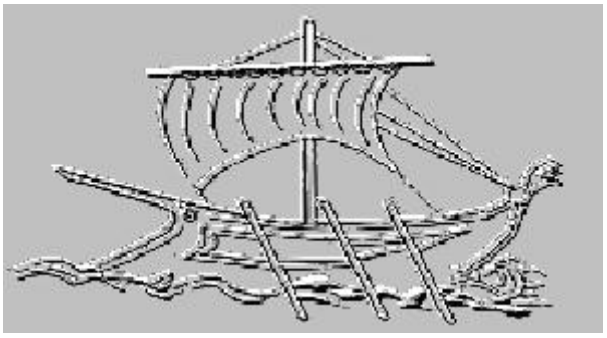
ΤΕΙ ΠΕΙΡΑΙΑ



ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΑΥΤΟΜΑΤΙΣΜΟΥ

SET ΕΝΤΟΛΩΝ PLC SIMATIC S7

Πειραιάς Δεκέμβριος 2005

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΤΟΥ PLC- SET ΕΝΤΟΛΩΝ

5.1 Γενικά

Τα PLC από μόνα τους είναι ουδέτερες συσκευές αφού δεν είναι από πριν κατασκευασμένες για μια συγκεκριμένη εφαρμογή. Κάθε φορά , ανάλογα με τις απαιτήσεις της εκάστοτε εγκατάστασης προγραμματίζονται να κάνουν την μεν ή δε ενέργεια.

Υπάρχουν διάφοροι τρόποι προγραμματισμού που ποικίλουν ακριβώς γιατί ποικίλουν και τα επίπεδα γνώσης και εμπειριών του κάθε προγραμματιστή. Οι ουσιαστικές διαφορές είναι στο τι βλέπουμε στην οθόνη του υπολογιστή μας ,αφού το τελικό αποτέλεσμα είναι πάντα ένα και το αυτό . Οι διάφορες γλώσσες μετατρέπονται σε γλωσσά μηχανής MC7 (Machine Code 7) κατά την μεταφορά του προγράμματος από την συσκευή προγραμματισμού στο PLC.

Υπάρχουν τρεις τυποποιημένες μορφές προγραμματισμού που έχουν επικρατήσει διεθνώς:

- Η λίστα εντολών **STL**
- Σχέδιο επαφών **LAD**
- Διάγραμμα λογικών πυλών **FDB**

Λεπτομερή αναφορά για την κάθε μια έχει γίνει σε προηγούμενο κεφάλαιο. Παρακάτω θα αναφερθούμε σε βασικές εντολές προγραμματισμού σε μορφή **STL**

5.2 Τύποι δεδομένων

Τα δεδομένα που πρόκειται να χρησιμοποιηθούν σε ένα πρόγραμμα προσδιορίζονται από κάποιο τύπο δεδομένων. Ορίζοντας έναν τύπο δεδομένων ορίζουμε και το μέγεθος των δεδομένων καθώς και τη δομή των bit τους. Το Step 7 προσφέρει διάφορες μορφές για την αναπαράσταση αριθμητικών τιμών που τις χρησιμοποιούμε για να δώσουμε στο σύστημα κάποιες τιμές ή να επιτηρούμε τα δεδομένα.

Ένα δεδομένο μπορεί να έχει μήκος από 1 bit ως και 32 bits. Από αυτή την άποψη λοιπόν τα δεδομένα χωρίζονται σε Bool, Byte, Char, Int, Word, Dint, Dword:

- **Bool:** Όπως ήδη ξέρουμε τα Bool δεδομένα είναι μήκους 1 bit και έτσι η τιμή τους μπορεί να είναι είτε Αληθής 1 είτε Ψευδής 0.

- **Byte:** Τα Bytes είναι προσημασμένοι αριθμοί με μήκος 8 bits και έτσι μπορεί να πάρει τιμή από -128 ως +127.
- **Char:** Αυτός ο τύπος δεδομένων περιλαμβάνει τους ASCII χαρακτήρες οι οποίοι έχουν μήκος 8 bits.
- **Int:** Είναι προσημασμένοι αριθμοί με μήκος 16 bit που μπορούν να πάρουν τιμή από –32768 ως +32767.
- **Word:** Έχει μήκος 2 Byte ή 16 bits. Η διαφορά του από τους Int αριθμούς, είναι ότι περιέχει δυο ανεξάρτητους διαδοχικούς αριθμούς των 8 bits η καθεμία, ενώ οι Int περιέχουν έναν ενιαίο αριθμό των 16 bits.
- **Dint:** Είναι 32 Bits προσημασμένοι αριθμοί με τιμή από – 2147483648 ως +2147483647.
- **Dword:** Είναι μήκους 4 Byte ή 32 bits. Όπως τα words, και τα Dwords περιέχουν τέσσερις διαδοχικούς αριθμούς των 8 bits.

