SIEMENS

SIMATIC

Langage LIST pour SIMATIC S7-300/400

Manuel de référence

Ce manuel est livré avec la documentations référencée :
6ES7810-4CA10-8CW1

05/2010
A5E02790284-01

Mentions légales

Signalétique d'avertissement

Ce manuel donne des consignes que vous devez respecter pour votre propre sécurité et pour éviter des dommages matériels. Les avertissements servant à votre sécurité personnelle sont accompagnés d'un triangle de danger, les avertissements concernant uniquement des dommages matériels sont dépourvus de ce triangle. Les avertissements sont représentés ci-après par ordre décroissant de niveau de risque.

⚠DANGER
signifie que la non-application des mesures de sécurité appropriées **entraîne** la mort ou des blessures graves.

⚠ATTENTION
signifie que la non-application des mesures de sécurité appropriées **peut entraîner** la mort ou des blessures graves.

⚠PRUDENCE
accompagné d' un triangle de danger, signifie que la non-application des mesures de sécurité appropriées peut entraîner des blessures légères.

PRUDENCE
non accompagné d' un triangle de danger, signifie que la non-application des mesures de sécurité appropriées peut entraîner un dommage matériel.

IMPORTANT
signifie que le non-respect de l'avertissement correspondant peut entraîner l'apparition d'un événement ou d'un état indésirable.

En présence de plusieurs niveaux de risque, c'est toujours l'avertissement correspondant au niveau le plus élevé qui est reproduit. Si un avertissement avec triangle de danger prévient des risques de dommages corporels, le même avertissement peut aussi contenir un avis de mise en garde contre des dommages matériels.

Personnes qualifiées

L' appareil/le système décrit dans cette documentation ne doit être manipulé que par du **personnel qualifié** pour chaque tâche spécifique. La documentation relative à cette tâche doit être observée, en particulier les consignes de sécurité et avertissements. Les personnes qualifiées sont, en raison de leur formation et de leur expérience, en mesure de reconnaître les risques liés au maniement de ce produit / système et de les éviter.

Utilisation des produits Siemens conforme à leur destination

Tenez compte des points suivants:

⚠ATTENTION
Les produits Siemens ne doivent être utilisés que pour les cas d'application prévus dans le catalogue et dans la documentation technique correspondante. S'ils sont utilisés en liaison avec des produits et composants d'autres marques, ceux-ci doivent être recommandés ou agréés par Siemens. Le fonctionnement correct et sûr des produits suppose un transport, un entreposage, une mise en place, un montage, une mise en service, une utilisation et une maintenance dans les règles de l'art. Il faut respecter les conditions d'environnement admissibles ainsi que les indications dans les documentations afférentes.

Marques de fabrique

Toutes les désignations repérées par ® sont des marques déposées de Siemens AG. Les autres désignations dans ce document peuvent être des marques dont l'utilisation par des tiers à leurs propres fins peut enfreindre les droits de leurs propriétaires respectifs.

Exclusion de responsabilité

Nous avons vérifié la conformité du contenu du présent document avec le matériel et le logiciel qui y sont décrits. Ne pouvant toutefois exclure toute divergence, nous ne pouvons pas nous porter garants de la conformité intégrale. Si l'usage de ce manuel devait révéler des erreurs, nous en tiendrons compte et apporterons les corrections nécessaires dès la prochaine édition.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
ALLEMAGNE

A5E02790284-01
℗ 02/2010

Avant-propos

Objet du manuel

Ce manuel vous aidera à écrire des programmes utilisateur en langage LIST.

Il contient une partie de référence décrivant la syntaxe et le fonctionnement des éléments du langage de programmation LIST.

Connaissances fondamentales requises

Ce manuel s'adresse aux programmeurs souhaitant élaborer des programmes S7 ainsi qu'au personnel chargé de la mise en service et de la maintenance.

La compréhension du manuel requiert des connaissances générales dans le domaine de la technique d'automatisation.

Nous supposerons en outre des connaissances dans l'utilisation d'ordinateurs ou autres équipements (par exemple consoles de programmation) analogues au PC et des systèmes d'exploitation MS Windows XP, MS Windows Server 2003 ou MS Windows 7.

Domaine de validité du manuel

Le présent manuel est valable pour le logiciel STEP 7 V5.5.

Norme

LIST correspond au langage « Liste d’instructions » défini dans la norme CEI 1131-3. Pour plus de renseignements à ce sujet, consultez la table de correspondance à la norme dans le fichier NORM_TBL.RTF (anglais) ou NORM_TAB.RTF (allemand) de STEP 7.

Connaissances requises

Vous trouverez dans l'aide en ligne de STEP 7 les connaissances thèoriques sur les programmes S7 nécessaires à la compréhension de ce manuel sur LIST. Les langages de programmation se basant sur le logiciel de base STEP 7, nous supposerons que vous savez utiliser ce logiciel et sa documentation.

Ce manuel fait partie de la documentation "STEP 7 Connaissances fondamentales“.

Le tableau suivant présente la documentation de STEP 7:

Manuel	Objet	Numéro de référence
STEP 7 Connaissances fondamentales avec • STEP 7 Getting Started • Programmer avec STEP 7 • Configuration matérielle et communication dans STEP 7 • STEP 7 Pour une transition facile de S5 à S7	Connaissances fondamentales pour le personnel technique. Décrit la marche à suivre pour réaliser des tâches d‘automatisation avec STEP 7 et S7-300/400.	6ES7810-4CA10-8CW0
STEP 7 Manuels de référence sur les • Langages CONT/LOG/LIST pour SIMATIC S7-300/400 • Logiciel système pour SIMATIC S7-300/400 Fonctions standard et fonctions système Volume 1 et Volume 2	Manuels de référence décrivant les langages de programmation CONT, LOG et LIST de même que les fonctions standard et les fonctions système en complément des connaissances fondamentales de STEP 7.	6ES7810-4CA10-8CW1

Aides en ligne	Objet	Numéro de référence
Aide de STEP 7	Connaissances fondamentales pour la programmation ainsi que pour la configuration du matériel avec STEP 7, sous forme d‘aide en ligne.	Fait partie du logiciel STEP 7
Aides de référence de LIST/CONT/LOG Aide de référence sur les SFB/SFC Aide de référence sur les blocs d‘organisation	Aides en ligne contextuelles de référence	Fait partie du logiciel STEP 7

Aide en ligne

En complément au manuel, l'aide en ligne intégrée au logiciel vous offre une assistance détaillée lors de l'utilisation du logiciel.

Ce système d'aide est intégré au logiciel grâce à plusieurs interfaces :

- L'aide contextuelle donne des informations sur le contexte actuel, par exemple sur une boîte de dialogue ouverte ou sur une fenêtre active. Vous l'appelez en cliquant sur le bouton "Aide" ou en appuyant sur la touche F1.
- Le menu d'aide **?** propose plusieurs commandes : **Rubrique d'aides** ouvre le sommaire de l'aide de STEP 7.
- Vous obtenez le glossaire relatif à toutes les applications de STEP 7 en cliquant sur **"Glossaire".**

Ce manuel est extrait de l' "Aide pour LIST". En raison de la structure similaire entre le manuel et l'aide en ligne, le passage de l'un à l'autre est aisé.

Assistance supplémentaire

Si des questions sont restées sans réponse dans ce manuel, veuillez vous adresser à votre interlocuteur Siemens dans la filiale ou l'agence de votre région.

Vous trouvez votre interloculeur sous :

http://www.siemens.com/automation/partner

Vous trouvez un fil rouge pour la recherche de documentations techniques sur les produits et systèmes SIMATIC à l'adresse suivante sur Internet :

http://www.siemens.com/simatic-tech-doku-portal

Le catalogue en ligne et le système de commande en ligne se trouvent à l'adresse :

http://mall.automation.siemens.com/

Centre de formation SIMATIC

Nous proposons des cours de formation pour vous faciliter l'apprentissage des automates programmables SIMATIC S7. Veuillez vous adresser à votre centre de formation régional ou au centre principal à D 90026 Nuremberg.

Internet: http://www.sitrain.com

Technical Support

Vous pouvez joindre le support technique pour tous les produits d'Industry Automation.

- Via le formulaire Web de demande d'assistance (Support Request)
  http://www.siemens.com/automation/support-request

Vous trouvez plus d'informations concernant notre Technical Support sur Internet à l'adresse suivante :

http://www.siemens.com/automation/service

Service & Support sur Internet

En plus de la documentation offerte, vous trouvez la totalité de notre savoir-faire en ligne sur Internet à l'adresse suivante :

http://www.siemens.com/automation/service&support

Vous y trouvez :

- le bulletin d'informations qui vous fournit constamment les dernières informations sur le produit,
- les documents dont vous avez besoin à l'aide de la fonction de recherche du Support produit,
- le forum où utilisateurs et spécialistes peuvent échanger des informations,
- votre interlocuteur Industry Automation sur site,
- des informations sur les réparations, pièces de rechange et la consultation.

Sommaire

1 Opérations combinatoires sur bits

1.1 Vue d'ensemble des opérations combinatoires sur bits

Description

Les opérations combinatoires sur bits utilisent deux chiffres : 1 et 0. Ces deux chiffres sont à la base du système de numération binaire et sont appelés chiffres binaires ou bits. Pour les contacts et les bobines, 1 signifie activé ou excité et 0 signifie désactivé ou désexcité.

Les opérations de combinaison sur bits évaluent les états de signal 1 et 0 et les combinent selon la logique booléenne. Le résultat de ces combinaisons est égal à 1 ou 0. Il s'agit du résultat logique (RLG).

Vous disposez des opérations de base suivantes :

- U ET
- UN ET NON
- O OU
- ON OU NON
- X OU exclusif
- XN OU NON exclusif

Les opérations suivantes permettent de combiner des parties de séquence combinatoire figurant entre parenthèses :

- U(ET d'une expression
- UN(ET NON d'une expression
- O(OU d'une expression
- ON(OU NON d'une expression
- X(OU exclusif d'une expression
- XN(OU NON exclusif d'une expression
-) Fermer la parenthèse d'une expression

Les opérations suivantes mettent fin à une séquence combinatoire :

- = Affectation
- R Mettre à 0
- S Mettre à 1

Les opérations suivantes vous permettent de modifier le résultat logique RLG :

- NOT Négation du RLG
- SET Mettre RLG à 1
- CLR Mettre RLG à 0
- SAVE Sauvegarder RLG dans le bit RB

Les opérations suivantes détectent les transitions dans le résultat logique RLG et y réagissent :

- FN Front descendant
- FP Front montant

1.2 U ET

Format

U <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

U (ET)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 1 et combine le résultat de l'interrogation au RLG selon la table de vérité ET.

Interrogation de l'état des bits du mot d'état :

L'opération **ET** vous permet d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	x	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais	
		Barre d'alimentation	
U	**E 1.0**	E 1.0 état de signal 1	Contact à fermeture
U	**E 1.1**	E 1.1 état de signal 1	Contact à fermeture
=	**A 4.0**	A 4.0 état de signal 1	Bobine
Montre un contact fermé.			

1.3 UN ET NON

Format

UN <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

UN (ET NON)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 0 et combine le résultat de l'interrogation au RLG selon la table de vérité ET.

Interrogation de l'état des bits du mot d'état :

L'opération **ET NON** vous permet d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	x	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais	
		Barre d'alimentation	
U	**E 1.0**	E 1.0 Etat de signal 0	Contact à fermeture
UN	**E 1.1**	E 1.1 Etat de signal 1	Contact à ouverture
=	**A 4.0**	A 4.0 Etat de signal 0	Bobine

1.4 O OU

Format

O <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

O (OU)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 1 et combine le résultat de l'interrogation au RLG selon la table de vérité OU.

Interrogation de l'état des bits du mot d'état :

L'opération **OU** vous permet d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
O	**E 1.0**	E 1.0 état de signal 1 Contact à fermeture
O	**E 1.1**	E 1.1 état de signal 0 Contact à fermeture
=	**A 4.0**	A 4.0 état de signal 1 Bobine
		Montre un contact fermé.

1.5 ON OU NON

Format

ON <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

ON (OU NON)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 0 et combine le résultat de l'interrogation au RLG selon la table de vérité OU.

Interrogation de l'état des bits du mot d'état :

L'opération **OU NON** vous permet aussi d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
O	**E 1.0**	E 1.0 Etat de signal 0 — Contact à fermeture
ON	**E 1.1**	E 1.1 Etat de signal 1 — Contact à ouverture
=	**A 4.0**	A 4.0 Etat de signal 1 — Bobine

1.6 X OU exclusif

Format

X <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

X (OU exclusif)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 1, et combine le résultat de cette interrogation au RLG selon la table de vérité OU exclusif.

Vous avez également la possibilité d'appliquer plusieurs fois d'affilée l'opération OU exclusif. Le RLG global sera alors égal à "1", lorsqu'un nombre impair des opérandes interrogés fournit le résultat "1".

Interrogation de l'état des bits du mot d'état :

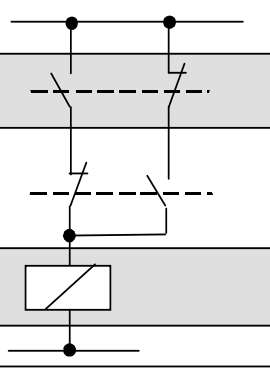
L'opération **OU exclusif** vous permet aussi d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
X	**E 1.0**	Contact E 1.0
X	**E 1.1**	Contact E 1.1
=	**A 4.0**	A 4.0 Bobine



1.7 XN OU NON exclusif

Format

XN <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D, T, Z

Description de l'opération

XN (OU NON exclusif)

Cette opération interroge le bit en accès afin de déterminer si son état de signal est 0, et combine le résultat de cette interrogation au RLG selon la table de vérité OU exclusif.

Interrogation de l'état des bits du mot d'état :

L'opération **OU NON exclusif** vous permet aussi d'interroger directement le mot d'état. A cet effet, utilisez les opérandes suivants : ==0, <>0, >0, <0, >=0, <=0, UO, BIE, OS, OV (BIE correspond à RB, OS à DM et OV à DEB).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
X	**E 1.0**	Contact E 1.0
XN	**E 1.1**	Contact E 1.1
=	**A 4.0**	A 4.0 Bobine

1.8 O ET avant OU

Format

O

Description de l'opération

O

Cette opération exécute, selon le principe ET avant OU, la combinaison OU sur des combinaisons ET.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	x	1	-	x

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
U	**E 0.0**	E 0.0, M 10.1
U	**M 10.0**	E 0.2
O		
U	**E 0.2**	M 10.0
U	**M 0.3**	M 0.3
O	**M 10.1**	
=	**A 4.0**	A 4.0 Bobine

1.9 U(ET d'une expression

Format

U(

Description de l'opération

U((ET d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La piles des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
U(		
O	**E 0.0**	E 0.0
O	**M 10.0**	M 10.0
)		
U(		
O	**E 0.2**	E 0.2
O	**M 10.3**	M 10.3
)		
U	**M 10.1**	M 10.1
=	**A 4.0**	A 4.0 Bobine

1.10 UN(ET NON d'une expression

Format

UN(

Description de l'opération

UN((ET NON d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La pile des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

1.11 O(OU d'une expression

Format

O(

Description de l'opération

O((OU d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La pile des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

1.12 ON(OU NON d'une expression

Format

ON(

Description de l'opération

ON((OU NON d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La pile des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

1.13 X(OU exclusif d'une expression

Format

X(

Description de l'opération

X((OU exclusif d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La pile des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

1.14 XN(OU NON exclusif d'une expression

Format

XN(

Description de l'opération

XN((OU NON exclusif d'une expression)

Cette opération sauvegarde les bits RLG et OU ainsi qu'un code d'opération dans la pile des parenthèses. La pile des parenthèses peut contenir jusqu'à 7 entrées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

1.15) Fermer la parenthèse d'une expression

Format

)

Description de l'opération

) (Fermer la parenthèse d'une expression)

Cette opération efface une entrée de la pile des parenthèses, restaure le bit OU, combine le bit RLG contenu dans l'entrée de la pile au RLG en cours conformément au code d'opération et affecte le résultat au RLG. S'il s'agit d'une opération ET ou ET NON, celle-ci tient également compte du bit OU.

Opérations ouvrant une expression entre parenthèses :

- U(ET d'une expression
- UN(ET NON d'une expression
- O(OU d'une expression
- ON(OU NON d'une expression
- X(OU exclusif d'une expression
- XN(OU NON exclusif d'une expression

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	x	1	x	1

Exemple

Programme LIST		Schéma de circuit à relais
		Barre d'alimentation
U(		
O	**E 0.0**	E 0.0
O	**M 10.0**	M 10.0
)		
U(		
O	**E 0.2**	E 0.2
O	**M 10.3**	M 10.3
)		
U	**M 10.1**	M 10.1
=	**A 4.0**	A 4.0 Bobine

1.16 = Affectation

Format

= <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D

Description de l'opération

= <bit>

Cette opération sauvegarde le RLG dans le bit en accès si le relais de masquage (Master Control Relay) est en fonction (MCR = 1). Si le relais MCR égale 0, c'est la valeur 0 et non le RLG qui est écrite dans le bit.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	-	0

Exemple

Programme LIST

```
U     E 1.0
=     A 4.0
```

Schéma de circuit à relais

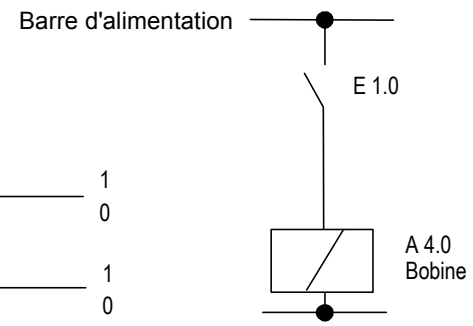
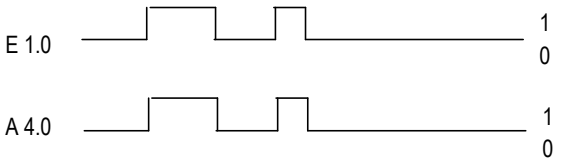


Schéma des états de signaux



1.17 R Mettre à 0

Format

R <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D

Description de l'opération

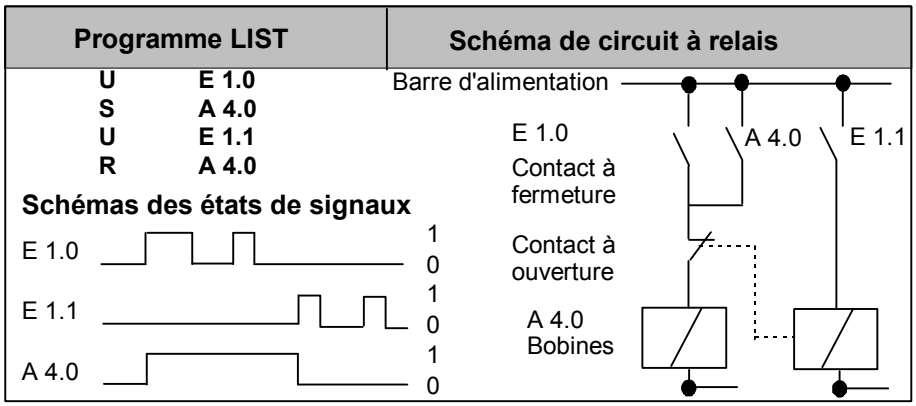
R (Mettre à 0)

Cette opération écrit la valeur 0 dans le bit en accès si le RLG égale 1 et si le relais de masquage (Master Control Relay) est en fonction (MCR = 1). Si le relais MCR égale 0, le bit indiqué n'est pas modifié.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	-	0

Exemple



1.18 S Mettre à 1

Format

S <bit>

Opérande	Type de données	Zone de mémoire
<bit>	BOOL	E, A, M, L, D

Description de l'opération

S (Mettre à 1)

Cette opération écrit la valeur 1 dans le bit en accès si le RLG égale 1 et si le relais de masquage (Master Control Relay) est en fonction (MCR = 1). Si le relais MCR égale 0, le bit indiqué n'est pas modifié.

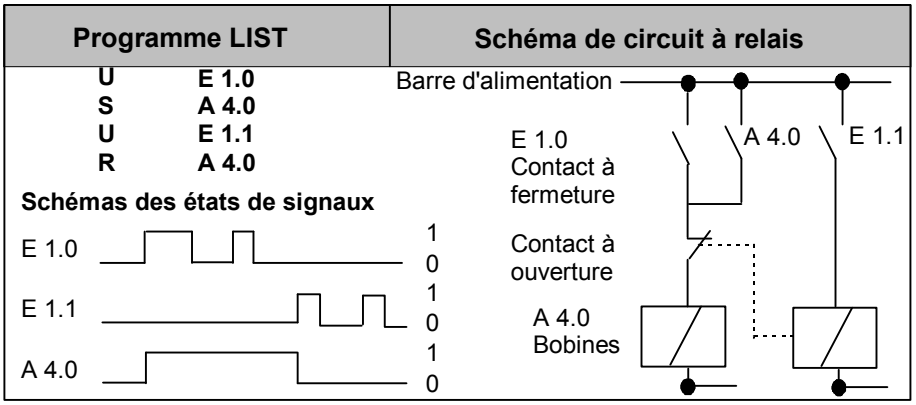
Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :		-							

Exemple

Programme LIST	Schéma de circuit à relais

```
U   E 1.0
S   A 4.0
U   E 1.1
R   A 4.0
```



1.19 NOT Négation du RLG

Format

NOT

Description de l'opération

NOT

Cette opération inverse le RLG.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	1	x	-

1.20 SET Mettre RLG à 1

Format

SET

Description de l'opération

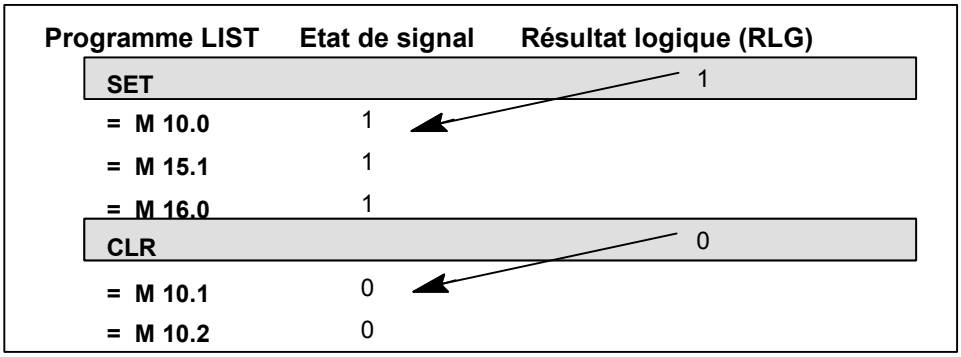
SET (Mettre RLG à 1)

Cette opération met le RLG à l'état de signal 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	1	0

Exemple


Programme LIST
Etat de signal
Résultat logique (RLG)
SET
1
= M 10.0
1
= M 15.1
1
= M 16.0
1
CLR
0
= M 10.1
0
= M 10.2
0

1.21 CLR Mettre RLG à 0

Format

CLR

Description de l'opération

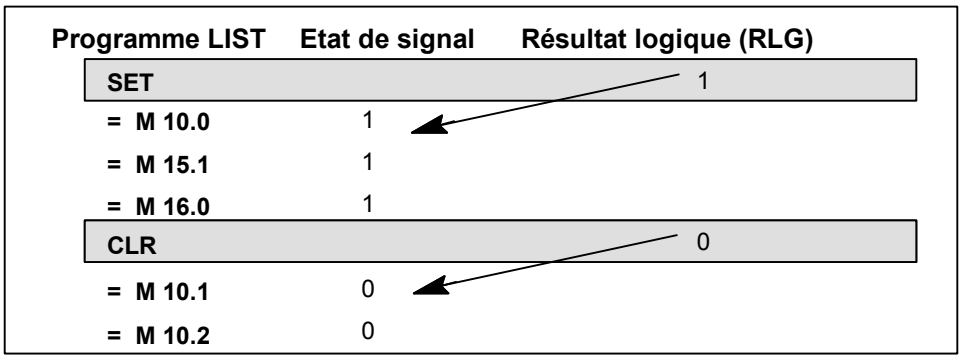
CLR (Mettre RLG à 0)

Cette opération met le RLG à l'état de signal 0.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	0	0	0

Exemple



1.22 SAVE Sauvegarder RLG dans le bit RB

Format

SAVE

Description de l'opération

SAVE (Sauvegarder RLG dans le bit RB)

Cette opération sauvegarde le RLG dans le bit RB, le bit de première interrogation /PI n'étant pas remis à 0.
Pour cette raison, une combinaison ET dans le réseau suivant prend en compte l'état du bit RB.

L'utilisation de l'opération **SAVE** suivie d'une interrogation du bit RB dans le même bloc ou dans un bloc subordonné n'est pas recommandée, le bit RB pouvant être modifié par un grand nombre d'opérations intermédiaires. Il est par contre judicieux d'utiliser l'opération **SAVE** avant de quitter un bloc, car ainsi la sortie ENO (= bit RB) prend la valeur du bit RLG, ce qui vous permet ensuite de poursuivre par un traitement des erreurs du bloc.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	x	-	-	-	-	-	-	-	-

1.23 FN Front descendant

Format

FN <bit>

Opérande	Type de données	Zone de mémoire	Description
<bit>	BOOL	E, A, M, L, D	Mémento de front, sauvegarde l'état de signal précédent du RLG

Description de l'opération

FN <bit> (Front descendant)

Cette opération permet de détecter un front descendant si le RLG passe de 1 à 0, et donne 1 comme résultat.

Au cours de chaque cycle de programme, l'état de signal du bit RLG est comparé à l'état de signal du bit RLG du cycle précédent pour déterminer toute modification de l'état. Pour que la comparaison s'exécute, l'état de signal du bit RLG précédent doit être sauvegardé dans l'adresse du mémento de front (**<bit>**). Si l'état de signal en cours diffère de l'état de signal précédent (1 ; détection d'un front descendant), le bit RLG égale 1 après cette opération.

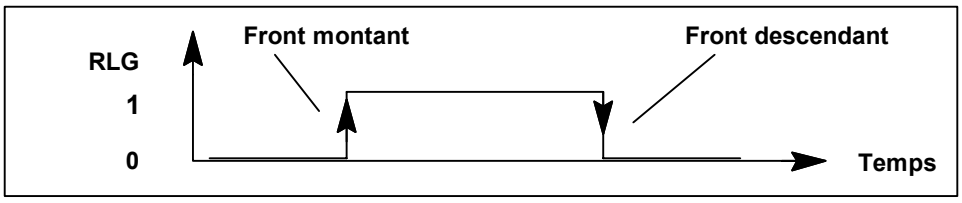
Nota

Lorsque le bit à contrôler se trouve dans la mémoire image, cette opération n'est pas significative. En effet, les données locales d'un bloc ne sont valides que pendant la durée d'exécution de ce bloc.

Mot d'état

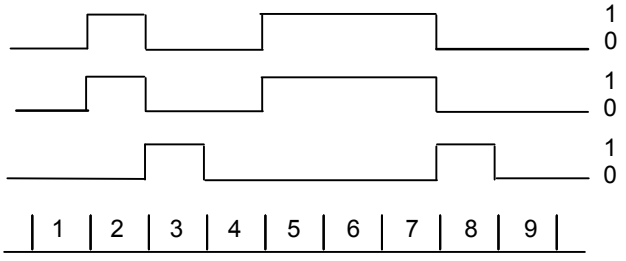
	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Définition



Exemple

Lorsque l'automate programmable détecte un front descendant au contact E 1.0, il active la sortie A 4.0 pour un cycle d'OB1.

Programme LIST		Schéma des états de signaux
U	**E 1.0**	
FN	**M 1.0**	
=	**A 4.0**	
Cycle OB1 n° :		

1.24 FP Front montant

Format

FP <bit>

Opérande	Type de données	Zone de mémoire	Description
<bit>	BOOL	E, A, M, L, D	Mémento de front, sauvegarde l'état de signal précédent du RLG

Description de l'opération

FP <bit> (Front montant)

Cette opération permet de détecter un front montant si le RLG passe de 0 à 1, et donne 1 comme résultat.

Au cours de chaque cycle de programme, l'état de signal du bit RLG est comparé à l'état de signal du bit RLG du cycle précédent pour déterminer toute modification de l'état. Pour que la comparaison s'exécute, l'état de signal du bit RLG précédent doit être sauvegardé dans l'adresse du mémento de front (**<bit>**). Si l'état de signal en cours diffère de l'état de signal précédent (0 ; détection d'un front montant), le bit RLG égale 1 après cette opération.

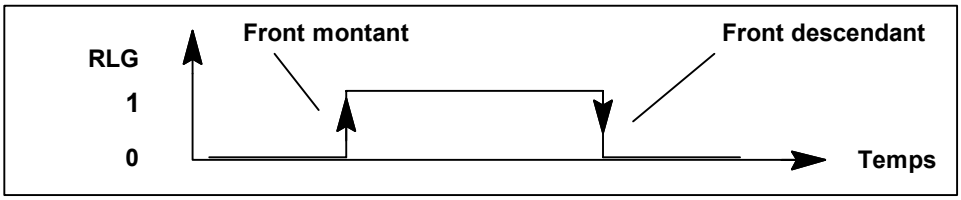
Nota

Lorsque le bit à contrôler se trouve dans la mémoire image, cette opération n'est pas significative. En effet, les données locales d'un bloc ne sont valides que pendant la durée d'exécution de ce bloc.

Mot d'état

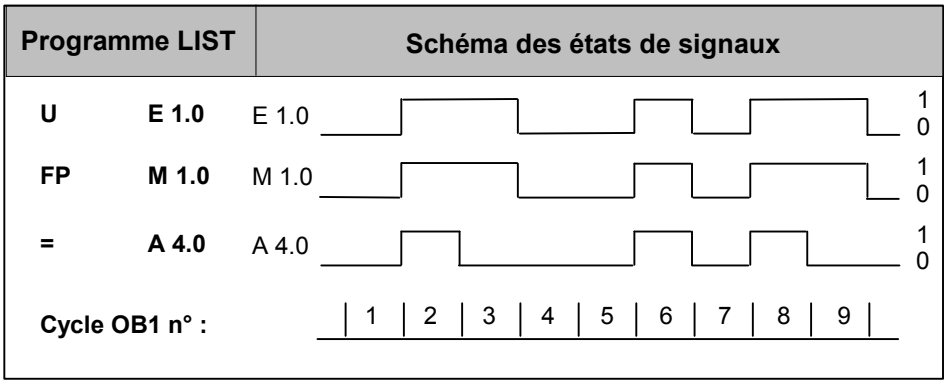
	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	x	x	1

Définition



Exemple

Lorsque l'automate programmable détecte un front montant au contact E 1.0, il active la sortie A 4.0 pour un cycle d'OB1.



2 Opérations de comparaison

2.1 Vue d'ensemble des opérations de comparaison

Description

Les opérations de comparaison comparent le contenu de l'accumulateur 2 à celui de l'accumulateur 1 selon les types de comparaison suivants :

== ACCU 2 est égal à ACCU 1
<> ACCU 2 est différent de ACCU 1
> ACCU 2 est supérieur à ACCU 1
< ACCU 2 est inférieur à ACCU 1
>= ACCU 2 est supérieur ou égal à ACCU 1
<= ACCU 2 est inférieur ou égal à ACCU 1

Si le RLG égale 1, le résultat de comparaison est vrai. Si le RLG égale 0, le résultat de comparaison est faux. Les bits BI1 et BI0 indiquent la relation "inférieur à", "égal à" ou "supérieur à".

Vous disposez des opérations de comparaison suivantes :

- ? I Comparer entiers de 16 bits
- ? D Comparer entiers de 32 bits
- ? R Comparer réels de 32 bits

2.2 ? I Comparer entiers de 16 bits

Format

==I, <>I, >I, <I, >=I, <=I

Description de l'opération

Les opérations de comparaison d'entiers de 16 bits comparent le contenu de l'accumulateur 2- L à celui de l'accumulateur 1- L. Les contenus de l'accumulateur 2-L et de l'accumulateur 1-L sont évalués comme nombres entiers de 16 bits. Le résultat de la comparaison est indiqué par le RLG et les bits significatifs du mot d'état. Si le RLG égale 1, le résultat de comparaison est vrai. Si le RLG égale 0, le résultat de comparaison est faux. Les bits BI1 et BI0 indiquent la relation "inférieur à", "égal à" ou "supérieur à".

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	0	-	0	x	x	1

Valeurs du RLG

Opération de comparaison exécutée	RLG si ACCU 2 > ACCU 1	RLG si ACCU 2 = ACCU 1	RLG si ACCU 2 < ACCU 1
==I	0	1	0
<>I	1	0	1
>I	1	0	0
<I	0	0	1
>=I	1	1	0
<=I	0	1	1

Exemple

```
LIST            Explication
L     MW10      //Charger le contenu de MW10 (entier de 16 bits).
L     EW24      //Charger le contenu de EW24 (entier de 16 bits).
>I              //Comparer si accumulateur 2-L (MW10) supérieur (>) à accumulateur 1-L (EW24).
=     M 2.0     //Le RLG égale 1 si MW10 > EW24.
```

2.3 ? D Comparer entiers de 32 bits

Format

==D, <>D, >D, <D, >=D, <=D

Description de l'opération

Les opérations de comparaison d'entiers de 32 bits comparent le contenu de l'accumulateur 2 à celui de l'accumulateur 1. Les contenus de l'accumulateur 2 et de l'accumulateur 1 sont évalués comme nombres entiers de 32 bits. Le résultat de la comparaison est indiqué par le RLG et les bits significatifs du mot d'état. Si le RLG égale 1, le résultat de comparaison est vrai. Si le RLG égale 0, le résultat de comparaison est faux. Les bits BI1 et BI0 indiquent la relation "inférieur à", "égal à" ou "supérieur à".

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	0	-	0	x	x	1

Valeurs du RLG

Opération de comparaison exécutée	RLG si ACCU 2 > ACCU 1	RLG si ACCU 2 = ACCU 1	RLG si ACCU 2 < ACCU 1
==D	0	1	0
<>D	1	0	1
>D	1	0	0
<D	0	0	1
>=D	1	1	0
<=D	0	1	1

Exemple

```
LIST            Explication
L       MD10    //Charger le contenu de MD10 (entier de 32 bits).
L       ED24    //Charger le contenu de ED24 (entier de 32 bits).
>D              //Comparer si accumulateur 2 (MD10) supérieur (>) à  accumulateur 1 (ED24).
=       M 2.0   //Le RLG égale 1 si MD10 > ED24.
```

2.4 ? R Comparer réels de 32 bits

Format

==R, <>R, >R, <R, >=R, <=R

Description de l'opération

Les opérations de comparaison de nombres à virgule flottante IEEE de 32 bits comparent le contenu de l'accumulateur 2 à celui de l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres à virgule flottante IEEE de 32 bits. Le résultat de la comparaison est indiqué par le RLG et les bits significatifs du mot d'état. Si le RLG égale 1, le résultat de comparaison est vrai. Si le RLG égale 0, le résultat de comparaison est faux. Les bits BI1 et BI0 indiquent la relation "inférieur à", "égal à" ou "supérieur à".

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture	-	x	x	x	x	0	x	x	1

Valeurs du RLG

Opération de comparaison exécutée	RLG si ACCU 2 > ACCU 1	RLG si ACCU 2 = ACCU 1	RLG si ACCU 2 < ACCU 1
==R	0	1	0
<>R	1	0	1
>R	1	0	0
<R	0	0	1
>=R	1	1	0
<=R	0	1	1

Exemple

```
LIST            Explication
L     MD10      //Charger le contenu de MD10 (nombre à virgule flottante).
L     1.359E+02 //Charger la constante 1.359E+02.
>R              //Comparer si l'accumulateur 2 (MD10) supérieur (>) à l'accumulateur 1
                //(1.359E+02).
=     M 2.0     //Le RLG égale 1 si MD10 > 1.359E+02.
```

3 Opérations de conversion

3.1 Vue d'ensemble des opérations de conversion

Description

Les opérations suivantes permettent de convertir des nombres décimaux codés binaires et des nombres entiers en d'autres types de nombres :

- BTI Convertir DCB en entier de 16 bits
- ITB Convertir entier de 16 bits en DCB
- BTD Convertir DCB en entier de 32 bits
- ITD Convertir entier de 16 bits en entier de 32 bits
- DTB Convertir entier de 32 bits en DCB
- DTR Convertir entier de 32 bits en réel (IEEE 754 32 bits)

Les opérations suivantes permettent de former le complément de nombres entiers ou de réaliser l'inversion de nombres à virgule flottante :

- INVI Complément à 1 d'entier de 16 bits
- INVD Complément à 1 d'entier de 32 bits
- NEGI Complément à 2 d'entier de 16 bits
- NEGD Complément à 2 d'entier de 32 bits
- NEGR Inverser nombre à virgule flottante (IEEE 754 32 bits)

Les opérations suivantes permettent de modifier l'ordre des octets dans le mot de poids faible de l'accumulateur 1 ou dans l'accumulateur 1 entier :

- TAW Modifier l'ordre dans l'accumulateur 1-L (16 bits)
- TAD Modifier l'ordre dans l'accumulateur 1 (32 bits)

Les opérations suivantes permettent de convertir le nombre à virgule flottante IEEE de 32 bits figurant dans l'accumulateur 1 en un nombre entier de 32 bits (entier double). Les différentes opérations se distinguent par leur façon d'arrondir :

- RND Arrondir à l'entier
- TRUNC Arrondir par troncature
- RND+ Arrondir à l'entier supérieur
- RND- Arrondir à l'entier inférieur

3.2 BTI Convertir DCB en entier de 16 bits

Format

BTI

Description de l'opération

BTI (Conversion en nombre entier d'un nombre décimal codé binaire à trois chiffres)

Cette opération évalue le contenu de l'accumulateur 1-L comme nombre décimal codé binaire (nombre DCB) à trois chiffres et le convertit en un nombre entier de 16 bits. Le résultat de la conversion est rangé dans l'accumulateur 1-L. L'accumulateur 1-H et l'accumulateur 2 restent inchangés.

