions de saut

SPMZ	Saut si inférieur ou égal à 0
SPZ	Saut si égal 0
SPPZ	Saut si supérieur ou égal à 0
SPN	Saut si différent de 0

V. / ASSISTANT INSTRUCTIONS DE PROGRAMME

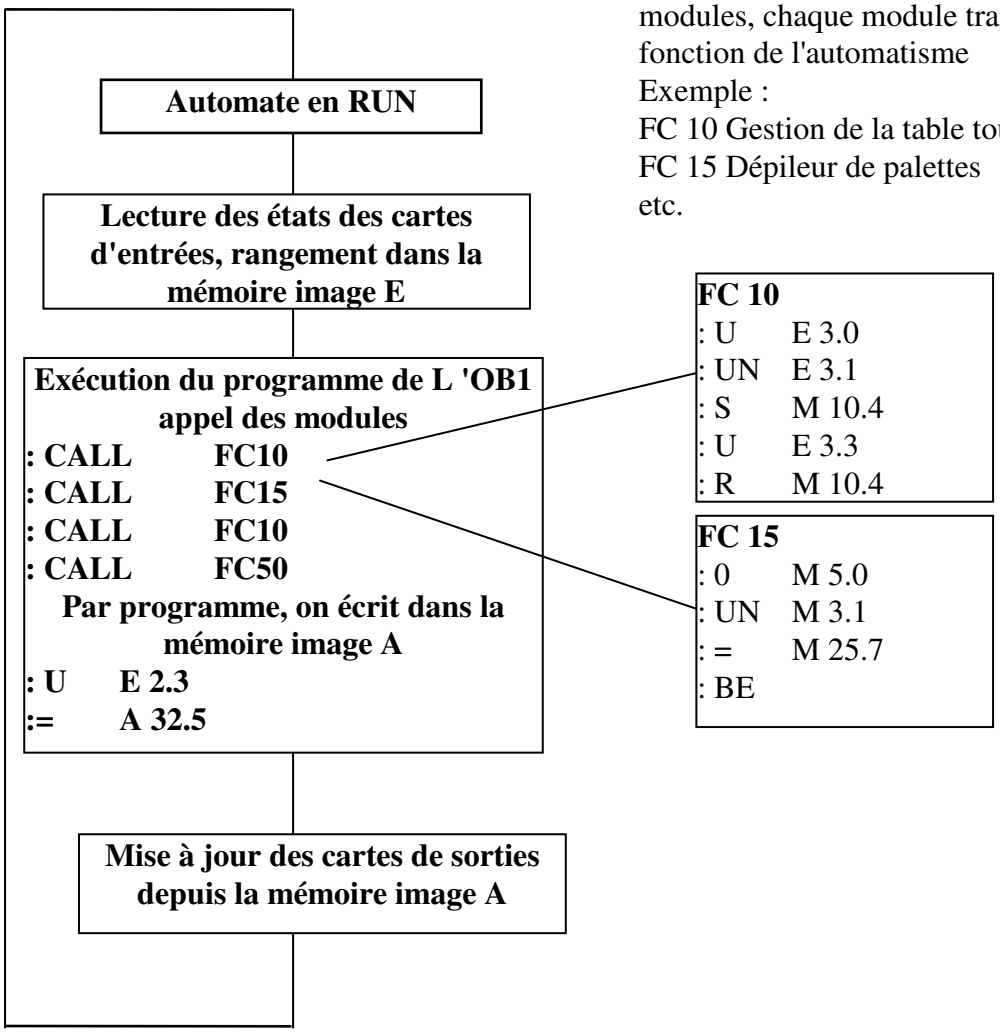
Menu Affichage puis Catalogue



On utilise beaucoup l'assistant quand on programme en Contact ou en Logigramme, cela permet de savoir quel sont les instructions disponibles.

VI. / STRUCTURE D'UN PROGRAMME

6.1 Exécution cyclique d'un programme OB1



Le programme est écrit dans différents modules, chaque module traitera une fonction de l'automatisme
Exemple :
FC 10 Gestion de la table tournante
FC 15 Dépileur de palettes
etc.

Il y a aussi des programmes qui ne sont pas cycliques et qui interrompent l'OB1, ce sont d'autres blocs d'organisations. Comme les OB d'horloge ou comme les OB de réaction aux erreurs voir le chapitre VIII

VII. / LES BLOCS DE CODE

Les OB **Blocs d'organisation** **OB 1 à OB 225**

Déterminent la structure du programme utilisateur. Les OB sont directement appelés par le système d'exploitation de la CPU en réaction à un événement (à condition toutefois de les avoir programmé et insérés dans l'automate).

Ils contiennent en général peut d'instructions, essentiellement des appels de blocs FC, FB

Les FB et FC **Fonctions et Blocs fonctionnels**

- Ce sont des fonctions écrites en LIST, CONTACT ou LOGIGRAMME,.
- Ils peuvent recevoir des paramètres d'entrées de sorties
- On peut y déclarer des variables locales temporaires
- Les blocs FB peuvent contenir des variables statiques qui sont sauvegardés dans un DB d'instance.

Les FB et FC s'adaptent particulièrement bien à la programmation de fonction récurrentes

☛ Contrairement aux FC, les FB sont des blocs avec mémoire, les paramètres transmis aux FB sont sauvegardés dans un bloc de donnée d'instance

Les SFB et SFC **Fonctions système et Blocs fonctionnels système**

Ce sont des blocs tout prêts. Il sont intégrés à la CPU S7 et ne peuvent être programmés par l'utilisateur.