5.3 Status Word

Σε κάθε μικροεπεξεργαστή οι εντολές επηρεάζουν κάποιες "σημαίες". Το ίδιο συμβαίνει και εδώ. Όλες οι σημαίες βρίσκονται σε μια λέξη που ονομάζεται **Status Word**. Μετά από κάθε εντολή μπορούμε να χρησιμοποιήσουμε άμεσα τα Bits που επηρεάστηκαν και έτσι να κατευθύνουμε το πρόγραμμά μας. Το Status word έχει 9 bits που η λειτουργία τους φαίνεται παρακάτω:

- **BR Binary Result:** Μας βοηθά να επεξεργαστούμε εντολές που αφορούν words σαν να ήταν εντολές που αφορούν Bits. Δηλαδή, αν μια εντολή έχει σαν αποτέλεσμα έναν οποιοδήποτε αριθμό (π.χ. πρόσθεση δυο ακεραίων), μέσω του BR μπορούμε να επεξεργαστούμε το αποτέλεσμα σαν να ήταν Bit και όχι αριθμός, γιατί αν γίνει η πρόσθεση το BR θα γίνει "1". Αν Δε γίνει θα μείνει "0".

- **CC1 και CC0 Condition Codes:** Επηρεάζονται από μαθηματικές εντολές και μας δίνουν πληροφορίες για το αποτέλεσμα. Π.χ. σε μια αφαίρεση, ανάλογα με πια bit είναι "1", μας λέει αν το αποτέλεσμα της ήταν μεγαλύτερο, μικρότερο ή ίσο του μηδενός.
- **OV Overflow:** Γίνεται "1" όταν προκύπτει κάποιο λάθος σε μαθηματικές πράξεις, όπως π.χ. το αποτέλεσμα να είναι μεγαλύτερο από το όριο. Γίνεται "0" αν το λάθος αποκατασταθεί.

- **OS Overflow Stored:** Γίνεται "1" αν και το Bit OV γίνει "1", και διατηρεί την κατάσταση αυτή ακόμα και αν το Bit OV γίνει "0". Δηλαδή να γίνει λάθος σε μια εντολή το Bit θα είναι Set σε όλη τη διάρκεια του προγράμματος.
- **OR:** Χρησιμοποιείται από την Statement List γλώσσα προγραμματισμού.
- **STA Status Bit:** Παίρνει την τιμή που έχει το τρέχον bit σε μια Bit Logic εντολή.
- **RLO Result of Logic Operation:** Αποθηκεύει το αποτέλεσμα της λογικής επεξεργασίας μετά από κάθε εντολή.
- **/FC First Check:** Είναι Bit ελέγχου του μικροεπεξεργαστή που δεν θα μας χρησιμεύσει η λειτουργία του.

5.4 Εντολές λογικών μανδαλώσεων

Οι εντολές αυτές χρησιμεύουν για την πραγματοποίηση λογικών μανδαλώσεων. Στον κλασσικό αυτοματισμό με ρελέ , τέτοιου είδους μανδαλώσεις δημιουργούνται συνδέοντας επαφές σε σειρά ή παράλληλα. Με το ίδιο σκεπτικό , υπάρχουν εντολές στον προγραμματιζόμενο αυτοματισμό, που πραγματοποιούν αυτή τη φιλοσοφία.