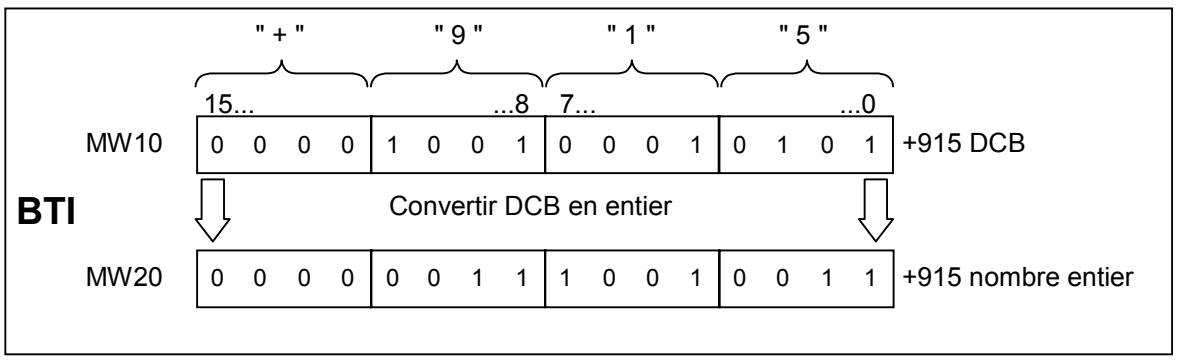
Nombre DCB contenu dans l'accumulateur 1-L : ce nombre DCB peut être compris entre -999 et +999. Les bits 0 à 11 indiquent la valeur et le bit 15 le signe (0 = positif, 1 = négatif) du nombre DCB. Les bits 12 à 14 ne sont pas utilisés pour la conversion. Si un chiffre décimal (une tétrade de 4 bits dans la représentation DCB) est compris dans la plage incorrecte de 10 à 15, une erreur BCDF est signalée lors de la tentative de conversion. En général, l'automate programmable passe alors à l'état de fonctionnement "Arrêt" (STOP). Cependant, l'OB121 vous permet de programmer une autre réaction à cette erreur synchrone.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
L     MW10      //Charger le nombre DCB dans l'accumulateur 1-L.
BTI             //Convertir le nombre DCB en un nombre entier et ranger le résultat dans
                //l'accumulateur 1-L.
T     MW20      //Transférer le résultat (nombre entier de 16 bits) dans le mot de mémento
                //MW20.
```



3.3 ITB Convertir entier de 16 bits en DCB

Format

ITB

Description de l'opération

ITB (Convertir entier de 16 bits en DCB)

Cette opération évalue le contenu de l'accumulateur 1-L comme nombre entier de 16 bits et le convertit en un nombre décimal codé binaire (nombre DCB) à trois chiffres. Le résultat est rangé dans l'accumulateur 1-L. Les bits 0 à 11 indiquent la valeur du nombre DCB. Les bits 12 à 15 contiennent l'état du signe du nombre DCB (0000 = positif, 1111 = négatif). L'accumulateur 1-H et l'accumulateur 2 restent inchangés.

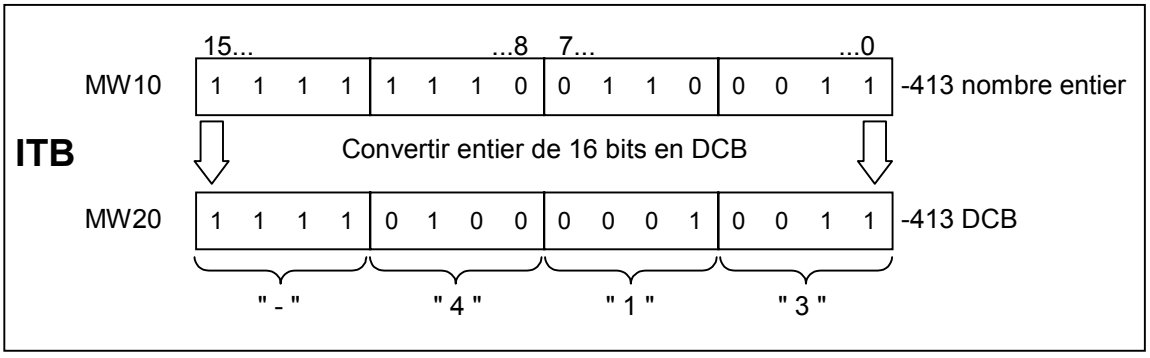
Le nombre DCB peut être compris entre -999 et +999. S'il se situe hors de la plage correcte, les bits d'état DEB et DM sont mis à 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST              Explication
L      MW10       //Charger l'entier dans l'accumulateur 1-L.
ITB               //Convertir l'entier de 16 bits en un nombre DCB et ranger le résultat dans
                  //l'accumulateur 1-L.
T      MW20       //Transférer le résultat (nombre DCB) dans le mot de mémento MW20.
```



3.4 BTD Convertir DCB en entier de 32 bits

Format

BTD

Description de l'opération

BTD (Conversion en nombre entier d'un nombre décimal codé binaire à sept chiffres)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre décimal codé binaire (nombre DCB) à sept chiffres et le convertit en un nombre entier de 32 bits. Le résultat de la conversion est rangé dans l'accumulateur 1. L'accumulateur 2 reste inchangé.

Nombre DCB contenu dans l'accumulateur 1 : ce nombre DCB peut être compris entre -9 999 999 et +9 999 999. Les bits 0 à 27 indiquent la valeur et le bit 31 le signe (0 = positif, 1 = négatif) du nombre DCB. Les bits 28 à 30 ne sont pas utilisés pour la conversion.

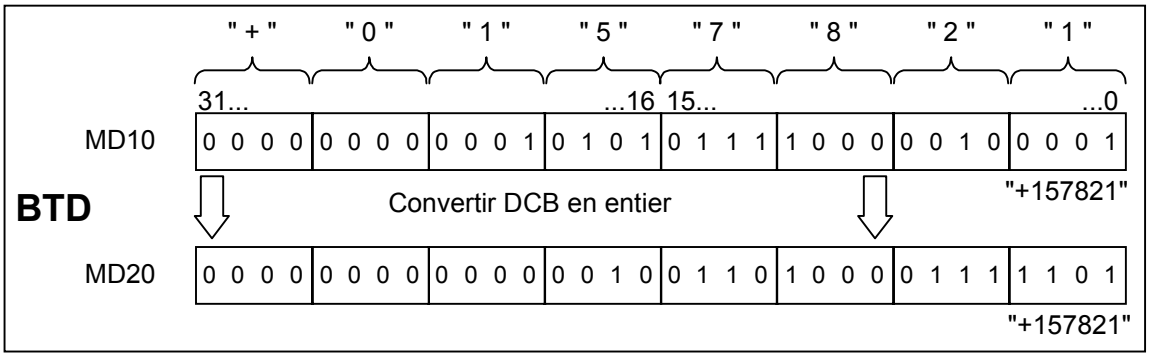
Si un chiffre décimal (une tétrade de 4 bits dans la représentation DCB) est compris dans la plage incorrecte de 10 à 15, une erreur BCDF est signalée lors de la tentative de conversion. En général, l'automate programmable passe alors à l'état de fonctionnement "Arrêt" (STOP). Cependant, l'OB121 vous permet de programmer une autre réaction à cette erreur synchrone.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
L     MD10      //Charger le nombre DCB dans l'accumulateur 1.
BTD             //Convertir le nombre DCB en un nombre entier et ranger le résultat dans
                //l'accumulateur 1.
T     MD20      //Transférer le résultat (nombre entier de 32 bits) dans le double mot de
                //mémento MD20.
```



3.5 ITD Convertir entier de 16 bits en entier de 32 bits

Format

ITD

Description de l'opération

ITD (Convertir entier de 16 bits en entier de 32 bits)

Cette opération évalue le contenu de l'accumulateur 1-L comme nombre entier de 16 bits et le convertit en un nombre entier de 32 bits. Le résultat est rangé dans l'accumulateur 1 ; l'accumulateur 2 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MW12       //Charger l'entier de 16 bits dans l'accumulateur 1.
ITD               //Convertir l'entier de 16 bits en un entier de 32 bits et ranger le résultat
                  //dans l'accumulateur 1.
T      MD20       //Transférer le résultat (nombre entier de 32 bits) dans le double mot de
                  //mémento MD20.
```

Exemple : MW12 = "-10" (nombre entier de 16 bits)

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **ITD**	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
après exécution de **ITD**	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 ou 1, bits non requis pour la conversion)							

3.6 DTB Convertir entier de 32 bits en DCB

Format

DTB

Description de l'opération

DTB (Conversion d'un nombre entier de 32 bits en nombre DCB)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre entier de 32 bits et le convertit en un nombre décimal codé binaire (nombre DCB) à sept chiffres. Le résultat est rangé dans l'accumulateur 1. Les bits 0 à 27 indiquent la valeur du nombre DCB. Les bits 28 à 31 représentent l'état du signe du nombre DCB (0000 = positif, 1111 = négatif). L'accumulateur 2 reste inchangé.

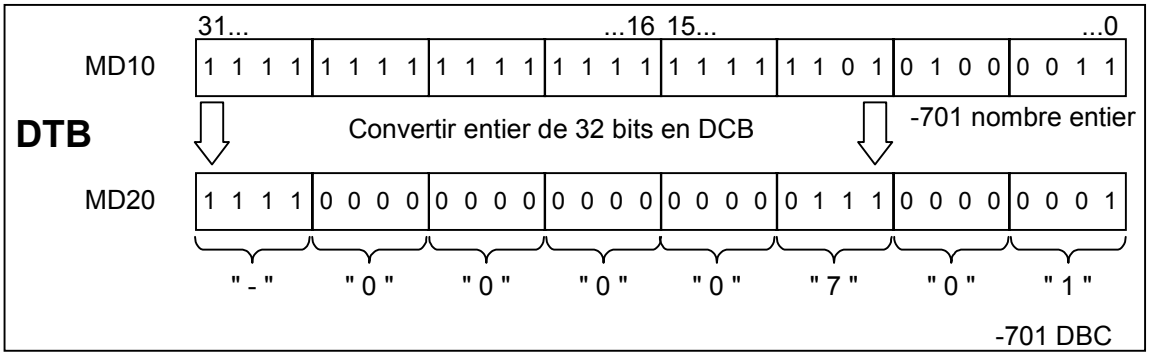
Le nombre DCB peut être compris entre -9 999 999 et +9 999 999. S'il se situe hors de la plage correcte, les bits d'état DEB et DM sont mis à 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST              Explication
L      MD10       //Charger l'entier de 32 bits dans l'accumulateur 1.
DTB               //Convertir l'entier de 32 bits en un nombre DCB et ranger le résultat dans
                  //l'accumulateur 1.
T      MD20       //Transférer le résultat (nombre DCB) dans le double mot de mémento MD20.
```



3.7 DTR Convertir entier de 32 bits en réel (IEEE 754 32 bits)

Format

DTR

Description de l'opération

DTR (Conversion d'un nombre entier de 32 bits en nombre à virgule flottante IEEE de 32 bits)

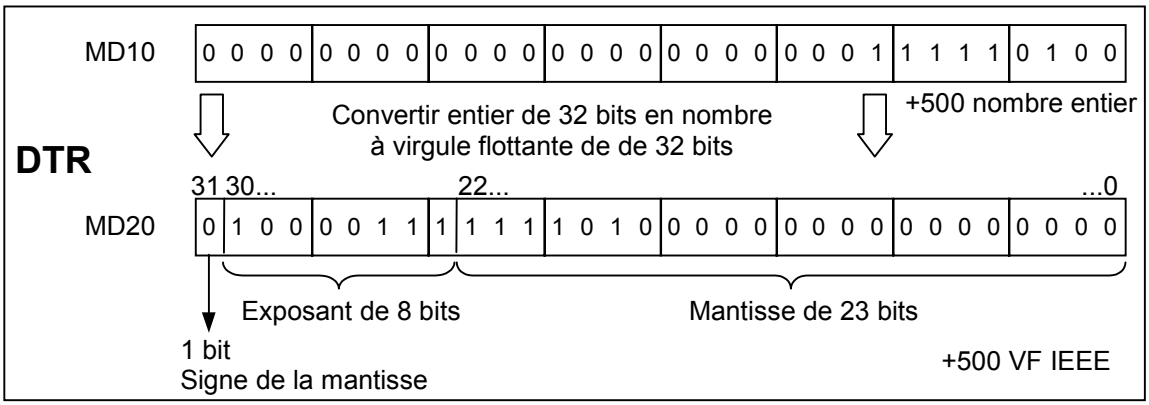
Cette opération évalue le contenu de l'accumulateur 1 comme nombre entier de 32 bits et le convertit en nombre à virgule flottante IEEE de 32 bits. Si nécessaire, l'opération arrondit le résultat (un entier de 32 bits a une plus grande précision qu'un nombre à virgule flottante IEEE de 32 bits). Le résultat est rangé dans l'accumulateur 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MD10       //Charger l'entier de 32 bits dans l'accumulateur  1.
DTR               //Convertir l'entier de 32 bits en un nombre à virgule flottante IEEE de
                  //32 bits et ranger le résultat dans l'accumulateur  1.
T      MD20       //Transférer le résultat (nombre DCB) dans le double mot de mémento MD20.
```



3.8 INVI Complément à 1 d'entier de 16 bits

Format

INVI

Description de l'opération

INVI (Complément à 1 d'entier de 16 bits)

Cette opération forme le complément à 1 de la valeur de 16 bits figurant dans l'accumulateur 1- L. Lors de la formation du complément à 1, les bits individuels sont inversés, c'est-à-dire que les zéros sont remplacés par des uns et les uns par des zéros. Le résultat est rangé dans l'accumulateur 1-L.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      EW8        //Charger la valeur dans l'accumulateur 1-L.
INVI              //Former le complément à 1 (16  bits).
T      MW10       //Transférer le résultat dans le mot de mémento MW10.
```

Contenu	ACCU1-L			
Bit	15 . . .	. .	. .	. . . 0
avant exécution de **INVI**	0110	0011	1010	1110
après exécution de **INVI**	1001	1100	0101	0001

3.9 INVD Complément à 1 d'entier de 32 bits

Format

INVD

Description de l'opération

INVD (Complément à 1 d'entier de 32 bits)

Cette opération forme le complément à 1 de la valeur de 32 bits figurant dans l'accumulateur 1. Lors de la formation du complément à 1, les bits individuels sont inversés, c'est-à-dire que les zéros sont remplacés par des uns et les uns par des zéros. Le résultat est rangé dans l'accumulateur 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      ED8        //Charger la valeur dans l'accumulateur 1.
INVD              //Former le complément à 1 (32 bits).
T      MD10       //Transférer le résultat dans le double mot de mémento MD10.
```

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **INVD**	0110	1111	1000	1100	0110	0011	1010	1110
après exécution de **INVD**	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI Complément à 2 d'entier de 16 bits

Format

NEGI

Description de l'opération

NEGI (Complément à 2 d'entier de 16 bits)

Cette opération forme le complément à 2 de la valeur de 16 bits figurant dans l'accumulateur 1-L. Lors de la formation du complément à 2, les bits individuels sont inversés, c'est-à-dire que les zéros sont remplacés par des uns et les uns par des zéros, puis on ajoute la valeur 1. Le résultat est rangé dans l'accumulateur 1-L. L'opération "Complément à 2 d'entier" est équivalente à une multiplication par -1. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état	BI1	BI0	DEB	DM
résultat = 0	0	0	0	-
-32768 <= résultat <= -1	0	1	0	-
32767 >= résultat >= 1	1	0	0	-
résultat = 2768	0	1	1	1

Exemple

```
LIST              Explication
L      EW8        //Charger la valeur dans l'accumulateur 1-L.
NEGI              //Former le complément à 2 (16 bits).
T      MW10       //Transférer le résultat dans le mot de mémento MW10.
```

Contenu	ACCU1-L			
Bit	15 . . .	. .	. .	. . . 0
avant exécution de **NEGI**	0101	1101	0011	1000
après exécution de **NEGI**	1010	0010	1100	1000

3.11 NEGD Complément à 2 d'entier de 32 bits

Format

NEGD

Description de l'opération

NEGD (Complément à 2 d'entier de 32 bits)

Cette opération forme le complément à 2 de la valeur de 32 bits figurant dans l'accumulateur 1. Lors de la formation du complément à 2, les bits individuels sont inversés, c'est-à-dire que les zéros sont remplacés par des uns et les uns par des zéros, puis on ajoute la valeur 1. Le résultat est rangé dans l'accumulateur 1. L'opération "Complément à 2 d'entier" est équivalente à une multiplication par -1. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état	BI1	BI0	DEB	DM
résultat = 0	0	0	0	-
-2 147 483 647 <= résultat <= -1	0	1	0	-
2 147 483 647 >= résultat >= 1	1	0	0	-
résultat = -2 147 483 648	0	1	1	1

Exemple

```
LIST              Explication
L      ED8        //Charger la valeur dans l'accumulateur 1.
NEGD              //Former le complément à 2 (32  bits).
T      MD10       //Transférer le résultat dans le double mot de mémento MD10.
```

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **NEGD**	0101	1111	0110	0100	0101	1101	0011	1000
après exécution de **NEGD**	1010	0000	1001	1011	1010	0010	1100	1000

3.12 NEGR Inverser nombre à virgule flottante (IEEE 754 32 bits)

Format

NEGR

Description de l'opération

NEGR (Inverser nombre à virgule flottante)

Cette opération inverse le nombre à virgule flottante IEEE de 32 bits figurant dans l'accumulateur 1. Elle inverse l'état du bit 31 contenu dans l'accumulateur 1 (signe de la mantisse). Le résultat est rangé dans l'accumulateur 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      ED8        //Charger la valeur dans l'accumulateur 1 (exemple : ED8 = 1.5E+02).
NEGR              //Inverser le nombre à virgule flottante IEEE de 32 bits et ranger le résultat
                  //dans l'accumulateur 1.
T      MD10       //Transférer le résultat dans le double mot de mémento MD10
                  //(exemple : résultat = −1.5E+02).
```

3.13 TAW Modifier l'ordre dans l'accumulateur 1-L (16 bits)

Format

TAW

Description de l'opération

TAW

Cette opération permet d'inverser l'ordre des octets dans l'accumulateur 1-L. Le résultat est rangé dans l'accumulateur 1-L. L'accumulateur 1-H et l'accumulateur 2 restent inchangés.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MW10       //Charger la valeur du mot de mémento MW10 dans l'accumulateur  1.
TAW               //Inverser l'ordre des octets dans l'accumulateur 1-L.
T      MW20       //Transférer le résultat dans le mot de mémento MW20.
```

Contenu	ACCU 1-H-H	ACCU 1-H-L	ACCU 1-L-H	ACCU 1-L-L
avant exécution de **TAW**	valeur A	valeur B	valeur C	valeur D
après exécution de **TAW**	valeur A	valeur B	valeur D	valeur C

3.14 TAD Modifier l'ordre dans l'accumulateur 1 (32 bits)

Format

TAD

Description de l'opération

TAD

Cette opération permet d'inverser l'ordre des octets dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1 ; l'accumulateur 2 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
L     MD10      //Charger la valeur du double mot de mémento MD10 dans l'accumulateur  1.
TAD             //Inverser l'ordre des octets dans l'accumulateur 1.
T     MD20      //Transférer le résultat dans le double mot de mémento MD20.
```

Contenu	ACCU 1-H-H	ACCU 1-H-L	ACCU 1-L-H	ACCU 1-L-L
avant exécution de **TAD**	valeur A	valeur B	valeur C	valeur D
après exécution de **TAD**	valeur D	valeur C	valeur B	valeur A

3.15 RND Arrondir à l'entier

Format

RND

Description de l'opération

RND (Conversion d'un nombre à virgule flottante IEEE de 32 bits en entier de 32 bits)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre à virgule flottante IEEE de 32 bits, le convertit en un nombre entier de 32 bits et arrondit le résultat au nombre entier le plus proche. Si la partie fractionnaire du nombre converti se situe exactement entre un résultat pair et un résultat impair, l'opération choisit le résultat pair. Si le nombre est hors de la plage correcte, les bits d'état DEB et DM sont mis à 1.

Si le nombre n'est pas un nombre à virgule flottante ou est un nombre à virgule flottante qui ne peut pas être représenté comme entier de 32 bits, la conversion n'a pas lieu et un débordement est signalé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST              Explication
L      MD10       //Charger dans l'accumulateur 1 le nombre à virgule flottante IEEE de 32 bits.
RND               //Convertir le nombre à virgule flottante IEEE de 32 bits en un nombre entier
                  //de 32 bits et arrondir au nombre entier le plus proche. Ranger le résultat
                  //dans l'accumulateur 1.
T      MD20       //Transférer le résultat (entier de 32 bits) dans le double mot de mémento
                  //MD20.
```

Valeur avant la conversion		Valeur après la conversion
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

3.16 TRUNC Arrondir par troncature

Format

TRUNC

Description de l'opération

TRUNC (Conversion d'un nombre à virgule flottante IEEE de 32 bits en nombre entier de 32 bits)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre à virgule flottante IEEE de 32 bits et le convertit en un nombre entier de 32 bits. Le résultat correspond à la partie entière du nombre à virgule flottante converti (mode d'arrondi IEEE "arrondi à zéro"). Si le nombre est hors de la plage correcte, les bits d'état DEB et DM sont mis à 1. Le résultat est rangé dans l'accumulateur 1.

Si le nombre n'est pas un nombre à virgule flottante ou est un nombre à virgule flottante qui ne peut pas être représenté comme entier de 32 bits, la conversion n'a pas lieu et un débordement est signalé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST              Explication
L      MD10       //Charger dans l'accumulateur 1 le nombre à virgule flottante IEEE de 32 bits.
TRUNC             //Convertir le nombre à virgule flottante de 32 bits en un nombre entier de
                  //32 bits et conserver la partie entière. Ranger le résultat dans
                  //l'accumulateur 1.
T      MD20       //Transférer le résultat (entier de 32 bits) dans le double mot de mémento
                  //MD20.
```

Valeur avant la conversion		Valeur après la conversion
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

3.17 RND+ Arrondir à l'entier supérieur

Format

RND+

Description de l'opération

RND+ (Conversion d'un nombre à virgule flottante IEEE de 32 bits en entier de 32 bits)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre à virgule flottante IEEE de 32 bits, le convertit en un nombre entier de 32 bits et arrondit le résultat au plus petit nombre entier supérieur ou égal au nombre à virgule flottante converti (mode d'arrondi IEEE "arrondi à +infini"). Si le nombre est hors de la plage correcte, les bits d'état DEB et DM sont mis à 1. Le résultat est rangé dans l'accumulateur 1.

Si le nombre n'est pas un nombre à virgule flottante ou est un nombre à virgule flottante qui ne peut pas être représenté comme entier de 32 bits, la conversion n'a pas lieu et un débordement est signalé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST            Explication
L     MD10      //Charger dans l'accumulateur 1 le nombre à virgule flottante IEEE de 32 bits.
RND+            //Convertir le nombre à virgule flottante IEEE de 32 bits en un nombre entier
                //de 32 bits et arrondir au plus petit nombre entier supérieur.
                //Ranger le résultat dans l'accumulateur 1.
T     MD20      //Transférer le résultat (entier de 32 bits) dans le double mot de mémento
                //MD20.
```

Valeur avant la conversion		Valeur après la conversion
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND- Arrondir à l'entier inférieur

Format

RND-

Description de l'opération

RND- (Conversion d'un nombre à virgule flottante IEEE de 32 bits en entier de 32 bits)

Cette opération évalue le contenu de l'accumulateur 1 comme nombre à virgule flottante IEEE de 32 bits, le convertit en un nombre entier de 32 bits et arrondit le résultat au plus grand nombre entier inférieur ou égal au nombre à virgule flottante converti (mode d'arrondi IEEE "arrondi à -infini"). Si le nombre est hors de la plage correcte, les bits d'état DEB et DM sont mis à 1. Le résultat est rangé dans l'accumulateur 1.

Si le nombre n'est pas un nombre à virgule flottante ou est un nombre à virgule flottante qui ne peut pas être représenté comme entier de 32 bits, la conversion n'a pas lieu et un débordement est signalé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	x	x	-	-	-	-

Exemple

```
LIST              Explication
L      MD10       //Charger dans l'accumulateur 1 le nombre à virgule flottante IEEE de 32 bits.
RND-              //Convertir le nombre à virgule flottante IEEE de 32 bits en un nombre entier
                  //de 32 bits et arrondir au plus grand nombre entier inférieur. Ranger le
                  //résultat dans l'accumulateur 1.
T      MD20       //Transférer le résultat (entier de 32 bits) dans le double mot de mémento
                  //MD20.
```

Valeur avant la conversion		Valeur après la conversion
MD10 = "100.5"	=> RND- =>	MD20 = "+100"
MD10 = "-100.5"	=> RND- =>	MD20 = "-101"

4 Opérations de comptage

4.1 Vue d'ensemble des opérations de comptage

Description

Un compteur est un élément fonctionnel du logiciel de programmation STEP 7. Une zone est réservée aux compteurs dans la mémoire de votre CPU. Cette zone de mémoire réserve un mot de 16 bits à chaque compteur. La programmation avec LIST prend en charge 256 compteurs. Le nombre de compteurs disponibles dans votre CPU figure dans les caractéristiques techniques. Les opérations de comptage sont les seules fonctions qui ont accès à la zone de mémoire réservée aux compteurs.

Vous disposez des opérations de comptage suivantes :

- FR Valider compteur
- L Charger valeur de comptage en cours comme entier dans l'accumulateur 1
- LC Charger valeur de comptage en cours comme nombre DCB dans l'accumulateur 1
- R Remettre compteur à zéro
- S Initialiser compteur
- ZV Incrémenter
- ZR Décrémenter

4.2 FR Valider compteur

Format

FR <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

FR <compteur> efface le mémento de front qui active l'incrémentation ou la décrémentation pour le compteur en accès si le résultat logique RLG passe de 0 à 1. La validation du compteur n'est pas requise pour initialiser un compteur ou exécuter la fonction de comptage normale. Cela signifie que, malgré un RLG constant de 1 aux instructions Initialiser compteur, Incrémenter ou Décrémenter, ces opérations sont à nouveau exécutées après la validation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
U     E 2.0     //Interroger l'état de signal à l'entrée E 2.0.
FR    Z3        //Valider le compteur Z3 si le RLG passe de 0 à 1.
```

4.3 L Charger valeur de comptage en cours comme entier dans l'accumulateur 1

Format

L <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

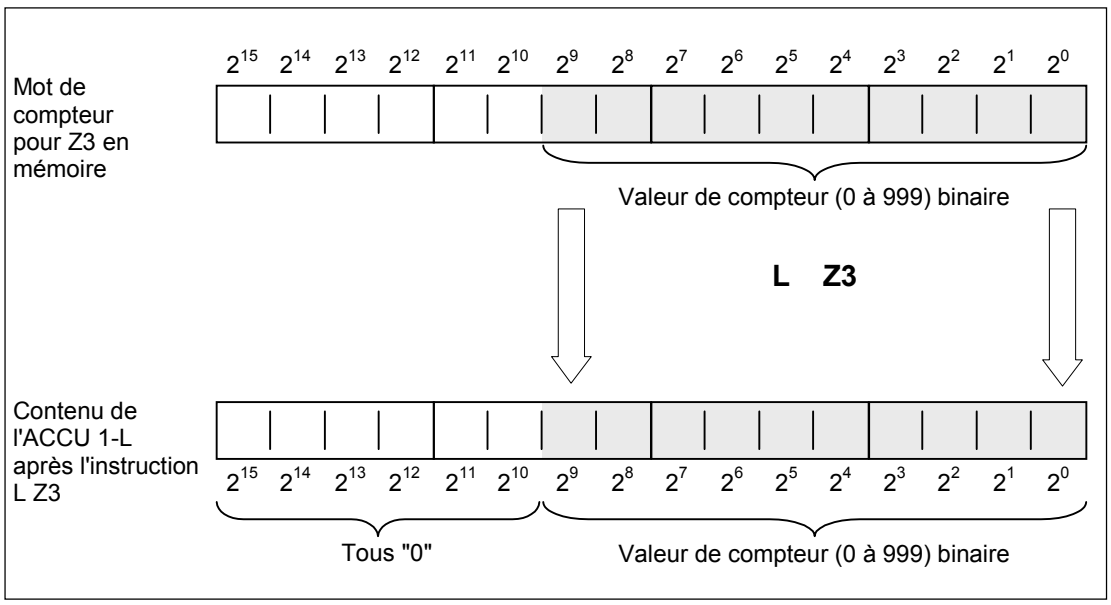
L <compteur> charge la valeur de comptage en cours du compteur en accès comme nombre entier dans l'accumulateur 1-L après sauvegarde du contenu de l'accumulateur 1 dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      Z3         //Charger la valeur de comptage du compteur Z3 en format binaire dans
                  //l'accumulateur 1-L.
```



4.4 LC Charger valeur de comptage en cours comme nombre DCB dans l'accumulateur 1

Format

LC <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

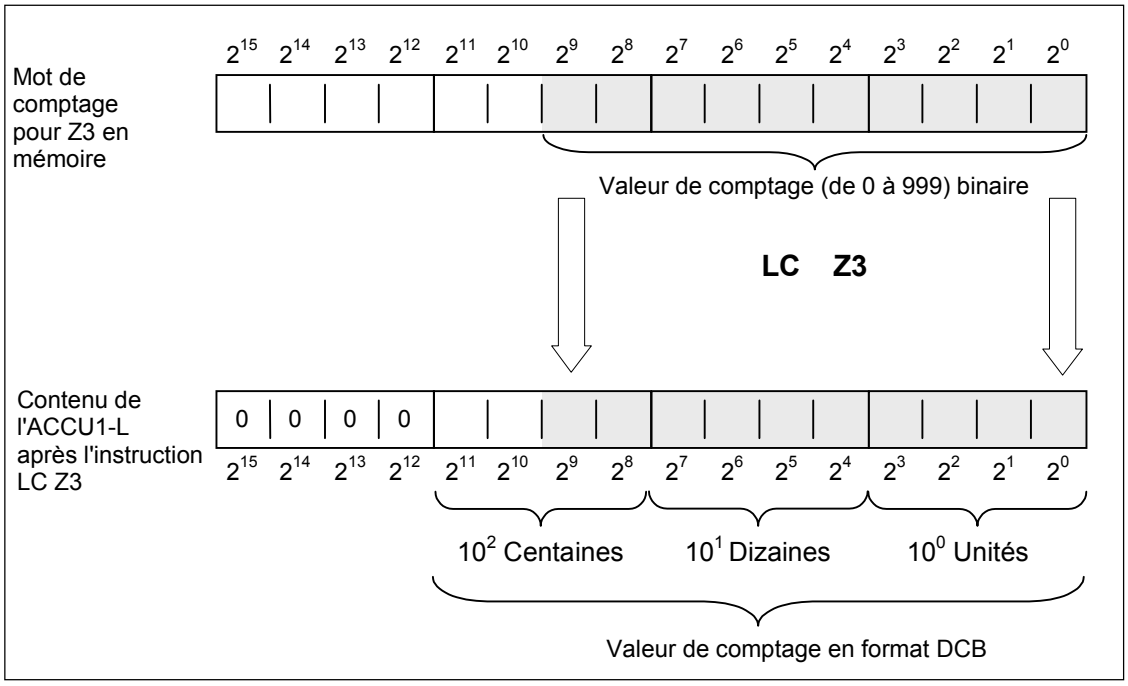
LC <compteur> charge la valeur de comptage en cours du compteur en accès comme nombre DCB dans l'accumulateur 1 après sauvegarde du contenu de l'accumulateur 1 dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
LC      Z3          //Charger la valeur de comptage du compteur Z3 en format DCB dans
                    //l'accumulateur 1-L.
```



4.5 R Remettre compteur à zéro

Format

R <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

R <compteur> charge la valeur de comptage 0 dans le compteur en accès si le RLG égale 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
U     E 2.3     //Interroger l'état de signal à l'entrée E 2.3.
R     Z3        //Remettre le compteur Z3 à 0 si le RLG passe de 0 à  1.
```

4.6 S Initialiser compteur

Format

S <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

S <compteur> charge la valeur de comptage figurant dans l'accumulateur 1-L dans le compteur concerné si le RLG passe de 0 à 1. La valeur de comptage dans l'accumulateur 1 doit être un nombre DCB compris entre 0 et 999.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST              Explication
U      E 2.3      //Interroger l'état de signal à l'entrée E 2.3.
L      C#3        //Charger la valeur de comptage 3 dans l'accumulateur 1-L.
S      Z1         //Initialiser le compteur Z1 à la valeur de comptage si le RLG passe de
                  //0 à 1.
```

4.7 ZV Incrémenter

Format

ZV <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

ZV <compteur> incrémente d'1 la valeur de comptage du compteur en accès si le résultat logique RLG passe de 0 à 1 et si la valeur de comptage est inférieure à 999. Si la valeur de comptage atteint sa limite supérieure de 999, l'incrémentation s'arrête. Une modification suivante du RLG n'a aucun effet. Le bit de débordement (DEB) n'est pas mis à 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST              Explication
U     E 2.1       //Interroger l'état de signal à l'entrée E 2.1.
ZV    Z3          //Incrémenter d'1 le compteur Z3 si le RLG passe de 0 à 1.
```

4.8 ZR Décrémenter

Format

ZR <compteur>

Paramètre	Type de données	Zone de mémoire	Description
<compteur>	COUNTER	Z	Compteur ; la plage dépend de la CPU.

Description de l'opération

ZR <compteur> décrémente d'1 la valeur de comptage du compteur en accès si le résultat logique RLG passe de 0 à 1 et si la valeur de comptage est supérieure à 0. Si le compteur atteint sa limite inférieure de 0, la décrémentation s'arrête. Une modification suivante du RLG n'a aucun effet, car le compteur n'opère pas avec des valeurs négatives.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
L     C#14      //Valeur prédéfinie du compteur
U     E 0.1     //Compteur initialisé après détection du front montant à l'entrée E 0.1.
S     Z1        //Charger la valeur prédéfinie dans le compteur Z1 si celui-ci est validé.
U     E 0.0     //Enlever 1 à chaque front montant à l'entrée E 0.0.
ZR    Z1        //Décrémenter d'1 le compteur Z1 si le RLG passe de 0 à 1 en fonction de l'entrée
                //E 0.0.
UN    Z1        //Détection de 0 avec le bit Z1
=     A 0.0     //Si la valeur du compteur Z1 égale 0, la sortie A 0.0 donne 1.
```

5 Opérations sur blocs de données

5.1 Vue d'ensemble des opérations sur blocs de données

Description

L'opération **AUF** (ouvrir bloc de données) permet d'ouvrir un bloc de données global ou un bloc de données d'instance. Un bloc de données global et un bloc de données d'instance peuvent être ouverts simultanément dans le programme.

Vous disposez des opérations sur bloc de données suivantes :

- AUF Ouvrir bloc de données
- TDB Permuter DB global et DB d'instance
- L DBLG Charger longueur de DB global dans l'accumulateur 1
- L DBNO Charger numéro de DB global dans l'accumulateur 1
- L DILG Charger longueur de DB d'instance dans l'accumulateur 1
- L DINO Charger numéro de DB d'instance dans l'accumulateur 1

5.2 AUF Ouvrir bloc de données

Format

AUF <bloc de données>

Opérande	Type de blocs de données	Adresse source
<bloc de données >	DB, DI	1 à 65535

Description de l'opération

AUF <bloc de données>

Cette opération ouvre un bloc de données comme bloc de données global ou comme bloc de données d'instance. C'est à chaque fois un bloc de données global et un bloc de données d'instance qui peuvent être ouverts simultanément.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
AUF     DB10    //Ouvrir le bloc de données DB10 comme bloc de données global.
L       DBW35   //Charger dans l'accumulateur 1-L le mot de données DBW35 du bloc de données
                //ouvert
T       MW22    //Transférer le contenu de l'accumulateur 1-L dans le mot de mémentos MW22.
AUF     DI20    //Ouvrir le bloc de données DB20 comme bloc de données d'instance.
L       DIB12   //Charger dans l'accumulateur 1-L-L. l'octet de données DIB12 du bloc de
                //données d'instance ouvert
T       DBB37   //Transférer le contenu de l'accumulateur 1-L-L dans l'octet de données DBB37
                //du bloc de données global ouvert.
```

5.3 TDB Permuter DB global et DB d'instance

Format

TDB

Description de l'opération

TDB permute les registres de bloc de données. Un bloc de données global devient ainsi bloc de données d'instance et vice versa.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

5.4 L DBLG Charger longueur de DB global dans l'accumulateur 1

Format

L DBLG

Description de l'opération

L DBLG (Charger longueur de DB global)

Cette opération charge la longueur du bloc de données global dans l'accumulateur 1 une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
AUF     DB10      //Ouvrir le bloc de données DB10 comme bloc de données global.
L       DBLG      //Charger la longueur du bloc de données global (longueur de DB10).
L       MD10      //Valeur de comparaison pour déterminer si le bloc de données est suffisamment
                  //long.
<D
SPB     ERRO      //Saut au repère de saut ERRO si la longueur est inférieure à la valeur figurant
                  //dans le mot de mémento MD10.
```

5.5 L DBNO Charger numéro de DB global dans l'accumulateur 1

Format

L DBNO

Description de l'opération

L DBNO (Charger le numéro de DB global)

Cette opération charge dans l'accumulateur 1 le numéro du bloc de données global ouvert une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

5.6 L DILG Charger longueur de DB d'instance dans l'accumulateur 1

Format

L DILG

Description de l'opération

L DILG (Charger longueur de DB d'instance)

Cette opération charge dans l'accumulateur 1 la longueur du bloc de données d'instance une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
AUF      DI20     //Ouvrir le bloc de données DB20 comme bloc de données d'instance.
L        DILG     //Charger la longueur du bloc de données d'instance (longueur de DB20).
L        MW10     //Valeur de comparaison pour déterminer si le bloc de données est suffisamment
                  //long.
<I
SPB      ERRO     //Saut au repère de saut ERRO si la longueur est inférieure à la valeur figurant
                  //dans le mot de mémento MW10.
```

5.7 L DINO Charger numéro de DB d'instance dans l'accumulateur 1

Format

L DINO

Description de l'opération

L DINO (Charger numéro de DB d'instance)

Cette opération charge dans l'accumulateur 1 le numéro du bloc de données d'instance ouvert une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

6 Opérations de saut

6.1 Vue d'ensemble des opérations de saut

Description

Les opérations de saut et de boucle suivantes permettent de gérer le déroulement de votre programme. Elles interrompent le déroulement linéaire de votre programme afin de reprendre son exécution à un endroit différent. L'opérande d'une opération de saut ou de boucle est un repère de saut.

Nota

Dans les programmes pour les CPU S7–300, veiller à ce que la destination du saut soit toujours le **début** d'une séquence d'instructions combinatoires (pas obligatoire pour 318-2). La destination du saut ne doit pas se trouver à l'intérieur de la séquence d'instructions combinatoires.