☛ Les blocs de code (OB, FB, FC) peuvent être chargés dans la CPU S7, ils sont soit créer et éditer directement dans des éditeurs incrémentaux, soit ils résultent de la compilation de source

VIII. / LES BLOCS D'ORGANISATION

Programme de démarrage **OB 100**

La CPU effectue une mise en route après la mise sous tension

- Lorsque le commutateur de mode de fonctionnement est commuté de STOP à RUN
- A la demande d'une fonction de commutation (par commande de menu depuis la PG)

Programme cyclique **OB 1**

Lors d'une exécution normale de programme, les traitements se font de façon cyclique. L'exécution du programme contenu dans l'OB 1 est démarrer une fois par cycle (quand il est fini, il recommence). On peut se servir de l'OB 1 pour appeler des blocs de type FC ou FB.

OB de déclenchement sur alarmes

Alarmes horaires	**OB 10 à OB 17**
Alarmes temporisés	**OB 20 à OB 23**
Alarme d'horloge	**OB 30 à OB 38**
Alarme process	**OB 40 à OB 47**

OB de réaction aux erreurs asynchrones

Erreur de temps **OB 80**

Dépassement du temps de cycle

Erreur d'alimentation **OB 81**

- Pile de sauvegarde manquante ou tension insuffisante
- Défaillance de l'alimentation 24V dans le chassis de base ou d'extension

Alarme diagnostic **OB 82**

Pour module avec fonction diagnostic

Alarme de débrochage et enfichage **OB 83**

Débrochage ou enfichage de module

Erreur matérielle sur CPU **OB 84**

Erreur sur une interface de la CPU (Réseau MPI, bus K)

Erreur d'exécution de programme **OB 85**

- Erreur d'accès à un bloc
- Erreur d'accès aux mémoires images E/S

Défaillance de profile **OB 86**

Défaillance d'une station centralisée ou décentralisée

Erreur de communication **OB 87**

Erreur de communication par données globales

OB de réaction aux erreurs synchrones

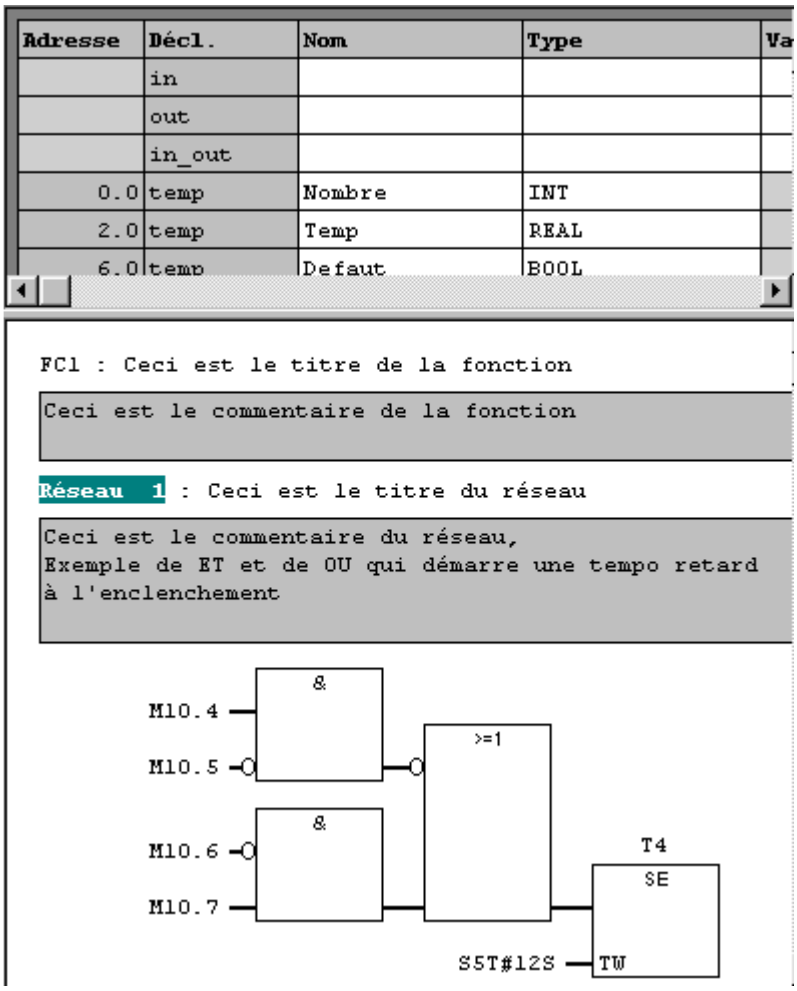
Erreur de programmation **OB 121**

- Erreur de conversion
- Bloc chargé

Erreur d'accès à la périphérie **OB 122**

Erreur d'accès en lecture ou en écriture à la périphérie

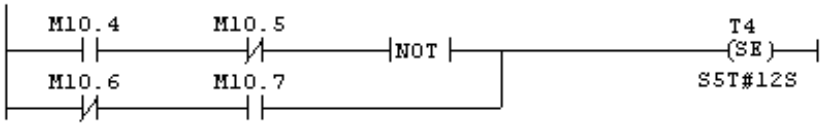
IX. / CONSTITUTION D'UN BLOC DE PROGRAMME