5.4.1 Εντολές And, And Not, Or, Or Not

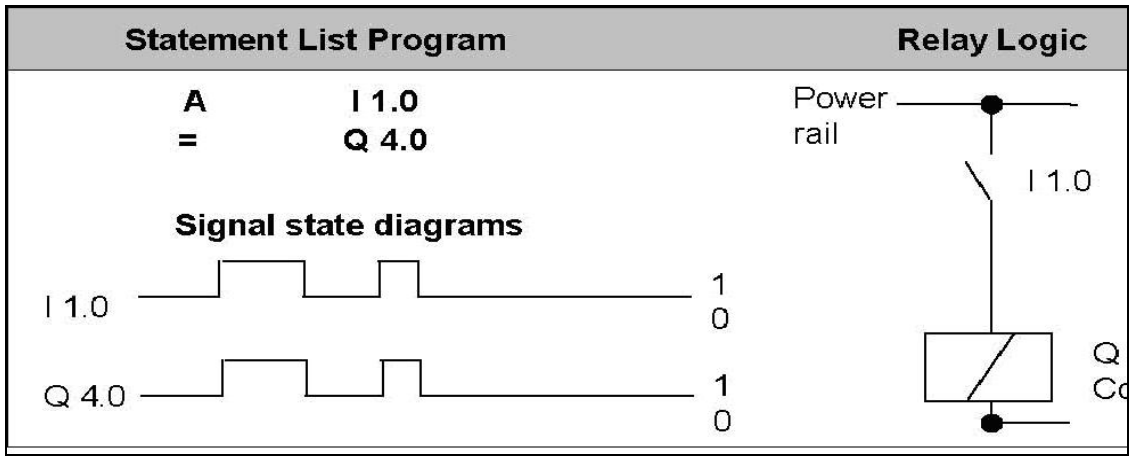
Η κανονικά ανοικτή επαφή είναι κλειστή όταν η τιμή του bit της διεύθυνσης n είναι ίση με 1.

Στην STL, η κανονικά ανοικτή επαφή αναπαριστάτε από τις εντολές And και Or. Αυτές οι εντολές φορτώνουν, κάνουν λογικό And ή λογικό Or αντίστοιχα, την τιμή του bit της διεύθυνσης n με την κορυφή του σωρού.

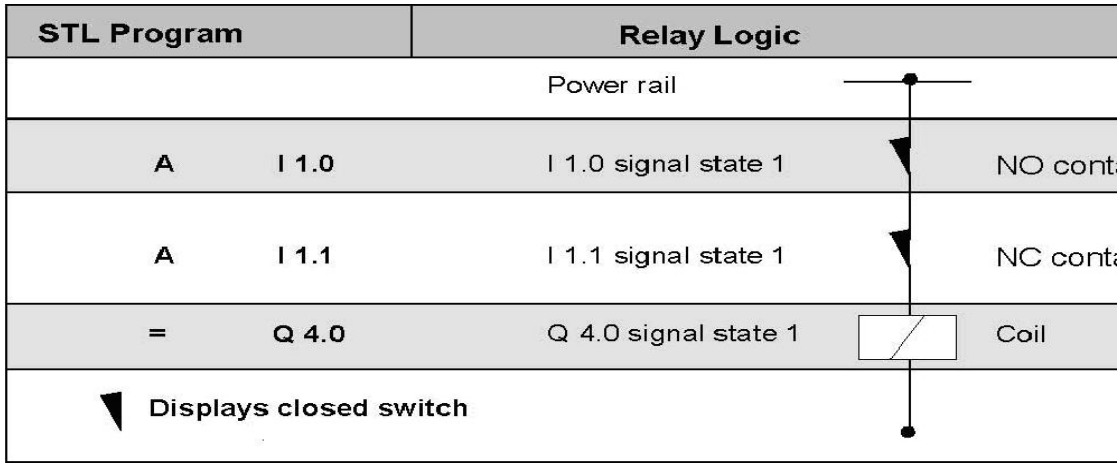
Η κανονικά κλειστή επαφή είναι κλειστή όταν η τιμή του bit της διεύθυνσης n είναι ίση με 0.

Στην STL, η κανονικά κλειστή επαφή αναπαρίσταται από τις εντολές And Not, και Or Not. Αυτές οι εντολές φορτώνουν, κάνουν λογικό And ή λογικό Or αντίστοιχα, το λογικό όχι της τιμής του bit της διεύθυνσης n με την κορυφή του σωρού.