Les opérations de saut suivantes permettent d'interrompre la séquence normale de votre programme de manière inconditionnelle :

- SPA Saut inconditionnel
- SPL Saut vers liste

Les opérations de saut suivantes interrompent la séquence normale dans votre programme selon le résultat logique RLG généré par l'instruction précédente :

- SPB Saut si RLG est 1
- SPBN Saut si RLG est 0
- SPBB Saut si RLG est 1 avec RB
- SPBNB Saut si RLG est 0 avec RB

Les opérations de saut suivantes interrompent la séquence normale dans votre programme selon l'état de signal d'un bit du mot d'état :

- SPBI Saut si RB est 1
- SPBIN Saut si RB est 0
- SPO Saut si DEB est 1
- SPS Saut si DM est 1

Les opérations de saut suivantes interrompent la séquence normale dans votre programme selon le résultat d'un calcul :

- SPZ Saut si égal à 0
- SPN Saut si différent de 0
- SPP Saut si plus
- SPM Saut si moins
- SPPZ Saut si supérieur ou égal à 0
- SPMZ Saut si inférieur ou égal à 0
- SPU Saut si illicite

6.2 SPA Saut inconditionnel

Format

SPA <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

SPA <repère de saut>

Cette opération interrompt la séquence logique normale de votre programme et provoque – quel que soit le contenu du mot d'état – le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
      U     E 1.0
      U     E 1.2
      SPB   EFFA   //Saut au repère de saut EFFA si le RLG égale 1.
      L     MB10
      INC   1
      T     MB10
      SPA   AVAN   //Saut inconditionnel au repère de saut AVAN.
EFFA: L     0
      T     MB10
AVAN: U     E 2.1  //La séquence de programme se poursuit ici après le saut au repère AVAN.
```

6.3 SPL Saut vers liste

Format

SPL <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

SPL <repère de saut>

L'opération "Saut vers liste" permet de programmer une série de sauts. La liste de destinations qui contient 255 entrées au maximum suit directement l'opération SPL et se termine avant le repère de saut précisé comme opérande de SPL. Chaque destination de saut correspond à une opération SPA. Le nombre des destinations de saut (0 à 255) figure dans l'accumulateur 1-L-L.

Tant que le contenu de l'ACCU est inférieur au nombre de destinations de saut entre l'instruction SPL et le repère de saut, l'opération SPL saute vers l'une des opérations SPA. Si l'accumulateur 1-L-L égale 0, le saut a lieu vers la première opération SPA ; si l'accumulateur 1-L-L égale 1, le saut s'exécute vers la deuxième opération SPA, etc. Si le nombre des destinations de saut est trop grand, l'opération SPL saute à la première instruction suivant la dernière opération SPA dans la liste de destinations.

La liste de destinations de saut doit être composée d'opérations SPA se situant avant le repère de saut précisé en opérande de l'instruction SPL. D'autres opérations dans la liste ne sont pas autorisées.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                    Explication
       L      MB0      //Charger le numéro de la destination de saut dans l'accumulateur 1-L-L.
       SPL    LSTX     //Destination de saut si l'accumulateur 1-L-L est supérieur à 3.
       SPA    SEG0     //Destination de saut si l'accumulateur 1-L-L égale  0.
       SPA    SEG1     //Destination de saut si l'accumulateur 1-L-L égale  1.
       SPA    COMM     //Destination de saut si l'accumulateur 1-L-L égale  2.
       SPA    SEG3     //Destination de saut si l'accumulateur 1-L-L égale  3.
LSTX:  SPA    COMM
SEG0:  *              //Instruction autorisée.
       *
       SPA    COMM
SEG1:  *              //Instruction autorisée.
       *
       SPA    COMM
SEG3:  *              //Instruction autorisée.
       *
       SPA    COMM
COMM:  *
       *
```

6.4 SPB Saut si RLG est 1

Format

SPB <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état RLG égale 1, l'opération **SPB <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Si RLG égale 0, le saut ne s'exécute pas. Le RLG est mis à 1 et la séquence normale de votre programme se poursuit par l'instruction suivante.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	1	0

Exemple

```
LIST                Explication
       U     E 1.0
       U     E 1.2
       SPB   SAUT   //Saut au repère de saut SAUT si RLG égale 1.
       L     EW8    //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       T     MW22
SAUT:  U     E 2.1  //La séquence de programme se poursuit ici après le saut au repère SAUT.
```

6.5 SPBN Saut si RLG est 0

Format

SPBN <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le RLG égale 1, l'opération **SPBN <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Si le RLG égale 1, le saut ne s'exécute pas. La séquence normale de votre programme se poursuit par l'instruction suivante.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	1	0

Exemple

```
LIST                Explication
      U     E 1.0
      U     E 1.2
      SPBN  SAUT   //Saut au repère SAUT si RLG égale 0.
      L     EW8    //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
      T     MW22
SAUT: U     E 2.1  //La séquence de programme se poursuit ici après le saut au repère SAUT.
```

6.6 SPBB Saut si RLG est 1 avec RB

Format

SPBB <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le RLG égale 1, l'opération **SPBB <repère de saut>** interrompt la séquence logique normale de votre programme et provoque un saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Si le RLG égale 0, le saut ne s'exécute pas. Le RLG est mis à 1 et la séquence normale de votre programme se poursuit par l'instruction suivante.

Avec l'opération **SPBB <repère de saut>**, le résultat logique est copié dans le bit RB, et ce indépendamment de la valeur du RLG.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	x	-	-	-	-	0	1	1	0

Exemple

```
LIST                Explication
       U     E 1.0
       U     E 1.2
       SPBB  SAUT  //Saut au repère SAUT si RLG égale 1. Copier le contenu du bit RLG dans le
                   //bit RB.
       L     EW8   //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       T     MW22
SAUT:  U     E 2.1 //La séquence de programme se poursuit ici après le saut au repère SAUT.
```

6.7 SPBNB Saut si RLG est 0 avec RB

Format

SPBNB <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le RLG égale 0, l'opération **SPBNB <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Si le RLG égale 1, le saut ne s'exécute pas. La séquence normale de votre programme se poursuit par l'instruction suivante.

Avec l'opération **SPBNB <repère de saut>**, le résultat logique est copié dans le bit RB, et ce indépendamment de la valeur du RLG.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	x	-	-	-	-	0	1	1	0

Exemple

```
LIST                Explication
      U     E 1.0
      U     E 1.2
      SPBNB SAUT  //Saut au repère SAUT si le RLG égale 0. Copier le contenu du bit RLG dans
                  //le bit RB.
      L     EW    //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
      T     MW22
SAUT: U     E 2.1 //La séquence de programme se poursuit ici après le saut au repère SAUT.
```

6.8 SPBI Saut si RB est 1

Format

SPBI <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état RB égale 1, l'opération **SPBI <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

6.9 SPBIN Saut si RB est 0

Format

SPBIN <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état RB égale 0, l'opération **SPBIN <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

6.10 SPO Saut si DEB est 1

Format

SPO <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état DEB égale 1, l'opération **SPO <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots). Pour une opération arithmétique combinée, il faut veiller, après chaque opération arithmétique individuelle, à ce qu'aucun débordement ne se produise afin de garantir que chaque résultat intermédiaire soit à l'intérieur de la plage autorisée. Sinon, utilisez l'opération SPS.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L     MW10
       L     3
       *I          //Multiplication du contenu de MW10 par 3.
       SPO   DEBO  //Saut si le résultat déborde de la plage maximale (DEB égale 1).
       T     MW10  //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       U     M 4.0
       R     M 4.0
       SPA   SUIV
DEBO:  UN    M 4.0 //La séquence de programme se poursuit ici après le saut au repère DEBO.
       S     M 4.0
SUIV:  NOP 0       //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.11 SPS Saut si DM est 1

Format

SPS <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état DM égale 1, l'opération **SPS <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	-	-	-	-

Exemple

```
LIST                Explication
       L      EW10
       L      MW12
       *I
       L      DBW25
       +I
       L      MW14
       -I
       SPS    DEBO   //Saut si débordement dans l'une des 3 opérations précédentes,
                     //DM égale 1 (voir  nota).
       T      MW16   //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       U      M 4.0
       R      M 4.0
       SPA    SUIV
DEBO:  UN     M 4.0  //La séquence de programme se poursuit ici après le saut au repère DEBO.
       S      M 4.0
SUIV:  NOP 0         //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

Nota

Dans pareil cas, n'utilisez jamais l'opération **SPO** qui ne testerait de débordement que pour l'opération -I précédente.

6.12 SPZ Saut si égal à 0

Format

SPZ <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état BI1 égale 0 et le bit d'état BI0 égale 0, l'opération **SPZ <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L      MW10
       SRW    1
       SPZ    ZERO   //Sauter au repère de saut ZERO si le bit décalé égale 0.
       L      MW2    //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       INC    1
       T      MW2
       SPA    SUIV
ZERO:  L      MW4    //La séquence de programme se poursuit ici après le saut au repère ZERO.
       INC    1
       T      MW4
SUIV:  NOP 0         //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.13 SPN Saut si différent de 0

Format

SPN <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le résultat indiqué par les bits BI1 et BI0 est inférieur ou égal à zéro (BI1 = 0/BI0 = 1 ou BI1 = 1/BI0 = 0), l'opération **SPN <repère de saut>** (Saut si différent de 0) interrompt la séquence logique normale de votre programme et provoque un saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L     EW8
       L     MW12
       XOW
       SPN   DIZE   //Saut si le contenu de l'accumulateur 1-L est différent de 0.
       UN    M 4.0  //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       S     M 4.0
       SPA   SUIV
DIZE:  UN    M 4.1  //La séquence de programme se poursuit ici après le saut au repère DIZE.
       S     M 4.1
SUIV:  NOP 0        //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.14 SPP Saut si plus

Format

SPP <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état BI1 égale 1 et si le bit d'état BI0 égale 0, l'opération **SPP <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L      EW8
       L      MW12
       -I             //Soustraction du contenu de MW12 du contenu de EW8.
       SPP    POS     //Saut si le résultat est supérieur à 0 (c'est-à-dire contenu de
                      //l'accumulateur 1 > 0).
       UN     M 4.0   //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       S      M 4.0
       SPA    SUIV
POS:   UN     M 4.1   //La séquence de programme se poursuit ici après le saut au repère POS.
       S      M 4.1
SUIV:  NOP 0          //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.15 SPM Saut si moins

Format

SPM <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état BI1 égale 0 et si le bit BI0 égale 1, l'opération **SPM <repère de saut>** interrompt la séquence logique normale de votre programme et provoque un saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L      EW8
       L      MW12
       -I           //Soustraction du contenu de MW12 du contenu de EW8.
       SPM    NEG   //Saut si résultat inférieur à 0 (c'est-à-dire contenu de l'accumulateur
                    //1 < 0).
       UN     M 4.0 //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       S      M 4.0
       SPA    SUIV
NEG:   UN     M 4.1 //La séquence de programme se poursuit ici après le saut au repère NEG.
       S      M 4.1
SUIV:  NOP 0        //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.16 SPPZ Saut si supérieur ou égal à 0

Format

SPPZ <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le résultat indiqué par les bits d'état BI1 et BI0 est supérieur ou égal à 0 (BI1 = 0/BI0 = 0 ou BI1 = 1/BI0 = 0), l'opération **SPPZ <repère de saut>** (Saut si >= 0) interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L     EW8
       L     MW12
       -I          //Soustraction du contenu de MW12 du contenu de EW8.
       SPPZ  REG0  //Saut si le résultat est supérieur ou égal à 0 (c'est-à-dire contenu de
                   //l'accumulateur 1 >= 0).
       UN    M 4.0 //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       S     M 4.0
       SPA   SUIV
REG0:  UN    M 4.1 //La séquence de programme se poursuit ici après le saut au repère REG0.
       S     M 4.1
SUIV:  NOP 0       //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.17 SPMZ Saut si inférieur ou égal à 0

Format

SPMZ <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le résultat indiqué par les bits BI1 et BI0 est inférieur ou égal à zéro (BI1 = 0/BI0 = 0 ou BI1 = 0/BI0 = 1), l'opération **SPMZ <repère de saut>** (Saut si <= 0) interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L     EW8
       L     MW12
       -I          //Soustraction du contenu de MW12 du contenu de EW8.
       SPMZ  RGE0  //Saut si résultat <= 0 (c'est-à-dire contenu de l'accumulateur 1 <= 0).
       UN    M 4.0 //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       S     M 4.0
       SPA   SUIV
RGE0:  UN    M 4.1 //La séquence de programme se poursuit ici après le saut au repère RGE0.
       S     M 4.1
SUIV:  NOP 0       //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.18 SPU Saut si illicite

Format

SPU <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

Si le bit d'état BI1 égale 1 et le bit d'état BI0 égale 1, l'opération **SPU <repère de saut>** interrompt la séquence logique normale de votre programme et provoque le saut à la destination où le traitement du programme doit continuer. La destination de saut est précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Les bits d'état BI1 et BI0 sont tous deux à 1 en cas :

- de division par 0,
- d'utilisation d'opérations illicites ou
- de résultat illicite d'une comparaison de nombres à virgule flottante (utilisation d'un format illicite).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST                Explication
       L     MD10
       L     ED2
       /D          //Division du contenu de MD10 par le contenu de ED2.
       SPU   ERRE  //Saut si division par 0 (c'est-à-dire ED2 = 0).
       T     MD14  //La séquence de programme se poursuit ici si le saut ne s'exécute pas.
       U     M 4.0
       R     M 4.0
       SPA   SUIV
ERRE:  UN    M 4.0 //La séquence de programme se poursuit ici après le saut au repère ERRE.
       S     M 4.0
SUIV:  NOP 0       //La séquence de programme se poursuit ici après le saut au repère SUIV.
```

6.19 LOOP Boucle de programme

Format

LOOP <repère de saut>

Opérande	Description
<repère de saut>	Nom de la destination de saut.

Description de l'opération

LOOP <repère de saut> (Décrémenter l'accumulateur 1-L et sauter si accumulateur 1-L différent de 0)

Cette opération simplifie la programmation de boucles. Le compteur de boucles est un nombre entier non signé de 16 bits qui se trouve dans l'accumulateur 1-L. L'instruction saute au repère de saut indiqué tant que le contenu de l'accumulateur 1-L est différent de 0. Le traitement du programme se poursuit à la destination de saut précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'opération "Boucle de programme" et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple de calcul de la factorielle de 5 (5!)

```
LIST                Explication
       L     L#1    //Charger la constante entière de 32 bits dans l'accumulateur 1.
       T     MD20   //Transférer le contenu de l'accumulateur 1 dans MD20 (initialisation).
       L     5      //Charger le nombre de boucles dans l'accumulateur 1-L.
SUIV:  T     MW10   //Repère de saut = début de la boucle/Transférer l'accumulateur 1-L dans
                    //le compteur de boucles.
       L     MD20
       *     D      //Multiplier le contenu en cours de MD20 par le contenu en cours de MB10.
       T     MD20   //Transférer le résultat de la multiplication dans MD20.
       L     MW10   //Charger le contenu du compteur de boucles dans l'accumulateur 1.
       LOOP  SUIV   //Décrémenter le contenu de l'accumulateur 1 et sauter au repère SUIV si
                    //l'accumulateur 1-L est supérieur à 0.
       L     MW24   //La séquence de programme se poursuit ici après la fin de la boucle.
       L     200
       >I
```

7 Fonctions sur nombres entiers

7.1 Vue d'ensemble des opérations arithmétiques sur nombre entiers

Description

Les opérations arithmétiques combinent le contenu des accumulateurs 1 et 2. Le résultat est rangé dans l'accumulateur 1. Le contenu de l'accumulateur 2 reste inchangé.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est ensuite copié dans l'accumulateur 2 et le contenu de l'accumulateur 4 est copié dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Les opérations arithmétiques sur nombres entiers permettent d'exécuter les fonctions arithmétiques suivantes sur **deux** nombres entiers (16 et 32 bits) :

- +I Additionner accumulateurs 1 et 2 (entiers de 16 bits)
- -I Soustraire accumulateur 1 de accumulateur 2 (entiers de 16 bits)
- /I Diviser accumulateur 2 par accumulateur 1 (entiers de 16 bits)
- *I Multiplier accumulateur 1 par accumulateur 2 (entiers de 16 bits)
- + Additionner constante entière (16, 32 bits)

- +D Additionner accumulateurs 1 et 2 (entiers de 32 bits)
- -D Soustraire accumulateur 1 de accumulateur 2 (entiers de 32 bits)
- *D Multiplier accumulateur 1 par accumulateur 2 (entiers de 32 bits)
- /D Diviser accumulateur 2 par accumulateur 1 (entiers de 32 bits)
- MOD Reste de division entière (32 bits)

7.2 Evaluation des bits du mot d'état dans les opérations sur nombres entiers

Description

Les opérations arithmétiques sur nombres entiers affectent les bits suivants du mot d'état :

- BI1 et BI0,
- DEB,
- DM.

Les tableaux ci-dessous montrent l'état de signal des bits du mot d'état pour les résultats d'opérations sur nombres entiers (16 et 32 bits) :

Plage autorisée	BI1	BI0	DEB	DM
0 (zéro)	0	0	0	*
16 bits : –32 768 <= résultat < 0 (nombre négatif) 32 bits : –2 147 483 648 <= résultat < 0 (nombre négatif)	0	1	0	*
16 bits : 32 767 > =résultat > 0 (nombre positif) 32 bits : 2 147 483 647 > = résultat > 0 (nombre positif)	1	0	0	*

* Le bit DM n'est pas influencé par le résultat de l'opération.

Plage non autorisée	BI1	BI0	DEB	DM
Dépassement négatif de la plage pour une addition 16 bits : résultat = –65536 32 bits : résultat = –4 294 967 296	0	0	1	1
Dépassement négatif de la plage pour une multiplication 16 bits : résultat < –32 768 (nombre négatif) 32 bits : résultat < –2 147 483 648 (nombre négatif)	0	1	1	1
Dépassement positif de la plage pour addition, soustraction 16 bits : résultat > 32 767 (nombre positif) 32 bits : résultat > 2 147 483 647 (nombre positif)	0	1	1	1
Dépassement positif de la plage pour multiplication, division 16 bits : résultat > 32 767 (nombre positif) 32 bits : résultat > 2 147 483 647 (nombre positif)	1	0	1	1
Dépassement négatif de la plage pour addition, soustraction 16 bits : résultat < –32 768 (nombre négatif) 32 bits : résultat < –2 147 483 648 (nombre négatif)	1	0	1	1
Division par zéro	1	1	1	1

Opération	BI1	BI0	DEB	DM
+D : résultat = –4 294 967 296	0	0	1	1
/D ou MOD : division par 0	1	1	1	1

7.3 +I Additionner accumulateurs 1 et 2 (entiers de 16 bits)

Format

+I

Description de l'opération

+I (Additionner entiers de 16 bits)

Cette opération additionne le contenu de l'accumulateur 1-L à celui de l'accumulateur 2-L et sauvegarde le résultat dans l'accumulateur 1-L. Les contenus de l'accumulateur 1-L et de l'accumulateur 2-L sont évalués comme nombres entiers de 16 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération. En cas de débordement haut ou bas, le résultat de l'opération n'est pas un entier de 32 bits, mais un entier de 16 bits.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état				BI1	BI0	DEB	DM
somme	=	0		0	0	0	-
-32768	<=	somme	< 0	0	1	0	-
32767	>=	somme	> 0	1	0	0	-
somme	=	-65536		0	0	1	1
65534	>=	somme	> 32767	0	1	1	1
-65535	<=	somme	< -32768	1	0	1	1

Exemple

```
LIST              Explication
L      EW10       //Charger dans l'accumulateur 1-L la valeur figurant dans le mot d'entrée EW10.
L      MW14       //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L. Charger
                  //la valeur figurant dans le mot de mémento MW14 dans l'accumulateur 1-L.
+I                //Additionner l'accumulateur 2-L et l'accumulateur 1-L et sauvegarder le
                  //résultat dans l'accumulateur 1-L.
T      DB1.DBW25  //Le contenu de l'accumulateur 1-L (résultat) est transféré au mot de données
                  //DBW25 dans DB1.
```

7.4 -I Soustraire accumulateur 1 de accumulateur 2 (entiers de 16 bits)

Format

-I

Description de l'opération

-I (Soustraire entiers de 16 bits) soustrait le contenu de l'accumulateur 1-L de celui de l'accumulateur 2-L et sauvegarde le résultat dans l'accumulateur 1-L. Les contenus de l'accumulateur 1-L et de l'accumulateur 2-L sont évalués comme nombres entiers de 16 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération. En cas de débordement haut ou bas, le résultat de l'opération n'est pas un entier de 32 bits, mais un entier de 16 bits.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état					BI1	BI0	DEB	DM
différence	=	0			0	0	0	-
-32768	<=	différence	<	0	0	1	0	-
32767	>=	différence	>	0	1	0	0	-
65535	>=	différence	>	32767	0	1	1	1
-65535	<=	différence	<	-32768	1	0	1	1

Exemple

```
LIST              Explication
L      EW10       //Charger dans l'accumulateur 1-L la valeur figurant dans le mot d'entrée EW10.
L      MW14       //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L. Charger
                  //la valeur figurant dans le mot de mémento MW14 dans l'accumulateur 1-L.
-I                //Soustraire l'accumulateur 1-L de l'accumulateur 2-L et sauvegarder le
                  //résultat dans l'accumulateur 1-L.
T      DB1.DBW25  //Le contenu de l'accumulateur 1-L (résultat) est transféré au mot de données
                  //DBW25 dans DB1.
```

7.5 *I Multiplier accumulateur 1 par accumulateur 2 (entiers de 16 bits)

Format

***I**

Description de l'opération

***I** (Multiplier entiers de 16 bits) multiplie le contenu de l'accumulateur 2-L par celui de l'accumulateur 1- L. Les contenus de l'accumulateur 1-L et de l'accumulateur 2-L sont évalués comme nombres entiers de 16 bits. Le résultat est rangé dans l'accumulateur 1 comme entier de 32 bits. Si les bits d'état DEB et DM sont tous deux égaux à 1, le résultat est hors de la plage d'un nombre entier de 16 bits.

L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état					BI1	BI0	DEB	DM
produit	=	0			0	0	0	-
-32768	<=	produit	<	0	0	1	0	-
32767	>=	produit	>	0	1	0	0	-
1.073.741.824	>=	produit	>	32767	1	0	1	1
-1.073.709.056	<=	produit	<	-32768	0	1	1	1

Exemple

```
LIST              Explication
L      EW10       //Charger dans l'accumulateur 1 la valeur figurant dans le mot d'entrée EW10.
L      MW14       //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L. Charger
                  //la valeur figurant dans le mot de mémento MW14 dans l'accumulateur 1-L.
*I                //Multiplier l'accumulateur 2-L par l'accumulateur 1-L et sauvegarder le
                  //résultat dans l'accumulateur 1.
T      DB1.DBD25  //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD25 dans DB1.
```

7.6 /I Diviser accumulateur 2 par accumulateur 1 (entiers de 16 bits)

Format

/I

Description de l'opération

/I (Diviser entiers de 16 bits) divise le contenu de l'accumulateur 2-L par celui de l'accumulateur 1-L. Les contenus de l'accumulateur 1-L et de l'accumulateur 2-L sont évalués comme nombres entiers de 16 bits. Le résultat est rangé dans l'accumulateur 1 : il comprend deux entiers de 16 bits, le quotient et le reste de la division. Le quotient est sauvegardé dans l'accumulateur 1-L et le reste de la division dans l'accumulateur 1-H. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état				BI1	BI0	DEB	DM
quotient = 0				0	0	0	-
-32768	<=	quotient <	0	0	1	0	-
32767	>=	quotient >	0	1	0	0	-
quotient	=	32768		1	0	1	1
Division par zéro				1	1	1	1

Exemple

```
LIST            Explication
L     EW10      //Charger dans l'accumulateur 1-L la valeur figurant dans le mot d'entrée EW10.
L     MW14      //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L. Charger
                //la valeur figurant dans le mot de mémento MW14 dans l'accumulateur 1-L.
/I              //Diviser l'accumulateur 2-L par l'accumulateur 1-L et sauvegarder le résultat
                //dans l'accumulateur 1 : ACCU 1-L : quotient, ACCU 1-H : reste de la division
T     MD20      //Le contenu de l'accumulateur 1 (résultat) est transféré dans le double mot
                //de mémento MD20.
```

Exemple : 13 divisé par 4

Contenu de l'accumulateur 2-L avant l'opération (EW10) :	"13"
Contenu de l'accumulateur 1-L avant l'opération (MW14) :	"4"
Opération /I (accumulateur 2-L / accumulateur 1-L) :	"13/4"
Contenu de l'accumulateur 1-L après l'opération (quotient) :	"3"
Contenu de l'accumulateur 1-H après l'opération (reste) :	"1"

7.7 + Additionner constante entière (16, 32 bits)

Format

+ <constante entière>

Opérande	Type de données	Description
<constante entière >	constante, (16 ou 32 bits)	Constante à additionner

Description de l'opération

+ <constante entière> additionne la constante entière au contenu de l'accumulateur 1 et sauvegarde le résultat dans l'accumulateur 1. L'opération s'exécute sans tenir compte des bits d'état ni influer sur eux.

+ <constante entière de 16 bits> additionne une constante entière de 16 bits (dans la plage de -32768 à +32767) au contenu de l'accumulateur 1-L et sauvegarde le résultat dans l'accumulateur 1-L.

+ <constante entière de 32 bits> additionne une constante entière de 32 bits (dans la plage de -2 147 483 648 à 2 147 483 647) au contenu de l'accumulateur 1 et sauvegarde le résultat dans l'accumulateur 1

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple 1

```
LIST            Explication
L     EW10      //Charger la valeur figurant dans le mot d'entrée EW10 dans l'accumulateur
                //1-L.
L     MW14      //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L. Charger
                //la valeur figurant dans le mot de mémento MW14 dans l'accumulateur 1-L.
+I              //Additionner l'accumulateur 2-L à l'accumulateur 1-L et sauvegarder le
                //résultat dans l'accumulateur 1-L.
+     25        //Additionner l'accumulateur 1-L à 25 et sauvegarder le résultat dans
                //l'accumulateur 1-L.
T     DB1.DBW25 //Transférer le contenu de l'accumulateur 1-L (résultat) au mot de données
                //DBW25 dans DB1.
```

Exemple 2

```
LIST              Explication
L      EW12
L      EW14
+      100        //Additionner l'accumulateur 1-L à 100 et sauvegarder le résultat dans
                  //l'accumulateur 1-L.
>I                //Si accumulateur 2 > accumulateur 1, c'est-à-dire EW12 > (EW14 + 100)
SPB    SUIV       //Alors sauter au repère de saut SUIV.
```

Exemple 3

```
LIST              Explication
L      MD20
L      MD24
+D                //Additionner l'accumulateur 1 à l'accumulateur 2 et sauvegarder le résultat
                  //dans l'accumulateur 1.
+      L#-200     //Additionner l'accumulateur 1 à -200 et sauvegarder le résultat dans
                  //l'accumulateur 1.
T      MD28
```

7.8 +D Additionner accumulateurs 1 et 2 (entiers de 32 bits)

Format

+D

Description de l'opération

+D (Additionner entiers de 32 bits)

Cette opération additionne le contenu de l'accumulateur 1 à celui de l'accumulateur 2 et sauvegarde le résultat dans l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres entiers de 32 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état				BI1	BI0	DEB	DM
somme	=	0		0	0	0	-
-2.147.483.648	<=	somme <	0	0	1	0	-
2.147.483.647	>=	somme >	0	1	0	0	-
somme	=	-4.294.967.296		0	0	1	1
4.294.967.294	>=	somme >	2.147.483.647	0	1	1	1
-4.294.967.295	<=	somme <	-2.147.483.648	1	0	1	1

Exemple

```
LIST              Explication
L      ED10       //Charger dans l'accumulateur 1 la valeur figurant dans le double mot d'entrée
                  //ED10.
L      MD14       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur figurant dans le double mot de mémento MD14 dans l'accumulateur 1.
+D                //Additionner l'accumulateur 2 à l'accumulateur 1 et sauvegarder le résultat
                  //dans l'accumulateur 1.
T      DB1.DBD25  //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD25 dans DB1.
```

7.9 -D Soustraire accumulateur 1 de accumulateur 2 (entiers de 32 bits)

Format

-D

Description de l'opération

-D (Soustraire entiers de 32 bits) soustrait le contenu de l'accumulateur 1 de celui de l'accumulateur 2 et sauvegarde le résultat dans l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres entiers de 32 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état					BI1	BI0	DEB	DM
différence	=	0			0	0	0	-
-2.147.483.648	<=	différence	<	0	0	1	0	-
2.147.483.647	>=	différence	>	0	1	0	0	-
4.294.967.295	>=	différence	> 2.147.483.647		0	1	1	1
-4.294.967.295	<=	différence	< -2.147.483.648		1	0	1	1

Exemple

```
LIST              Explication
L      ED10       //Charger dans l'accumulateur 1 la valeur figurant dans le double mot d'entrée
                  //ED10.
L      MD14       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                  //contenu du double mot de données MD14 dans l'accumulateur 1.
-D                //Soustraire l'accumulateur 1 de l'accumulateur 2 et sauvegarder le résultat
                  //dans l'accumulateur 1.
T      DB1.DBD25  //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD25 dans DB1.
```

7.10 *D Multiplier accumulateur 1 par accumulateur 2 (entiers de 32 bits)

Format

***D**

Description de l'opération

***D** (Multiplier entiers de 32 bits)

Cette opération multiplie le contenu de l'accumulateur 1 par celui de l'accumulateur 2. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres entiers de 32 bits. Le résultat est rangé dans l'accumulateur 1 comme entier de 32 bits. Si les bits d'état DEB et DM sont tous deux égaux à 1, le résultat est hors de la plage d'un nombre entier de 32 bits.

L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état			BI1	BI0	DEB	DM
produit	=	0	0	0	0	-
-2.147.483.648	<=	produit < 0	0	1	0	-
2.147.483.647	>=	produit > 0	1	0	0	-
produit	>	2.147.483.647	1	0	1	1
produit	<	-2.147.483.648	0	1	1	1

Exemple

```
LIST            Explication
L     ED10      //Charger dans l'accumulateur 1 la valeur figurant dans le double mot d'entrée
                //ED10.
L     MD14      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur figurant dans le double mot de mémento MD14 dans l'accumulateur 1.
*D              //Multiplier l'accumulateur 2 par l'accumulateur 1 et sauvegarder le résultat
                //dans l'accumulateur 1.
T     DB1.DBD25 //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                //données DBD25 dans DB1.
```

7.11 /D Diviser accumulateur 2 par accumulateur 1 (entiers de 32 bits)

Format

/D

Description de l'opération

/D (Diviser entiers de 32 bits) divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres entiers de 32 bits. Le résultat est rangé dans l'accumulateur 1. Le résultat contient uniquement le quotient et non pas le reste de la division que vous obtenez à l'aide de l'opération MOD.

L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état				BI1	BI0	DEB	DM
Quotient = 0				0	0	0	-
-2147483648	<=	quotient <	0	0	1	0	-
2147483647	>=	quotient >	0	1	0	0	-
quotient	=	2147483648		1	0	1	1
Division par zéro				1	1	1	1

Exemple

```
LIST            Explication
L     ED10      //Charger dans l'accumulateur 1 la valeur figurant dans le double mot d'entrée
                //ED10.
L     MD14      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                //contenu du double mot de mémento MD14 dans l'accumulateur 1.
/D              //Diviser l'accumulateur 2 par l'accumulateur 1 et sauvegarder le résultat
                //(quotient) dans l'accumulateur 1.
T     MD20      //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                //mémento MD20.
```

Exemple : 13 divisé par 4

Contenu de l'accumulateur 2 avant l'opération (ED10) : "13"

Contenu de l'accumulateur 1 avant l'opération (MD14) : "4"

Opération /D (accumulateur 2 / accumulateur 1) : "13/4"

Contenu de l'accumulateur 1 après l'opération (quotient) : "3"

7.12 MOD Reste de division entière (32 bits)

Format

MOD

Description de l'opération

MOD (Reste de division entière de 32 bits) divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres entiers de 32 bits. Le résultat est sauvegardé dans l'accumulateur 1. Il contient uniquement le reste de la division et non pas le quotient que vous obtenez à l'aide de l'opération /D.

L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Formation des bits d'état					BI1	BI0	DEB	DM
reste	=	0			0	0	0	-
-2147483648	<=	reste	<	0	0	1	0	-
2147483647	>=	reste	>	0	1	0	0	-
Division par zéro					1	1	1	1

Exemple

```
LIST            Explication
L     ED10      //Charger dans l'accumulateur 1 la valeur figurant dans le double mot d'entrée
                //ED10.
L     MD14      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                //contenu du double mot de mémento MD14 dans l'accumulateur 1.
MOD             //Diviser l'accumulateur 2 par l'accumulateur 1 et sauvegarder le résultat
                //(reste de la division) dans l'accumulateur 1.
T     MD20      //Le contenu de l'accumulateur 1 (résultat) est transféré dans le double mot
                //de mémento MD20.
```

Exemple : 13 divisé par 4

Contenu de l'accumulateur 2 avant l'opération (ED10) : "13"

Contenu de l'accumulateur 1 avant l'opération (MD14) : "4"

Opération /MOD (accumulateur 2 / accumulateur 1) : "13/4"

Contenu de l'accumulateur 1 après l'opération (reste de la division) : "1"

8 Fonctions sur nombres à virgule flottante

8.1 Vue d'ensemble des opérations arithmétiques sur nombres à virgule flottante

Description

Les opérations arithmétiques combinent le contenu des accumulateurs 1 et 2. Le résultat est rangé dans l'accumulateur 1, le contenu de l'accumulateur 2 reste inchangé.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est ensuite copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu précédent de l'accumulateur 4 reste inchangé.

Les nombres à virgule flottante IEEE de 32 bits ont le type de données REAL. Les opérations arithmétiques sur nombres à virgule flottante permettent d'exécuter les fonctions arithmétiques suivantes sur **deux** nombres à virgule flottante IEEE de 32 bits :

- +R Additionner accumulateurs 1 et 2 (réels VF IEEE, 32 bits)
- -R Soustraire accumulateur 1 d'accumulateur 2 (réels VF IEEE, 32 bits)
- *R Multiplier accumulateur 1 par accumulateur 2 (réels VF IEEE, 32 bits)
- /R Diviser accumulateur 2 par accumulateur 1 (réels VF IEEE, 32 bits)

Les opérations suivantes permettent d'exécuter les fonctions arithmétiques suivantes sur **un** nombre à virgule flottante IEEE de 32 bits :

- ABS Valeur absolue d'un nombre à virgule flottante (VF IEEE, 32 bits)
- SQR Carré d'un nombre à virgule flottante (32 bits)
- SQRT Racine carrée d'un nombre à virgule flottante (32 bits)
- EXP Valeur exponentielle d'un nombre à virgule flottante (32 bits)
- LN Logarithme naturel d'un nombre à virgule flottante (32 bits)

- SIN Sinus d'un angle comme nombres à virgule flottante (32 bits)
- COS Cosinus d'un angle comme nombres à virgule flottante (32 bits)
- TAN Tangente d'un angle comme nombres à virgule flottante (32 bits)
- ASIN Arc sinus d'un nombre à virgule flottante (32 bits)
- ACOS Arc cosinus d'un nombre à virgule flottante (32 bits)
- ATAN Arc tangente d'un nombre à virgule flottante (32 bits)

8.2 Evaluation des bits du mot d'état dans les opérations sur nombres à virgule flottante

Description

Les opérations arithmétiques sur nombres à virgule flottante affectent les bits suivants du mot d'état :

- BI1 et BI0,
- DEB,
- DM.

Les tableaux ci-dessous montrent l'état de signal des bits du mot d'état pour les résultats d'opérations sur nombres à virgule flottante (32 bits) :

Plage autorisée	BI1	BI0	DEB	DM
+0, -0 (zéro)	0	0	0	*
-3.402823E+38 < résultat < -1.175494E-38 (nombre négatif)	0	1	0	*
+1.175494E-38 < résultat < 3.402824E+38 (nombre positif)	1	0	0	*

* Le bit DM n'est pas influencé par le résultat de l'opération.

Plage non autorisée	BI1	BI0	DEB	DM
Dépassement bas -1.175494E-38 < résultat < - 1.401298E-45 (nombre négatif)	0	0	1	1
Dépassement bas +1.401298E-45 < résultat < +1.175494E-38 (nombre positif)	0	0	1	1
Débordement résultat < -3.402823E+38 (nombre négatif)	0	1	1	1
Débordement résultat > 3.402823E+38 (nombre positif)	1	0	1	1
Pas un nombre réel correct ou opération illicite (valeur d'entrée hors de la plage de valeurs autorisée)	1	1	1	1

8.3 Opérations de base

8.3.1 +R Additionner accumulateurs 1 et 2 (réels VF IEEE, 32 bits)

Format

+R

Description de l'opération

+R (Additionner nombres à virgule flottante VF IEEE, 32 bits)

Cette opération additionne les contenus des accumulateurs 1 et 2 et sauvegarde le résultat dans l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres à virgule flottante VF IEEE de 32 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Exemple

```
LIST            Explication
AUF    DB10
L      ED10     //Charger dans l'accumulateur 1 la valeur du double mot d'entrée ED10.
L      MD14     //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du double mot de mémento MD14 dans l'accumulateur 1.
+R              //Additionner l'accumulateur 2 et l'accumulateur 1 ; sauvegarder le résultat
                //dans l'accumulateur 1.
T      DBD25    //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                //données DBD25 dans DB10.
```

8.3.2 -R Soustraire accumulateur 1 d'accumulateur 2 (réels VF IEEE, 32 bits)

Format

-R

Description de l'opération

-R (Soustraire nombres à virgule flottante VF IEEE, 32 bits)

Cette opération soustrait le contenu de l'accumulateur 1 de celui de l'accumulateur 2 et sauvegarde le résultat dans l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres à virgule flottante VF IEEE de 32 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Exemple

```
LIST              Explication
AUF    DB10
L      ED10       //Charger dans l'accumulateur 1 la valeur du double mot d'entrée ED10.
L      MD14       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur du double mot de mémento MD14 dans l'accumulateur 1.
-R                //Soustraire l'accumulateur 1 de l'accumulateur 2 ; sauvegarder le résultat
                  //dans l'accumulateur 1.
T      DBD25      //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD25 dans DB10.
```

8.3.3 *R Multiplier accumulateur 1 par accumulateur 2 (réels VF IEEE, 32 bits)

Format

***R**

Description de l'opération

***R** (Multiplier nombres à virgule flottante VF IEEE, 32 bits)

Cette opération multiplie le contenu de l'accumulateur 2 par le contenu de l'accumulateur 1 – ces contenus sont évalués comme nombre à virgule flottante VF IEEE de 32 bits – et range le résultat comme nombre à virgule flottante VF IEEE de 32 bits dans l'accumulateur 1. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Exemple

```
LIST              Explication
AUF     DB10
L       ED10      //Charger dans l'accumulateur 1 la valeur du double mot d'entrée ED10.
L       MD14      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur du double mot de mémento MD14 dans l'accumulateur 1.
*R                //Multiplier l'accumulateur 2 par l'accumulateur 1 et sauvegarder le résultat
                  //dans l'accumulateur 1.
T       DBD25     //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD25 dans DB10.
```

8.3.4 /R Diviser accumulateur 2 par accumulateur 1 (réels VF IEEE, 32 bits)

Format

/R

Description de l'opération

/R (Diviser nombres à virgule flottante VF IEEE, 32 bits)

Cette opération divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1 et sauvegarde le résultat dans l'accumulateur 1. Les contenus de l'accumulateur 1 et de l'accumulateur 2 sont évalués comme nombres à virgule flottante VF IEEE de 32 bits. L'opération s'exécute sans tenir compte du RLG ni influer sur lui. Les bits d'état BI1, BI0, DM et DEB sont mis à 1 comme résultat de l'opération.