Dans cette zone on peut déclarer des paramètres d'entrées, de sortie, d'entrées et sortie, ainsi que des variables temporaires qui ne sont accessible que par ce bloc

Tous ces paramètres sont facultatifs

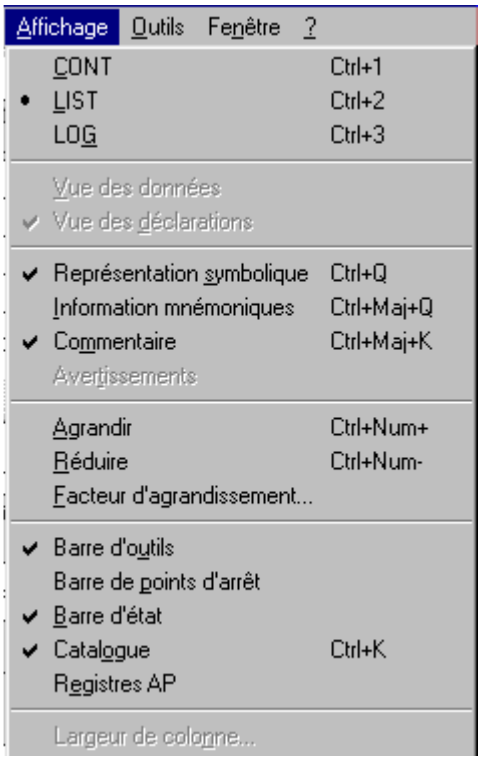
Représentation en contact



Représentation en LIST

```
U     M     10.4
UN    M     10.5
NOT
O
UN    M     10.6
U     M     10.7
L     S5T#12S
SE    T      4
```

Le menu Affichage permet de changer la représentation



Paramétrage des Fonctions et Blocs fonctionnels

Paramètre / variables	Description	Autorisé pour
IN	Paramètre d'entrée dont la valeur est fournie par le bloc de code appelant	FB, FC
OUT	Paramètre de sortie dont la valeur est fournie par le bloc de code appelé	FB, FC
IN_OUT	Paramètre dont la valeur est fournie après le bloc de code appelé	FB, FC
STAT	Variable statique contenue dans un DB d'instance	FB
TEMP	Variable temporaire rangée dans la pile des données locales. Les valeurs de ces variables sont perdues une fois l'exécution du bloc achevée.	FB, FC, OB

EXEMPLE

Adresse	Décl.	Nom	Type	Valeur initiale	Comme
0.0	in	hauteur	REAL		
4.0	in	Largeur	REAL		
8.0	out	Surface	REAL		
	in_out				
0.0	temp	i	INT		
2.0	temp	j	INT		
4.0	temp	tableau	ARRAY[1..5]		
*2.0	temp		INT		

Pour utiliser un paramètre de bloc dans un programme, on doit le faire précéder du symbole « # », exemple :

```
:L    #hauteur        // Charge la hauteur
:L    #Largeur        // Charge la largeur
:*R                   // Multiplie en nombre réel
:T    #Surface        // Transfert dans surface
```

TYPE DES PARAMETRES :

Les paramètres peuvent être des types simples tel que BYTE, BOOL, WORD, INT, DINT, REAL, S5TIME, etc.

Ou de types complexes tel que : TIMER, COUNTER, BLOCK_OB, BLOCK_FB, BLOCK_FC, BLOCK_DB

Ou de type tableau ARRAY[1..X]
Ou du type structure STRUCT

X. / INSTRUCTIONS SUR MOTS

Les opérations sur mots ne peuvent se faire que par l'intermédiaire des accumulateurs, les automates disposent de 2 à 4 ACCU de 16 à 64 bits suivant le modèle d'UC.

ACCU 1	ACCU 2

Opérations arithmétiques

ACCU1 = ACCU2 op ACCU1			**L MW10**
+	**I, D, R**	Addition	**L 25**
-	**I, D, R**	Soustraction	**+F**
*	**I, D, R**	Multiplication	**T MW10**
/	**I, D, R**	Division	

Comparaison des ACCUS

RLG = ACCU2 comp ACCU1			**L MW10**
>	**I, D, R**	Supérieur	**L 25**
<	**I, D, R**	Inférieur	**==I**
><	**I, D, R**	Différent	**= M10.4**
>=	**I, D, R**	Supérieur ou égal	
<=	**I, D, R**	Inférieur ou égal	
==	**I, D, R**	Egal	

Le terme **I** signifie une opération en entier (INT), le terme **D** signifie une opération en double entier (DINT) et le terme **R** signifie une opération en réel (REAL)

Opérations logiques

ACCU1 = ACCU2 op ACCU1			**L MW10**
UW	**UD**	ET bit à bit	**L W#16#000F**
OW	**OD**	OU bit à bit	**UW**
XOW	**XOD**	OU EXCLUSIF bit à bit	**T MW10**

Le terme **W** signifie une opération en mot (WORD) de 16 bit et le terme **D** signifie une opération en double mot (DWORD) de 32 bit

Opérations de conversions (voir §14.2)

ACCU1 = op ACCU1		**Exemple**
BTI	BCD vers entier	
DTR	Mot double entier vers flottant	**L MW10**
SLW n	Décalage à gauche	**SLW 8**
SRW n	Décalage à droite	**T MW10**

Remarque : la lettre **I** indique que l'opération se fait sur un nombre entier (INT) de 16 bits, **D** un nombre entier sur 32 bits (DINT), **R** un nombre réel (REAL) sur 32 bits.