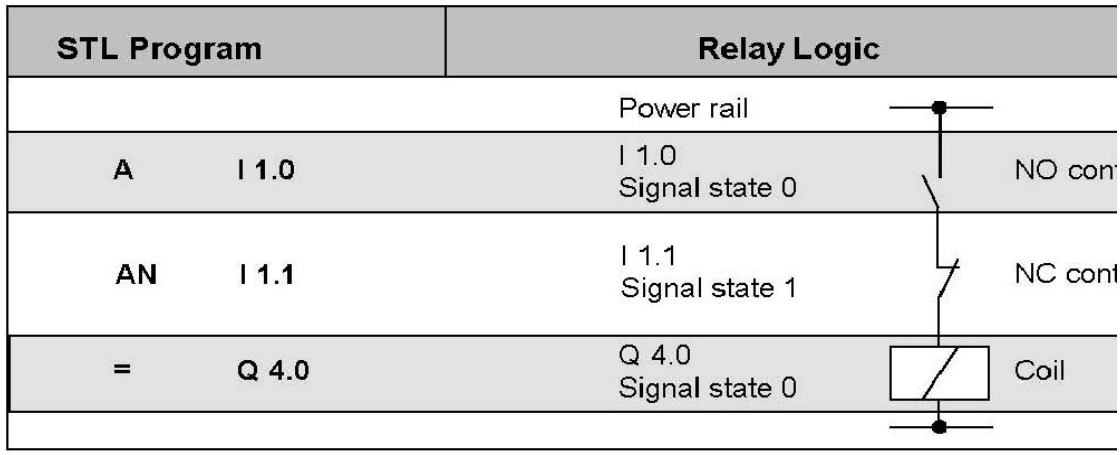
Στα παρακάτω σχήματα έχουμε κάποια παραδείγματα σύνταξης των εντολών στην STL.