Le contenu de l'accumulateur 2 reste inchangé pour les CPU à deux accumulateurs.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est copié dans l'accumulateur 2 et celui de l'accumulateur 4 dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	x	-	-	-	-

Exemple

```
LIST              Explication
AUF     DB10
L       ED10      //Charger dans l'accumulateur 1 la valeur du double mot d'entrée ED10.
L       MD14      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur du double mot de mémento MD14 dans l'accumulateur 1.
/R                //Diviser l'accumulateur 2 par l'accumulateur 1 ; sauvegarder le résultat dans
                  //l'accumulateur 1.
T       DBD20     //Le contenu de l'accumulateur 1 (résultat) est transféré au double mot de
                  //données DBD20 dans DB10.
```

8.3.5 ABS Valeur absolue d'un nombre à virgule flottante (VF IEEE, 32 bits)

Format

ABS

Description de l'opération

ABS (Valeur absolue d'un nombre à virgule flottante VF IEEE, 32 bits)

Cette opération forme la valeur absolue d'un nombre à virgule flottante IEEE de 32 bits figurant dans l'accumulateur 1. Le résultat est sauvegardé dans l'accumulateur 1. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      ED8        //Charger la valeur dans l'accumulateur 1 (exemple : ED8 = -1.5E+02).
ABS               //Former la valeur absolue et sauvegarder le résultat dans l'accumulateur 1.
T      MD10       //Transférer le résultat au double mot de mémento MD10
                  //(exemple : résultat = 1.5E+02).
```

8.4 Opérations étendues

8.4.1 SQR Carré d'un nombre à virgule flottante (32 bits)

Format

SQR

Description de l'opération

L'opération **SQR** (Carré d'un nombre à virgule flottante IEEE de 32 bits) calcule le carré d'un nombre à virgule flottante IEEE de 32 bits contenu dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Voir aussi Evaluation des bits du mot d'état dans les opérations sur nombres à virgule flottante.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST              Explication
      OPN   DB17  //Ouvrir le bloc de données DB17.
      L     DBD0  //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de données
                  //DBD0 (doit être un nombre à virgule flottante).
      SQR         //Calculer le carré du nombre à virgule flottante IEEE de 32 bits contenu dans
                  //l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV    //Tester à 0 le bit DEB du mot d'état.
      SPB   OK    //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération SQR.
BEA               //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                  //SQR.
OK:   T     DBD4  //Transférer le résultat de l'accumulateur 1 dans le double mot de données DBD4.
```

8.4.2 SQRT Racine carrée d'un nombre à virgule flottante (32 bits)

Format

SQRT

Description de l'opération

L'opération **SQRT** (Racine carrée d'un nombre à virgule flottante IEEE de 32 bits) calcule la racine carrée d'un nombre à virgule flottante IEEE de 32 bits contenu dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. La valeur d'entrée doit être positive ou nulle, exception faite de -0 dont la racine carrée est
-0. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST                Explication
      L     MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                    //MD10 (doit être un nombre à virgule flottante).
      SQRT          //Calculer la racine carrée du nombre à virgule flottante IEEE de 32 bits contenu
                    //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV      //Tester à 0 le bit DEB du mot d'état.
      SPB   OK      //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération SQRT.
BEA                 //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                    //SQRT.
            MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
OK:   T     DBD4    //Transférer le résultat de l'accumulateur 1 dans le double mot de données DBD4.
```

8.4.3 EXP Valeur exponentielle d'un nombre à virgule flottante (32 bits)

Format

EXP

Description de l'opération

L'opération **EXP** (Valeur exponentielle d'un nombre à virgule flottante IEEE de 32 bits) calcule la valeur exponentielle (de base e) d'un nombre à virgule flottante IEEE de 32 bits contenu dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST                Explication
      L     MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                    //MD10 (doit être un nombre à virgule flottante).
      EXP           //Calculer la valeur exponentielle (de base e) du nombre à virgule flottante
                    //IEEE de 32 bits contenu dans l'accumulateur 1 et ranger le résultat dans
                    //l'accumulateur 1.
      UN    OV      //Tester à 0 le bit DEB du mot d'état.
      SPB   OK      //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération EXP.
BEA                 //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                    //EXP.
OK:   T     MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

8.4.4 LN Logarithme naturel d'un nombre à virgule flottante (32 bits)

Format

LN

Description de l'opération

L'opération **LN** (Logarithme naturel d'un nombre à virgule flottante IEEE de 32 bits) calcule le logarithme naturel – logarithme de base e – d'un nombre à virgule flottante IEEE de 32 bits contenu dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. La valeur d'entrée doit être strictement positive. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
- zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
- infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Exemple

```
LIST              Explication
      L     MD10  //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                  //MD10 (doit être un nombre à virgule flottante).
      LN          //Calculer le logarithme naturel du nombre à virgule flottante IEEE de 32 bits
                  //contenu dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV    //Tester à 0 le bit DEB du mot d'état.
      SPB   OK    //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération LN.
BEA               //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                  //LN.
OK:   T     MD20  //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

8.4.5 SIN Sinus d'un angle comme nombres à virgule flottante (32 bits)

Format

SIN

Description de l'opération

L'opération **SIN** (Sinus d'angles comme nombres à virgule flottante IEEE de 32 bits) calcule le sinus d'un angle indiqué en radians. L'angle doit figurer sous forme de nombre à virgule flottante dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement
+zéro	0	0	0	-	
+infini	1	0	1	1	
- zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST            Explication
L       MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de
                //mémento MD10 (doit être un nombre à virgule flottante).
SIN             //Calculer le sinus du nombre à virgule flottante IEEE de 32 bits contenu dans
                //l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
T       MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento
                //MD20.
```

8.4.6 COS Cosinus d'un angle comme nombres à virgule flottante (32 bits)

Format

COS

Description de l'opération

L'opération **COS** (Cosinus d'angles comme nombres à virgule flottante IEEE de 32 bits) calcule le cosinus d'un angle indiqué en radians. L'angle doit figurer sous forme de nombre à virgule flottante dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement
+zéro	0	0	0	-	
- zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST            Explication
L       MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de
                //mémento MD10 (doit être un nombre à virgule flottante).
COS             //Calculer le cosinus du nombre à virgule flottante IEEE de 32 bits contenu
                //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
T       MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento
                //MD20.
```

8.4.7 TAN Tangente d'un angle comme nombres à virgule flottante (32 bits)

Format

TAN

Description de l'opération

L'opération **TAN** (Tangente d'angles comme nombres à virgule flottante IEEE de 32 bits) calcule la tangente d'un angle indiqué en radians. L'angle doit figurer sous forme de nombre à virgule flottante dans l'accumulateur 1. Le résultat est rangé dans l'accumulateur 1. Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+infini	1	0	1	1	Débordement
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement bas
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
- infini	0	1	1	1	Débordement
-qNaN	1	1	1	1	

Exemple

```
LIST                Explication
      L     MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                    //MD10 (doit être un nombre à virgule flottante).
      TAN           //Calculer la tangente du nombre à virgule flottante IEEE de 32 bits contenu
                    //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV      //Tester à 0 le bit DEB du mot d'état.
      SPB   OK      //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération TAN.
BEA                 //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                    //TAN.
OK:   T     MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

8.4.8 ASIN Arc sinus d'un nombre à virgule flottante (32 bits)

Format

ASIN

Description de l'opération

L'opération **ASIN** (Arc sinus d'un nombre à virgule flottante IEEE de 32 bits) calcule l'arc sinus d'un nombre à virgule flottante figurant dans l'accumulateur 1. La valeur d'entrée doit être comprise entre :

-1 <= valeur d'entrée <= +1

Le résultat est un angle indiqué en radians. Sa valeur est comprise dans la plage suivante :

$-\pi / 2$ <= arc sinus (ACCU 1) <= $+\pi / 2$, avec π = 3,14159...

Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST              Explication
      L     MD10  //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                  //MD10 (doit être un nombre à virgule flottante).
      ASIN        //Calculer l'arc sinus du nombre à virgule flottante IEEE de 32 bits contenu
                  //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV    //Tester à 0 le bit DEB du mot d'état.
      SPB   OK    //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération ASIN.
BEA               //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                  //ASIN.
OK:   T     MD20  //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

8.4.9 ACOS Arc cosinus d'un nombre à virgule flottante (32 bits)

Format

ACOS

Description de l'opération

L'opération **ACOS** (Arc cosinus d'un nombre à virgule flottante IEEE de 32 bits) calcule l'arc cosinus d'un nombre à virgule flottante figurant dans l'accumulateur 1. La valeur d'entrée doit être comprise entre :

-1 <= valeur d'entrée <= +1

Le résultat est un angle indiqué en radians. Sa valeur est comprise dans la plage suivante :

0 <= arc cosinus (ACCU 1) <= π, avec π = 3,14159...

Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST                Explication
      L     MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                    //MD10 (doit être un nombre à virgule flottante).
      ACOS          //Calculer l'arc cosinus du nombre à virgule flottante IEEE de 32 bits contenu
                    //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV      //Tester à 0 le bit DEB du mot d'état.
      SPB   OK      //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération ACOS.
BEA                 //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                    //ACOS.
OK:   T     MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

8.4.10 ATAN Arc tangente d'un nombre à virgule flottante (32 bits)

Format

ATAN

Description de l'opération

L'opération **ATAN** (Arc tangente d'un nombre à virgule flottante IEEE de 32 bits) calcule l'arc tangente d'un nombre à virgule flottante figurant dans l'accumulateur 1. Le résultat est un angle indiqué en radians. Sa valeur est comprise dans la plage suivante :

$-\pi / 2 <=$ arc tangente (ACCU 1) $<= +\pi / 2$, avec $\pi = 3{,}14159...$

Cette opération influe sur les bits BI1, BI0, DEB et DM du mot d'état.

Le contenu de l'accumulateur 2 – et des accumulateurs 3 et 4 pour les CPU à quatre accumulateurs – reste inchangé.

Résultat

Le résultat dans ACCU 1 est	BI1	BI0	DEB	DM	Indication
+qNaN	1	1	1	1	
+normalisé	1	0	0	-	
+dénormalisé	0	0	1	1	Débordement
+zéro	0	0	0	-	
-zéro	0	0	0	-	
-dénormalisé	0	0	1	1	Débordement bas
-normalisé	0	1	0	-	
-qNaN	1	1	1	1	

Exemple

```
LIST                Explication
      L     MD10    //Charger dans l'accumulateur 1 la valeur figurant dans le double mot de mémento
                    //MD10 (doit être un nombre à virgule flottante).
      ATAN          //Calculer l'arc tangente du nombre à virgule flottante IEEE de 32 bits contenu
                    //dans l'accumulateur 1 et ranger le résultat dans l'accumulateur 1.
      UN    OV      //Tester à 0 le bit DEB du mot d'état.
      SPB   OK      //Sauter au repère OK si aucune erreur n'est apparue lors de l'opération ATAN.
BEA                 //Fin de bloc inconditionnelle si une erreur est apparue lors de l'opération
                    //ATAN.
OK:   T     MD20    //Transférer le résultat de l'accumulateur 1 dans le double mot de mémento MD20.
```

9 Opérations de chargement et de transfert

9.1 Vue d'ensemble des opérations de chargement et de transfert

Description

Les opérations L (charger) et T (transférer) vous permettent de programmer l'échange d'informations entre des modules d'entrées ou de sorties, d'une part, et des zones de mémoire, d'autre part, ou bien entre des zones de mémoire. La CPU exécute ces opérations inconditionnellement à chaque cycle, c'est-à-dire quel que soit le résultat logique RLG d'une opération.

Vous disposez des opérations de chargement et de transfert suivantes :

- L Charger
- L STW Charger mot d'état dans l'accumulateur 1
- LAR1 Charger contenu de l'accumulateur 1 dans registre d'adresse 1
- LAR1 <D> Charger pointeur de 32 bits dans registre d'adresse 1
- LAR1 AR2 Charger contenu du registre d'adresse 2 dans registre d'adresse 1
- LAR2 Charger contenu de l'accumulateur 1 dans registre d'adresse 2
- LAR2 <D> Charger pointeur de 32 bits dans registre d'adresse 2

- T Transférer
- T STW Transférer accumulateur 1 dans mot d'état
- TAR Permuter registre d'adresse 1 avec registre d'adresse 2
- TAR1 Transférer registre d'adresse 1 dans l'accumulateur 1
- TAR1 <D> Transférer registre d'adresse 1 à l'adresse de destination (32 bits)
- TAR1 AR2 Transférer registre d'adresse 1 dans registre d'adresse 2
- TAR2 Transférer registre d'adresse 2 dans l'accumulateur 1
- TAR2 <D> Transférer registre d'adresse 2 à l'adresse de destination (32 bits)

9.2 L Charger

Format

L <opérande>

	Type de données	Zone de mémoire	Adresse source
<opérande>	BYTE WORD DWORD	E, A, PE, M, L, D, pointeur, paramètre	0...65535 0...65534 0...65532

Description de l'opération

L <opérande> charge dans l'accumulateur 1 l'octet, le mot ou le double mot indiqué une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2 et que l'accumulateur 1 a été mis à 0.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      EB10       //Charger l'octet d'entrée EB10 dans l'accumulateur 1-L-L.
L      MB120      //Charger l'octet de mémento MB120 dans l'accumulateur 1-L-L.
L      DBB12      //Charger l'octet de données DBB12 dans l'accumulateur 1-L-L.
L      DIW15      //Charger le mot de données d'instance DIW15 dans l'accumulateur 1-L.
L      LD252      //Charger le double mot de données locales LD252 dans l'accumulateur 1.
L      P# E 8.7   //Charger le pointeur dans l'accumulateur 1.
L      OTTO       //Charger le paramètre "OTTO" dans l'accumulateur 1.
L      P# ANNA    //Charger dans l'accumulateur 1 le pointeur sur le paramètre spécifié.
                  //(Cette instruction charge le décalage d'adresse relatif du paramètre
                  //spécifié ; pour déterminer le décalage absolu dans le bloc de données
                  //d'instance de blocs fonctionnels multi-instance, il faut encore ajouter à
                  //cette valeur le contenu du registre d'adresse 2.)
```

Contenu de l'accumulateur 1

Contenu	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
avant exécution de l'opération de chargement	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
après exécution de **L MB10** (L <octet>)	00000000	00000000	00000000	<MB10>
après exécution de **L MW10** (L <mot>)	00000000	00000000	<MB10>	<MB11>
après exécution de **L MD10** (L <double mot>)	<MB10>	<MB11>	<MB12>	<MB13>
après exécution de **L P# ANNA** (dans le FB)	<86>	<décalage d'un bit de ANNA par rapport au début du FB> Pour déterminer le décalage absolu dans le bloc de données d'instance de blocs fonctionnels multi-instance, il faut encore ajouter à cette valeur le contenu du registre d'adresse 2.		
après exécution de **L P# ANNA** (dans la FC)	<une adresse interzone de la donnée transmise à ANNA>			
	X = "1" ou "0"			

9.3 L STW Charger mot d'état dans l'accumulateur 1

Format

L STW

Description de l'opération

L STW (Opération L avec l'opérande STW)

Cette opération charge le contenu du mot d'état dans l'accumulateur 1. Cette opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Nota

Dans les CPU de la famille S7-300, l'opération **L STW** ne charge pas les bits /PI, ETAT et OU du mot d'état. Seuls les bits 1, 4, 5, 6, 7 et 8 sont chargés dans les emplacements correspondants du mot de poids faible de l'accumulateur 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
L       STW     //Charger le contenu du mot d'état dans l'accumulateur 1.
```

Le contenu de l'accumulateur 1 après exécution de **L STW** est le suivant :

Bits	31-9	8	7	6	5	4	3	2	1	0
Contenu :	0	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI

9.4 LAR1 Charger contenu de l'accumulateur 1 dans registre d'adresse 1

Format

LAR1

Description de l'opération

LAR1

Cette opération charge dans le registre d'adresse 1 (AR1) le contenu de l'accumulateur 1 (32 bits). L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.5 LAR1 <D> Charger pointeur de 32 bits dans registre d'adresse 1

Format

LAR1 <D>

Paramètre	Type de données	Zone de mémoire	Adresse source
<D>	DWORD Constante pointeur	D, M, L	0...65532

Description de l'opération

LAR1 <D>

Cette opération charge dans le registre d'adresse 1 (AR1) le contenu du double mot <D> indiqué ou une constante de pointeur. L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemples : Adresses directes

```
LIST              Explication
LAR1   DBD 20     //Charger dans AR1 le pointeur figurant dans le double mot de données DBD20.
LAR1   DID 30     //Charger dans AR1 le pointeur figurant dans le double mot d'instance DID30.
LAR1   LD 180     //Charger dans AR1 le pointeur figurant dans le double mot de données locales
                  //LD180.
LAR1   MD 24      //Charger dans AR1 le pointeur figurant dans le double mot de mémento MD24.
```

Exemple : Constante pointeur

```
LIST              Explication
LAR1   P#M100.0   //Charger une constante de pointeur de 32 bits dans AR1.
```

9.6 LAR1 AR2 Charger contenu du registre d'adresse 2 dans registre d'adresse 1

Format

LAR1 AR2

Description de l'opération

LAR1 AR2 (Opération LAR1 avec l'opérande AR2)

Cette opération charge dans le registre d'adresse 1 (AR1) le contenu du registre d'adresse 2 (AR2). L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.7 LAR2 Charger contenu de l'accumulateur 1 dans registre d'adresse 2

Format

LAR2

Description de l'opération

LAR2 charge dans le registre d'adresse 2 (AR2) le contenu de l'accumulateur 1 (32 bits).

L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.8 LAR2 <D> Charger pointeur de 32 bits dans registre d'adresse 2

Format

LAR2 <D>

Paramètre	Type de données	Zone de mémoire	Adresse source
<D>	DWORD Constante pointeur	D, M, L	0...65532

Description de l'opération

LAR2 <D>

Cette opération charge dans le registre d'adresse 2 (AR2) le contenu du double mot <D> indiqué ou une constante de pointeur. L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemples : Adresses directes

```
LIST              Explication
LAR2   DBD 20     //Charger dans AR2 le pointeur figurant dans le double mot de données DBD20.
LAR2   DID 30     //Charger dans AR2 le pointeur figurant dans le double mot d'instance DID30.
LAR2   LD 180     //Charger dans AR2 le pointeur figurant dans le double mot de données locales
                  //LD180.
LAR2   MD 24      //Charger dans AR2 le pointeur figurant dans le double mot de mémento MD24.
```

Exemple : Constante pointeur

```
LIST              Explication
LAR2   P#M100.0   //Charger une constante de pointeur de 32 bits dans AR2.
```

9.9 T Transférer

Format

T <opérande>

Paramètre	Type de données	Zone de mémoire	Adresse source
<opérande>	BYTE	E, A, PA, M, L, D	0...65535
	WORD		0...65534
	DWORD		0...65532

Description de l'opération

T <opérande>

Cette opération transfère (copie) le contenu de l'accumulateur 1 dans l'adresse de destination si le relais de masquage (Master Control Relay) est en fonction (MCR = 1). Si le relais MCR égale 0, c'est la valeur 0 qui est écrite dans l'adresse de destination. Le nombre d'octets copiés à partir de l'accumulateur 1 dépend de la taille indiquée dans l'adresse de destination. L'accumulateur 1 conserve les données même après exécution de l'opération de transfert. Le transfert dans la zone de périphérie directe (zone de mémoire PA) exécute également un transfert du contenu de l'accumulateur 1 ou de l'état de signal 0 (si MCR égale 0) à l'adresse correspondante dans la mémoire image des sorties (zone de mémoire A). L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemples

```
LIST            Explication
T     AB10      //Transférer le contenu de l'accumulateur 1-L-L dans l'octet de sortie AB10.
T     MW14      //Transférer le contenu de l'accumulateur 1-L dans le mot de mémento MW14.
T     DBD2      //Transférer le contenu de l'accumulateur 1 dans le double mot de données DBD2.
```

9.10 T STW Transférer accumulateur 1 dans mot d'état

Format

T STW

Description de l'opération

T STW (Opération T avec l'opérande STW)

Cette opération transfère dans le mot d'état les bits 0 à 8 figurant dans l'accumulateur 1.

L'opération s'exécute sans tenir compte des bits du mot d'état.

Remarque : dans la gamme de CPU S7-300, les bits du mot d'état /ER, STA et OR ne sont pas décrits par l'instruction T STW. Seuls les bit 1, 4, 5, 6, 7 et 8 sont écrits selon l'affectation de bits de l'AKKU1.

Mot d'état

	RB	**BI1**	**BI0**	**DEB**	**DM**	**OU**	**ETAT**	**RLG**	**/PI**
Ecriture :	x	x	x	x	x	x	x	x	x

Exemple

```
LIST              Explication
T     STW         //Transférer les bits 0 à 8 de l'accumulateur 1 dans le mot d'état.
```

Les bits dans l'accumulateur 1 contiennent les bits d'état suivants :

Bits	**31-9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Contenu :	*	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI

* Ces bits ne sont pas transférés.

9.11 TAR Permuter registre d'adresse 1 avec registre d'adresse 2

Format

TAR

Description de l'opération

TAR (Permuter registre d'adresse)

Cette opération permute les contenus des registres d'adresse AR 1 et AR 2. Cette opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Le contenu du registre d'adresse 1 est déplacé dans le registre d'adresse 2 et celui du registre d'adresse 2 dans le registre d'adresse 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.12 TAR1 Transférer registre d'adresse 1 dans l'accumulateur 1

Format

TAR1

Description de l'opération

TAR1 transfère le contenu du registre d'adresse 1 (AR1) dans l'accumulateur 1 (32 bits). Le contenu de l'accumulateur 1 a auparavant été rangé dans l'accumulateur 2. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.13 TAR1 <D> Transférer registre d'adresse 1 à l'adresse de destination (32 bits)

Format

TAR1 <D>

Paramètre	Type de données	Zone de mémoire	Adresse source
<D>	DWORD	D, M, L	0...65532

Description de l'opération

TAR1 <D>

Cette opération transfère le contenu du registre d'adresse 1 (AR1) dans le double mot <D> indiqué. Les zones de destination possibles sont les suivantes : doubles mots de mémento (MD), doubles mots de données locales (LD), doubles mots de données (DBD) et doubles mots d'instance (DID).

Les accumulateurs 1 et 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemples

```
LIST              Explication
TAR1   DBD20      //Transférer le contenu de AR1 dans le double mot de données DBD20.
TAR1   DID30      //Transférer le contenu de AR1 dans le double mot d'instance DID30.
TAR1   LD18       //Transférer le contenu de AR1 dans le double mot de données locales LD18.
TAR1   MD24       //Transférer le contenu de AR1 dans le double mot de mémento MD24.
```

9.14 TAR1 AR2 Transférer registre d'adresse 1 dans registre d'adresse 2

Format

TAR1 AR2

Description de l'opération

TAR1 AR2 (Opération TAR1 avec l'opérande AR2)

Cette opération transfère le contenu du registre d'adresse 1 (AR1) dans le registre d'adresse 2 (AR2).

L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.15 TAR2 Transférer registre d'adresse 2 dans l'accumulateur 1

Format

TAR2

Description de l'opération

TAR2 transfère le contenu du registre d'adresse 2 (AR2) dans l'accumulateur 1 (32 bits). Le contenu de l'accumulateur 1 a auparavant été sauvegardé dans l'accumulateur 2. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

9.16 TAR2 <D> Transférer registre d'adresse 2 à l'adresse de destination (32 bits)

Format

TAR2 <D>

Paramètre	Type de données	Zone de mémoire	Adresse source
<D>	DWORD	D, M, L	0...65532

Description de l'opération

TAR2 <D>

Cette opération transfère le contenu du registre d'adresse 2 (AR2) dans le double mot <D> indiqué. Les zones de destination possibles sont les suivantes : doubles mots de mémento (MD), doubles mots de données locales (LD), doubles mots de données (DBD) et doubles mots d'instance (DID).

L'accumulateur 1 et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemples

```
LIST              Explication
TAR2   DBD20      //Transférer le contenu de AR2 dans le double mot de données DBD20.
TAR2   DID30      //Transférer le contenu de AR2 dans le double mot d'instance DID30.
TAR2   LD18       //Transférer le contenu de AR2 dans le double mot de données locales LD18.
TAR2   MD24       //Transférer le contenu de AR2 dans le double mot de mémento MD24.
```

10 Opérations de gestion d'exécution de programme

10.1 Vue d'ensemble des opérations de gestion d'exécution de programme

Description

Vous disposez des opérations de gestion de programme suivantes :

- BE Fin de bloc
- BEB Fin de bloc conditionnelle
- BEA Fin de bloc inconditionnelle
- CALL Appel de bloc
- CC Appel de bloc conditionnel
- UC Appel de bloc inconditionnel

- Appeler FB
- Appeler FC
- Appeler SFB
- Appeler SFC
- Appeler multi-instance
- Appeler un bloc dans une bibliothèque

- Relais de masquage (Master Control Relay, MCR)
- Remarques importantes sur l'utilisation de la fonctionnalité MCR
- MCR(Sauvegarder RLG dans pile MCR, début de zone MCR
-)MCR Fin de zone MCR
- MCRA Activer la zone MCR
- MCRD Désactiver la zone MCR

10.2 BE Fin de bloc

Format

BE

Description de l'opération

BE (Fin de bloc)

Cette opération interrompt la séquence normale de votre programme dans le bloc en cours et saute au bloc ayant appelé le bloc en cours. Le programme se poursuit avec la première instruction suivant l'appel du bloc. La zone de données locales en cours est libérée et la zone de données locales précédentes redevient la zone de données locales en cours. Les blocs de données qui étaient ouverts au moment de l'appel sont à nouveau ouverts. De plus, la dépendance par rapport au MCR du bloc appelant est restaurée et le RLG est transféré du bloc en cours au bloc appelant. L'opération BE prend effet sans aucune condition. Si l'opération BE est sautée, le déroulement de votre programme ne s'achève pas, mais se poursuit à la destination de saut, à l'intérieur du bloc.

L'opération BE ne correspond pas à celle du logiciel S5. Pour le matériel S7, l'opération BE a la même fonction que l'opération BEA de S5.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple

```
LIST                Explication
       U      E 1.0
       SPB    SUIV   //Saut au repère de saut SUIV si le résultat logique égale 1 (E 1.0 = 1).
       L      EW4    //Poursuivre ici si le saut ne s'exécute pas.
       T      EW10
       U      E 6.0
       U      E 6.1
       S      M
              12.0
       BE            //Fin de bloc.
SUIV:  NOP 0         //Poursuivre ici si le saut s'exécute.
```

10.3 BEB Fin de bloc conditionnelle

Format

BEB

Description de l'opération

Si le RLG égale 1, l'opération **BEB** (Fin de bloc conditionnelle) interrompt la séquence normale de votre programme dans le bloc en cours et saute au bloc ayant appelé le bloc en cours. Le programme se poursuit avec la première instruction suivant l'appel du bloc. La zone de données locales en cours est libérée et la zone de données locales précédentes redevient la zone de données locales en cours. Les blocs de données qui étaient ouverts au moment de l'appel sont à nouveau ouverts. La dépendance par rapport au MCR du bloc appelant est restaurée.

Le RLG (égal à 1) est reporté du bloc qui s'est achevé dans le bloc appelant. Si le RLG égale 0, l'opération BEB ne s'exécute pas. Le RLG est mis à 1 et le programme se poursuit avec l'instruction suivante.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	x	0	1	1	0

Exemple

```
LIST              Explication
U       E 1.0     //Actualiser le RLG.
BEB               //Terminer le bloc si le RLG égale 1.
L       EW4       //Poursuivre ici si l'instruction BEB ne s'exécute pas (RLG égal à 0).
T       MW10
```

10.4 BEA Fin de bloc inconditionnelle

Format

BEA

Description de l'opération

BEA (Fin de bloc inconditionnelle)

Cette opération interrompt la séquence normale de votre programme dans le bloc en cours et saute au bloc ayant appelé le bloc en cours. Le programme se poursuit avec la première instruction suivant l'appel du bloc. La zone de données locales en cours est libérée et la zone de données locales précédentes redevient la zone de données locales en cours. Les blocs de données qui étaient ouverts au moment de l'appel sont à nouveau ouverts. De plus, la dépendance par rapport au MCR du bloc appelant est restaurée et le RLG est transféré du bloc en cours au bloc appelant. L'opération BEA prend effet sans condition. Si l'opération BEA est sautée, le déroulement de votre programme ne s'achève pas, mais se poursuit à la destination de saut, à l'intérieur du bloc.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple

```
LIST                Explication
       U      E 1.0
       SPB    SUIV   //Saut au repère de saut SUIV si le résultat logique RLG égale 1
                     //(E 1.0 = 1).
       L      EW4    //Poursuivre ici si le saut ne s'exécute pas.
       T      EW10
       U      E 6.0
       U      E 6.1
       S      M
              12.0
       BEA           //Fin de bloc inconditionnelle.
SUIV:  NOP 0         //Poursuivre ici si le saut s'exécute.
```

10.5 CALL Appel de bloc

Format

CALL <ID du bloc de code>

Description de l'opération

CALL <ID du bloc de code>

Cette opération permet l'appel des fonctions (FC) et blocs fonctionnels (FB) ou celui des fonctions standard (SFC) et blocs fonctionnels standard (SFB) livrés par Siemens. CALL appelle la FC, le FB, la SFC ou le SFB que vous avez indiqué en opérande, et ce indépendamment du RLG ou de toute autre condition. Si vous appelez un FB ou un SFB à l'aide de CALL, vous devez préciser un bloc de données d'instance. Une fois le bloc appelé traité, le programme se poursuit dans le bloc appelant. Vous pouvez indiquer l'identificateur de bloc sous forme absolue ou symbolique. Les contenus des registres sont restaurés après un appel de SFB/SFC.

Exemple : CALL FB1, DB1 ou CALL REMPLIR1, SUBST1

Bloc de code	Type de bloc	Syntaxe pour l'appel (adressage absolu)
FC	Fonction	CALL FCn
SFC	Fonction système	CALL SFCn
FB	Bloc fonctionnel	CALL FBn1,DBn2
SFB	Bloc fonctionnel système	CALL SFBn1,DBn2

Nota

Si vous utilisez l'éditeur LIST, les indications (n, n1 ou n2) du tableau ci-dessus doivent se rapporter à des blocs corrects déjà présents. Vous devez aussi définir préalablement les mnémoniques.

Transfert de paramètres (à cet effet, travaillez en mode de traitement incrémental)

Le bloc appelant peut échanger des paramètres avec le bloc appelé à l'aide de la liste de variables. Cette liste est automatiquement complétée dans votre programme LIST si vous entrez une instruction CALL correcte.

Si vous appelez un FB, un SFB, une FC ou une SFC et si la table de déclaration des variables du bloc appelé comporte des déclarations de type IN, OUT et IN_OUT, ces variables seront ajoutées dans le programme du bloc appelant comme liste des paramètres formels.

Vous devez, lors de l'appel de FC et de SFC, affecter des paramètres effectifs du bloc de code appelant aux paramètres formels.

Lors de l'appel de FB et de SFB, vous devez uniquement indiquer les paramètres effectifs devant changer par rapport au dernier appel, car les paramètres effectifs sont sauvegardés dans le DB d'instance après le traitement du FB. Si le paramètre effectif est un DB, il faut toujours indiquer l'adresse absolue entière, par exemple DB1, DBW2.

Il est possible d'indiquer les paramètres IN comme constantes ou comme adresses absolues ou symboliques. Les paramètres OUT et IN_OUT doivent être indiqués comme adresses absolues ou symboliques. Veillez à ce que toutes les adresses et constantes soient compatibles avec les types de données transférés.

L'opération CALL sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données ouverts et le bit MA dans la pile des blocs. Elle désactive en outre la relation de dépendance par rapport au MCR et définit la zone de données locales du bloc qui doit être appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple 1 : Affectation de paramètres à l'appel de la fonction FC6

```
CALL    FC6
        Paramètre formel     Paramètre  effectif
        NO OF TOOL           := MW100
        TIME OUT             := MW110
        FOUND                := A 0.1
        ERROR                := A 100.0
```

Exemple 2 : Appel d'une SFC sans paramètre

```
LIST              Explication
CALL     SFC43    //Appeler SFC43 pour redémarrer la surveillance du temps (sans paramètre).
```

Exemple 3 : Appel du FB99 avec le bloc de données d'instance DB1

```
CALL    FB99,DB1
        Paramètre formel        Paramètre effectif
        MAX_RPM                 := #RPM1_MAX
        MIN_RPM                 := #RPM2
        MAX_POWER               := #POWER
        MAX_TEMP                := #TEMP
```

Exemple 4 : Appel du FB99 avec le bloc de données d'instance DB2

```
CALL    FB99,DB2
        Paramètre formel        Paramètre effectif
        MAX_RPM                 := #RPM3_MAX
        MIN_RPM                 := #RPM2
        MAX_POWER               := #POWER1
        MAX_TEMP                := #TEMP
```

Nota

Chaque appel de bloc fonctionnel ou de bloc fonctionnel système nécessite un bloc de données d'instance. Dans l'exemple ci-dessus, les blocs DB1 et DB2 doivent être présents avant l'appel.

10.6 Appeler FB

Format

CALL FB n1, DB n2

Description de l'opération

Cette opération permet l'appel des blocs fonctionnels (FB) que vous avez programmés. CALL appelle le FB que vous avez indiqué en opérande, et ce indépendamment du RLG ou de toute autre condition. Si vous appelez un FB à l'aide de CALL, vous devez préciser un bloc de données d'instance. Une fois le bloc appelé traité, le programme se poursuit dans le bloc appelant. Vous pouvez indiquer l'identificateur de bloc sous forme absolue ou symbolique.

Transfert de paramètres (à cet effet, travaillez en mode de traitement incrémental)

Le bloc appelant peut échanger des paramètres avec le bloc appelé à l'aide de la liste de variables. Cette liste est automatiquement complétée dans votre programme LIST si vous entrez une instruction CALL correcte.

Si vous appelez un FB dont la table de déclaration des variables comporte des déclarations de type IN, OUT et IN_OUT, ces variables seront ajoutées dans le programme du bloc appelant comme liste des paramètres formels.

Lors de l'appel de FB, vous devez uniquement indiquer les paramètres effectifs devant changer par rapport au dernier appel, car les paramètres effectifs sont sauvegardés dans le DB d'instance après le traitement du FB. Si le paramètre effectif est un DB, il faut toujours indiquer l'adresse absolue entière, par exemple DB1, DBW2.

Il est possible d'indiquer les paramètres IN comme constantes ou comme adresses absolues ou symboliques. Les paramètres OUT et IN_OUT doivent être indiqués comme adresses absolues ou symboliques. Veillez à ce que toutes les adresses et constantes soient compatibles avec les types de données transférés.

L'opération CALL sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données ouverts et le bit MA dans la pile des blocs. Elle désactive en outre la relation de dépendance par rapport au MCR et définit la zone de données locales du bloc qui doit être appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple 1 : Appel du FB99 avec le bloc de données d'instance DB1

```
CALL    FB99,DB1
        Paramètre formel     Paramètre  effectif
        MAX_RPM              := #RPM1_MAX
        MIN_RPM              := #RPM1
        MAX_POWER            := #POWER1
        MAX_TEMP             := #TEMP1
```

Exemple 2 : Appel du FB99 avec le bloc de données d'instance DB2

```
CALL    FB99,DB2
        Paramètre formel     Paramètre  effectif
        MAX_RPM              := #RPM2_MAX
        MIN_RPM              := #RPM2
        MAX_POWER            := #POWER2
        MAX_TEMP             := #TEMP2
```

Nota

Chaque appel de bloc fonctionnel nécessite un bloc de données d'instance. Dans l'exemple ci-dessus, les blocs DB1 et DB2 doivent être présents avant l'appel.

10.7 Appeler FC

Format

CALL FC n

Nota

Si vous utilisez l'éditeur LIST, l'indication (n) doit se rapporter à des blocs corrects déjà présents. Vous devez aussi définir préalablement les mnémoniques.

Description de l'opération

Cette opération permet l'appel de fonctions (FC). CALL appelle la FC que vous avez indiquée en opérande, et ce indépendamment du RLG ou de toute autre condition. Une fois le bloc appelé traité, le programme se poursuit dans le bloc appelant. Vous pouvez indiquer l'identificateur de bloc sous forme absolue ou symbolique.

Transfert de paramètres (à cet effet, travaillez en mode de traitement incrémental)

Le bloc appelant peut échanger des paramètres avec le bloc appelé à l'aide de la liste de variables. Cette liste est automatiquement complétée dans votre programme LIST si vous entrez une instruction CALL correcte.

Si vous appelez une FC dont la table de déclaration des variables comporte des déclarations de type IN, OUT et IN_OUT, ces variables seront ajoutées dans le programme du bloc appelant comme liste des paramètres formels.

Vous devez, lors de l'appel de FC, affecter des paramètres effectifs du bloc de code appelant aux paramètres formels.

Il est possible d'indiquer les paramètres IN comme constantes ou comme adresses absolues ou symboliques. Les paramètres OUT et IN_OUT doivent être indiqués comme adresses absolues ou symboliques. Veillez à ce que toutes les adresses et constantes soient compatibles avec les types de données transférés.

L'opération CALL sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données ouverts et le bit MA dans la pile des blocs. Elle désactive en outre la relation de dépendance par rapport au MCR et définit la zone de données locales du bloc qui doit être appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple : Affectation de paramètres à l'appel de la fonction FC6

```
CALL    FC6
        Paramètre formel     Paramètre  effectif
        NO OF TOOL           := MW100
        TIME OUT             := MW110
        FOUND                := A 0.1
        ERROR                := A 100.0
```

10.8 Appeler SFB

Format

CALL SFB n1, DB n2

Description de l'opération

Cette opération permet l'appel des blocs fonctionnels standard (SFB) livrés par Siemens. CALL appelle le SFB que vous avez indiqué en opérande, et ce indépendamment du RLG ou de toute autre condition. Si vous appelez un SFB à l'aide de CALL, vous devez préciser un bloc de données d'instance. Une fois le bloc appelé traité, le programme se poursuit dans le bloc appelant. Vous pouvez indiquer l'identificateur de bloc sous forme absolue ou symbolique.

Transfert de paramètres (à cet effet, travaillez en mode de traitement incrémental)

Le bloc appelant peut échanger des paramètres avec le bloc appelé à l'aide de la liste de variables. Cette liste est automatiquement complétée dans votre programme LIST si vous entrez une instruction CALL correcte.

Si vous appelez un FB dont la table de déclaration des variables comporte des déclarations de type IN, OUT et IN_OUT, ces variables seront ajoutées dans le programme du bloc appelant comme liste des paramètres formels.

Lors de l'appel de SFB, vous devez uniquement indiquer les paramètres effectifs devant changer par rapport au dernier appel, car les paramètres effectifs sont sauvegardés dans le DB d'instance après le traitement du SFB. Si le paramètre effectif est un DB, il faut toujours indiquer l'adresse absolue entière, par exemple DB1, DBW2.

Il est possible d'indiquer les paramètres IN comme constantes ou comme adresses absolues ou symboliques. Les paramètres OUT et IN_OUT doivent être indiqués comme adresses absolues ou symboliques. Veillez à ce que toutes les adresses et constantes soient compatibles avec les types de données transférés.

L'opération CALL sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données ouverts et le bit MA dans la pile des blocs. Elle désactive en outre la relation de dépendance par rapport au MCR et définit la zone de données locales du bloc qui doit être appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple

```
CALL      SFB4,DB4
          Paramètre formel       Paramètre  effectif
          IN:                    E0.1
          PT:                    T#20s
          Q:                     M0.0
          ET:                    MW10
```

Nota

Chaque appel de bloc fonctionnel nécessite un bloc de données d'instance. Dans l'exemple ci-dessus, les blocs SFB4 et DB4 doivent être présents avant l'appel.

10.9 Appeler SFC

Format

CALL SFC n

Nota

Si vous utilisez l'éditeur LIST, l'indication (n) doit se rapporter à des blocs corrects déjà présents. Vous devez aussi définir préalablement les mnémoniques.

Description de l'opération

Cette opération permet l'appel de fonctions standard (SFC) livrées par Siemens. CALL appelle la SFC que vous avez indiquée en opérande, et ce indépendamment du RLG ou de toute autre condition. Une fois le bloc appelé traité, le programme se poursuit dans le bloc appelant. Vous pouvez indiquer l'identificateur de bloc sous forme absolue ou symbolique.

Transfert de paramètres (à cet effet, travaillez en mode de traitement incrémental)

Le bloc appelant peut échanger des paramètres avec le bloc appelé à l'aide de la liste de variables. Cette liste est automatiquement complétée dans votre programme LIST si vous entrez une instruction CALL correcte.

Si vous appelez une FC dont la table de déclaration des variables comporte des déclarations de type IN, OUT et IN_OUT, ces variables seront ajoutées dans le programme du bloc appelant comme liste des paramètres formels.

Vous devez, lors de l'appel de SFC, affecter des paramètres effectifs du bloc de code appelant aux paramètres formels.

Il est possible d'indiquer les paramètres IN comme constantes ou comme adresses absolues ou symboliques. Les paramètres OUT et IN_OUT doivent être indiqués comme adresses absolues ou symboliques. Veillez à ce que toutes les adresses et constantes soient compatibles avec les types de données transférés.

L'opération CALL sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données ouverts et le bit MA dans la pile des blocs. Elle désactive en outre la relation de dépendance par rapport au MCR et définit la zone de données locales du bloc qui doit être appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple : Appel d'une SFC sans paramètre

```
LIST              Explication
CALL     SFC43    //Appeler SFC43 pour redémarrer la surveillance du temps (sans paramètre).
```

10.10 Appeler multi-instance

Format

CALL # nom-variable

Description de l'opération

Vous créez une multi-instance par la déclaration d'une variable statique de type de données "bloc fonctionnel". Seules les multi-instances déjà déclarées apparaissent dans le catalogue des éléments de programme.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	X	X	X

10.11 Appeler un bloc dans une bibliothèque

La liste des bibliothèques connues dans le gestionnaire de projets SIMATIC vous est proposée.

Vous pouvez y choisir des blocs

- qui sont intégrés dans le système d'exploitation de votre CPU (bibliothèque "Standard Library") ;
- que vous avez rangés vous-même dans des bibliothèques parce que vous avez l'intention de les utiliser plusieurs fois.

10.12 CC Appel de bloc conditionnel

Format

CC <ID du bloc de code>

Description de l'opération

CC <ID du bloc de code>

L'opération "Appel de bloc conditionnel" permet d'appeler, si le RLG égale 1, un bloc de code de type FC ou SFC sans paramètre. L'opération CC est quasiment identique à l'opération CALL à la différence qu'il n'est pas possible de transmettre des paramètres. L'opération sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données en cours ainsi que le bit MA dans la pile des blocs, désactive la dépendance par rapport au MCR, crée la zone de données locales du bloc qui doit être appelé et commence l'exécution du code appelé. Vous pouvez indiquer l'identificateur du bloc sous forme absolue ou symbolique.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	1	0

Exemple

```
LIST            Explication
U       E 2.0   //Interroger l'état de signal à l'entrée E 2.0.
CC      FC6     //Appeler la fonction FC6 si l'état de signal en E 2.0 égale 1.
U       M3.0    //S'exécute après retour de la fonction appelée (si E 2.0 = 1) ou juste après
                l'instruction U E 2.0 si E 2.0 égale 0.
```

Nota

Lorsque vous appelez un bloc fonctionnel (FB) ou un bloc fonctionnel système (SFB) avec l'opération CALL, vous devez préciser un bloc de données d'instance (n° de DB) dans l'instruction. L'utilisation d'une variable de type "BlockFB" ou "BlockFC" est interdite avec l'opération CC. Comme vous ne pouvez pas indiquer de bloc de données dans l'opérande de l'instruction pour un appel avec l'opération CC, vous ne pouvez utiliser cette opération que pour des blocs sans paramètres de bloc et données locales statiques.

Selon le réseau avec lequel vous travaillez, la fenêtre de programmation de blocs en CONT ou LIST génère en partie l'opération UC et en partie l'opération CC lors de la conversion du langage de programmation CONT (schéma à contacts) en langage de programmation LIST (liste d'instructions). Utilisez, en règle générale, l'opération CALL pour que des erreurs n'apparaissent pas dans les programmes que vous avez créés.

10.13 UC Appel de bloc inconditionnel

Format

UC <ID du bloc de code>

Description de l'opération

UC <ID du bloc de code>

L'opération "Appel de bloc inconditionnel" permet d'appeler un bloc de code de type FC ou SFC. L'opération UC ressemble à l'opération CALL à la différence qu'il n'est pas possible de transmettre des paramètres. L'opération sauvegarde l'adresse de retour (sélecteur et adresse relative), les sélecteurs des deux blocs de données en cours ainsi que le bit MA dans la pile des blocs. Elle désactive la relation de dépendance par rapport au MCR, crée la zone de données locales du bloc qui doit être appelé et commence l'exécution du code appelé.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	0	0	1	-	0

Exemple 1

```
LIST            Explication
UC      FC6     //Appeler la fonction FC6 (sans paramètre).
```

Exemple 2

```
LIST            Explication
UC      SFC43   //Appeler la fonction système SFC43 (sans paramètre).
```

Nota

Lorsque vous appelez un bloc fonctionnel (FB) ou un bloc fonctionnel système (SFB) avec l'opération **CALL**, vous devez préciser un bloc de données d'instance (n° de DB) dans l'instruction. L'utilisation d'une variable de type "BlockFB" ou "BlockFC" est interdite avec l'opération **UC**. Comme vous ne pouvez pas indiquer de bloc de données dans l'opérande de l'instruction pour un appel avec l'opération **UC**, vous ne pouvez utiliser cette opération que pour des blocs sans paramètres de bloc et données locales statiques.

Selon le réseau avec lequel vous travaillez, la fenêtre de programmation de blocs en CONT ou LIST génère en partie l'opération **UC** et en partie l'opération **CC** lors de la conversion du langage de programmation CONT (schéma à contacts) en langage de programmation LIST (liste d'instructions). Utilisez, en règle générale, l'opération **CALL** pour que des erreurs n'apparaissent pas dans les programmes que vous avez créés.

10.14 Relais de masquage (Master Control Relay, MCR)

Remarques importantes sur l'utilisation de la fonctionnalité MCR

Avertissement

Pour exclure tout risque de blessures ou de dégâts matériels, n'utilisez jamais le MCR pour remplacer un relais de masquage mécanique câblé servant à une fonction d'arrêt d'urgence.

Description

Le relais de masquage Master Control Relay (MCR) est utilisé dans les schémas à contacts de relais pour exciter et désexciter le flux d'énergie. Les opérations déclenchées par les opérations suivantes de combinaison sur bits et de transfert dépendent du MCR :

- **=** <bit>
- **S** <bit>
- **R** <bit>
- **T** <octet>, **T** <mot>, **T** <double mot>

L'opération **T** avec octet, mot ou double mot écrit un 0 en mémoire si le MCR égale 0. Les opérations **S** et **R** laissent la valeur existante inchangée. L'opération **=** écrit un 0 dans le bit adressé.

Réactions des opérations dépendant de l'état de signal du bit MCR

Etat de signal du MCR	**= <bit>**	**S <bit>, R <bit>**	**T <octet>, T <mot> T <double mot>**
0 (HORS FONCTION)	Ecrit 0. Imite un relais passant à l'état de repos en cas de perte de tension.	N'écrit rien. Imite un relais restant à l'état en vigueur en cas de perte de tension.	Ecrit 0. Imite un composant produisant une valeur de 0 en cas de perte de tension.
1 (EN FONCTION)	Exécution normale.	Exécution normale.	Exécution normale.

MCR(: Début de zone MCR et)MCR : Fin de zone MCR

Le MCR est commandé par une pile de un bit de large et de huit bits de profondeur. Le MCR est actif tant que toutes les huit entrées égalent 1. L'opération MCR(copie le bit RLG dans la pile MCR. L'opération)MCR annule la dernière entrée de la pile et met l'emplacement libéré à 1. Les opérations MCR(et)MCR doivent toujours être utilisées par paires. Une erreur MCR (MCRF) est signalée si plus de huit opérations MCR(se suivent ou si une opération)MCR doit être exécutée alors que la pile est vide.

MCRA : Activer la zone MCR et MCRD : Désactiver la zone MCR

Les opérations MCRA et MCRD doivent toujours être utilisées par paires. Les instructions entre MCRA et MCRD dépendent de l'état de signal du bit MCR. En revanche, les instructions en dehors de la séquence MCRA-MCRD ne dépendent pas de l'état de signal de ce bit.

Vous devez programmer la relation de dépendance au MCR des fonctions (FC) et des blocs fonctionnels (FB) dans les blocs eux-mêmes en utilisant l'opération MCRA dans le bloc appelé.

10.15 Remarques importantes sur l'utilisation de la fonctionnalité MCR

Attention avec les blocs dans lesquels le relais de masquage a été activé par l'instruction MCRA :

- Lorsque le relais de masquage (MCR) est hors fonction, la valeur 0 est écrite par toutes les affectation (T, =) dans les sections de programme entre **MCR(** et **)MCR** !
- Le MCR se trouve précisément hors fonction lorsque le RLG était égal à 0 avant une instruction **MCR(**.

Danger : arrêt de l'AP ou comportement indéfini de la durée d'exécution !

Pour les calculs d'adresses, le compilateur accède également en écriture aux données locales suivant les variables temporaires définies dans VAR_TEMP. De ce fait, les séquences d'instructions suivantes mettent l'AP à l'arrêt ou conduisent à des comportements indéfinis de la durée d'exécution :

Accès à des paramètres formels

- Accès à des composants de paramètres FC complexes de type STRUCT, UDT, ARRAY, STRING
- Accès à des composants de paramètres FB complexes de type STRUCT, UDT, ARRAY, STRING de la zone IN_OUT dans un bloc admettant les multi-instances (bloc de version 2).
- Accès aux paramètres d'un FB admettant les multi-instances (bloc de version 2) lorsque leur adresse est supérieure à 8180.0.
- L'accès à un paramètre de type BLOCK_DB dans un FB admettant les multi-instances (bloc de version 2) ouvre le DB 0. Les accès ultérieurs aux données mettent la CPU à l'arrêt. Pour TIMER, COUNTER, BLOCK_FC, BLOCK_FB se sont aussi toujours T 0, Z 0, FC 0 ou FB 0 qui sont utilisés.

Transmission des paramètres

Appels pour lesquels des paramètres sont transmis.

CONT/LOG

- Dans CONT ou LOG, les branches T et les connecteurs débutent par RLG = 0.

Remède

Séparez les instructions concernées de la dépendance par rapport au relais de masquage :

1er Désactivez le relais de masquage en utilisant l'instruction MCRD avant l'instruction ou le réseau concernés.

2e Activez le relais de masquage en utilisant l'instruction MCRA après l'instruction ou le réseau concernés.

10.16 MCR(Sauvegarder RLG dans pile MCR, début de zone MCR

Remarques importantes sur l'utilisation de la fonctionnalité MCR

Format

MCR(

Description de l'opération

MCR((Ouvrir une zone MCR)

Cette opération sauvegarde le RLG dans la pile MCR (Master Control Relay : relais de masquage) et ouvre une zone MCR. Une zone MCR comprend les instructions entre l'opération **MCR(** et l'opération **)MCR** correspondante. Il faut toujours utiliser les opérations **MCR(** et **)MCR** par paires.

Si le RLG égale 1, le relais de masquage est "en fonction". Les instructions dépendant du MCR à l'intérieur de cette zone MCR s'exécutent normalement.

Si le RLG égale 0, le relais de masquage est "hors fonction". Les instructions dépendant du MCR à l'intérieur de cette zone MCR s'exécutent comme indiqué dans le tableau ci-dessous.

Réactions des opérations dépendant de l'état de signal du bit MCR

Etat de signal du MCR	**= <bit>**	**S <bit>, R <bit>**	**T <octet>, T <mot> T <double mot>**
0 (HORS FONCTION)	Ecrit 0. Imite un relais passant à l'état de repos en cas de perte de tension.	N'écrit rien. Imite un relais restant à l'état en vigueur en cas de perte de tension.	Ecrit 0. Imite un composant produisant une valeur de 0 en cas de perte de tension.
1 (EN FONCTION)	Exécution normale.	Exécution normale.	Exécution normale.

Vous pouvez imbriquer les opérations MCR(et)MCR. La profondeur d'imbrication maximale est de huit opérations. La pile peut, par conséquent, contenir huit entrées au maximum. Si l'opération MCR(est exécutée à pile pleine, une erreur de pile MCR (MCRF) est alors signalée.

Mot d'état

	RB	**BI1**	**BI0**	**DEB**	**DM**	**OU**	**ETAT**	**RLG**	**/PI**
Ecriture :	-	-	-	-	-	0	1	-	0

Exemple

```
LIST                Explication
MCRA                //Activer la zone MCR.
U       E 1.0
MCR(                //Sauvegarder le RLG dans la pile MCR, ouvrir une zone MCR. Le MCR est en
                    //fonction si le RLG égale 1 (E 1.0 =  1). Le MCR est hors fonction si le RLG
                    //égale 0 (E 1.0 = 0).
U       E 4.0
=       A 8.0       //Si le MCR est hors fonction, la sortie A 8.0 est mise à 0 sans considérer
                    //l'état de signal à l'entrée E 4.0.
L       MW20
T       AW10        //Si le MCR est hors fonction, la valeur 0 est transférée dans AW10.
)MCR                //Mettre fin à la zone MCR.
MCRD                //Désactiver la zone MCR.
U       E 1.1
=       A 8.1       //Ces instructions sont à l'extérieur de la zone MCR et ne dépendent pas du
                    //bit MCR.
```

10.17)MCR Fin de zone MCR

Remarques importantes sur l'utilisation de la fonctionnalité MCR

Format

)MCR

Description de l'opération

)MCR (Fin de zone MCR)

Cette opération retire une entrée de la pile MCR et termine une zone MCR. La dernière entrée dans la pile MCR est libérée et mise à 1. Les opérations MCR(et)MCR doivent toujours être utilisées par paires. Si l'opération)MCR s'exécute si la pile est vide, une erreur de pile MCR (MCRF) est signalée.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	1	-	0

Exemple

```
LIST              Explication
MCRA              //Activer la zone MCR.
U       E 1.0
MCR(              //Sauvegarder le RLG dans la pile MCR, ouvrir une zone MCR. Le MCR est en
                  //fonction si le RLG égale 1 (E 1.0 = 1). Le MCR est hors fonction si le RLG
                  //égale 0 (E 1.0 = 0).
U       E 4.0
=       A 8.0     //Si le MCR est hors fonction, la sortie A 8.0 est mise à 0 sans considérer
                  //l'état de signal à l'entrée E 4.0.
L       MW20
T       AW10      //Si le MCR est hors fonction, la valeur 0 est transférée dans AW10.
)MCR              //Mettre fin à la zone MCR.
MCRD              //Désactiver la zone MCR.
U       E 1.1
=       A 8.1     //Ces instructions sont à l'extérieur de la zone MCR et ne dépendent pas du
                  //bit MCR.
```

10.18 MCRA Activer la zone MCR

Remarques importantes sur l'utilisation de la fonctionnalité MCR

Format

MCRA

Description de l'opération

MCRA (Activer le relais de masquage)

Cette opération permet d'activer la dépendance par rapport au MCR des instructions qui suivent cette opération. Les opérations MCRA et MCRD (désactivation du relais de masquage) doivent toujours être utilisées par paires. Les instructions figurant entre MCRA et MCRD dépendent de l'état de signal du bit MCR.

L'opération s'exécute sans tenir compte des bits d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
MCRA              //Activer la zone MCR.
U       E 1.0
MCR(              //Sauvegarder le RLG dans la pile MCR, ouvrir une zone MCR. Le MCR est en
                  //fonction si le RLG égale 1 (E 1.0 = 1). Le MCR est hors fonction si le RLG
                  //égale 0 (E 1.0 = 0).
U       E 4.0
=       A 8.0     //Si le MCR est hors fonction, la sortie A 8.0 est mise à 0 sans considérer
                  //l'état de signal à l'entrée E 4.0.
L       MW20
T       AW10      //Si le MCR est hors fonction, la valeur 0 est transférée dans AW10.
)MCR              //Mettre fin à la zone MCR.
MCRD              //Désactiver la zone MCR.
U       E 1.1
=       A 8.1     //Ces instructions sont à l'extérieur de la zone MCR et ne dépendent pas du
                  //bit MCR.
```

10.19 MCRD Désactiver la zone MCR

Remarques importantes sur l'utilisation de la fonctionnalité MCR

Format

MCRD

Description de l'opération

MCRD (Désactiver le relais de masquage)

Cette opération permet de désactiver la dépendance par rapport au MCR des instructions qui suivent cette opération. Les opérations MCRA et MCRD (activation du relais de masquage) doivent toujours être utilisées par paires. Les instructions figurant entre MCRA et MCRD dépendent de l'état de signal du bit MCR.

L'opération s'exécute sans tenir compte des bits d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
MCRA              //Activer la zone MCR.
U       E 1.0
MCR(              //Sauvegarder le RLG dans la pile MCR, ouvrir une zone MCR. Le MCR est en
                  //fonction si le RLG égale 1 (E 1.0 = 1). Le MCR est hors fonction si le RLG
                  //égale 0 (E 1.0 = 0).
U       E 4.0
=       A 8.0     //Si le MCR est hors fonction, la sortie A 8.0 est mise à 0 sans considérer
                  //l'état de signal à l'entrée E 4.0.
L       MW20
T       AW10      //Si le MCR est hors fonction, la valeur 0 est transférée dans AW10.
)MCR              //Mettre fin à la zone MCR.
MCRD              //Désactiver la zone MCR.
U       E 1.1
=       A 8.1     //Ces instructions sont à l'extérieur de la zone MCR et ne dépendent pas du
                  //bit MCR.
```

11 Opérations de décalage et de rotation

11.1 Opérations de décalage

11.1.1 Vue d'ensemble des opérations de décalage

Description

Les opérations de décalage permettent de décaler bit par bit le contenu du mot de poids faible de l'accumulateur 1 ou de l'accumulateur entier vers la gauche ou vers la droite (voir aussi Registres de la CPU). Le décalage vers la gauche multiplie le contenu de l'accumulateur par des puissances de 2 ; le décalage vers la droite le divise par des puissances de 2. Si, par exemple, vous décalez de 3 bits vers la gauche l'équivalent binaire de la valeur décimale 3, vous obtenez l'équivalent binaire de la valeur décimale 24 dans l'accumulateur. Si vous décalez de 2 bits vers la droite l'équivalent binaire de la valeur décimale 16, vous obtenez l'équivalent binaire de la valeur décimale 4 dans l'accumulateur.

Le nombre de bits de décalage est précisé dans l'instruction de décalage même ou est pris dans l'octet de poids faible du mot de poids faible de l'accumulateur 2. Les positions binaires libérées par l'opération de décalage sont soit remplies par des zéros, soit par l'état de signal du bit de signe (0 signifie positif et 1 négatif). Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Les bits BI0 et DEB du mot d'état sont remis à 0. Vous pouvez évaluer le bit BI1 à l'aide d'opérations de saut.

Les opérations de décalage sont inconditionnelles : leur exécution ne dépend d'aucune condition spéciale. Elles n'affectent pas le résultat logique RLG.

Vous disposez des opérations de décalage suivantes :

- SSI Décalage vers la droite d'un entier avec signe (16 bits)
- SSD Décalage vers la droite d'un entier avec signe (32 bits)
- SLW Décalage vers la gauche d'un mot (16 bits)
- SRW Décalage vers la droite d'un mot (16 bits)
- SLD Décalage vers la gauche d'un double mot (32 bits)
- SRD Décalage vers la droite d'un double mot (32 bits)

11.1.2 SSI Décalage vers la droite d'un entier avec signe (16 bits)

Formats

SSI
SSI <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 15

Description de l'opération

SSI (Décalage vers la droite d'un entier avec signe)

Cette opération permet de décaler bit par bit le contenu de l'accumulateur 1-L vers la droite. L'état de signal du bit de signe (bit 15) est écrit dans les bits libérés par le décalage. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L.

SSI <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 15 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SSI : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 16 donne toujours le même résultat : accumulateur 1 = 16#0000, BI1 = 0 ou accumulateur 1 = 16#FFFF, BI1 = 1. Si le nombre de bits de décalage est supérieur à 0, les bits d'état BI0 et DEB sont mis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SSI 6**	0101	1111	0110	0100	1001	1101	0011	1011
après exécution de **SSI 6**	0101	1111	0110	0100	1111	1110	0111	0100

Exemple 1

```
LIST            Explication
L     MW4       //Charger la valeur de MW4 dans l'accumulateur 1.
SSI   6         //Décaler les bits dans l'accumulateur 1, signe inclus, de 6 positions vers
                //la droite.
T     MW8       //Transférer le résultat dans le mot de mémento MW8.
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MW20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du mot de mémento MW20 dans l'accumulateur  1.
SSI             //Le nombre pour le décalage égale la valeur de l'accumulateur 2-L-L => décaler
                //les bits dans l'accumulateur 1-L, signe inclus, de 3 positions vers la
                //droite, mettre les positions libérées à l'état de signal du bit de signe.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.1.3 SSD Décalage vers la droite d'un entier avec signe (32 bits)

Formats

SSD
SSD <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 32

Description de l'opération

SSD (Décalage vers la droite d'un entier avec signe, 32 bits)

Cette opération permet de décaler bit par bit le contenu entier de l'accumulateur 1 vers la droite. L'état de signal du bit de signe est écrit dans les bits libérés par le décalage. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2- L-L.

SSD <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 32 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SSD : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 32 donne toujours le même résultat : accumulateur 1 = 32#00000000, BI1 = 0 ou accumulateur 1 = 32#FFFFFFFF, BI1 = 1. Si le nombre de bits de décalage est supérieur à 0, les bits d'état BI0 et DEB sont mis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SSD 7**	1000	1111	0110	0100	0101	1101	0011	1011
après exécution de **SSD 7**	1111	1111	0001	1110	1100	1000	1011	1010

Exemple 1

```
LIST              Explication
L      MD4        //Charger la valeur de MD4 dans l'accumulateur 1.
SSD    7          //Décaler les bits dans l'accumulateur 1, signe inclus, de 7 positions vers
                  //la droite.
T      MD8        //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST              Explication
L      +3         //Charger la valeur +3 dans l'accumulateur 1.
L      MD20       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur du double mot de mémento MD20 dans l'accumulateur 1.
SSD               //Le nombre pour le décalage égale la valeur de l'accumulateur 2-L-L => décaler
                  //les bits dans l'accumulateur 1, signe inclus, de 3 positions vers la droite,
                  //mettre les positions libérées à l'état de signal du bit de signe.
SPP    SUIV       //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.1.4 SLW Décalage vers la gauche d'un mot (16 bits)

Formats

SLW

SLW <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 15

Description de l'opération

SLW (Décalage vers la gauche d'un mot)

Cette opération permet de décaler bit par bit le contenu de l'accumulateur 1-L vers la gauche. Les positions binaires libérées par le décalage sont complétées par des zéros. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L.

SLW <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 15 sont autorisées. Les bits d'état BI0 et DEB sont remis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SLW : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 16 donne toujours le même résultat : accumulateur 1-L = 0, BI1 = 0, BI0 = 0, DEB = 0. Si 0 < nombre de bits décalés <= 16, les bits d'état BI0 et DEB sont mis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SLW 5**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **SLW 5**	0101	1111	0110	0100	1010	0111	0110	0000

Exemple 1

```
LIST            Explication
L     MW4       //Charger la valeur de MW4 dans l'accumulateur 1.
SLW   5         //Décaler les bits dans l'accumulateur 1 de 5 positions vers la gauche.
T     MW8       //Transférer le résultat dans le mot de mémento MW8.
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MW20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du mot de mémento MW20 dans l'accumulateur  1.
SLW             //Le nombre pour le décalage égale la valeur de l'accumulateur 2-L-L => décaler
                //les bits dans l'accumulateur 1-L de 3 positions vers la gauche.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.1.5 SRW Décalage vers la droite d'un mot (16 bits)

Formats

SRW
SRW <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 15

Description de l'opération

SRW (Décalage vers la droite d'un mot)

Cette opération permet de décaler bit par bit le contenu de l'accumulateur 1-L vers la droite. Les positions binaires libérées par le décalage sont complétées par des zéros. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2 L-L.

SRW <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 15 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SRW : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 16 donne toujours le même résultat : accumulateur 1-L = 0, BI1 = 0, BI0 = 0, DEB = 0. Si 0 < nombre de bits de décalage <= 16, les bits d'état BI0 et DEB sont mis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SRW 6**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **SRW 6**	0101	1111	0110	0100	0000	0001	0111	0100

Exemple 1

```
LIST            Explication
L     MW4       //Charger la valeur de MW4 dans l'accumulateur 1.
SRW   6         //Décaler les bits dans l'accumulateur 1 de 6 positions vers la droite.
T     MW8       //Transférer le résultat dans le mot de mémento MW8
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MW20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du mot de mémento MW20 dans l'accumulateur 1.
SRW             //Le nombre pour le décalage égale la valeur de l'accumulateur 2-L-L => décaler
                //les bits dans l'accumulateur 1-L de 3 positions vers la droite.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.1.6 SLD Décalage vers la gauche d'un double mot (32 bits)

Formats

SLD
SLD <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 32

Description de l'opération

SLD (Décalage vers la gauche d'un double mot)

Cette opération permet de décaler bit par bit le contenu entier de l'accumulateur 1 vers la gauche. Les positions binaires libérées par le décalage sont complétées par des zéros. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits à décaler est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L.

SLD <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 32 sont autorisées. Si <nombre> est supérieur à 0, les bits BI0 et DEB sont mis à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SLD : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 32 donne toujours le même résultat : accumulateur 1 = 0, BI1 = 0, BI0 = 0, DEB = 0. Si 0 < nombre de bits de décalage <= 32, les bits d'état BI0 et DEB sont mis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SLD 5**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **SLD 5**	1110	1100	1000	1011	1010	0111	0110	0000

Exemple 1

```
LIST            Explication
L     MD4       //Charger la valeur de MD4 dans l'accumulateur 1.
SLD   5         //Décaler les bits dans l'accumulateur 1 de 5 positions vers la gauche.
T     MD8       //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MD20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du double mot de mémento MD20 dans l'accumulateur 1.
SLD             //Le nombre pour le décalage égale la valeur de l'accumulateur 2-L-L => décaler
                //les bits dans l'accumulateur 1 de 3 positions vers la gauche.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.1.7 SRD Décalage vers la droite d'un double mot (32 bits)

Formats

SRD
SRD <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits à décaler ; plage de décalage de 0 à 32

Description de l'opération

SRD (Décalage vers la droite d'un double mot)

Cette opération permet de décaler bit par bit le contenu entier de l'accumulateur 1 vers la droite. Les positions binaires libérées par le décalage sont complétées par des zéros. Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L.

SRD <nombre> : Le nombre de bits de décalage est précisé par l'opérande <nombre>. Des valeurs entre 0 et 32 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

SRD : Le nombre de bits de décalage est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Un nombre de bits de décalage supérieur à 32 donne toujours le même résultat : accumulateur 1 = 0, BI1 = 0, BI0 = 0, DEB = 0. Si 0 < nombre de bits de décalage <= 32, les bits BI0 et DEB sont remis à 0. Si le nombre de bits de décalage égale 0, l'opération de décalage s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **SRD 7**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **SRD 7**	0000	0000	1011	1110	1100	1000	1011	1010

Exemple 1

```
LIST            Explication
L     MD4       //Charger la valeur de MD4 dans l'accumulateur 1.
SRD   7         //Décalage des bits dans l'accumulateur 1 de 7 positions vers la droite.
T     MD8       //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MD20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du double mot de mémento MD20 dans l'accumulateur 1.
SRD             //Le nombre pour le décalage égale la valeur de l'accumulateur  2-L-L =>
                //décaler les bits dans l'accumulateur 1 de 3 positions vers la droite.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit décalé (BI1) égale 1.
```

11.2 Opérations de rotation

11.2.1 Vue d'ensemble des opérations de rotation

Description

Les opérations de rotation permettent d'effectuer la rotation bit par bit vers la gauche ou vers la droite du contenu entier de l'accumulateur 1 (voir aussi Registres de la CPU). Ces opérations déclenchent des fonctions similaires aux opérations de décalage. Toutefois, les positions binaires libérées sont remplies avec l'état de signal des bits qui ont été décalés hors de l'accumulateur.

Le nombre de bits de rotation est précisé dans l'instruction de rotation même ou est pris dans l'octet de poids faible du mot de poids faible de l'accumulateur 2. Selon l'opération, la rotation s'effectue via le bit BI1 du mot d'état. Le bit BI0 du mot d'état est remis à 0.

Vous disposez des opérations de rotation suivantes :

- RLD Rotation vers la gauche d'un double mot (32 bits)
- RRD Rotation vers la droite d'un double mot (32 bits)
- RLDA Rotation vers la gauche de l'accumulateur 1 via BI1 (32 bits)
- RRDA Rotation vers la droite de l'accumulateur 1 via BI1 (32 bits)

11.2.2 RLD Rotation vers la gauche d'un double mot (32 bits)

Formats

RLD
RLD <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits objet de la rotation ; plage de rotation de 0 à 32

Description de l'opération

RLD (Rotation vers la gauche d'un double mot)

Cette opération permet de déplacer bit par bit le contenu entier de l'accumulateur vers la gauche. Les bits libérés par la rotation sont remplis avec l'état de signal des bits qui ont été décalés hors de l'accumulateur 1. Le bit décalé en dernier est mémorisé dans le bit BI1 du mot d'état. Le nombre de positions binaires objet de la rotation est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L.

RLD <nombre> : Le nombre de bits objet de la rotation est précisé par l'opérande <nombre>. Des valeurs entre 0 et 32 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si <nombre> est supérieur à zéro. Si <nombre> égale zéro, l'opération de rotation s'exécute comme une opération nulle (NOP).

RLD : Le nombre de bits objet de la rotation est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Les bits d'état BI0 et DEB sont mis à 0 si le contenu de l'accumulateur 2-L-L est supérieur à zéro. Si le nombre de bits objet de la rotation égale 0, l'opération de rotation s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **RLD 4**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **RLD 4**	1111	0110	0100	0101	1101	0011	1011	0101

Exemple 1

```
LIST              Explication
L      MD2        //Charger la valeur de MD2 dans l'accumulateur 1.
RLD    4          //Rotation des bits dans l'accumulateur 1 de 4 positions vers la gauche.
T      MD8        //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST              Explication
L      +3         //Charger la valeur +3 dans l'accumulateur 1.
L      MD20       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                  //valeur du double mot de mémento MD20 dans l'accumulateur 1.
RLD               //Le nombre pour la rotation égale la valeur de l'accumulateur 2-L-L =>
                  //effectuer la rotation des bits dans l'accumulateur 1 de 3 positions vers
                  //la gauche.
SPP    SUIV       //Sauter au repère de saut SUIV si le dernier bit objet de la rotation (BI1)
                  //égale 1.
```

11.2.3 RRD Rotation vers la droite d'un double mot (32 bits)

Formats

RRD
RRD <nombre>

Opérande	Type de données	Description
<nombre>	Entier non signé	Nombre de bits objet de la rotation ; plage de rotation de 0 à 32

Description de l'opération

RRD (Rotation vers la droite d'un double mot)

Cette opération permet de déplacer bit par bit le contenu entier de l'accumulateur vers la droite. Les bits libérés par la rotation sont remplis avec l'état de signal des bits qui ont été décalés hors de l'accumulateur 1. Le bit déplacé en dernier est mémorisé dans le bit BI1 du mot d'état. Le nombre de bits objet de la rotation est précisé soit par l'opérande, soit par une valeur figurant dans l'accumulateur 2-L-L.

RRD <nombre> : Le nombre de bits objet de la rotation est précisé par l'opérande <nombre>. Des valeurs entre 0 et 32 sont autorisées. Les bits BI0 et DEB sont mis à 0 si <nombre> est supérieur à 0. Si <nombre> égale 0, l'opération de décalage s'exécute comme une opération NOP.

RRD : Le nombre de bits objet de la rotation est précisé par la valeur figurant dans l'accumulateur 2-L-L. Des valeurs entre 0 et 255 sont autorisées. Si le contenu de l'accumulateur 2-L-L est supérieur à 0, les bits BI0 et DEB sont mis à 0. Si le nombre de bits de rotation égale 0, l'opération de rotation s'exécute comme une opération NOP.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	x	x	-	-	-	-	-

Exemples

Contenu	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **RRD 4**	0101	1111	0110	0100	0101	1101	0011	1011
après exécution de **RRD 4**	1011	0101	1111	0110	0100	0101	1101	0011

Exemple 1

```
LIST            Explication
L     MD2       //Charger la valeur de MD2 dans l'accumulateur 1.
RRD   4         //Effectuer la rotation des bits dans l'accumulateur 1 de 4 positions vers
                la droite.
T     MD8       //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST            Explication
L     +3        //Charger la valeur +3 dans l'accumulateur 1.
L     MD20      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger la
                //valeur du double mot de mémento MD20 dans l'accumulateur 1.
RRD             //Le nombre pour la rotation égale la valeur de l'accumulateur 2-L-L =>
                //effectuer la rotation des bits dans l'accumulateur 1 de 3 positions vers
                //la droite.
SPP   SUIV      //Sauter au repère de saut SUIV si le dernier bit objet de la rotation (BI1)
                //égale 1.
```

11.2.4 RLDA Rotation vers la gauche de l'accumulateur 1 via BI1 (32 bits)

Format

RLDA

Description de l'opération

RLDA (Rotation vers la gauche d'un double mot via BI1)

Cette opération permet de déplacer le contenu entier de l'accumulateur 1 d'un bit vers la gauche via le bit indicateur BI1. Les bits BI0 et DEB sont remis à 0.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Contenu	BI1	ACCU1-H				ACCU1-L			
Bit		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **RLDA**	**X**	**0**101	1111	0110	0100	0101	1101	0011	1011
après exécution de **RLDA**	**0**	1011	1110	1100	1000	1011	1010	0111	011**X**
	(X égale 0 ou 1, ancien état de signal de BI1)								

```
LIST              Explication
L      MD2        //Charger la valeur de MD2 dans l'accumulateur 1.
RLDA              //Effectuer la rotation des bits dans l'accumulateur 1 d'une position vers la
                  //gauche via BI1.
SPP    SUIV       //Sauter au repère de saut SUIV si le dernier bit objet de la rotation (BI1)
                  //égale 1.
```

11.2.5 RRDA Rotation vers la droite de l'accumulateur 1 via BI1 (32 bits)

Format

RRDA

Description de l'opération

RRDA (Rotation vers la droite d'un double mot via BI1)

Cette opération permet de déplacer le contenu entier de l'accumulateur 1 d'un bit vers la droite via le bit indicateur BI1. Les bits BI0 et DEB sont remis à 0.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Contenu	BI1	ACCU1-H				ACCU1-L			
Bit		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
avant exécution de **RRDA**	**X**	0101	1111	0110	0100	0101	1101	0011	101**1**
après exécution de **RRDA**	**1**	**X**010	1111	1011	0010	0010	1110	1001	1101
	(X égale 0 ou 1, ancien état de signal de BI1)								

```
LIST              Explication
L      MD2        //Charger la valeur de MD2 dans l'accumulateur 1.
RRDA              //Effectuer la rotation des bits dans l'accumulateur 1 d'une position vers
                  //la droite via BI1.
SPP    SUIV       //Sauter au repère de saut SUIV si le dernier bit objet de la rotation (BI1)
                  //égale 1.
```

12 Opérations de temporisation

12.1 Vue d'ensemble des opérations de temporisation

Description

Vous trouverez des informations sur le réglage et la sélection de la bonne temporisation sous "Adresse d´une temporisation en mémoire et composants d´une temporisation".

Le langage de programmation LIST du logiciel de programmation STEP 7 met les opérations de temporisation suivantes à votre disposition :

- FR Valider temporisation
- L Charger valeur de temps en cours comme nombre entier dans l'accumulateur 1
- LC Charger valeur de temps en cours comme nombre DCB dans l'accumulateur 1
- R Remettre temporisation à 0
- SI Temporisation sous forme d'impulsion
- SV Temporisation sous forme d'impulsion prolongée
- SE Temporisation sous forme de retard à la montée
- SS Temporisation sous forme de retard à la montée mémorisé
- SA Temporisation sous forme de retard à la retombée

12.2 Adresse d´une temporisation en mémoire et composants d´une temporisation

Zone de mémoire

Une zone de mémoire est réservée aux temporisations dans votre CPU. Un mot de 16 bits y est réservé pour chaque opérande de temporisation. La programmation avec LIST permet d'utiliser jusqu'à 256 temporisations. Le nombre de mots de temporisation disponibles dans votre CPU figure dans les caractéristiques de la CPU.

Les fonctions suivantes ont accès à la zone de mémoire réservée aux temporisations :

- opérations de temporisation,
- actualisation des mots de temporisation avec une horloge. Cette fonction décrémente, à l'état de marche (RUN) de la CPU, une valeur donnée d'une unité dans un intervalle défini par la base de temps, et ce, jusqu'à ce que la valeur de temps soit égale à zéro.

Valeur de temps

La valeur de temps est contenue sous forme binaire dans les bits 0 à 9 du mot de temporisation. Elle détermine un nombre d'unités. L'actualisation de l'heure décrémente la valeur de temps d'une unité dans un intervalle défini par la base de temps. La décrémentation se poursuit jusqu'à ce que la valeur de temps soit égale à zéro. Pour charger une valeur de temps, vous pouvez utiliser le format binaire, hexadécimal ou décimal codé binaire (DCB). La plage de temps est comprise entre 0 et 9 990 secondes.