Charger des valeurs dans les accumulateurs

Pour charger une valeur dans l'ACCU 1 on utilise l'instruction **L**, lorsqu'on charge l'ACCU1, l'ancienne valeur de l'ACCU1 se place dans l'ACCU 2.

Exemple :

MW10 = 5		**ACCU1**	**ACCU2**	
L	**MW10**	**5**	**?**	**On charge 5 dans l'ACCU1**
L	**7**	**7**	**5**	**On charge 7 dans l'ACCU1**
+F		**12**	**5**	**On fait la somme ACCU2 + ACCU1**
T	**MW14**	**12**	**5**	**Transfert de l'ACCU1 dans MW14**

XI. / LES REGISTRES DU PROCESSEUR

Il y a 4 accumulateurs 32 bits , ACCU1, ACCU2, ACCU3, ACCU4

Les instructions sur ACCU sont :

L	Chargement de l'accu1
T	Transfert de l'accu 1
TAK	Permutation de l'accu1 avec l'accu2
ENT	Transfert de l'accu2 dans Accu3 et accu3 dans Accu4
LEAVE	Transfert de l'accu4 dans Accu3 et Accu3 dans Accu2
PUSH	Accu1 -> Accu2 -> Accu3 -> Accu4
POP	Accu4 -> Accu3 -> Accu2 -> Accu1
TAW	Inversion des mots poids faible, poids fort de l'accu1
TAD	Inversion des mots double poids faible, poids fort de l'accu1

Exemple de programme : soit réaliser l'opération suivante:

(MW5 + 12) / (MW15 – MW10)

		ACCU1	ACCU2	ACCU3	
:L	MW5	MW5	?	?	// Charge MW5
:L	12	12	MW5	?	// Charge la valeur 12
:+I		MW5+12	?	?	// Additionne les accus
:TAK		?	MW5+12	?	// Inverse ACCU1 et ACCU2
:ENT		?	MW5+12	MW5+12	// Déplace l'ACCU2 vers ACCU3
:L	MW15	MW15	?	MW5+12	// Charge MW15 dans ACCU1
:L	MW10	MW10	MW15	MW5+12	// Charge 10 dans ACCU1
:-I		MW15-MW10	MW5+12	?	//Soustraits MW10 de MW15
:					// L'accu 3 remonte dans l'ACCU 2
:/I					// Divise Accu2 par accu1
:T	MW20				// Transfert le résultat dans MW20

Remarque:
A la place de TAK puis ENT on aurait put utiliser PUSH puis PUSH (nouvelle instruction qui n'existait pas sur Siemens S5).
L'ACCU3 remonte automatiquement dans l'ACCU2 à la suite d'une opération (+, -, *, /), L'ACCU3 ne se charge pas automatiquement, il faut une instruction comme ENT ou PUSH

Il y a aussi deux registres d'adressage indirect AR1 et AR2 voir § XIX

XII. / LES TEMPORISATEURS

Les temporisateurs sont des mots de 16 bits représentés par la lettre **T**.

Différents temporisateurs

SE T_____0 Retard à l'enclenchement (tempo travail)
SA 0_____T Retard au déclenchement (tempo repos)
SS T_____S Retard à l'enclenchement puis mémorisation à l'état « 1 », faire un reset
SI Limiteur d'impulsion
SV _____V Générateur d'impulsion (Monostable)

Principe

1 Charger une valeur dans l'ACCU1
2 Démarrer la tempo (l'ACCU1 se range dans la tempo)
3 La valeur de la tempo décompte jusqu'à 0, au rythme d'une base de temps

Les bases de temps sont

0 10 ms
1 100 ms
2 1 sec.
3 10 sec

4 Tester l'état de la tempo
5 Eventuellement tester la valeur courante de la tempo

Les instructions de temporisation

FR Valider (n'est pas utile)
L Charger l'ACCU 1 avec la valeur binaire pure du compteur
LC Charger L'ACCU 1 avec la valeur BCD du compteur
R Entrée de remise à zéro

Exemple :

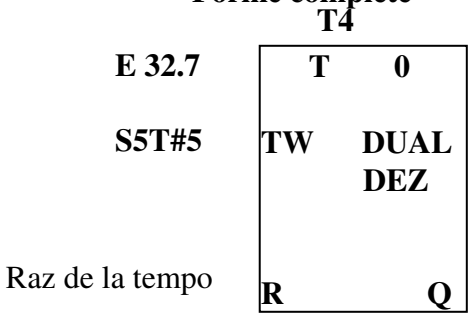
```
L     S5T5s      Charger l'ACCU1 avec une valeur de tempo 5 sec
U     E 32.7     Tester une condition de démarrage de la tempo
SE    T 4        Démarrage de la tempo avec la valeur contenue dans
l'ACCU 1

U     T4         Tester l'état de la tempo (si = « 1 » ⇒ tempo écoulée)
=     A 32.7
```

Facultatif

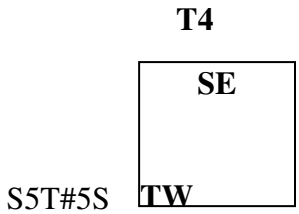
```
L     T4         Charger la valeur courante de la tempo
L     25         Charger la valeur 25
>F               Tester si valeur courante > 25
=     M 10.4     Affecter le résultat au bit M 10.4
```

Représentation graphique en contact ou logigramme

Forme complète



Forme simplifiée



XIII. / LES COMPTEURS

Dans les automatismes, on est souvent amené à compter (Nb de tour, Nb de paquets, Nb de palette, Nb de pesées etc...)