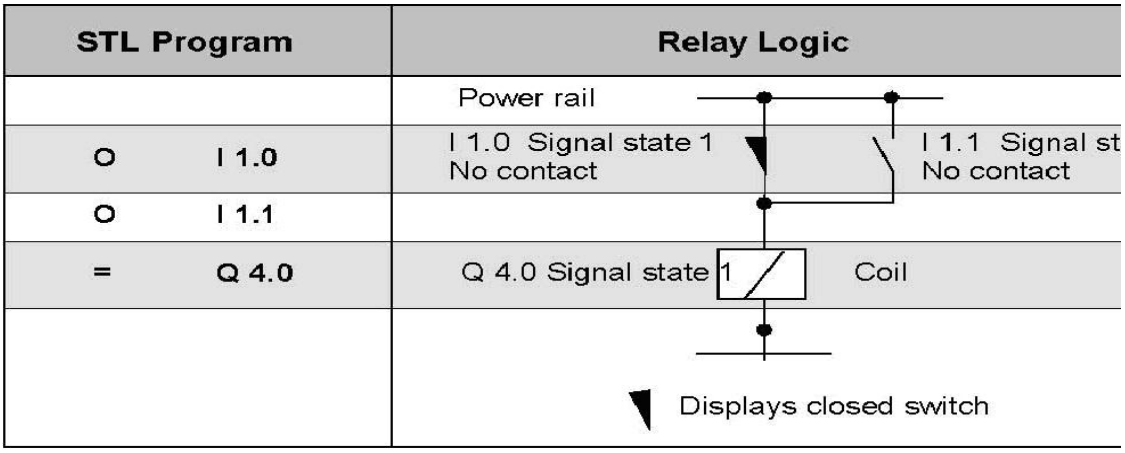
Σχήμα 5.1. Σύνταξη εντολής And



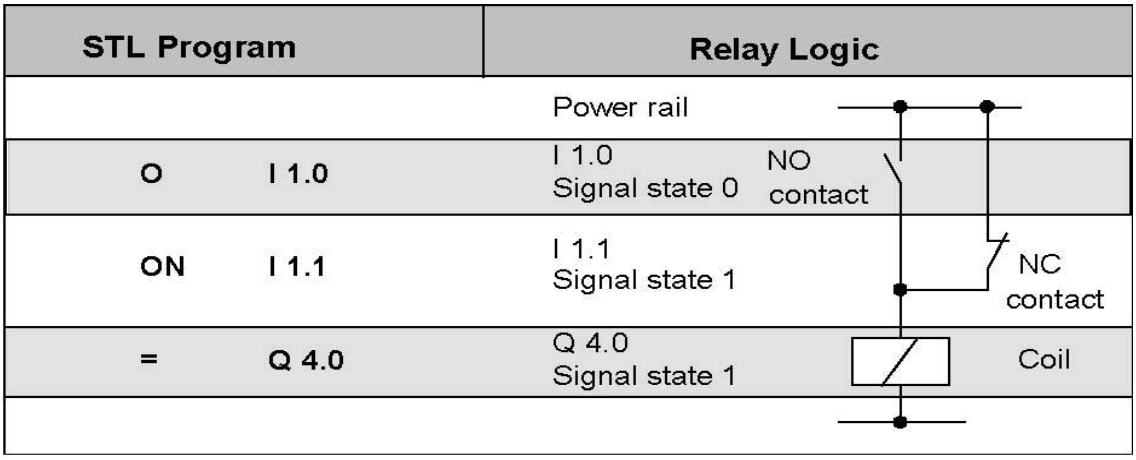
Σχήμα 5.2. Σύνταξη δύο εντολών And



Σχήμα 5.3. Σύνταξη μιας εντολής And και μιας εντολής And Not



Σχήμα 5.4. Σύνταξη εντολής Or



Σχήμα 5.5. Σύνταξη μιας εντολής Or και μιας εντολής Or Not

5.5 Εντολές διέγερσης

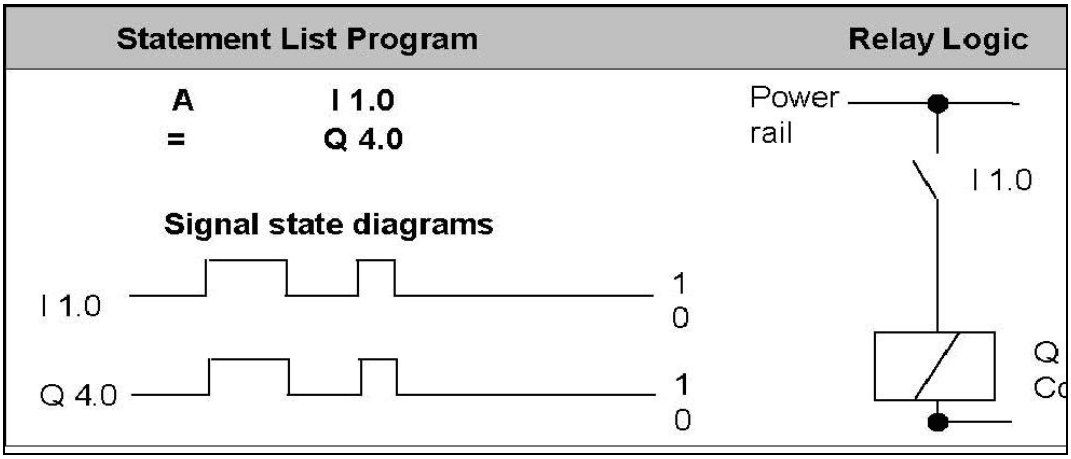
Οι εντολές διέγερσης μεταφέρουν στο στοιχείο πάνω στο οποίο επενεργούν το αποτέλεσμα της λογικής πράξης το οποίο προέκυψε από μια λογική μανδάλωση.

Οι εντολές αυτές μπορούν να είναι:

- Διέγερσης χωρίς αυτοσυγκράτηση (εντολή =)
- Διέγερση με αυτοσυγκράτηση (εντολή Set,Reset)

5.5.1 Εντολή =

Η εντολή = είναι δυναμική και έχει την ιδιότητα να κρατά ενεργοποιημένη μια έξοδο για όσο διάστημα το RLO που δημιουργήθηκε από τις εντολές που προηγήθηκαν είναι 1. Η κατάσταση του RLO μεταφέρεται στο στοιχείο που ορίζουμε (Q,M,D)



Σχήμα 5.6.