Vous pouvez charger une valeur de temps prédéfinie en utilisant l'un des deux formats suivants :

- w#16#txyz où

  t = base de temps (c'est-à-dire l'intervalle de temps ou la résolution)

  xyz = valeur de temps en format décimal codé binaire (DCB)

- S5T#a**H**_b**M**_c**S**_d**MS**

  H (heures), M (minutes), S (secondes) et MS (millisecondes) ;
  a, b, c, d sont definies par l'utilisateur

  la base de temps est choisie automatiquement et la valeur est arrondie au nombre inférieur le plus proche avec cette base de temps.

La valeur de temps maximale que vous pouvez indiquer est égale à 9 990 secondes ou 2H_46M_30S.
Exemples :

S5TIME#4S = 4 secondes
s5t#2h_15m = 2 heures et 15 minutes
S5T#1H_12M_18S = 1 heure, 12 minutes et 18 secondes

Base de temps

La base de temps est contenue en code binaire dans les bits 12 et 13 du mot de temporisation. Elle détermine à quel intervalle la valeur de temps va être décrémentée. La base de temps minimale est égale à 10 ms ; la base de temps maximale à 10 s.

Base	Code binaire de la base de temps
10 ms	00
100 ms	01
1 s	10
10 s	11

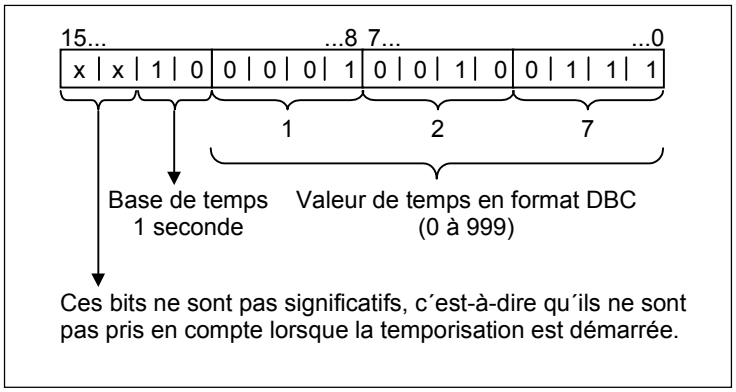
Les valeurs ne doivent pas excéder 2H_46M_30S. Les valeurs présentant une plage trop importante ou une trop grande résolution sont arrondies en fonction des valeurs limites de plages et de résolutions. Le format général S5TIME a les valeurs limites suivantes pour la plage et la résolution :

Résolution	Plage
0,01 seconde	10MS à 9S_990MS
0,1 seconde	100MS à 1M_39S_900MS
1 seconde	1S à 16M_39S
10 secondes	10S à 2H_46M_30S

Configuration des bits dans la cellule de temporisation

Lorsqu'une temporisation est démarrée, le contenu de la cellule de temporisation est utilisé comme valeur de temps. Les bits 0 à 11 de la cellule de temporisation contiennent la valeur de temps en format décimal codé binaire (format DCB : chaque groupe de quatre bits contient le code binaire d'une valeur décimale). Les bits 12 et 13 contiennent la base de temps en code binaire.

La figure suivante montre le contenu de la cellule de temporisation dans laquelle vous avez chargé la valeur de temps 127 et une base de temps de 1 seconde.

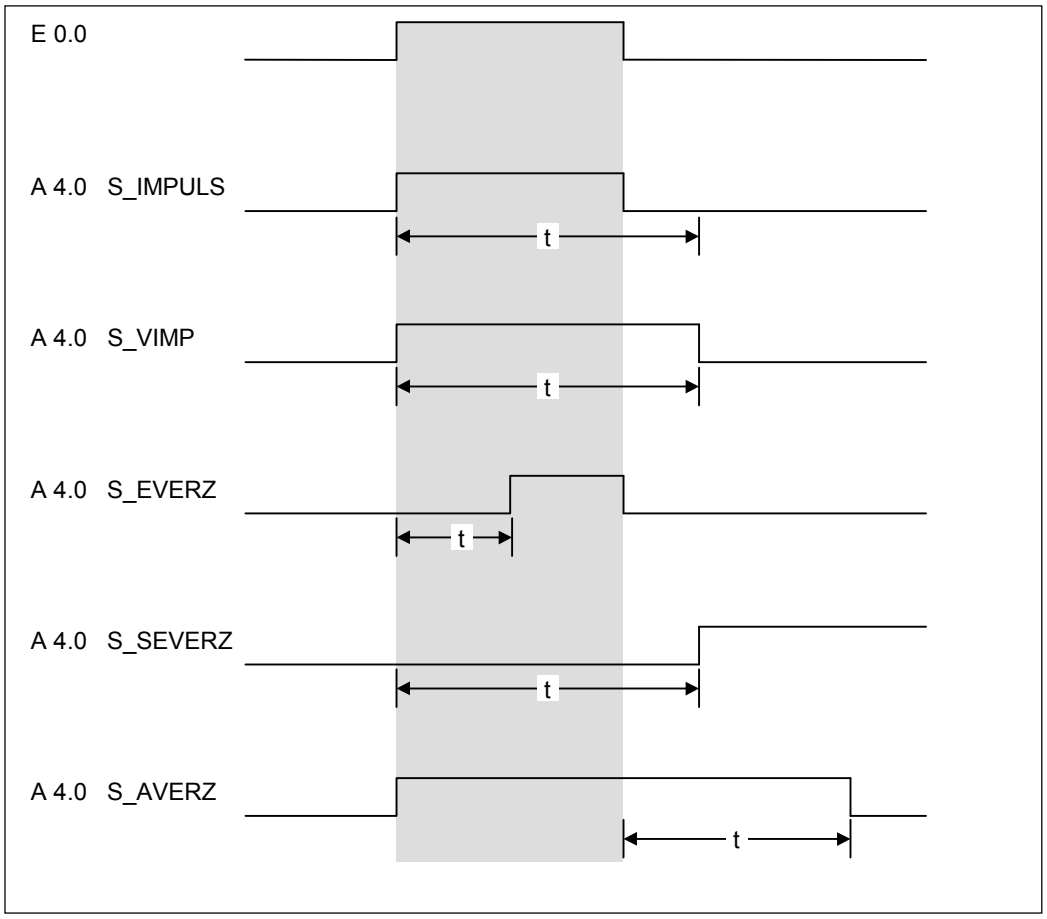


Lecture de la valeur et de la base de temps

Chaque boîte de temporisation possède deux sorties, DUAL (BI) et DEZ (BCD), pour lesquelles vous pouvez indiquer une adresse de mot. La sortie DUAL fournit la valeur de temps en format binaire. La sortie DEZ fournit la base de temps et la valeur de temps en format décimal codé binaire (DCB).

Choix de la temporisation correcte

La vue d'ensemble des cinq types de temporisations doit vous aider à choisir la temporisation qui répond le mieux à vos besoins.



Temporisations	Description
S_IMPULS temporisation sous forme d´impulsion	La durée maximale pendant laquelle le signal de sortie reste à 1 est la même que la valeur de temps « t » programmée. Le signal de sortie reste à 1 pour une durée plus courte si le signal d'entrée passe à 0.
S_VIMP temporisation sous forme d´impulsion prolongée	Le signal de sortie reste à 1 pendant la durée programmée, quelle que soit la durée pendant laquelle le signal d'entrée reste à 1.
S_EVERZ temporisation sous forme de retard à la montée	Le signal de sortie est égal à 1 uniquement lorsque le temps programmé s'est écoulé et que le signal d'entrée est toujours à 1.
S_SEVERZ temporisation sous forme de retard à la montée mémorisé	Le signal de sortie passe de 0 à 1 uniquement lorsque le temps programmé s'est écoulé, quelle que soit la durée pendant laquelle le signal d'entrée reste à 1.
S_AVERZ temporisation sous forme de retard à la retombée	Le signal de sortie est égal à 1 lorsque le signal d'entrée est égal à 1 ou lorsque la temporisation s'exécute. La temporisation est démarrée lorsque le signal d'entrée passe de 1 à 0.

12.3 FR Valider temporisation

Format

FR <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

FR <temporisation>

Cette opération efface le mémento de front utilisé pour démarrer la temporisation indiquée si le RLG passe de 0 à 1. Le passage du résultat logique RLG de 0 à 1 avant une opération de validation de temporisation FR valide une temporisation.

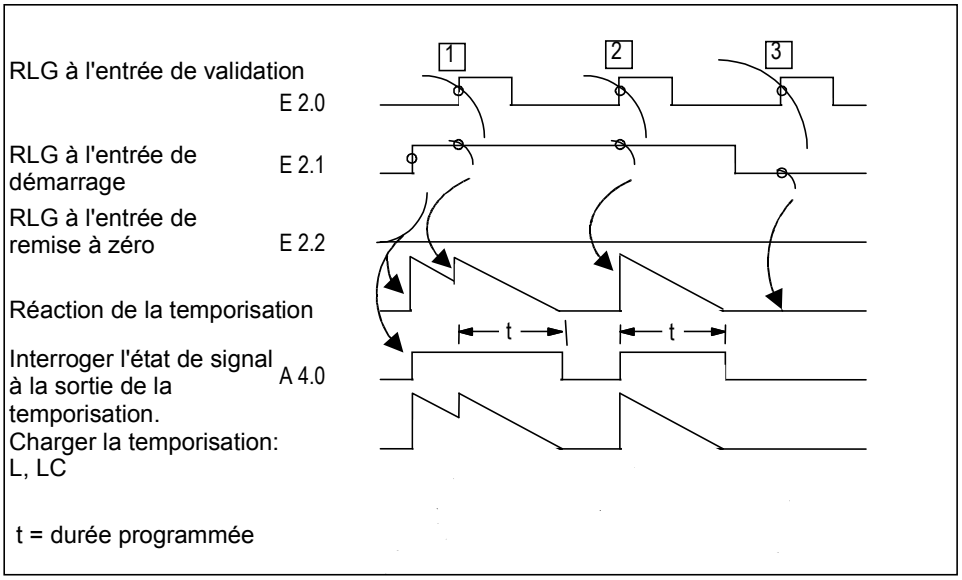
L'opération **Valider temporisation** n'est pas obligatoire pour démarrer une temporisation ni pour les temporisations normales. Elle sert essentiellement à redémarrer une temporisation en cours. Cela n'est possible que si l'opération de démarrage continue à être traitée avec un RLG de 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST              Explication
U       E 2.0
FR      T1        //Valider la temporisation T1.
U       E 2.1
L       S5T#10s   //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SI      T1        //Démarrer la temporisation T1 sous forme d'impulsion.
U       E 2.2
R       T1        //Remettre la temporisation T1 à zéro.
U       T1        //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1        //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                  //nombre binaire.
T       MW10
```



(1) Le passage du RLG de 0 à 1 à l'entrée de validation, alors que la temporisation s'exécute, redémarre la temporisation. Le temps programmé est utilisé comme temps en cours pour le redémarrage. Le passage du RLG de 1 à 0 à l'entrée de validation n'a pas d'influence.

(2) Si le RLG passe de 0 à 1 à l'entrée de validation alors que la temporisation ne s'exécute pas et qu'un RLG de 1 reste appliqué à l'entrée de démarrage, la temporisation est également démarrée sous forme d'impulsion avec la valeur de temps programmée.

(3) Le passage du RLG de 0 à 1 à l'entrée de validation alors qu'un RLG égal à 0 reste appliqué à l'entrée de démarrage n'a aucun effet sur la temporisation.

12.4 L Charger valeur de temps en cours comme nombre entier dans l'accumulateur 1

Format

L <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

L <temporisation>

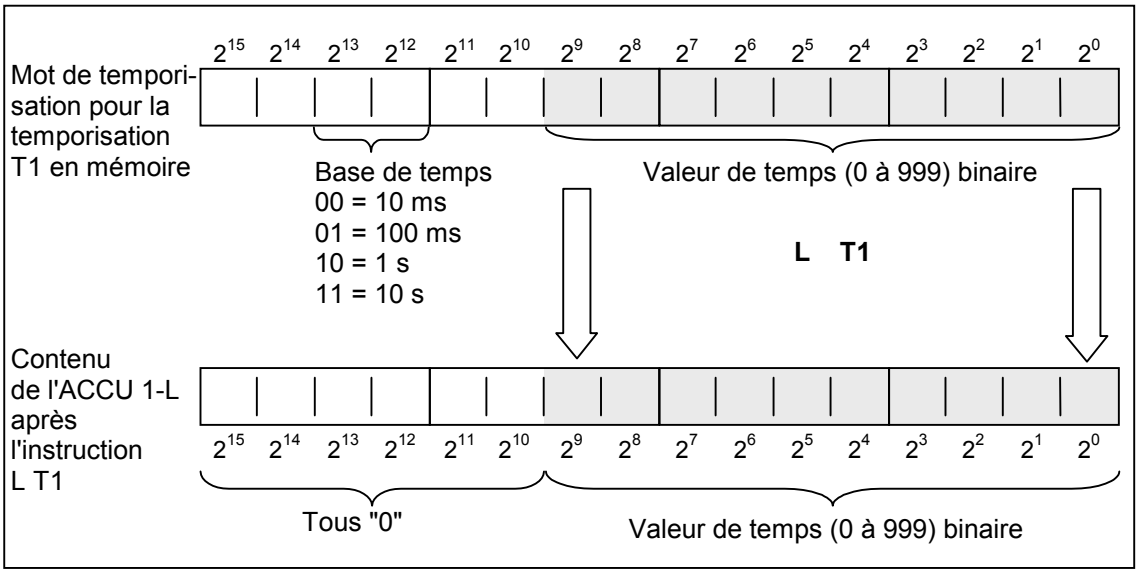
Cette opération charge, comme nombre entier binaire dans l'accumulateur 1-L, la valeur de temps en cours figurant dans le mot de temporisation indiqué sans la base de temps, et ce après sauvegarde préalable du contenu de l'accumulateur 1 dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
L       T1      //Charger dans l'accumulateur 1-L la valeur de temps de la temporisation T1
                //en format binaire.
```



Nota

L <temporisation> charge uniquement le code binaire de la valeur de temps en cours dans l'accumulateur 1-L et non la base de temps. La valeur de temps chargée est la valeur initiale de la temporisation moins le temps qui s'est écoulé depuis le démarrage de la fonction de temporisation.

12.5 LC Charger valeur de temps en cours comme nombre DCB dans l'accumulateur 1

Format

LC <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

LC <temporisation>

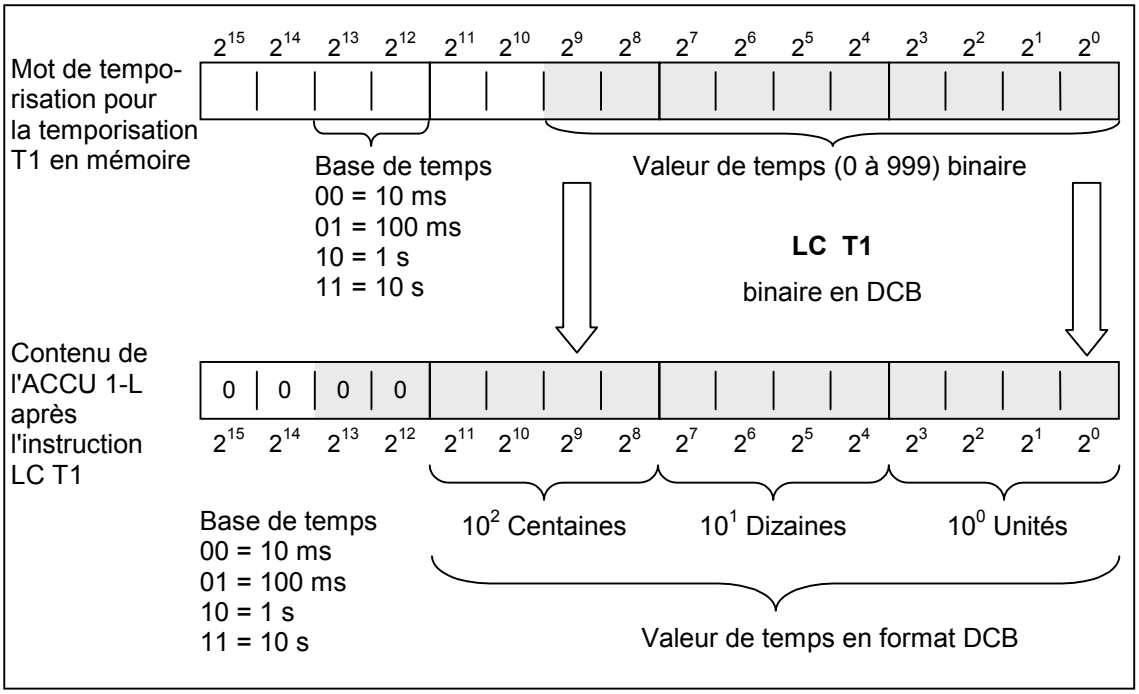
Cette opération charge, comme nombre DCB dans l'accumulateur 1, la valeur de temps et la base de temps figurant dans le mot de temporisation indiqué, et ce après sauvegarde préalable du contenu de l'accumulateur 1 dans l'accumulateur 2.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
LC      T1        //Charger dans l'accumulateur 1 la valeur de temps en cours et la base de temps
                  //de la temporisation T1 en format DCB.
```



12.6 R Remettre temporisation à 0

Format

R <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

R <temporisation>

Cette opération permet de remettre une temporisation à zéro et d'effacer la valeur de temps et la base de temps du mot de temporisation indiqué si le résultat logique RLG passe de 0 à 1.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
U       E 2.1
R       T1      //Interroger l'état de signal à l'entrée E 2.1. Si le RLG passe de 0 à 1,
                //remettre la temporisation T1 à zéro.
```

12.7 SI Temporisation sous forme d'impulsion

Format

SI <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

SI <temporisation>

Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule tant que le RLG égale 1. Si le RLG passe à 0 avant que cette durée n'ait expiré, la temporisation s'arrête. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps figurent en format DCB dans l'accumulateur 1-L.

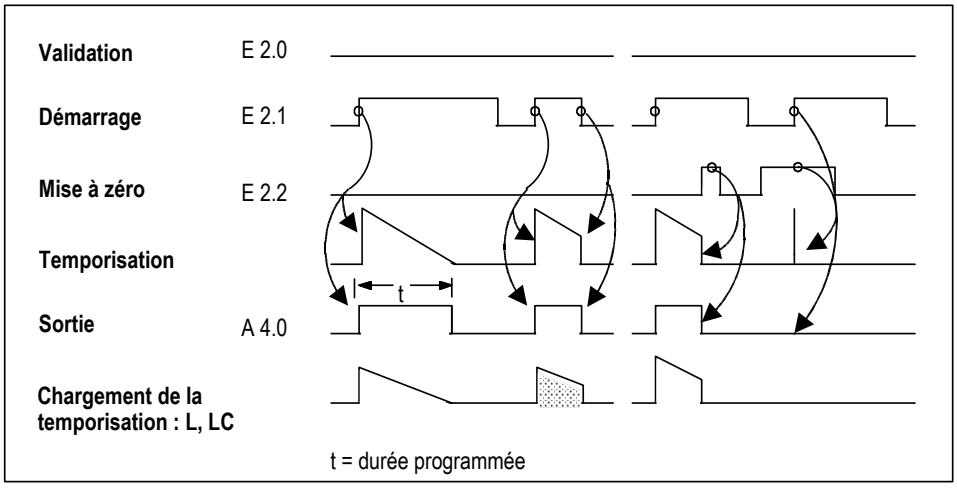
Voir aussi Adresse d´une temporisation en mémoire et composants d´une temporisation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST                Explication
U       E 2.0
FR      T1          //Valider la temporisation T1.
U       E 2.1
L       S5T#10s     //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SI      T1          //Démarrer la temporisation T1 sous forme d'impulsion.
U       E 2.2
R       T1          //Remettre la temporisation T1 à zéro.
U       T1          //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1          //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                    //nombre binaire.
T       MW10
LC      T1          //Charger la valeur de temps en cours de la temporisation T1 en format DCB.
T       MW12
```



12.8 SV Temporisation sous forme d'impulsion prolongée

Format

SV <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

SV <temporisation>

Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule même si le RLG passe entre-temps à 0. Si le RLG passe de 0 à 1 avant que la durée programmée n'ait expiré, le temps redémarre. La valeur de temps et la base de temps doivent figurer en format DCB dans l'accumulateur 1-L pour que la temporisation démarre.

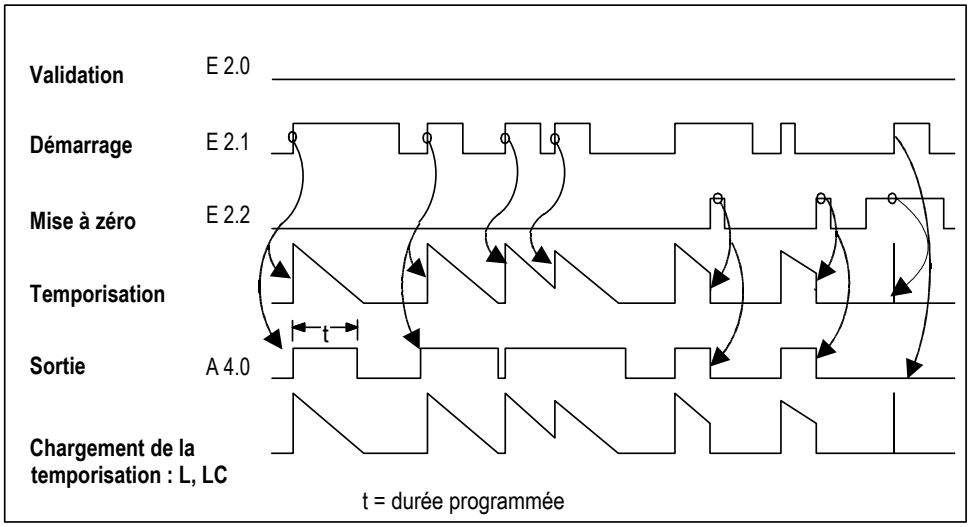
Voir aussi Adresse d´une temporisation en mémoire et composants d´une temporisation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
U       E 2.0
FR      T1      //Valider la temporisation T1.
U       E 2.1
L       S5T#10s //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SV      T1      //Démarrer la temporisation T1 sous forme d'impulsion prolongée.
U       E 2.2
R       T1      //Remettre la temporisation T1 à zéro.
U       T1      //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1      //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                //nombre binaire.
T       MW10
LC      T1      //Charger la valeur de temps en cours de la temporisation T1 en format DCB.
T       MW12
```



12.9 SE Temporisation sous forme de retard à la montée

Format

SE <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

SE <temporisation>

Cette opération démarre la temporisation indiquée si le résultat logique passe de 0 à 1. La durée initiale programmée s'écoule tant que le RLG égale 1. Si le RLG passe à 0 avant que cette durée n'ait expiré, la temporisation s'arrête. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps sont mémorisées dans l'accumulateur 1-L en format DCB.

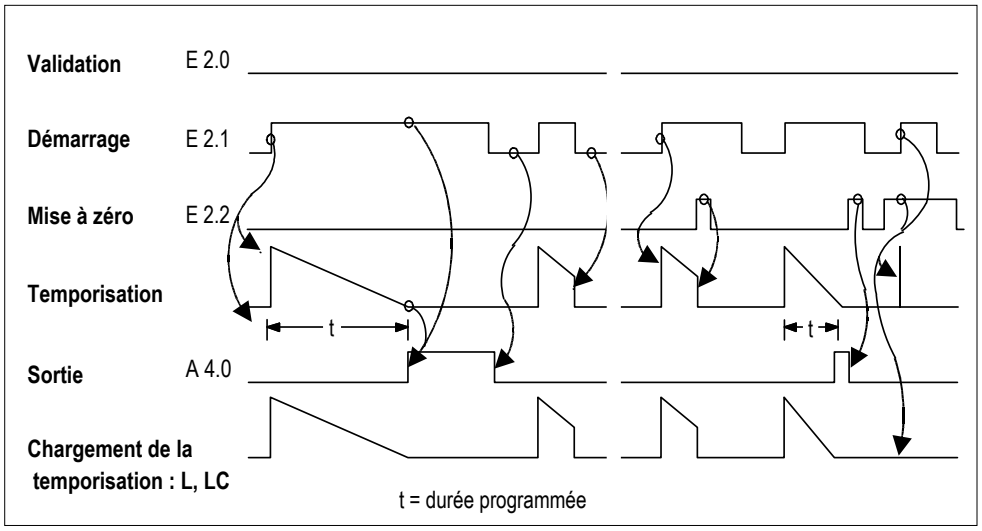
Voir aussi Adresse d´une temporisation en mémoire et composants d´une temporisation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST              Explication
U       E 2.0
FR      T1        //Valider la temporisation T1.
U       E 2.1
L       S5T#10s   //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SE      T1        //Démarrer la temporisation T1 sous forme de retard à la montée.
U       E 2.2
R       T1        //Remettre la temporisation T1 à zéro.
U       T1        //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1        //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                  //nombre binaire.
T       MW10
LC      T1        //Charger la valeur de temps en cours de la temporisation T1 en format DCB.
T       MW12
```



12.10 SS Temporisation sous forme de retard à la montée mémorisé

Format

SS <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

SS <temporisation> (Démarrer temporisation sous forme de retard à la montée mémorisé)

Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule même si le RLG passe entre-temps à 0. Si le RLG passe de 0 à 1 avant que cette durée n'ait expiré, le temps redémarre. La valeur de temps et la base de temps doivent figurer en format DCB dans l'accumulateur 1-L pour que la temporisation démarre.

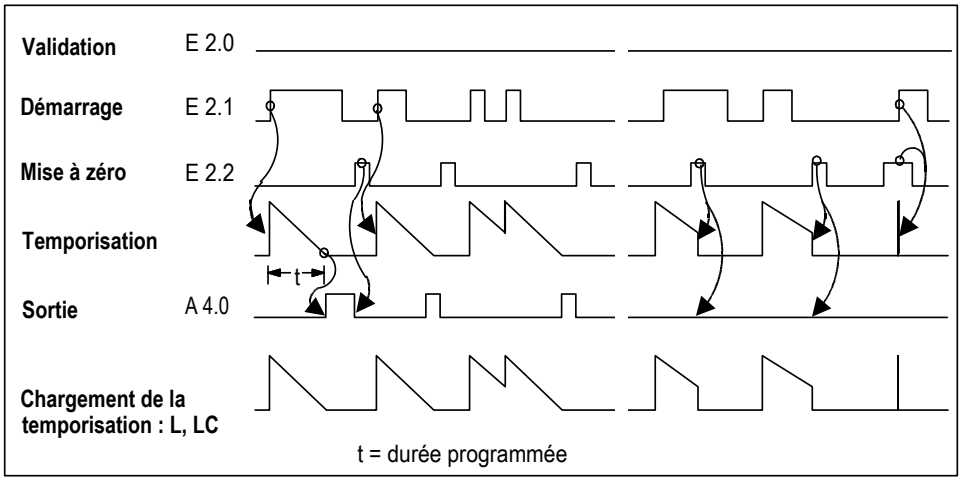
Voir aussi Adresse d´une temporisation en mémoire et composants d´une temporisation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST                Explication
U       E 2.0
FR      T1          //Valider la temporisation T1.
U       E 2.1
L       S5T#10s     //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SS      T1          //Démarrer la temporisation T1 sous forme de retard à la montée mémorisé.
U       E 2.2
R       T1          //Remettre la temporisation T1 à zéro.
U       T1          //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1          //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                    //nombre binaire.
T       MW10
LC      T1          //Charger la valeur de temps en cours de la temporisation T1 en format DCB.
T       MW12
```



12.11 SA Temporisation sous forme de retard à la retombée

Format

SA <temporisation>

Opérande	Type de données	Zone de mémoire	Description
<temporisation>	TIMER	T	Numéro de la temporisation ; la plage dépend de la CPU.

Description de l'opération

SA <temporisation>

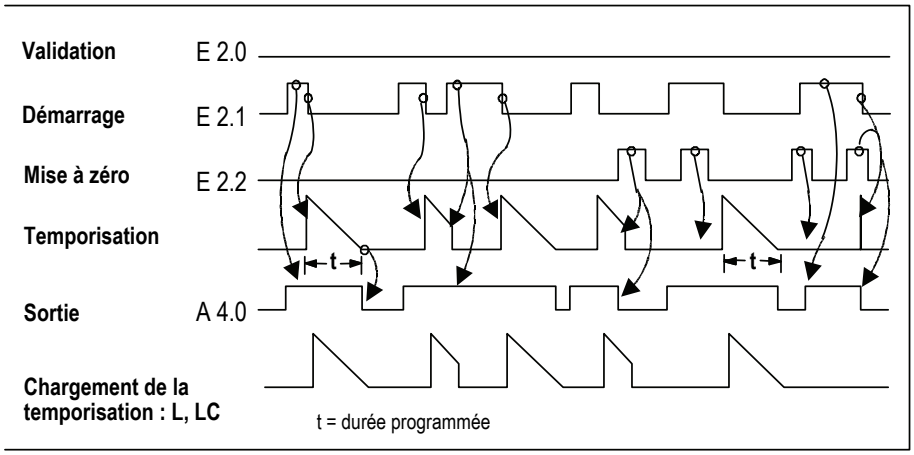
Cette opération démarre la temporisation indiquée si le résultat logique passe de 1 à 0. La durée programmée s'écoule tant que le RLG égale 0. Si le RLG passe à 1 avant que cette durée n'ait expiré, la temporisation s'arrête. La valeur de temps et la base de temps doivent figurer en format DCB dans l'accumulateur 1-L pour que la temporisation démarre.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	0	-	-	0

Exemple

```
LIST            Explication
U       E 2.0
FR      T1      //Valider la temporisation T1.
U       E 2.1
L       S5T#10s //Définir une initialisation de 10 secondes dans l'accumulateur 1.
SA      T1      //Démarrer la temporisation T1 sous forme de retard à la retombée.
U       E 2.2
R       T1      //Remettre la temporisation T1 à zéro.
U       T1      //Interroger l'état de signal de la temporisation T1.
=       A 4.0
L       T1      //Charger la valeur de temps en cours de la temporisation T1 sous forme de
                //nombre binaire.
T       MW10
LC      T1      //Charger la valeur de temps en cours de la temporisation T1 en format DCB.
T       MW12
```



13 Opérations combinatoires sur mots

13.1 Vue d'ensemble des opérations combinatoires sur mots

Description

Les opérations combinatoires sur mots combinent deux mots (16 bits) ou deux doubles mots (32 bits) bit par bit selon les combinaisons booléennes. Chaque mot ou double mot doit se trouver dans l'un des deux accumulateurs.

Lors de la combinaison de mots, le contenu du mot de poids faible de l'accumulateur 2 est combiné au contenu du mot de poids faible de l'accumulateur 1. Le résultat de la combinaison se substitue à l'ancien contenu du mot de poids faible de l'accumulateur 1.

Lors de la combinaison de doubles mots, le contenu de l'accumulateur 2 est combiné au contenu de l'accumulateur 1. Le résultat de la combinaison se substitue à l'ancien contenu de l'accumulateur 1.

Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération.

Vous disposez des opérations combinatoires sur mots suivantes :

- UW ET mot (16 bits)
- OW OU mot (16 bits)
- XOW OU exclusif mot (16 bits)
- UD ET double mot (32 bits)
- OD OU double mot (32 bits)
- XOD OU exclusif double mot (32 bits)

13.2 UW ET mot (16 bits)

Formats

UW
UW <constante>

Opérande	Type de données	Description
<constante>	WORD, constante (16 bits)	Profil binaire à combiner à l'accumulateur 1-L selon la table de vérité ET

Description de l'opération

UW (ET mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1-L au contenu de l'accumulateur 2-L ou à une constante de 16 bits selon la table de vérité ET. C'est uniquement si les bits correspondants des deux mots à combiner sont à 1 que le bit dans le mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1-L. L'accumulateur 1-H et l'accumulateur 2, ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs, restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

UW : combine l'accumulateur 1-L à l'accumulateur 2-L.

UW <constante> : combine l'accumulateur 1-L à une constante de 16 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	15 . .	. .	. .	. . . 0
Accumulateur 1-L avant exécution de **UW**	0101	1001	0011	1011
Accumulateur 2-L ou constante (16 bits)	1111	0110	1011	0101
Résultat (ACCU 1-L) après exécution de **UW**	0101	0000	0011	0001

Exemple 1

```
LIST              Explication
L      EW20       //Charger le contenu de EW20 dans l'accumulateur 1-L.
L      EW22       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                  //contenu de EW22 dans l'accumulateur 1-L.
UW                //Combiner, bit par bit, le contenu de l'accumulateur 1-L au contenu de
                  //l'accumulateur 2-L selon la table de vérité ET ; ranger le résultat dans
                  //l'accumulateur 1-L.
T      MW 8       //Transférer le résultat dans le mot de mémento MW8.
```

Exemple 2

```
LIST              Explication
L      EW20       //Charger le contenu de EW20 dans l'accumulateur 1-L.
UW     W#16#0FFF  //Combiner les bits de l'accumulateur 1-L au profil binaire de la constante
                  //de 16 bits (0000_1111_1111_1111) selon la table de vérité ET ; ranger le
                  //résultat dans l'accumulateur 1-L.
SPP    SUIV       //Sauter au repère de saut SUIV si le résultat est différent de 0 (BI1 égale 1).
```

13.3 OW OU mot (16 bits)

Formats

OW
OW <constante>

Opérande	Type de données	Description
<constante>	WORD, constante (16 bits)	Profil binaire à combiner à l'accumulateur 1-L selon la table de vérité OU

Description de l'opération

OW (OU mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1-L au contenu de l'accumulateur 2-L ou à une constante de 16 bits selon la table de vérité OU. C'est uniquement si au moins un des bits correspondants des deux mots à combiner est à 1 que le bit dans le mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1-L. L'accumulateur 1-H et l'accumulateur 2, ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs, restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

OW : combine l'accumulateur 1-L à l'accumulateur 2-L.

OW <constante> : combine l'accumulateur 1-L à une constante de 16 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	**15 . .**	**. .**	**. .**	**. . . 0**
Accumulateur 1-L avant exécution de **OW**	0101	0101	0011	1011
Accumulateur 2-L ou constante (16 bits)	1111	0110	1011	0101
Résultat (ACCU 1-L) après exécution de **OW**	1111	0111	1011	1111

Exemple 1

```
LIST            Explication
L    EW20       //Charger le contenu de EW20 dans l'accumulateur 1-L.
L    EW22       //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                //contenu de EW22 dans l'accumulateur 1-L.
OW              //Combiner, bit par bit, le contenu de l'accumulateur 1-L au contenu de
                //l'accumulateur 2-L selon la table de vérité OU ; ranger le résultat dans
                //l'accumulateur 1-L.
T    MW8        //Transférer le résultat dans le mot de mémento MW8.
```

Exemple 2

```
LIST            Explication
L    EW20       //Charger le contenu de EW20 dans l'accumulateur 1-L.
OW   W#16#0FFF  //Combiner les bits de l'accumulateur 1-L au profil binaire de la constante
                //de 16 bits (0000_1111_1111_1111) selon la table de vérité OU ; ranger le
                //résultat dans l'accumulateur  1-L.
SPP  SUIV       //Sauter au repère de saut SUIV si le résultat est différent de 0
                //(BI1 égale 1).
```

13.4 XOW OU exclusif mot (16 bits)

Formats

XOW
XOW <constante>

Opérande	Type de données	Description
<constante>	WORD, constante (16 bits)	Profil binaire à combiner à l'accumulateur 1-L selon la table de vérité OU exclusif

Description de l'opération

XOW (OU exclusif mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1-L au contenu de l'accumulateur 2-L ou à une constante de 16 bits selon la table de vérité OU exclusif. C'est uniquement si exactement un des bits correspondants des deux mots à combiner est à 1 que le bit dans le mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1-L. L'accumulateur 1-H et l'accumulateur 2, ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs, restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

Vous pouvez également utiliser la fonction OU exclusif plusieurs fois d'affilée. Le RLG global est alors égal à "1", lorsqu'un nombre impair des opérandes interrogés fournit le résultat "1".

XOW : combine l'accumulateur 1-L à l'accumulateur 2-L.

XOW <constante> : combine l'accumulateur 1-L à une constante de 16 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	**15 . .**	**. .**	**. .**	**. . . 0**
Accumulateur 1-L avant exécution de **XOW**	0101	0101	0011	1011
Accumulateur 2-L ou constante (16 bits)	1111	0110	1011	0101
Résultat (ACCU 1-L) après exécution de **XOW**	1010	0011	1000	1110

Exemple 1

```
LIST            Explication
L     EW20      //Charger le contenu de EW20 dans l'accumulateur 1-L.
L     EW22      //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                //contenu de EW22 dans l'accumulateur 1-L.
XOW             //Combiner, bit par bit, le contenu de l'accumulateur 1-L au contenu de
                //l'accumulateur 2-L selon la table de vérité OU exclusif ; ranger le résultat
                //dans l'accumulateur 1-L.
T     MW8       //Transférer le résultat dans le mot de mémento MW8.
```

Exemple 2

```
LIST            Explication
L     EW20      //Charger le contenu de EW20 dans l'accumulateur 1-L.
XOW   16#0FFF   //Combiner les bits de l'accumulateur 1-L au profil binaire de la constante
                //de 16 bits (0000_1111_1111_1111) selon la table de vérité OU exclusif ;
                //ranger le résultat dans l'accumulateur 1-L.
SPP   SUIV      //Sauter au repère de saut SUIV si le résultat est différent de 0
                //(BI1 égale 1).
```

13.5 UD ET double mot (32 bits)

Formats

UD
UD <constante>

Opérande	Type de données	Description
<constante>	DWORD, constante (32 bits)	Profil binaire à combiner à l'accumulateur 1 selon la table de vérité ET

Description de l'opération

UD (ET double mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1 au contenu de l'accumulateur 2 ou à une constante de 32 bits selon la table de vérité ET. C'est uniquement si les bits correspondants des deux doubles mots à combiner sont à 1 que le bit dans le double mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1. L'accumulateur 2 ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

UD : combine l'accumulateur 1 à l'accumulateur 2.