Les compteurs sont des mots de 16 bits représentés par la lettre **Z**, ils peuvent évoluer de 000 à 999, il y a 128 mots pour 128 compteurs (**Z 0** à **Z 127**).

Différents types de compteurs

- les compteurs d'incrémentation et de décrémentation,
- les compteurs d'incrémentation
- compteurs de décrementation.

Les instructions de comptage / décomptage

ZV	Entrée d'incrémentation
ZR	Entrée de décrémentation
R	RAZ compteur
S	Initialisation du compteur à la valeur contenue dans l'ACCU1
L	Charger l'ACCU 1 avec la valeur binaire pure du compteur
LC	Charger L'ACCU 1 avec la valeur BCD du compteur
FR	Valider

Exemple

```
:U     E 2.5      Si condition de comptage ( capteur par exemple )
:ZV    Z 5        Compter + 1 ( Le comptage se fait sur front montant du RLG )
:
:U     E 2.6      Si condition de décomptage
ZR     Z5         Enlever 1
:
:L     Z 5        Charger dans l'ACCU 1 la valeur courante du compteur
:L     KF + 50    Charger la valeur 50  dans l'ACCU1 (  Z5 passe dans l'ACCU2 )
:>=F              Si Z 5 supérieur ou égal à 50
:R     Z 5        Mettre le compteur à 0
:
:
:UN    Z 5        Si compteur vaut 0
:=     A 7.2      Signale compteur vide
```

Représentation graphique en contacts ou logigramme

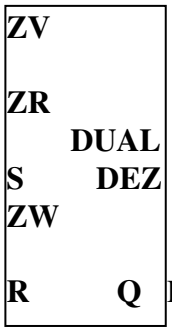
Forme complète

Z 5

E 2.5 — Entrée de décomptage

Entrée d'initialisation

Valeur d'initialisation **M 50.1**

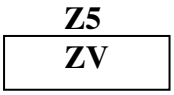


Valeur courante en BCD

Valeur courante en binaire

Etat du compteur = "0" si compteur = 0

Forme simplifiée



XIV. / INSTRUCTIONS COMPLEMENTAIRES

14.1 Opérations arithmétiques sur ACCU1

SQRT	**Racine carré**
SQR	**carré**
LN	**Logarithme naturel**
EXP	**Exponentiel**
SIN	**Sinus**
COS	**Cosinus**
TAN	**Tangente**
ASIN	**Arc Sinus**
ACOS	**Arc Cosinus**
ATAN	**Arc Tangente**

14.2 Opérations de conversion sur ACCU1

BTI	**Conversion BCD ➔ binaire 16 bits**
BTD	**Conversion BCD ➔ binaire 32 bits**
ITB	**Conversion entier ➔ BCD**
ITD	**Conversion entier 16 bits➔ entier 32 bits**
DTB	**Conversion entier 32 bits ➔ BCD 32 bits**
DTR	**Conversion entier 32 bits ➔ réel**
RND	**Arrondi au plus proche**
RND+	**Arrondi entier supérieur**
RND-	**Arrondi entier inférieur**
TRUNC	**Tronquer une partie entiére**
INVI	**Complément à 1 sur 16 bits**
INVD	**Complément à 1 sur 32 bits**
NEGI	**Complément à 2 sur 16 bits**
NEGD	**Complément à 2 sur 32 bits**
NEGR	**Inversion d'un nombre réel**

14.3 Décalages

SLWn	**Décalage à gauche 16 bits**
SLW	**Décalage à gauche 16 bits défini dans ACCU 2**
SLDn	**Décalage à gauche 32 bits**
SLD	**Décalage à gauche 32 bits défini dans ACCU 2**
SRWn	**Décalage à droite 16 bits**
SRW	**Décalage à droite 16 bits défini dans ACCU 2**
SRDn	**Décalage à droite 32 bits**
SRD	**Décalage à droite 32 bits défini dans ACCU 2**
SSIn	**Décalage arithmétique à droite 16 bits**
SSI	**Décalage arithmétique à droite 16 bits défini dans ACCU 2**
SSD	**Décalage arithmétique à droite 16 bits**
SSDn	**Décalage arithmétique à droite 16 bits défini dans ACCU 2**
RLDn	**Rotation à gauche 32 bits**
RLD	**Rotation à gauche 32 bits dans ACCU 2**
RRDn	**Rotation à droite 32 bits**
RRD	**Rotation à droite 32 bits dans ACCU 2**
RLDA	**Rotation d'un bit vers la gauche sur 33 bits**
RRDA	**Rotation d'un bit vers la droite sur 33 bits**

14.4 Les opérateurs de saut

SPA **Saut inconditionnel à une étiquette**
SPL **Saut vers liste (équivalent à 1 case)**

Exemple :

```
        :L    MB100 (MB 100 : variable a tester)
        :SPL  LIST
        :SPA  CAS0
        :SPA  CAS1
        :SPA  CAS2
LIST    :SPA  ELSE
CAS0 :
      :SPA FIN
                                  CAS1 :
:SPA FIN
                                  CAS2 :
:SPA FIN
FIN   :
```

SPB **Saut si RLG =1 vrai**
SPBN **Saut si RLG = 0**

SPBB

SPBNB **Saut si pas vrai**
LOOP Boucle de programme

Exemple :

```
        :L     +5     // Init de la boucle FOR , 5 fois
SUIV    :T     MB10
        : // instructions qui seront exécutés 5 fois
        :L     MB10 // Charge le compteur de boucle
        :LOOP  SUIV  // Décrémente l'ACCU1 et saute
```

XV. / REPRESENTATION DES NOMBRES

Les nombres sont des valeurs binaires constitués de « 1 » et de « 0 », on peut représenter ces nombres sous différentes formes, de manière à faciliter leurs lecture, il est par exemple plus compréhensible de lire **« 25 »** que sa représentation binaire **« 11001 »,** ou si on veut représenter un code ASCII il sera plus facile de lire **« A »** que son code binaire **« 100 0000 »,** etc.

Forme générale d'écriture d'une valeur

<Symbole de type> <Symbole de format> <Valeur>

15.1 Les types de données

Syntaxe de déclaration du type	Type	taille	Symbole	
BOOL	Bit	1	TRUE ou FALSE	
BYTE	Octet	8	B#	
WORD	Mot	16	W#	
DWORD	Double mot	32	DW#	
INT	Entier	16	rien	
DINT	Entier long	32	L#	
REAL	Réel	32	rien	
S5TIME	Durée de temporisation	16	S5T#	
TIME	Durée au standard CEI	32	T#	
DATE	Date au standard CEI	16	D#	
TIME_OF_DAY	Heure du jour	32	TOD#	
DATE_AND_TIME	Date et Heure	64	DT#	
CHAR	Caractère ASCII	8	' '	
POINTER	Pointeur	48	P#	

15.2 Les formats de représentation

Format	Représentation	Exemples	Commentaires
Décimal		214, -58	
Hexadécimal	16#	16#5FA2	
Binaire	2#	2#1001101011	
BCD	C#	C#5798	La valeur doit être BCD
Réel		12.21 $-458^{e}-24$	-458×10^{-24}
Durée de tempo	H_M_S_MS	2H_50M_35S_12MS 5S	
Durée CEI	D_H_M_S_MS	4D_11H_18M_50S_254MS	Durée en milisecondes
Date CEI	aaa-mm-jj	1998-12-28	
Heure	hh:mm:ss	10:32:50	
Date et Heure	aaaa-mm-jj-hh:mm:ss	1998-10-25-07:15:00	
Caractère	' '	'A', 'B', '*' etc.	

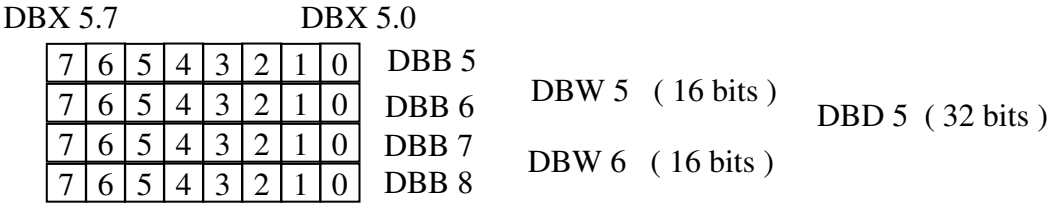
Exemple de représentation des valeurs :

W#16#FA5C	**Type WORD, Base 16, Valeur héxa FA5C**
• 15 245	La valeur décimale - 15 245
12^{e}-5	**12×10^{-5}**
L#254 789	**Type DINT, La valeur 254 789 tient sur 32 bits**
S5T#35S	**Durée de tempo = 35 secondes**
S5T#35S_10MS	**Durée de tempo = 35 seconde et 10 milisecondes**
T#21H_3M_5S	**Durée = 21 heures, 3 minutes et 5 secondes**
B#16#A5	**Type BYTE, Base 16, valeur = A5**
W#2#10010110111	**Type WORD, Base 2, valeur 10010110111**

XVI. / LES BLOCS DE DONNEES

Il s'agit de zones de données utilisées par les zones de code de programme utilisateur pour sauvegarder des valeurs.

Les blocs de données contiennent des octets DBB, 2 octets consécutifs forme un mot DBW, 4 octets forme un double mot DBD



X : Représente un bit, B : un octet, W : un mot, D : un double mot

16.1 Type de blocs de données :

Les Blocs de données globaux

Blocs de données auxquels peuvent accéder tous les blocs du code du programme utilisateur S7. Toutes FB, FC ou OB peut y lire, ou écraser des données.

Les Blocs de données d'instance

Les bloc de données d'instances sont des blocs ordinaires, dans lesquels sont stockés tous les paramètres d'un FB, à chaque fois qu'on appel un FB on doit préciser quel est le DB d'instance, il peut y avoir plusieurs DB d'instance pour un même FB. Les données sont locales.

16.2 Utilisation des blocs de données

On peut utiliser les octets (ou mot ou double ou bits) de données de deux manières, soit en déclarant le DB au préalable grâce à l'instruction **AUF DB** soit en précisant le nom du DB devant la variable.