UD <constante> : combine l'accumulateur 1 à une constante de 32 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	**31 . .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. . . 0**
Accumulateur 1 avant exécution de **UD**	0101	0000	1111	1100	1000	1001	0011	1011
Accumulateur 2 ou constante (32 bits)	1111	0011	1000	0101	0111	0110	1011	0101
Résultat (ACCU 1) après exécution de **UD**	0101	0000	1000	0100	0000	0000	0011	0001

Exemple 1

```
LIST                Explication
L   ED20            //Charger le contenu de ED20 dans l'accumulateur 1.
L   ED24            //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                    //contenu de ED24 dans l'accumulateur 1.
UD                  //Combiner, bit par bit, le contenu de l'accumulateur 1 au contenu de
                    //l'accumulateur 2 selon la table de vérité ET ; ranger le résultat dans
                    //l'accumulateur 1.
T   MD8             //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST                Explication
L   ED 20           //Charger le contenu de ED20 dans l'accumulateur 1.
UD  DW#16#0FFF_EF21 //Combiner les bits de l'accumulateur 1 au profil binaire de la constante
                    //de 32 bits (0000_1111_1111_1111_1110_1111_0010_0001) selon la table de
                    //vérité ET ; ranger le résultat dans l'accumulateur 1.
SPP SUIV            //Sauter au repère de saut SUIV si le résultat est différent de 0
                    //(BI1 égale 1).
```

13.6 OD OU double mot (32 bits)

Formats

OD
OD <constante>

Opérande	Type de données	Description
<constante>	DWORD, constante (32 bits)	Profil binaire à combiner à l'accumulateur 1 selon la table de vérité OU

Description de l'opération

OD (OU double mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1 au contenu de l'accumulateur 2 ou à une constante de 32 bits selon la table de vérité OU. C'est uniquement si au moins un des bits correspondants des deux doubles mots à combiner est à 1 que le bit dans le double mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1. L'accumulateur 2 ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

OD : combine l'accumulateur 1 à l'accumulateur 2.

OD <constante> : combine l'accumulateur 1 à une constante de 32 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	**31 . .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. . . 0**
Accumulateur 1 avant exécution de **OD**	0101	0000	1111	1100	1000	0101	0011	1011
Accumulateur 2 ou constante (32 bits)	1111	0011	1000	0101	0111	0110	1011	0101
Résultat (ACCU 1) après exécution de **OD**	1111	0011	1111	1101	1111	0111	1011	1111

Exemple 1

```
LIST                    Explication
L     ED20              //Charger le contenu de ED20 dans l'accumulateur 1.
L     ED24              //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger
                        //le contenu de ED24 dans l'accumulateur 1.
OD                      //Combiner, bit par bit, le contenu de l'accumulateur 1 au contenu de
                        //l'accumulateur 2 selon la table de vérité OU ; ranger le résultat dans
                        //l'accumulateur 1.
T     MD8               //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST                    Explication
L     ED20              //Charger le contenu de ED20 dans l'accumulateur 1.
OD    DW#16#0FFF_EF21   //Combiner les bits de l'accumulateur 1 au profil binaire de la constante
                        //de 32 bits (0000_1111_1111_1111_1110_1111_0010_0001) selon la table
                        //de vérité OU ; ranger le résultat dans l'accumulateur 1.
SPP   SUIV              //Sauter au repère de saut SUIV si le résultat est différent de 0
                        //(BI1 égale 1).
```

13.7 XOD OU exclusif double mot (32 bits)

Formats

XOD
XOD <constante>

Opérande	Type de données	Description
<constante>	DWORD, constante (32 bits)	Profil binaire à combiner à l'accumulateur 1 selon la table de vérité OU exclusif

Description de l'opération

XOD (OU exclusif double mot)

Cette opération combine, bit par bit, le contenu de l'accumulateur 1 au contenu de l'accumulateur 2 ou à une constante de 32 bits selon la table de vérité OU exclusif. C'est uniquement si exactement un des bits correspondants des deux doubles mots à combiner est à 1 que le bit dans le double mot de résultat sera à 1. Le résultat est rangé dans l'accumulateur 1. L'accumulateur 2 ainsi que les accumulateurs 3 et 4 pour les CPU à quatre accumulateurs restent inchangés. Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération. Les bits d'état BI0 et DEB sont mis à 0.

Vous pouvez également utiliser la fonction OU exclusif plusieurs fois d'affilée. Le RLG global est alors égal à "1", lorsqu'un nombre impair des opérandes interrogés fournit le résultat "1".

XOD : combine l'accumulateur 1 à l'accumulateur 2.

XOD <constante> : combine l'accumulateur 1 à une constante de 32 bits.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	x	0	0	-	-	-	-	-

Exemples

Bit	**31 . .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. . . 0**
Accumulateur 1 avant exécution de **XOD**	0101	0000	1111	1100	1000	0101	0011	1011
Accumulateur 2 ou constante (32 bits)	1111	0011	1000	0101	0111	0110	1011	0101
Résultat (ACCU 1) après exécution de **XOD**	1010	0011	0111	1001	1111	0011	1000	1110

Exemple 1

```
LIST              Explication
L     ED20        //Charger le contenu de ED20 dans l'accumulateur 1.
L     ED24        //Charger le contenu de l'accumulateur 1 dans l'accumulateur 2. Charger le
                  //contenu de ED24 dans l'accumulateur 1.
XOD               //Combiner, bit par bit, le contenu de l'accumulateur 1 au contenu de
                  //l'accumulateur 2 selon la table de vérité OU exclusif ; ranger le résultat
                  //dans l'accumulateur 1.
T     MD8         //Transférer le résultat dans le double mot de mémento MD8.
```

Exemple 2

```
LIST                    Explication
L     ED20              //Charger le contenu de ED20 dans l'accumulateur 1.
XOD   DW#16#0FFF_EF21   //Combiner les bits de l'accumulateur 1 au profil binaire de la constante
                        //de 32 bits (0000_1111_1111_1111_1111_1110_0010_0001) selon la table
                        //de vérité OU exclusif ; ranger le résultat dans l'accumulateur 1.
SPP   SUIV              //Sauter au repère de saut SUIV si le résultat est différent de 0
                        //(BI1 égale 1).
```

14 Opérations sur les accumulateurs

14.1 Vue d'ensemble des opérations sur les accumulateurs

Description

Les opérations suivantes permettent de traiter le contenu d’un ou plusieurs accumulateurs ou registres d’adresse :

- TAK Permuter accumulateur 1 et accumulateur 2
- PUSH CPU avec deux accumulateurs
- PUSH CPU avec quatre accumulateurs
- POP CPU avec deux accumulateurs
- POP CPU avec quatre accumulateurs

- ENT Entrer dans pile accumulateur
- LEAVE Quitter pile accumulateur
- INC Incrémenter accumulateur 1-L-L
- DEC Décrémenter accumulateur 1-L-L

- +AR1 Additionner accumulateur 1 au registre d'adresse 1
- +AR2 Additionner accumulateur 1 au registre d'adresse 2
- BLD Opération de composition d'image (opération nulle)
- NOP 0 Opération nulle
- NOP 1 Opération nulle

14.2 TAK Permuter accumulateur 1 et accumulateur 2

Format

TAK

Description de l'opération

TAK (Permuter accumulateur 1 et accumulateur 2)

Cette opération permet de permuter le contenu de l'accumulateur 1 et celui de l'accumulateur 2. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux. Les contenus des accumulateurs 3 et 4 ne sont pas modifiés (CPU avec quatre accumulateurs).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple : Soustraire la plus petite valeur de la plus grande valeur

```
LIST                    Explication
        L     MW10      //Charger le contenu du  mot de mémento MW10 dans l'accumulateur 1-L.
        L     MW12      //Charger le contenu de l'accumulateur 1-L dans l'accumulateur 2-L.
                        //Charger le contenu du mot de mémento MW12 dans l'accumulateur 1-L.
        >I              //Vérifier si l'accumulateur 2-L (MW10) est plus grand que l'accumulateur
                        //1-L (MW12).
        SPB   SUIV      //Sauter au repère de saut SUIV si l'accumulateur 2 (MW10) est plus grand
                        //que l'accumulateur 1 (MW12).
        TAK             //Permuter le contenu de l'accumulateur 1 et celui de l'accumulateur 2.
SUIV:   -I              //Soustraire le contenu de l'accumulateur 1-L du contenu de l'accumulateur
                        //2-L.
        T     MW14      //Transférer le résultat (= valeur plus grande moins la valeur plus petite)
                        //dans MW14.
```

Contenu	ACCU 1	ACCU 2
Avant exécution de l'opération **TAK**	<MW12>	<MW10>
Après exécution de l'opération **TAK**	<MW**10**>	<MW**12**>

14.3 PUSH CPU avec deux accumulateurs

Format

PUSH

Description de l'opération

PUSH (Accumulateur 1 dans accumulateur 2)

Cette opération copie le contenu complet de l'accumulateur 1 dans l'accumulateur 2 sans modifier l'accumulateur 1. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	**BI1**	**BI0**	**DEB**	**DM**	**OU**	**ETAT**	**RLG**	**/PI**
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MW10       //Charger le contenu du mot de mémento MW10 dans l'accumulateur 1.
PUSH              //Copier le contenu total de l'accumulateur 1 dans l'accumulateur 2.
```

Contenu	**ACCU 1**	**ACCU 2**
Avant exécution de l'opération **PUSH**	<MW10>	<X>
Après exécution de l'opération **PUSH**	<MW10>	<**MW10**>

14.4 PUSH CPU avec quatre accumulateurs

Format

PUSH

Description de l'opération

PUSH (CPU avec quatre accumulateurs)

Cette opération copie le contenu de l'accumulateur 3 dans l'accumulateur 4, le contenu de l'accumulateur 2 dans l'accumulateur 3 et le contenu de l'accumulateur 1 dans l'accumulateur 2. L'accumulateur 1 n'est pas modifié. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MW10       //Charger le contenu du mot de mémento MW10 dans l'accumulateur 1.
PUSH              //Copier le contenu total de l'accumulateur 1 dans l'accumulateur 2, le contenu
                  //de l'accumulateur 2 dans l'accumulateur 3  et le contenu de l'accumulateur
                  //3 dans l'accumulateur 4.
```

Contenu	ACCU 1	ACCU 2	ACCU 3	ACCU 4
Avant exécution de l'opération **PUSH**	valeur A	valeur B	valeur C	valeur D
Après exécution de l'opération **PUSH**	valeur A	valeur A	valeur B	valeur C

14.5 POP CPU avec deux accumulateurs

Format

POP

Description de l'opération

POP (CPU avec deux accumulateurs)

Cette opération copie le contenu complet de l'accumulateur 2 dans l'accumulateur 1 sans modifier l'accumulateur 2. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
T      MD10       //Transférer le contenu de l'accumulateur 1 (= valeur A) dans le double mot
                  //de mémento MD10.
POP               //Copier le contenu total de l'accumulateur 2 dans l'accumulateur 1.
T      MD14       //Transférer le contenu de l''accumulateur 1 (= valeur B) dans le double mot
                  //de mémento MD14.
```

Contenu	ACCU 1	ACCU 2
Avant exécution de l'opération **POP**	valeur A	valeur B
Après exécution de l'opération **POP**	valeur B	valeur B

14.6 POP CPU avec quatre accumulateurs

Format

POP

Description de l'opération

POP (CPU avec quatre accumulateurs)

Cette opération copie le contenu de l'accumulateur 2 dans l'accumulateur 1, le contenu de l'accumulateur 3 dans l'accumulateur 2 et le contenu de l'accumulateur 4 dans l'accumulateur 3. L'accumulateur 4 n'est pas modifié. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST            Explication
T     MD10      //Transférer le contenu de l'accumulateur 1 (= valeur A) dans le double mot
                //de mémento MD10.
POP             //Copier le contenu de l'accumulateur 2 dans l'accumulateur 1, le contenu de
                //l'accumulateur 3 dans l'accumulateur 2 et le contenu de l'accumulateur 4
                //dans l'accumulateur 3.
T     MD14      //Transférer le contenu de l'accumulateur 1 (= valeur B) dans le double mot
                //de mémento MD14.
```

Contenu	ACCU 1	ACCU 2	ACCU 3	ACCU 4
Avant exécution de l'opération **POP**	valeur A	valeur B	valeur C	valeur D
Après exécution de l'opération **POP**	valeur B	valeur C	valeur D	valeur D

14.7 ENT Entrer dans pile accumulateur

Format

ENT

Description de l'opération

ENT (Entrer dans pile accumulateur)

Cette opération copie le contenu de l'accumulateur 3 dans l'accumulateur 4 et le contenu de l'accumulateur 2 dans l'accumulateur 3. En programmant l'opération **ENT** directement avant une opération de chargement, vous pouvez sauvegarder un résultat intermédiaire dans l'acccumulateur 3.

Exemple

```
LIST            Explication
L       DBD0    //Charger la valeur (nombre à virgule flottante) contenue dans le double mot de
                données DBD0 dans l'accumulateur 1.
L       DBD4    //Copier la valeur de l'accumulateur 1 dans l'accumulateur 2. Charger la valeur
                //(nombre à virgule flottante) contenue dans le double mot de données DBD4 dans
                //l'accumulateur 1.
+R              //Additionner les contenus des accumulateurs 1 et 2 (nombres à virgule flottante
                //IEEE 754 de 32 bits) et sauvegarder le résultat dans l'accumulateur 1.
L       DBD8    //Copier la valeur de l'accumulateur 1 dans l'accumulateur 2 et celle du double
                //mot de données DBD8 dans l'accumulateur 1.
ENT             //Copier le contenu de l'accumulateur 3 dans l'accumulateur 4. Copier le contenu
                //de l'accumulateur 2 (résultat intermédiaire) dans l'accumulateur  3.
L       DBD12   //Charger la valeur du double mot de données DBD12 dans l'accumulateur 1.
-R              //Soustraire le contenu de l'accumulateur 1 du contenu de l'accumulateur 2 et
                //sauvegarder le résultat dans l'accumulateur  1. Copier le contenu de
                //l'accumulateur  3 dans l'accumulateur 2 et le contenu de l'accumulateur 4 dans
                //l'accumulateur 3.
/R              //Diviser le contenu de l'accumulateur 2 (DBD0 + DBD4) par le contenu de
                //l'accumulateur 1 (DBD8 - DBD12) et sauvegarder le résultat dans
                //l'accumulateur  1.
T       DBD16   //Transférer le résultat (accumulateur 1) dans le double mot de données DBD16.
```

14.8 LEAVE Quitter pile accumulateur

Format

LEAVE

Description de l'opération

LEAVE (Quitter pile accumulateur)

Cette opération copie le contenu de l'accumulateur 3 dans l'accumulateur 2 et le contenu de l'accumulateur 4 dans l'accumulateur 3. Si vous programmez l'opération **LEAVE** directement avant une opération de décalage ou de rotation qui combine les accumulateurs, **LEAVE** agit comme une opération arithmétique. Les contenus des accumulateurs 1 et 4 restent inchangés.

14.9 INC Incrémenter accumulateur 1-L-L

Format

INC <entier, 8 bits>

Opérande	Type de données	Description
<entier, 8 bits>	Constante (entier, 8 bits)	Constante à additionner au contenu de l'accumulateur 1-L-L, plage de 0 à 255

Description de l'opération

INC <entier, 8 bits> (Incrémenter accumulateur 1-L-L)

Cette opération additionne l'entier de 8 bits indiqué au contenu de l'accumulateur 1-L-L et sauvegarde le résultat dans l'accumulateur 1-L-L. L'accumulateur 1-L-H, l'accumulateur 1-H et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Attention

Ces opérations ne conviennent pas aux opérations arithmétiques (16 ou 32 bits), car elles n'effectuent pas de report de l'octet de poids faible du mot de poids faible de l'accumulateur 1 dans l'octet de poids fort du mot de poids faible de l'accumulateur. Vous devez utiliser +I ou +D pour les opérations arithmétiques (16 ou 32 bits).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MB22       //Charger la valeur de MB22.
INC    1          //Incrémenter d'1 l'accumulateur 1 (MB 22), sauvegarder le résultat dans
                  //l'accumulateur 1-L-L.
T      MB22       //Retransférer le contenu de l'accumulateur 1-L-L (résultat) dans l'octet de
                  //mémento MB22.
```

14.10 DEC Décrémenter accumulateur 1-L-L

Format

DEC <entier, 8 bits>

Opérande	Type de données	Description
<entier, 8 bits>	Constante (entier, 8 bits)	Constante à soustraire du contenu de l'accumulateur 1-L-L, plage de 0 à 255

Description de l'opération

DEC <entier, 8 bits> (Décrémenter accumulateur 1-L-L)

Cette opération soustrait l'entier de 8 bits indiqué du contenu de l'accumulateur 1-L-L et sauvegarde le résultat dans l'accumulateur 1-L-L. L'accumulateur 1-L-H, l'accumulateur 1-H et l'accumulateur 2 restent inchangés. L'opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

Attention

Ces opérations ne conviennent pas aux opérations arithmétiques (16 ou 32 bits), car elles n'effectuent pas de report de l'octet de poids faible du mot de poids faible de l'accumulateur 1 dans l'octet de poids fort du mot de poids faible de l'accumulateur. Vous devez utiliser +I ou +D pour les opérations arithmétiques (16 ou 32 bits).

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple

```
LIST              Explication
L      MB250      //Charger la valeur de MB250.
DEC    1          //Décrémenter d'1 l'accumulateur 1-L-L, sauvegarder le résultat dans
                  //l'accumulateur 1-L-L.
T      MB250      //Retransférer le contenu de l'accumulateur 1-L- L (résultat) dans l'octet
                  //de mémento MB250.
```

14.11 +AR1 Additionner accumulateur 1 au registre d'adresse 1

Formats

+AR1
+AR1 <P#octet.bit>

Opérande	Type de données	Description
<P#octet.bit>	Constante pointeur	Adresse additionnée au registre d'adresse 1

Description de l'opération

+AR1 (Additionner au registre d'adresse 1)

Cette opération ajoute au contenu du registre d'adresse 1 (AR1) le décalage précisé soit dans l'instruction soit dans l'accumulateur 1-L. Le nombre entier de 16 bits est tout d'abord étendu à 24 bits (en tenant compte du signe), puis additionné aux 24 bits de poids faible (partie de l'adresse relative) du registre d'adresse 1. Les bits 24, 25 et 26 qui identifient la zone dans AR1 restent inchangés. Cette opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

+AR1 : Le nombre entier de 16 bits qui doit être additionné au contenu du registre d'adresse 1 correspond à la valeur contenue dans l'accumulateur 1-L. Les valeurs comprises entre -32768 et +32767 sont autorisées.

+AR1 <P#octet.bit> : Le décalage qui doit être additionné correspond à l'opérande **<P#octet.bit>**.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple 1

```
LIST            Explication
L      +300     //Charger la valeur dans l'accumulateur 1-L.
+AR1            //Additionner l'accumulateur 1-L (entier de 16 bits) au registre d'adresse 1.
```

Exemple 2

```
LIST            Explication
+AR1   P#300.0  //Additionner le décalage 300.0 au registre d'adresse 1.
```

14.12 +AR2 Additionner accumulateur 1 au registre d'adresse 2

Formats

+AR2
+AR2 <P#octet.bit>

Opérande	Type de données	Description
<P#octet.bit>	Constante pointeur	Adresse additionnée au registre d'adresse 2

Description de l'opération

+AR2 (Additionner au registre d'adresse 2)

Cette opération ajoute au contenu du registre d'adresse 2 (AR2) le décalage précisé soit dans l'instruction soit dans l'accumulateur 1-L. Le nombre entier de 16 bits est tout d'abord étendu à 24 bits (en tenant compte du signe), puis additionné aux 24 bits de poids faible (partie de l'adresse relative) du registre d'adresse 2. Les bits 24, 25 et 26 qui identifient la zone dans AR2 restent inchangés. Cette opération s'exécute sans tenir compte des bits du mot d'état ni influer sur eux.

+AR2 : Le nombre entier de 16 bits qui doit être additionné au contenu du registre d'adresse 2 correspond à la valeur contenue dans l'accumulateur 1-L. Les valeurs comprises entre -32768 et +32767 sont autorisées.

+AR2 <P#octet.bit> : Le décalage qui doit être additionné correspond à l'opérande **<P#octet.bit>**.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

Exemple 1

```
LIST              Explication
L      +300       //Charger la valeur dans l'accumulateur 1-L.
+AR2              //Additionner l'accumulateur 1-L (entier de 16-bits) au registre
                  //d'adresse 2.
```

Exemple 2

```
LIST              Explication
+AR2   P#300.0    //Additionner le décalage 300.0 au registre d'adresse 2.
```

14.13 BLD Opération de composition d'image (opération nulle)

Format

BLD <nombre>

Opérande	Description
<nombre>	Numéro de l'opération BLD ; plage de 0 à 255

Description de l'opération

BLD <nombre> (Composition d'image ; opération nulle)

Cette opération n'exécute pas de fonction et n'influe pas sur les bits du mot d'état. Elle sert à la composition d'image graphique de la console de programmation. Elle est automatiquement générée lorsqu'un programme CONT ou LOG est affiché en LIST. L'opérande <nombre> identifie l'opération BLD, il est généré par la console de programmation.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

14.14 NOP 0 Opération nulle

Format

NOP 0

Description de l'opération

NOP 0 (Opération NOP avec l'opérande 0)

Cette opération n'exécute aucune fonction et n'influe pas sur les bits du mot d'état. Le code d'opération contient un profil binaire de 16 zéros. L'opération ne revêt d'importance que pour la console de programmation lorsqu'un programme est affiché.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

14.15 NOP 1 Opération nulle

Format

NOP 1

Description de l'opération

NOP 1 (Opération NOP avec l'opérande 1)

Cette opération n'exécute aucune fonction et n'influe pas sur les bits du mot d'état. Le code d'opération contient un profil binaire de 16 uns. L'opération ne revêt d'importance que pour la console de programmation lorsqu'un programme est affiché.

Mot d'état

	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
Ecriture :	-	-	-	-	-	-	-	-	-

A Présentation de toutes les opérations LIST

A.1 Opérations LIST classées d'après les abréviations allemandes (SIMATIC)

Abréviation allemande	Abréviation anglaise	Catalogue des éléments de programme	Description
		Combinaison sur bits	Affectation
)	)	Combinaison sur bits	Fermer la parenthèse d'une expression
*D	*D	Fonction sur nombres entiers	Multiplier accumulateur 1 par accumulateur 2 (entiers de 32 bits)
*I	*I	Fonction sur nombres entiers	Multiplier accumulateur 1 par accumulateur 2 (entiers de 16 bits)
*R	*R	Fonction sur nombres à virgule flottante	Multiplier accumulateur 1 par accumulateur 2 (réels IEEE 754, 32 bits)
/D	/D	Fonction sur nombres entiers	Diviser accumulateur 2 par accumulateur 1 (entiers de 32 bits)
/I	/I	Fonction sur nombres entiers	Diviser accumulateur 2 par accumulateur 1 (entiers de 16 bits)
/R	/R	Fonction sur nombres à virgule flottante	Diviser accumulateur 2 par accumulateur 1 (réels IEEE 754, 32 bits)
? D	? D	Comparaison	Comparer entiers de 32 bits , <>, >, <, >, <
? I	? I	Comparaison	Comparer entiers de 16 bits , <>, >, <, >, <
? R	? R	Comparaison	Comparer réels de 32 bits , <>, >, <, >, <
+	+	Fonction sur nombres entiers	Additionner constante entière (16, 32 bits)
+AR1	+AR1	Accumulateurs	Additionner accumulateur 1 au registre d'adresse 1
+AR2	+AR2	Accumulateurs	Additionner accumulateur 1 au registre d'adresse 2
+D	+D	Fonction sur nombres entiers	Additionner accumulateurs 1 et 2 (entiers de 32 bits)
+I	+I	Fonction sur nombres entiers	Additionner accumulateurs 1 et 2 (entiers de 16 bits)
+R	+R	Fonction sur nombres à virgule flottante	Additionner accumulateurs 1 et 2 (réels IEEE 754, 32 bits)
ABS	ABS	Fonction sur nombres à virgule flottante	Valeur absolue d'un nombre à virgule flottante (IEEE 754, 32 bits)
ACOS	ACOS	Fonction sur nombres à virgule flottante	Arc cosinus d'un nombre à virgule flottante (32 bits)
ASIN	ASIN	Fonction sur nombres à virgule flottante	Arc sinus d'un nombre à virgule flottante (32 bits)
ATAN	ATAN	Fonction sur nombres à virgule flottante	Arc tangente d'un nombre à virgule flottante (32 bits)
AUF	OPN	Blocs de données	Ouvrir bloc de données
BE	BE	Gestion d'exécution de programme	Fin de bloc

Abréviation allemande	Abréviation anglaise	Catalogue des éléments de programme	Description
BEA	BEU	Gestion d'exécution de programme	Fin de bloc inconditionnelle
BEB	BEC	Gestion d'exécution de programme	Fin de bloc conditionnelle
BLD	BLD	Accumulateurs	Opération de composition d'image (opération nulle)
BTD	BTD	Conversions	Convertir DCB en entier de 32 bits
BTI	BTI	Conversions	Convertir DCB en entier de 16 bits
CALL	CALL	Gestion d'exécution de programme	Appel de bloc
CALL	CALL	Gestion d'exécution de programme	Appel de multi-instance
CALL	CALL	Gestion d'exécution de programme	Appel de bloc dans une bibliothèque
CC	CC	Gestion d'exécution de programme	Appel de bloc conditionnel
CLR	CLR	Combinaison sur bits	Mettre RLG à 0
COS	COS	Fonction sur nombres à virgule flottante	Cosinus d'angles comme nombres à virgule flottante (32 bits)
–D	–D	Fonction sur nombres entiers	Soustraire accumulateur 1 de accumulateur 2 (entiers de 32 bits)
DEC	DEC	Accumulateurs	Décrémenter accumulateur 1-L-L
DTB	DTB	Conversions	Convertir entier de 32 bits en DCB
DTR	DTR	Conversions	Convertir entier de 32 bits en réel (IEEE 754 32 bits)
ENT	ENT	Chargement/ Transfert	Entrer dans pile accumulateur
EXP	EXP	Fonction sur nombres à virgule flottante	Valeur exponentielle d'un nombre à virgule flottante (32 bits)
FN	FN	Combinaison sur bits	Front descendant
FP	FP	Combinaison sur bits	Front montant
FR	FR	Compteurs	Valider compteur (validé, FR Z 0 à Z 255)
FR	FR	Temporisations	Valider temporisation
–I	–I	Fonction sur nombres entiers	Soustraire accumulateur 1 de accumulateur 2 (entiers de 16 bits)
INC	INC	Accumulateurs	Incrémenter accumulateur 1-L-L
INVD	INVD	Conversions	Complément à 1 d'entier de 32 bits
INVI	INVI	Conversions	Complément à 1 d'entier de 16 bits
ITB	ITB	Conversions	Convertir entier de 16 bits en DCB
ITD	ITD	Conversions	Convertir entier de 16 bits en entier de 32 bits
L	L	Accumulateurs	Charger
L STW	L STW	Chargement/ Transfert	Charger mot d'état dans l'accumulateur 1
L	L	Temporisations	Charger valeur de temps en cours comme nombre entier dans l'accumulateur 1 (la valeur de temps en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : L T 32)
L	L	Compteurs	Charger valeur de comptage en cours comme entier dans l'accumulateur 1 (la valeur de comptage en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : L Z 15)
L DBLG	L DBLG	Blocs de données	Charger longueur de DB global dans l'accumulateur 1
L DBNO	L DBNO	Blocs de données	Charger numéro de DB global dans l'accumulateur 1

Abréviation allemande	Abréviation anglaise	Catalogue des éléments de programme	Description
L DILG	L DILG	Blocs de données	Charger longueur de DB d'instance dans l'accumulateur 1
L DINO	L DINO	Blocs de données	Charger numéro de DB d'instance dans l'accumulateur 1
LAR1	LAR1	Chargement/ Transfert	Charger contenu de l'accumulateur 1 dans registre d'adresse 1
LAR1	LAR1	Chargement/ Transfert	Charger pointeur de 32 bits dans registre d'adresse 1
LAR1	LAR1	Chargement/ Transfert	Charger contenu du registre d'adresse 2 dans registre d'adresse 1
LAR2	LAR2	Chargement/ Transfert	Charger contenu de l'accumulateur 1 dans registre d'adresse 2
LAR2	LAR2	Chargement/ Transfert	Charger pointeur de 32 bits dans registre d'adresse 2
LC	LC	Compteurs	Charger valeur de comptage en cours comme nombre DCB dans l'accumulateur 1 (la valeur de comptage en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : LC Z 15)
LC	LC	Temporisations	Charger valeur de temps en cours comme nombre DCB dans l'accumulateur 1 (la valeur de temps en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : LC T 32)
LEAVE	LEAVE	Accumulateurs	Quitter pile accumulateur
LN	LN	Fonction sur nombres à virgule flottante	Logarithme naturel d'un nombre à virgule flottante (32 bits)
LOOP	LOOP	Sauts	Boucle de programme
MCR(	MCR(	Gestion d'exécution de programme	Sauvegarder RLG dans pile MCR, début de zone MCR
)MCR	)MCR	Gestion d'exécution de programme	Fin de zone MCR
MCRA	MCRA	Gestion d'exécution de programme	Activer la zone MCR
MCRD	MCRD	Gestion d'exécution de programme	Désactiver la zone MCR
MOD	MOD	Fonction sur nombres entiers	Reste de division entière (32 bits)
NEGD	NEGD	Conversions	Complément à 2 d'entier de 32 bits
NEGI	NEGI	Conversions	Complément à 2 d'entier de 16 bits
NEGR	NEGR	Conversions	Inverser nombre à virgule flottante (IEEE 754 32 bits)
NOP 0	NOP 0	Accumulateurs	Opération nulle
NOP 1	NOP 1	Accumulateurs	Opération nulle
NOT	NOT	Combinaison sur bits	Négation du RLG
O	O	Combinaison sur bits	OU
O(	O(	Combinaison sur bits	OU d'une expression
OD	OD	Combinaison sur mots	OU double mot (32 bits)
ON	ON	Combinaison sur bits	OU NON
ON(	ON(	Combinaison sur bits	OU NON d'une expression
OW	OW	Combinaison sur mots	OU mot (16 bits)
POP	POP	Accumulateurs	POP CPU avec deux accumulateurs
POP	POP	Accumulateurs	POP CPU avec quatre accumulateurs
PUSH	PUSH	Accumulateurs	PUSH CPU avec deux accumulateurs
PUSH	PUSH	Accumulateurs	PUSH CPU avec quatre accumulateurs
R	R	Combinaison sur bits	Mettre à 0

Abréviation allemande	Abréviation anglaise	Catalogue des éléments de programme	Description
R	R	Compteurs	Remettre compteur à 0 (le compteur actuel peut être un nombre compris dans la plage de 0 à 255, par exemple : R Z 15)
R	R	Temporisations	Remettre temporisation à 0 (la temporisation actuelle peut être un nombre compris dans la plage de 0 à 255, par exemple : R T 32)
–R	–R	Fonction sur nombres à virgule flottante	Soustraire accumulateur 1 de accumulateur 2 (réels IEEE 754, 32 bits)
RLD	RLD	Décalage/Rotation	Rotation vers la gauche d'un double mot (32 bits)
RLDA	RLDA	Décalage/Rotation	Rotation vers la gauche de l'accumulateur 1 via BI1 (32 bits)
RND	RND	Conversions	Arrondir à l'entier
RND–	RND–	Conversions	Arrondir à l'entier inférieur
RND+	RND+	Conversions	Arrondir à l'entier supérieur
RRD	RRD	Décalage/Rotation	Rotation vers la droite d'un double mot (32 bits)
RRDA	RRDA	Décalage/Rotation	Rotation vers la droite de l'accumulateur 1 via BI1 (32 bits)
S	S	Combinaison sur bits	Mettre à 1
S	S	Compteurs	Initialiser compteur (le compteur actuel peut être un nombre compris dans la plage de 0 à 255, par exemple : S Z 15)
SA	SF	Temporisations	Temporisation sous forme de retard à la retombée
SAVE	SAVE	Combinaison sur bits	Sauvegarder RLG dans le bit RB
SE	SD	Temporisations	Temporisation sous forme de retard à la montée
SET	SET	Combinaison sur bits	Mettre à 1
SI	SP	Temporisations	Temporisation sous forme d'impulsion
SIN	SIN	Fonction sur nombres à virgule flottante	Sinus d'angles comme nombres à virgule flottante (32 bits)
SLD	SLD	Décalage/Rotation	Décalage vers la gauche d'un double mot (32 bits)
SLW	SLW	Décalage/Rotation	Décalage vers la gauche d'un mot (16 bits)
SPA	JU	Sauts	Saut inconditionnel
SPB	JC	Sauts	Saut si RLG est 1
SPBB	JCB	Sauts	Saut si RLG est 1 avec RB
SPBI	JBI	Sauts	Saut si RB est 1
SPBIN	JNBI	Sauts	Saut si RB est 0
SPBN	JCN	Sauts	Saut si RLG est 0
SPBNB	JNB	Sauts	Saut si RLG est 0 avec RB
SPL	JL	Sauts	Saut vers liste
SPM	JM	Sauts	Saut si moins
SPMZ	JMZ	Sauts	Saut si inférieur ou égal à 0
SPN	JN	Sauts	Saut si différent de 0
SPO	JO	Sauts	Saut si DEB est 1
SPP	JP	Sauts	Saut si plus
SPPZ	JPZ	Sauts	Saut si supérieur ou égal à 0
SPS	JOS	Sauts	Saut si DM est 1
SPU	JUO	Sauts	Saut si illicite
SPZ	JZ	Sauts	Saut si égal à 0
SQR	SQR	Fonction sur nombres à virgule flottante	Carré d'un nombre à virgule flottante (32 bits)
SQRT	SQRT	Fonction sur nombres à virgule flottante	Racine carrée d'un nombre à virgule flottante (32 bits)
SRD	SRD	Décalage/Rotation	Décalage vers la droite d'un double mot (32 bits)
SRW	SRW	Décalage/Rotation	Décalage vers la droite d'un mot (16 bits)

Abréviation allemande	Abréviation anglaise	Catalogue des éléments de programme	Description
SS	SS	Temporisations	Temporisation sous forme de retard à la montée mémorisé
SSD	SSD	Décalage/Rotation	Décalage vers la droite d'un entier avec signe (32 bits)
SSI	SSI	Décalage/Rotation	Décalage vers la droite d'un entier avec signe (16 bits)
SV	SE	Temporisations	Temporisation sous forme d'impulsion prolongée
T	T	Chargement/ Transfert	Transférer
T STW	T STW	Chargement/ Transfert	Transférer accumulateur 1 dans mot d'état
TAD	CAD	Conversions	Modifier l'ordre dans l'accumulateur 1 (32 bits)
TAK	TAK	Accumulateurs	Permuter accumulateur 1 et accumulateur 2
TAN	TAN	Fonction sur nombres à virgule flottante	Tangente d'angles comme nombres à virgule flottante (32 bits)
TAR	CAR	Chargement/ Transfert	Permuter registre d'adresse 1 avec registre d'adresse 2
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 dans l'accumulateur 1
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 à l'adresse de destination (32 bits)
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 dans registre d'adresse 2
TAR2	TAR2	Chargement/ Transfert	Transférer registre d'adresse 2 dans l'accumulateur 1
TAR2	TAR2	Chargement/ Transfert	Transférer registre d'adresse 2 à l'adresse de destination (32 bits)
TAW	CAW	Conversions	Modifier l'ordre dans l'accumulateur 1 (16 bits)
TDB	CDB	Blocs de données	Permuter DB global et DB d'instance
TRUNC	TRUNC	Conversions	Arrondir par troncature
U	A	Combinaison sur bits	ET
U(	A(	Combinaison sur bits	ET d'une expression
UC	UC	Gestion d'exécution de programme	Appel de bloc inconditionnel
UD	AD	Combinaison sur mots	ET double mot
UN	AN	Combinaison sur bits	ET NON
UN(	AN(	Combinaison sur bits	ET NON d'une expression
UW	AW	Combinaison sur mots	ET mot (16 bits)
X	X	Combinaison sur bits	OU exclusif
X(	X(	Combinaison sur bits	OU exclusif d'une expression
XN	XN	Combinaison sur bits	OU NON exclusif
XN(	XN(	Combinaison sur bits	OU NON exclusif d'une expression
XOD	XOD	Combinaison sur mots	OU exclusif double mot (32 bits)
XOW	XOW	Combinaison sur mots	OU exclusif mot (16 bits)
ZR	CD	Compteurs	Décrémenter
ZV	CU	Compteurs	Incrémenter

A.2 Opérations LIST classées d'après les abréviations anglaises (internationales)

Abréviation anglaise	Abréviation allemande	Catalogue des éléments de programme	Description
		Combinaison sur bits	Affectation
)	)	Combinaison sur bits	Fermer la parenthèse d'une expression
*D	*D	Fonction sur nombres entiers	Multiplier accumulateur 1 par accumulateur 2 (entiers de 32 bits)
*I	*I	Fonction sur nombres entiers	Multiplier accumulateur 1 par accumulateur 2 (entiers de 16 bits)
*R	*R	Fonction sur nombres à virgule flottante	Multiplier accumulateur 1 par accumulateur 2 (réels IEEE 754, 32 bits)
/D	/D	Fonction sur nombres entiers	Diviser accumulateur 2 par accumulateur 1 (entiers de 32 bits)
/I	/I	Fonction sur nombres entiers	Diviser accumulateur 2 par accumulateur 1 (entiers de 16 bits)
/R	/R	Fonction sur nombres à virgule flottante	Diviser accumulateur 2 par accumulateur 1 (réels IEEE 754, 32 bits)
? D	? D	Comparaison	Comparer entiers de 32 bits , <>, >, <, >, <
? I	? I	Comparaison	Comparer entiers de 16 bits , <>, >, <, >, <
? R	? R	Comparaison	Comparer réels de 32 bits , <>, >, <, >, <
+	+	Fonction sur nombres entiers	Additionner constante entière (16, 32 bits)
+AR1	+AR1	Accumulateurs	Additionner accumulateur 1 au registre d'adresse 1
+AR2	+AR2	Accumulateurs	Additionner accumulateur 1 au registre d'adresse 2
+D	+D	Fonction sur nombres entiers	Additionner accumulateurs 1 et 2 (entiers de 32 bits)
+I	+I	Fonction sur nombres entiers	Additionner accumulateurs 1 et 2 (entiers de 16 bits)
+R	+R	Fonction sur nombres à virgule flottante	Additionner accumulateurs 1 et 2 (réels IEEE 754, 32 bits)
A	U	Combinaison sur bits	ET
A(	U(	Combinaison sur bits	ET d'une expression
ABS	ABS	Fonction sur nombres à virgule flottante	Valeur absolue d'un nombre à virgule flottante (IEEE 754, 32 bits)
ACOS	ACOS	Fonction sur nombres à virgule flottante	Arc cosinus d'un nombre à virgule flottante (32 bits)
AD	UD	Combinaison sur mots	ET double mot
AN	UN	Combinaison sur bits	ET NON
AN(	UN(	Combinaison sur bits	ET NON d'une expression
ASIN	ASIN	Fonction sur nombres à virgule flottante	Arc sinus d'un nombre à virgule flottante (32 bits)
ATAN	ATAN	Fonction sur nombres à virgule flottante	Arc tangente d'un nombre à virgule flottante (32 bits)
AW	UW	Combinaison sur mots	ET mot (16 bits)
BE	BE	Gestion d'exécution de programme	Fin de bloc
BEC	BEB	Gestion d'exécution de programme	Fin de bloc conditionnelle

Abréviation anglaise	Abréviation allemande	Catalogue des éléments de programme	Description
BEU	BEA	Gestion d'exécution de programme	Fin de bloc inconditionnelle
BLD	BLD	Accumulateurs	Opération de composition d'image (opération nulle)
BTD	BTD	Conversions	Convertir DCB en entier de 32 bits
BTI	BTI	Conversions	Convertir DCB en entier de 16 bits
CAD	TAD	Conversions	Modifier l'ordre dans l'accumulateur 1 (32 bits)
CALL	CALL	Gestion d'exécution de programme	Appel de bloc
CALL	CALL	Gestion d'exécution de programme	Appel de multi-instance
CALL	CALL	Gestion d'exécution de programme	Appel de bloc dans une bibliothèque
CAR	TAR	Chargement/ Transfert	Permuter registre d'adresse 1 avec registre d'adresse 2
CAW	TAW	Chargement/ Transfert	Modifier l'ordre dans l'accumulateur 1 (16 bits)
CC	CC	Gestion d'exécution de programme	Appel de bloc conditionnel
CD	ZR	Compteurs	Décrémenter
CDB	TDB	Blocs de données	Permuter DB global et DB d'instance
CLR	CLR	Combinaison sur bits	Mettre RLG à 0
COS	COS	Fonction sur nombres à virgule flottante	Cosinus d'angles comme nombres à virgule flottante (32 bits)
CU	ZV	Compteurs	Incrémenter
–D	–D	Fonction sur nombres entiers	Soustraire accumulateur 1 de accumulateur 2 (entiers de 32 bits)
DEC	DEC	Accumulateurs	Décrémenter accumulateur 1-L-L
DTB	DTB	Conversions	Convertir entier de 32 bits en DCB
DTR	DTR	Conversions	Convertir entier de 32 bits en réel (IEEE 754 32 bits)
ENT	ENT	Chargement/ Transfert	Entrer dans pile accumulateur
EXP	EXP	Fonction sur nombres à virgule flottante	Valeur exponentielle d'un nombre à virgule flottante (32 bits)
FN	FN	Combinaison sur bits	Front descendant
FP	FP	Combinaison sur bits	Front montant
FR	FR	Compteurs	Valider compteur (validé, FR Z 0 à Z 255)
FR	FR	Temporisations	Valider temporisation
–I	–I	Fonction sur nombres entiers	Soustraire accumulateur 1 de accumulateur 2 (entiers de 16 bits)
INC	INC	Accumulateurs	Incrémenter accumulateur 1-L-L
INVD	INVD	Conversions	Complément à 1 d'entier de 32 bits
INVI	INVI	Conversions	Complément à 1 d'entier de 16 bits
ITB	ITB	Conversions	Convertir entier de 16 bits en DCB
ITD	ITD	Conversions	Convertir entier de 16 bits en entier de 32 bits
JBI	SPBI	Sauts	Saut si RB est 1
JC	SPB	Sauts	Saut si RLG est 1
JCB	SPBB	Sauts	Saut si RLG est 1 avec RB
JCN	SPBN	Sauts	Saut si RLG est 0
JL	SPL	Sauts	Saut vers liste

Abréviation anglaise	Abréviation allemande	Catalogue des éléments de programme	Description
JM	SPM	Sauts	Saut si moins
JMZ	SPMZ	Sauts	Saut si inférieur ou égal à 0
JN	SPN	Sauts	Saut si différent de 0
JNB	SPBNB	Sauts	Saut si RLG est 0 avec RB
JNBI	SPBIN	Sauts	Saut si RB est 0
JO	SPO	Sauts	Saut si DEB est 1
JOS	SPS	Sauts	Saut si DM est 1
JP	SPP	Sauts	Saut si plus
JPZ	SPPZ	Sauts	Saut si supérieur ou égal à 0
JU	SPA	Sauts	Saut inconditionnel
JUO	SPU	Sauts	Saut si illicite
JZ	SPZ	Sauts	Saut si égal à 0
L	L	Accumulateurs	Charger
L STW	L STW	Chargement/ Transfert	Charger mot d'état dans l'accumulateur 1
L	L	Temporisations	Charger valeur de temps en cours comme nombre entier dans l'accumulateur 1 (la valeur de temps en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : L T 32)
L	L	Compteurs	Charger valeur de comptage en cours comme entier dans l'accumulateur 1 (la valeur de comptage en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : L Z 15)
L DBLG	L DBLG	Blocs de données	Charger longueur de DB global dans l'accumulateur 1
L DBNO	L DBNO	Blocs de données	Charger numéro de DB global dans l'accumulateur 1
L DILG	L DILG	Blocs de données	Charger longueur de DB d'instance dans l'accumulateur 1
L DINO	L DINO	Blocs de données	Charger numéro de DB d'instance dans l'accumulateur 1
LAR1	LAR1	Chargement/ Transfert	Charger contenu de l'accumulateur 1 dans registre d'adresse 1
LAR1	LAR1	Chargement/ Transfert	Charger pointeur de 32 bits dans registre d'adresse 1
LAR1	LAR1	Chargement/ Transfert	Charger contenu du registre d'adresse 2 dans registre d'adresse 1
LAR2	LAR2	Chargement/ Transfert	Charger contenu de l'accumulateur 1 dans registre d'adresse 2
LAR2	LAR2	Chargement/ Transfert	Charger pointeur de 32 bits dans registre d'adresse 2
LC	LC	Compteurs	Charger valeur de comptage en cours comme nombre DCB dans l'accumulateur 1 (la valeur de comptage en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : LC Z 15)
LC	LC	Temporisations	Charger valeur de temps en cours comme nombre DCB dans l'accumulateur 1 (la valeur de temps en cours peut être un nombre compris dans la plage de 0 à 255, par exemple : LC T 32)
LEAVE	LEAVE	Accumulateurs	Quitter pile accumulateur
LN	LN	Fonction sur nombres à virgule flottante	Logarithme naturel d'un nombre à virgule flottante (32 bits)
LOOP	LOOP	Sauts	Boucle de programme
MCR(	MCR(	Gestion d'exécution de programme	Sauvegarder RLG dans pile MCR, début de zone MCR
)MCR	)MCR	Gestion d'exécution de programme	Fin de zone MCR

Abréviation anglaise	Abréviation allemande	Catalogue des éléments de programme	Description
MCRA	MCRA	Gestion d'exécution de programme	Activer la zone MCR
MCRD	MCRD	Gestion d'exécution de programme	Désactiver la zone MCR
MOD	MOD	Fonction sur nombres entiers	Reste de division entière (32 bits)
NEGD	NEGD	Conversions	Complément à 2 d'entier de 32 bits
NEGI	NEGI	Conversions	Complément à 2 d'entier de 16 bits
NEGR	NEGR	Conversions	Inverser nombre à virgule flottante (IEEE 754 32 bits)
NOP 0	NOP 0	Accumulateurs	Opération nulle
NOP 1	NOP 1	Accumulateurs	Opération nulle
NOT	NOT	Combinaison sur bits	Négation du RLG
O	O	Combinaison sur bits	OU
O(	O(	Combinaison sur bits	OU d'une expression
OD	OD	Combinaison sur mots	OU double mot (32 bits)
ON	ON	Combinaison sur bits	OU NON
ON(	ON(	Combinaison sur bits	OU NON d'une expression
OPN	AUF	Blocs de données	Ouvrir bloc de données
OW	OW	Combinaison sur mots	OU mot (16 bits)
POP	POP	Accumulateurs	POP CPU avec deux accumulateurs
POP	POP	Accumulateurs	POP CPU avec quatre accumulateurs
PUSH	PUSH	Accumulateurs	PUSH CPU avec deux accumulateurs
PUSH	PUSH	Accumulateurs	PUSH CPU avec quatre accumulateurs
R	R	Combinaison sur bits	Mettre à 0
R	R	Compteurs	Remettre compteur à 0 (le compteur actuel peut être un nombre compris dans la plage de 0 à 255, par exemple : R Z 15)
R	R	Temporisations	Remettre temporisation à 0 (la temporisation actuelle peut être un nombre compris dans la plage de 0 à 255, par exemple :R T 32)
–R	–R	Fonction sur nombres à virgule flottante	Soustraire accumulateur 1 de accumulateur 2 (réels IEEE 754, 32 bits)
RLD	RLD	Décalage/Rotation	Rotation vers la gauche d'un double mot (32 bits)
RLDA	RLDA	Décalage/Rotation	Rotation vers la gauche de l'accumulateur 1 via BI1(32 bits)
RND	RND	Conversions	Arrondir à l'entier
RND–	RND–	Conversions	Arrondir à l'entier inférieur
RND+	RND+	Conversions	Arrondir à l'entier supérieur
RRD	RRD	Décalage/Rotation	Rotation vers la droite d'un double mot (32 bits)
RRDA	RRDA	Décalage/Rotation	Rotation vers la droite de l'accumulateur 1 via BI1 (32 bits)
S	S	Combinaison sur bits	Mettre à 1
S	S	Compteurs	Initialiser compteur (le compteur actuel peut être un nombre compris dans la plage de 0 à 255, par exemple : S Z 15)
SAVE	SAVE	Combinaison sur bits	Sauvegarder RLG dans le bit RB
SD	SE	Temporisations	Temporisation sous forme de retard à la montée
SE	SV	Temporisations	Temporisation sous forme d'impulsion prolongée
SET	SET	Combinaison sur bits	Mettre à 1
SF	SA	Temporisations	Temporisation sous forme de retard à la retombée
SIN	SIN	Fonction sur nombres à virgule flottante	Sinus d'angles comme nombres à virgule flottante (32 bits)

Abréviation anglaise	Abréviation allemande	Catalogue des éléments de programme	Description
SLD	SLD	Décalage/Rotation	Décalage vers la gauche d'un double mot (32 bits)
SLW	SLW	Décalage/Rotation	Décalage vers la gauche d'un mot (16 bits)
SP	SI	Temporisations	Temporisation sous forme d'impulsion
SQR	SQR	Fonction sur nombres à virgule flottante	Carré d'un nombre à virgule flottante (32 bits)
SQRT	SQRT	Fonction sur nombres à virgule flottante	Racine carrée d'un nombre à virgule flottante (32 bits)
SRD	SRD	Décalage/Rotation	Décalage vers la droite d'un double mot (32 bits)
SRW	SRW	Décalage/Rotation	Décalage vers la droite d'un mot (16 bits)
SS	SS	Temporisations	Temporisation sous forme de retard à la montée mémorisé
SSD	SSD	Décalage/Rotation	Décalage vers la droite d'un entier avec signe (32 bits)
SSI	SSI	Décalage/Rotation	Décalage vers la droite d'un entier avec signe (16 bits)
T	T	Chargement/ Transfert	Transférer
T STW	T STW	Chargement/ Transfert	Transférer accumulateur 1 dans mot d'état
TAK	TAK	Accumulateurs	Permuter accumulateur 1 et accumulateur 2
TAN	TAN	Fonction sur nombres à virgule flottante	Tangente d'angles comme nombres à virgule flottante (32 bits)
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 dans l'accumulateur 1
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 à l'adresse de destination (32 bits)
TAR1	TAR1	Chargement/ Transfert	Transférer registre d'adresse 1 dans registre d'adresse 2
TAR2	TAR2	Chargement/ Transfert	Transférer registre d'adresse 2 dans l'accumulateur 1
TAR2	TAR2	Chargement/ Transfert	Transférer registre d'adresse 2 à l'adresse de destination (32 bits)
TRUNC	TRUNC	Conversions	Arrondir par troncature
UC	UC	Gestion d'exécution de programme	Appel de bloc inconditionnel
X	X	Combinaison sur bits	OU exclusif
X(	X(	Combinaison sur bits	OU exclusif d'une expression
XN	XN	Combinaison sur bits	OU NON exclusif
XN(	XN(	Combinaison sur bits	OU NON exclusif d'une expression
XOD	XOD	Combinaison sur mots	OU exclusif double mot (32 bits)
XOW	XOW	Combinaison sur mots	OU exclusif mot (16 bits)

B Exemples de programmation

B.1 Vue d'ensemble des exemples de programmation

Applications pratiques

Chacune des opérations LIST déclenche une fonction précise. En combinant ces opérations dans un programme, vous pouvez exécuter une grande variété de tâches d'automatisation. Vous trouvez dans la suite quelques exemples d'applications pratiques des opérations LIST :

- Commande d'un tapis roulant à l'aide d'opérations de combinaison sur bits
- Détection du sens de déplacement d'un tapis roulant à l'aide d'opérations de combinaison sur bits
- Génération d'une période d'horloge à l'aide d'opérations de temporisation
- Surveillance de l'espace de stockage à l'aide à l'aide d'opérations de comptage et de comparaison
- Calculs à l'aide d'opérations arithmétiques sur nombres entiers
- Réglage de la durée de chauffage d'un four

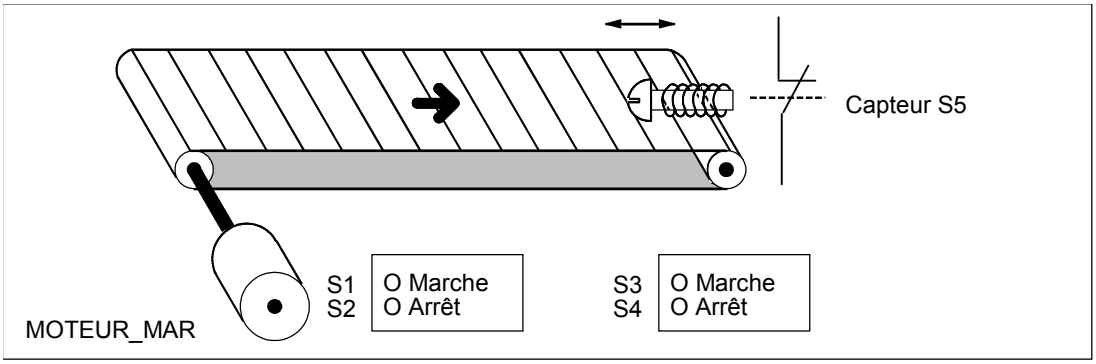
Opérations utilisées

Abréviation Allemande	Catalogue des éléments de programme	Description
UW	Combinaison sur mots	ET mot
OW	Combinaison sur mots	OU mot
ZV, ZR	Compteurs	Décrémenter, Incrémenter
S, R	Combinaison sur bits	Mettre à 1, Mettre à 0
NOT	Combinaison sur bits	Négation du RLG
FP	Combinaison sur bits	Front montant
+I	Fonction sur nombres entiers	Additionner entiers de 16 bits
/I	Fonction sur nombres entiers	Diviser entiers de 16 bits
*I	Fonction sur nombres entiers	Multiplier entiers de 16 bits
>=I, <=I	Comparaison	Comparer entiers de 16 bits
U, UN	Combinaison sur bits	ET, ET NON
O, ON	Combinaison sur bits	OU, OU NON
=	Combinaison sur bits	Affectation
INC	Accumulateurs	Incrémenter accumulateur 1
BE, BEB	Gestion d'exécution de programme	Fin de bloc, Fin de bloc conditionnelle
L, T	Chargement/Transfert	Charger, Transférer
SV	Temporisations	Temporisation sous forme d'impulsion prolongée

B.2 Exemples : Opérations combinatoires sur bits

Exemple 1 : Commande d'un tapis roulant

La figure suivante montre un tapis roulant pouvant être mis en route électriquement. Deux boutons-poussoirs, S1 pour MARCHE et S2 pour ARRET, se situent au début du tapis et deux, S3 pour MARCHE et S4 pour ARRET, à la fin du tapis. Il est donc possible de démarrer et d'arrêter le tapis à ses deux extrémités. D'autre part, le capteur S5 arrête le tapis lorsqu'un objet atteint la fin du tapis.



Programmation absolue et symbolique

Vous pouvez écrire le programme de commande du tapis roulant en représentant les diverses composants du système convoyeur à l'aide **d'adresses absolues** ou à l'aide de **mnémoniques**.

Vous mettez les mnémoniques choisis dans la table des mnémoniques en relation avec les adresses absolues (voir l'aide en ligne de STEP 7).

Composant du système	Adresse absolue	Mnémonique	Table de mnémoniques
Bouton-poussoir Marche	E 1.1	S1	E 1.1 S1
Bouton-poussoir Arrêt	E 1.2	S2	E 1.2 S2
Bouton-poussoir Marche	E 1.3	S3	E 1.3 S3
Bouton-poussoir Arrêt	E 1.4	S4	E 1.4 S4
Capteur	E 1.5	S5	E 1.5 S5
Moteur	A 4.0	MOTEUR_MAR	A 4.0 MOTEUR_MAR

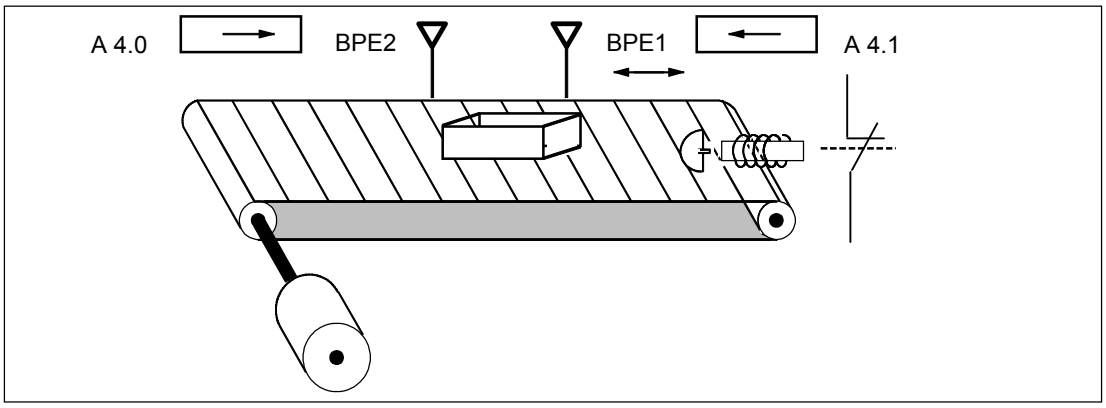
Programme absolue		Programme symbolique	
O	E 1.1	O	S1
O	E 1.3	O	S3
S	A 4.0	S	MOTOR_MAR
O	E 1.2	O	S2
O	E 1.4	O	S4
ON	E 1.5	ON	S5
R	A 4.0	R	MOTOR_MAR

Programme LIST pour commander un tapis roulant