Exemple :

```
   // Ouverture préalable d'un DB
:AUF  DB 10          Déclare le DB 10 ouvert
:L    DW 50          Ajouter 5 au mot DW 5 du DB 10
:L    5
   :+F
   :T DW 50
:
:L    DW 2           Comparer le mot 2 du DB 10 à la valeur 25
:L    25
   :<F
   := M 10.4
:
   // On accède à la variable en précisant le DB
:L    DB20.DBW 2              Charger le DBW 2 du DB 20
:T    DB25.DBW 2              Le ranger dans le mot DBW 2 du DB 25
:
:
:L    DB25.DBB 8        Charger l'octet DBB 8 du DB 25
:T    DB25.DBB 6        Le ranger dans le DBB 6 du DB 25
:
:U    DB25.DBX 200.4    Tester le bit 4 de l'octet DBB200 du DB 25
:=    M 120.0
:
     :L    «blocDB ».ref    Accès au champs « ref » du bloc de données dont le symbole est
   blocDB
:
: AUF DI    4               Ouvre le DB d'instance 4, voir chapitre suivant
```

16.3 Utilisation de l'éditeur de bloc de données

On a accès à tout les types de données INT, DINT, WORD, DWORD , REAL, BOOL, S5TIME POINTER, TIME, CHAR etc..

On a accès aux structures STRUCT

Chaque variable dans un DB contient un nom, un type, un commentaire, mais vous pouvez aussi leur affecter une valeur par défaut

Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	Nb	INT	12	Nombre Mesure
+2.0	TachyTremie1	STRUCT		
+0.0	pv	REAL	0.000000e+000	Valeur mise à l'échelle Vites
+4.0	pts	INT	0	Valeur en points Vitesse du t
+6.0	min	REAL	0.000000e+000	Valeur mini echelle Vitesse d
+10.0	max	REAL	0.000000e+000	Valeur maxi echelle Vitesse d
+14.0	mini	INT	0	Nombre de pt mini Vitesse du
+16.0	maxi	INT	10000	Nombre de pt maxi Vitesse du
+18.0	hgmin	INT	0	Valeur hors gamme mini Vitess
+20.0	hgmax	INT	12000	Valeur hors gamme maxi Vitess
+22.0	def	BOOL	FALSE	En défaut Vitesse du tapis tr
+22.1	acq	BOOL	FALSE	Acquittement Vitesse du tapis

On accède à une variable soit par sont adresse défini sur la colonne de gauche, par exemple DBW14 correspond à la variable « mini » soit par son nom par exemple « Nb » correspond au « Nombre Mesure »

Supposons que ce DB soit le DB25

DB25.Nb donne accès au Nombre Mesure

DB25.TachyTremie1.mini donne accès au « mini » de la structure TachyTremie1

Supposons que le DB25 s'appelle lui-même « Mesure »

Mesure.Nb donne accès au Nombre Mesure

Mesure.TachyTremie1.mini donne accès au « mini » de la structure TachyTremie1

XVII. / LES BLOCS DE DONNEES D'INSTANCE

Les blocs de données d'instance sont affectés au FB/SFB au moment de l'appel du bloc. Ils sont générés automatiquement lors de la compilation.

```
CALL  FB    10 , DB10          // Appel le FB 10 avec le DB 10
CALL  FB    10 , DB11          // Appel le FB 10 avec le DB 11
```

Les DB d'instances, servent à sauvegarder le contexte d'un FB entre 2 appels, les paramètres sauvegardés sont : IN, OUT, IN_OUT et STATIC, les paramètres TEMP ne sont pas sauvegardés.

17.1 Création d'un DB

Lors de la création d'un DB, le système demande comment on veut le créer :

Bloc de données: On définira chaque valeurs du DB manuellement

Bloc de données associé à un type de données utilisateur : le DB se crée automatiquement d'après une définition, il faut au préalable créer un «UDT»

Bloc de données associé à un bloc fonctionnel (FB) : le DB se crée automatiquement avec les paramètres du FB, le FB doit existé.

XVIII. / ADRESSAGE INDIRECT ZONE MEMOIRE

18.1 Pointeur 32 bits

Les espaces mémoires E, A, M, L, DB sont accessibles par un pointeur 32 bits structuré de la manière suivante :

ZONES E, A, M, L, DBB

a	0	0	0	0	t	t	t	0	0	0	0	0	o	o	o
o	o	o	o	o	o	o	o	o	o	o	o	o	b	b	b

a : Mode d'adressage (0: Indirect intra zone, 1: indirect inter zone)

ttt : Identificateur de zone si a=1 sinon ttt= 000
oooooooooooooo : N° de l'octet de 0 à 65535
bbb : N° du bit de 0 à 7

EX : 00000000000000000000000000010100 = Octet 2, Bit 4

La valeur du pointeur peut être représenté sous la forme:
P#octet.bit

Exemples de programme : (Ces programme ne sont que des exemples d'utilisation d'adressage)

```
// Le but est tester le bit M 2.4
L     P#2.4                  // Chargement de la valeur du pointeur
T     MD100                  // Transfert dans le mot MD100
U     M[MD100]               // Test de M2.4

// Le but est de transferer le Mot DBW40 dans le mot DBW42, on aurait put faire bcp plus simple
:L    P#40.0                 // Chargement de la valeur de pointeur
:T    MD100                  // Transfert dans le pointer MD100
:AUF  DB50                   // Ouverture du DB 50
:L    DBW[MD100]             // Lecture du mot DBW40
:L    MD100                  // Lecture du pointeur MD100
:L    P#2.0                  // Chargement de la valeur 2.0
:+I                          // Ajoute
:T    MD100                  // Transfert dans le pointeur MD100
:AUF  DB51                   // Ouverture DB 50
:T    DBB[MD100]             // Ecriture du mot DBW42 ( = 40 + 2 )
```

18.2 Pointeur 16 bits

Les espaces mémoires T, Z, BLOCS sont accessibles par un pointeur 16 bits structuré de la manière suivante :

POINTEURS 16 BITS
ZONES T, Z, BLOCS (DB, FB, FC, OB)

0	0	0	0	0	0	0	0	0	0	0	0	0	o	o	o

Exemple :

```
L     30              // Charge le N° de DB 30
T     MW80                  // Transfert dans pointeur MW80
AUF   DB[MW80]        // Ouvre le DB 30
U     E 0.0           // Teste l'entrée E 0.0
SE    T[MW80]         // Demarre la tempo T 30
```

XIX. / ADRESSSAGE INDIRECT PAR REGISTRE (AR1 ET AR2)

Les registres **AR1** et **AR2** sont des registres d'adresses, ils sont chargé par les instructions LAR1 et LAR2
Exemple :
LAR1 P#4.5 // Chargement du registre AR1 avec la valeur 4.5
LAR1 P#M34.5 // Chargement du registre AR1 avec la valeur 34.5 et 83h dans le poids fort, le poids fort précise le type de mémoire (M = 83h)

Exemple d'utilisation des registres AR1 et AR2

```
LAR1  P#0.0                 // Chargement du  registre AR1 avec la valeur 0.0
L     MB[AR1,P#50.0]        // Chargement de l'octet d'adresse AR1+P#50.0
```

Exemple de programme
Transfert de 10 octets MB 50 à 59 dans le DB25 de DBB10 à DBB 19

```
      :LAR1 P#0.0                 // Initialisation du pointeur à 0
      :L    10                    // Chargement valeur boucle = 10 fois
BOUC:T      MW100                 // Transfert dans indice de boucle
      :L    MB[AR1,P#50.0]        // Charge MB 50 + AR1 ( MB50 à MB59)
      :AUF  DB25                  // Ouvre le DB 25
      :T    DBB[AR1,P#10.0]       // Transfert dans DBB 10 + AR1
      :+AR1 P#1.0                 // Ajoute 1 au pointeur
      :L    MW100                 // Charge l'indice de boucle
      :LOOP BOUC                  // Décrémente accu1 et boucle tant que > 0
```

XX. / PARAMETRES DE TYPE ANY

Le parametre de type ANY à une taille de 10 octets, définissant le type de données, le nombre d'élément, le N° DU DB, la zone mémoire, le N° d'octet et le N° de bit

Toujours = 10h	Type de données	
Nombre d'éléments		
N° de DB ou 0 si pas de DB		
Zone mémoire (h8x)		
N° d'octet		N° Bit

Type de données :

Type	Code Hexa
BOOL	01
BYTE	02
CHAR	03
WORD	04
INT	05
DWORD	06
DINT	07
REAL	08
DATE	09
TOD	0A
TIME	0B
S5TIME	0C
DT	0E
STRING	13
BLOCK_FB	17
BLOCK_FC	18
BLOCK_DB	19
BLOCK_SDB	1A
COUNTER	1C
TIMER	1D

Zone mémoire

Zone	Quartet fort	Quartet faible
E	8	1
A	8	2
M	8	3
DB	8	4
DI	8	5
L	8	6

Exemple d'utilisation du parametre ANY en langage source

```
FUNCTION   FC 10:VOID
VAR_TEMP
        Source              ANY;
        Destination   ANY;
END_VAR
BEGIN
LAR1  P#Source;          // Registre d'adresse = adresse du parametre #source
LAR2  P#Destination;     // Registre d'adresse = adresse du parametre #destination

L       B#16#10;         // Charge l'Identificateur 10h
T       LB [AR1,P#0.0];
L       B#16#02;         // Charge le type BYTE
T       LB [AR1,P#1.0];
L       10;              // Charge le nombre d'éléments ( 10 octets )
T       LW [AR1,P#2.0];
L       22;              // Charge le N° de DB 22
T       LW [AR1,P#4.0];
L       P#DBX11.0;       // Charge le type DB et l'adresse 11
T       LD [AR1,P#6.0];

L       B#16#10;         // Charge l'Identificateur 10h
T       LB [AR2,P#0.0];
L       B#16#02;         // Charge le type BYTE
T       LB [AR2,P#1.0];
L       10;              // Charge le nombre d'éléments ( 10 octets )
T       LW [AR2,P#2.0];
L       15;              // Charge le N° de DB 15
T       LW [AR2,P#4.0];
L       P#DBX50.0;       // Charge le type DB et l'adresse 50
T       LD [AR1,P#6.0];

// Appel la fonction block move
CALL SFC 20 ( SRC_BLK := #Source, RET_VAL := MW12, DST_BLK := #Destination);
END_FUNCTION
```