```
LIST            Explication
O     E 1.1     //Appuyer sur l'un des deux boutons Marche fait démarrer le moteur.
O     E 1.3
S     A 4.0
O     E 1.2     //Appuyer sur l'un des deux boutons Arrêt ou ouvrir le contact à ouverture
                //à la fin du tapis arrête le moteur.
O     E 1.4
ON    E 1.5
R     A 4.0
```

Exemple 2 : Détection du sens de déplacement d'un tapis roulant

La figure suivante montre un tapis roulant équipé de deux barrières photoélectriques (BPE1 et BPE2) chargées de détecter le sens dans lequel se déplace un paquet sur le tapis. Chaque barrière photoélectrique fonctionne comme un contact à fermeture.



Programmation absolue et symbolique

Vous pouvez écrire le programme de commande du tapis roulant en représentant les diverses composants du système convoyeur à l'aide **d'adresses absolues** ou à l'aide de **mnémoniques**.

Vous mettez les mnémoniques choisis dans la table des mnémoniques en relation avec les adresses absolues (voir l'aide en ligne de STEP 7).

Composant du système	Adresse absolue	Mnémonique	Table de mnémoniques
Barrière photoélectrique 1	E 1.1	BPE1	E 0.0 BPE1
Barrière photoélectrique 2	E 0.0	BPE2	E 0.1 BPE2
Affichage pour mouvement vers la droite	A 4.0	DROITE	A 4.0 DROITE
Affichage pour mouvement vers la gauche	A 4.1	GAUCHE	A 4.1 GAUCHE
Mémento de cadence 1	M 0.0	MP1	M 0.0 MP1
Mémento de cadence 2	M 0.1	MP2	M 0.1 MP2

Programme absolue		Programme symbolique	
U	E 0.0	U	BPE 1
FP	M 0.0	FP	MP1
UN	E 0.1	UN	BPE 2
S	A 4.1	S	GAUCHE
U	E 0.1	U	BPE 2
FP	M 0.1	FP	MP2
UN	E 0.0	UN	BPE 1
S	A 4.0	S	DROITE
UN	E 0.0	UN	BPE 1
UN	E 0.1	UN	BPE 2
R	A 4.0	R	DROITE
R	A 4.1	R	GAUCHE

Programme LIST pour détecter le sens de déplacement d'un tapis roulant

```
LIST                Explication
U       E 0.0       //Si l'état de signal à l'entrée E 0.0 passe de 0 à 1 (front montant) et si
                    //l'état de signal à l'entrée E 0.1 est simultanément à 0, le paquet sur le
                    //tapis se déplace vers la gauche.
FP      M 0.0
UN      E 0.1
S       A 4.1
U       E 0.1       //Si l'état de signal à l'entrée E 0.1 passe de 0 à 1 (front montant) et si
                    //l'état de signal à l'entrée E 0.0 est simultanément à 0, le paquet sur le
                    //tapis se déplace vers la droite. Si l'une des barrières photoélectriques
                    //est interrompue, cela signifie qu'un paquet se trouve entre les deux
                    //barrières.
FP      M 0.1
UN      E 0.0
S       A 4.0
UN      E 0.0       //Si aucune des barrières photoélectriques n'est interrompue, aucun paquet
                    //ne se trouve entre les barrières. L'indicateur de sens se désactive.
UN      E 0.1
R       A 4.0
R       A 4.1
```

B.3 Exemple : Opérations de temporisation

Générateur d'horloge

Vous pouvez utiliser, pour produire un signal qui se répète périodiquement, un générateur d'impulsions d'horloge ou un relais clignotant. On trouve souvent des générateurs d'horloge dans les systèmes de signalisation qui commandent le clignotement des lampes de signalisation.

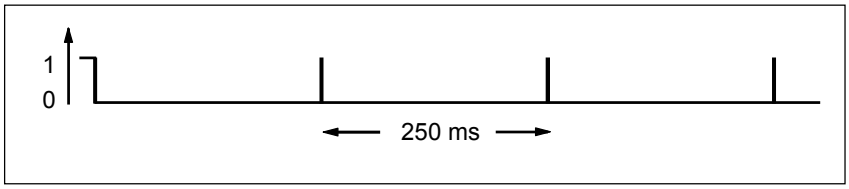
Dans l'automate S7-300, vous pouvez réaliser la génération d'impulsions d'horloge en utilisant le traitement commandé par horloge dans des blocs d'organisation spéciaux. Toutefois, l'exemple présenté dans le programme LIST suivant illustre l'utilisation de fonctions de temporisation pour générer une période d'horloge.

Programme LIST pour générer une période d'horloge (rapport d'impulsion 1:1)

```
LIST              Explication
UN      T1        //Lorsque la temporisation T1 s'est écoulée,
L       S5T#250ms //charger la valeur de temps 250 ms dans T1 et
SV      T1        //démarrer T1 sous forme d'impulsion prolongée.
NOT               //Inverser le résultat logique.
BEB               //Mettre fin au bloc en cours si la temporisation s'exécute.
L       MB100     //Après l'exécution de la temporisation, charger le contenu de l'octet de
                  //mémento MB100,
INC     1         //incrémenter le contenu de 1 et
T       MB100     //transférer le résultat dans l'octet de mémento MB100.
```

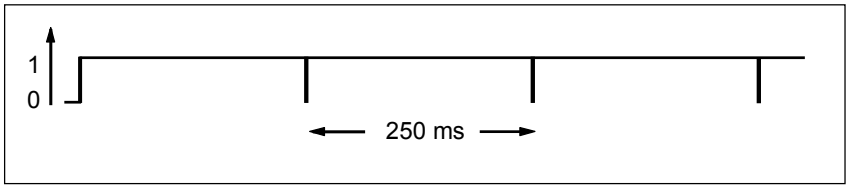
L'interrogation de l'état de signal

L'interrogation de l'état de signal de la temporisation T1 fournit le résultat logique RLG :



La temporisation est redémarrée une fois le temps écoulé. De ce fait, l'interrogation de l'état de signal par l'instruction **UN T1** ne délivre l'état de signal 1 que brièvement.

La figure montre comment se présente le bit RLG inversé.



Le bit RLG est égal à 0 toutes les 250 ms. L'opération BEB ne met alors pas fin au traitement du bloc, mais incrémente le contenu de l'octet de mémento MB100 de 1.

Le contenu de l'octet de mémento MB100 change toutes les 250 ms de la manière suivante :

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

Obtenir une fréquence précise

Vous pouvez obtenir les fréquences suivantes avec les bits de l'octet de mémento MB100 :

Bits de MB100	Fréquence en hertz	Durée	
M 100.0	2.0	0.5 s	(250 ms marche / 250 ms arrêt)
M 100.1	1.0	1 s	(0.5 s marche / 0.5 s arrêt)
M 100.2	0.5	2 s	(1 s marche / 1 s arrêt)
M 100.3	0.25	4 s	(2 s marche / 2 s arrêt)
M 100.4	0.125	8 s	(4 s marche / 4 s arrêt)
M 100.5	0.0625	16 s	(8 s marche / 8 s arrêt)
M 100.6	0.03125	32 s	(16 s marche / 16 s arrêt)
M 100.7	0.015625	64 s	(32 s marche / 32 s arrêt)

Programme LIST

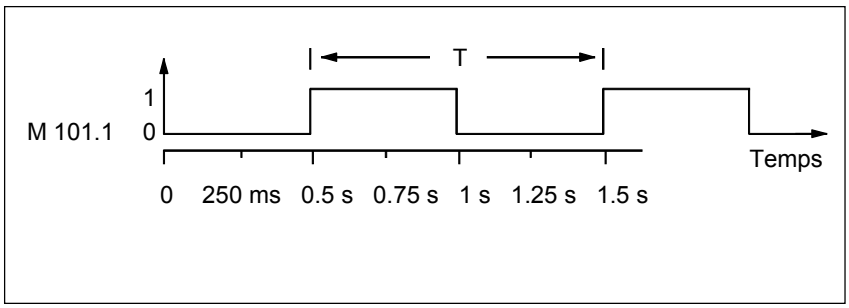
```
LIST              Explication
U     M10.0       //M 10.0 = 1 en cas d'erreur. La lampe d'erreur clignote à la fréquence de
                  //1 Hz en cas d'erreur.
U     M100.1
=     A 4.0
```

Etat de signal des bits de l'octet de mémento MB101

Cycle	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valeur de temps (ms)
0	0	0	0	0	0	0	**0**	0	250
1	0	0	0	0	0	0	**0**	1	250
2	0	0	0	0	0	0	1	0	250
3	0	0	0	0	0	0	1	1	250
4	0	0	0	0	0	1	**0**	0	250
5	0	0	0	0	0	1	**0**	1	250
6	0	0	0	0	0	1	1	0	250
7	0	0	0	0	0	1	1	1	250
8	0	0	0	0	1	0	**0**	0	250
9	0	0	0	0	1	0	**0**	1	250
10	0	0	0	0	1	0	1	0	250
11	0	0	0	0	1	0	1	1	250
12	0	0	0	0	1	1	**0**	0	250

Etat de signal du bit 1 du MB101 (M 101.1)

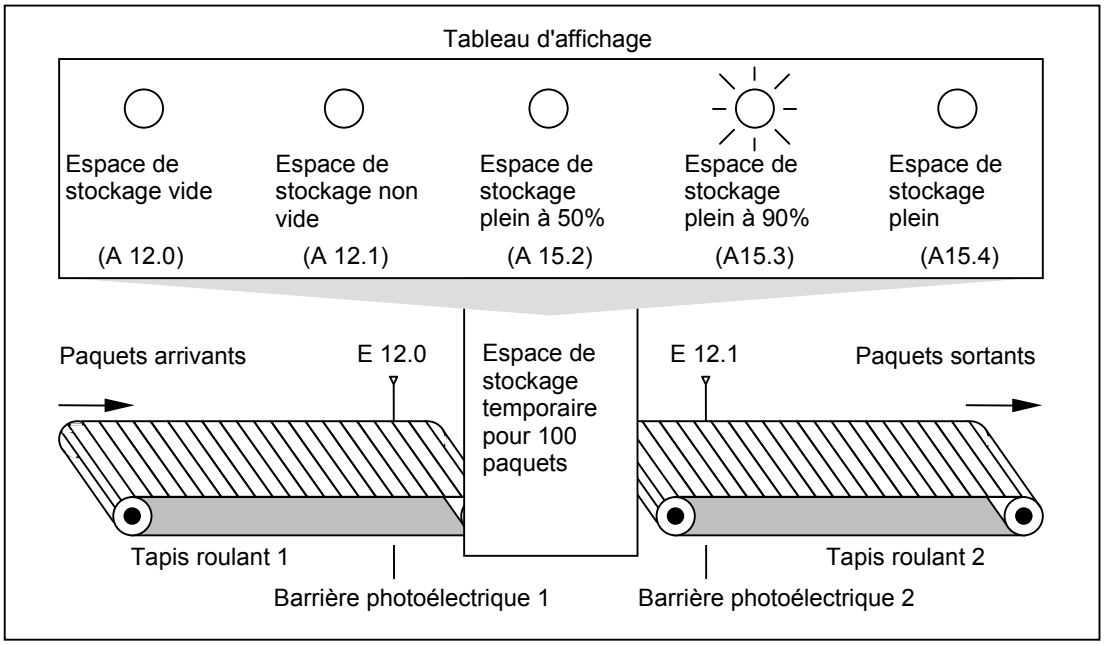
Fréquence = 1/T = 1/1 s = 1 Hz



B.4 Exemple : Opérations de comptage et de comparaison

Espace de stockage avec compteur et comparateur

La figure suivante montre un système avec deux tapis roulants et un espace de stockage temporaire entre eux. La tapis roulant 1 transporte les paquets dans l'espace de stockage. Une barrière photoélectrique à l'extrémité du tapis roulant 1, près de l'espace de stockage, détermine le nombre de paquets qui y sont amenés. Le tapis roulant 2 transporte les paquets de l'espace de stockage temporaire à une rampe de chargement d'où ils sont chargés dans des camions afin d'être livrés aux clients. Une barrière photoélectrique à l'extrémité du tapis roulant 2 près de l'espace de stockage détermine le nombre de paquets transportés de l'espace de stockage à la rampe de chargement. Un tableau d'affichage avec cinq lampes indique le niveau de remplissage de l'espace de stockage temporaire.



Programme LIST pour activer les lampes de signalisation sur un tableau d'affichage

```
U     E 0.0   //Chaque impulsion générée par la barrière photoélectrique 1
ZV    Z1      //augmente d'un la valeur du compteur Z1, comptant ainsi le nombre de paquets
              //transportés dans l'espace de stockage.
              //
U     E 0.1   //Chaque impulsion générée par la barrière photoélectrique 2
ZR    Z1      //diminue d'un la valeur du compteur Z1, comptant ainsi le nombre de paquets
              //quittant l'espace de stockage.
              //
UN    Z1      //Si la valeur du compteur est 0,
=     A 4.0   //la lampe de signalisation « Espace de stockage vide » s'allume.
              //
U     Z1      //Si elle est différente de 0,
=     A 4.1   //la lampe de signalisation « Espace de stockage non vide » s'allume.
              //
L     50
L     Z1
<=I           //Si 50 est inférieur ou égal à la valeur du compteur,
=     A 4.2   //la lampe de signalisation « Espace de stockage plein à 50 % » s'allume.
              //
L     90
>=I           //Si la valeur du compteur est supérieure ou égale à 90,
=     A 4.3   //la lampe de signalisation « Espace de stockage plein à 90 % » s'allume.
              //
L     Z1
L     100
>=I           //Si la valeur du compteur est supérieure ou égale à 100,
=     A 4.4   //la lampe de signalisation « Espace de stockage plein » s'allume (vous pouvez
              //également bloquer le tapis roulant 1 via la sortie A 4.4).
```

B.5 Exemple : Opérations arithmétiques sur nombres entiers

Calcul d'une Équation

L'exemple de programme suivant montre comment obtenir en utilisant trois opérations arithmétiques sur nombres entiers le même résultat que montre l'équation suivante :

MD4 = ((EW0 + DBW3) x 15) / MW2

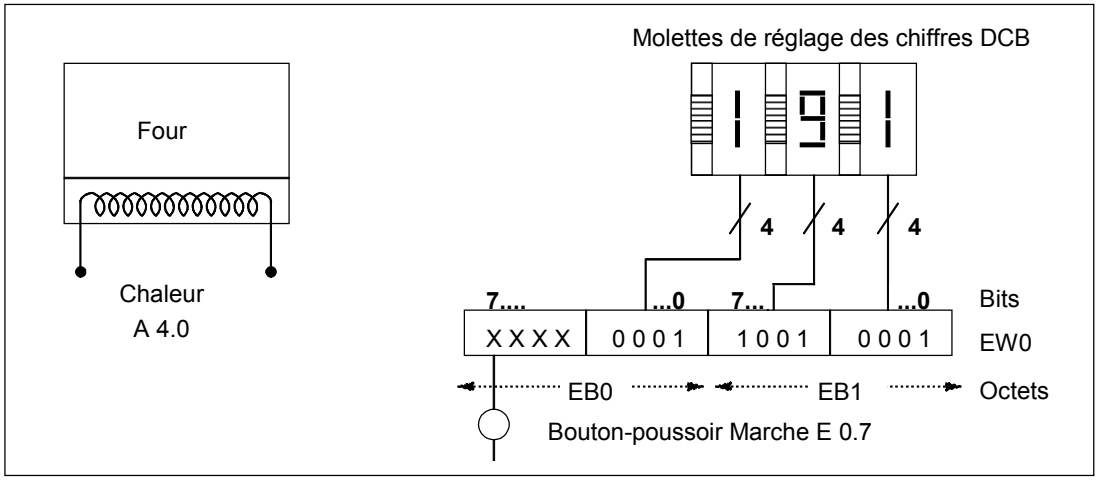
Programme LIST

```
LIST                Explication
L       EW0         //Charger dans l'accumulateur 1 la valeur figurant dans le mot d'entrée EW0.
L       DB5.DBW3    //Charger dans l'accumulateur 1 la valeur du mot de données global DBW3 du
                    //DB5. L'ancien contenu de l'accumulateur 1 est déplacé dans l'accumulateur 2.
+I      E 0.1       //Additionner le contenu des mots de poids faible des accumulateurs 1 et 2 et
                    //ranger le résultat dans le mot de poids faible de l'accumulateur 1. Le contenu
                    //de l'accumulateur 2 et le mot de poids fort de l'accumulateur 1 restent
                    //inchangés.
L       +15         //Charger dans l'accumulateur 1 la valeur constante +15. L'ancien contenu de
                    //l'accumulateur 1 est déplacé dans l'accumulateur 2.
*I                  //Multiplier le contenu du mot de poids faible de l'accumulateur 2 par le contenu
                    //du mot de poids faible de l'accumulateur 1 et ranger le résultat dans
                    //l'accumulateur 1. Le contenu de l'accumulateur 2 reste inchangé.
L       MW2         //Charger dans l'accumulateur 1 la valeur figurant dans le mot de mémento MW2.
                    //L'ancien contenu de l'accumulateur 1 est déplacé dans l'accumulateur 2.
/I                  //Diviser le contenu du mot de poids faible de l'accumulateur 2 par le contenu
                    //du mot de poids faible de l'accumulateur 1 et ranger le résultat dans
                    //l'accumulateur 1. Le contenu de l'accumulateur 2 reste inchangé.
T       MD4         //Transférer le résultat final dans le double mot de mémento MD4. Le contenu
                    //des deux accumulateurs reste inchangé.
```

B.6 Exemple : Opérations combinatoires sur mots

Chauffage d'un Four

L'opérateur du four déclenche le chauffage du four en appuyant sur le bouton-poussoir Marche. Il peut régler la durée du chauffage à l'aide des molettes représentées dans la figure La valeur indiquée donne les secondes en format décimal codé binaire (DCB).



Composant du système	Adresse absolue
Bouton-poussoir Marche	E 0.7
Molette de réglage des unités	E 1.0 à E 1.3
Molette de réglage des dizaines	E 1.4 à E 1.7
Molette de réglage des centaines	E 0.0 à E 0.3
Déclenchement du chauffage	A 4.0

Programme LIST

```
LIST            Explication
U     T1        //Si la temporisation s'exécute,
=     A 4.0     //déclencher le chauffage.
BEB             //Si la temporisation s'exécute, arrêter le traitement ici. Ainsi,
                //la temporisation T1 n'est pas redémarrée si le bouton-poussoir Marche
                //est enfoncé.
L     EW0
UW    W#16#0FFF //Masquer les bits d'entrée E 0.4 à E 0.7 (c'est-à-dire les remettre à 0).
                //La valeur de temps en secondes se trouve dans le mot de poids faible de
                //l'accumulateur 1 en format DCB.
OW    W#16#2000 //Affecter la base de temps en secondes dans les bits 12 et 13 du mot de poids
                //faible de l'accumulateur 1.
U     E 0.7
SV    T1        //Démarrer la temporisation T1 sous forme d'impulsion prolongée lorsque le
                //bouton-poussoir est enfoncé.
```

C Transmission de paramètres

Les paramètres d'un bloc sont transmis sous forme de valeur. Pour les blocs fonctionnels, une copie de la valeur du paramètre effectif est utilisée dans le DB d'instance au sein du bloc appelé. Pour les fonctions, une copie de la valeur effective se trouve dans la pile des données locales. Les pointeurs ne sont pas copiés. Avant l'appel, les valeurs INPUT sont copiées dans le DB d'instance ou la pile L. Après l'appel, les valeurs OUTPUT sont recopiées dans les variables. Seules des copies sont utilisées au sein du bloc appelé. Les instructions LIST requises se trouvent dans le bloc appelant et restent transparentes à l'utilisateur.

Nota

Si des mémentos, entrées, sorties, périphéries d'entrée ou de sortie sont utilisés en tant qu'opérandes effectifs dans une fonction, ils sont traités de manière différente que les autres opérandes. Leur actualisation n'est effectuée au moyen de la pile L, mais de manière directe.

Important

Lors de la programmation du bloc appelé, veillez à compléter les paramètres déclarés comme OUTPUT, sans quoi les valeurs fournies seront aléatoires ! Pour les blocs fonctionnels, on obtiendrait la valeur du DB d'instance inscrite lors du dernier appel, pour les fonctions, la valeur aléatoire se trouvant dans la pile L.

Tenez compte des points suivants :

- Si possible, initialisez tous les paramètres OUTPUT.
- Evitez l'utilisation d'instructions de mise à 1 et de remise à 0, car elles dépendent du RLG. Lorsque le RLG prend la valeur 0, c'est la valeur aléatoire qui est conservée !
- Lorsque vous effectuez un saut au sein du bloc, faites attention de ne pas sauter une ligne dans laquelle sont décrits des paramètres OUTPUT. Tenez également compte de BEB et de l'effet des instructions MCR.

Index

D

E

F

I

L

M

N

O

P

Q

R

S

T

U

V

X

Z