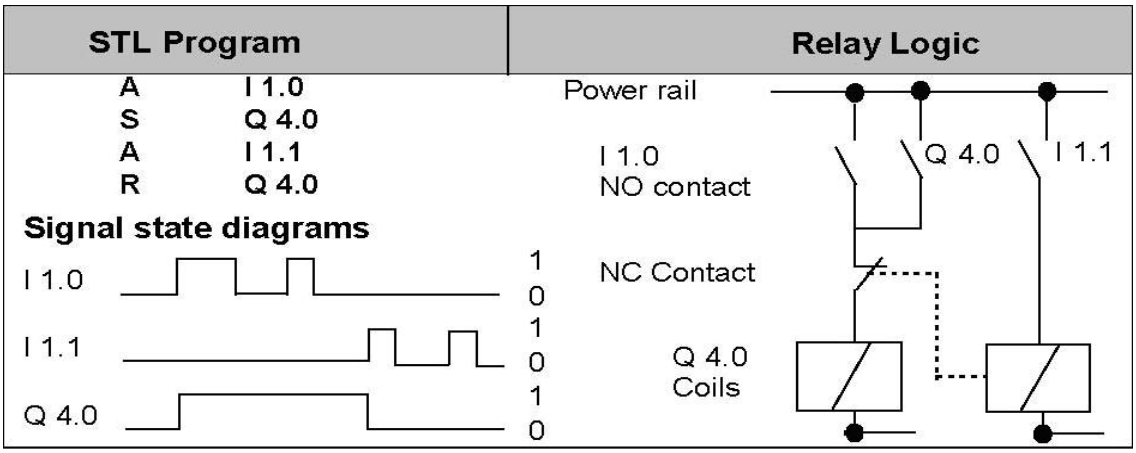
Στο σχήμα 5.6 για όσο διάστημα η είσοδος I1.0 θα έχει σήμα <<1>> ,θα είναι ενεργοποιημένη και η έξοδος Q4.0

5.5.2 Εντολές RESET και SET

Η εντολή **Reset** θα δώσει αποτέλεσμα "0", δηλαδή θα κάνει Reset τη διεύθυνση που αντιπροσωπεύει, αν φτάσει μέχρι εκεί ρεύμα. Αν δεν φτάσει η διεύθυνση θα διατηρήσει την προηγούμενη κατάστασή της.

Η εντολή **Set** Θα δώσει αντίστοιχα αποτέλεσμα "1", δηλαδή θα κάνει Set την αντίστοιχη διεύθυνση, αν φτάσει ρεύμα μέχρι εκεί.

Οι εντολές αυτές συνήθως χρησιμοποιούνται για θέσεις μνήμης, όμως μπορούν να χρησιμοποιηθούν και για εξόδους. Για να κατανοήσετε καλύτερα τη διαφορά των εντολών SET, RESET με την εντολή Πηνίο Εξόδου, όταν αυτές χρησιμοποιούνται για εξόδους και όχι για θέσεις μνήμης, θέτουμε το παρακάτω παράδειγμα:



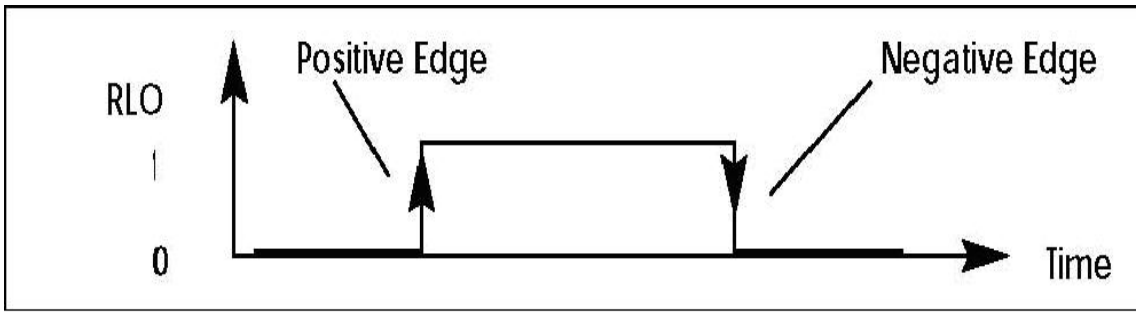
Σχήμα 5.7

Στο σχήμα 5.7 η έξοδος Q4.0 θα ενεργοποιηθεί μόνο όταν στην είσοδο I1.0 έχουμε σήμα <<1>> και στην είσοδο I1.1 έχουμε σήμα <<0>>. Αν η είσοδος I1.0 χάσει το σήμα 1 η έξοδος Q4.0 θα παραμείνει ενεργοποιημένη , θα απενεργοποιηθεί μόνο όταν η είσοδος I1.1 έχει σήμα <<1>>.

Να σημειώσουμε εδώ ότι οι δύο εντολές Set και Reset αλλάζουν την κατάσταση των διευθύνσεων που αναφέρονται μόνο με "1" στην είσοδό τους. Με "0" η διεύθυνση διατηρεί την προηγούμενη κατάστασή της.

5.6 Εντολές αναγνώρισης παρυφών

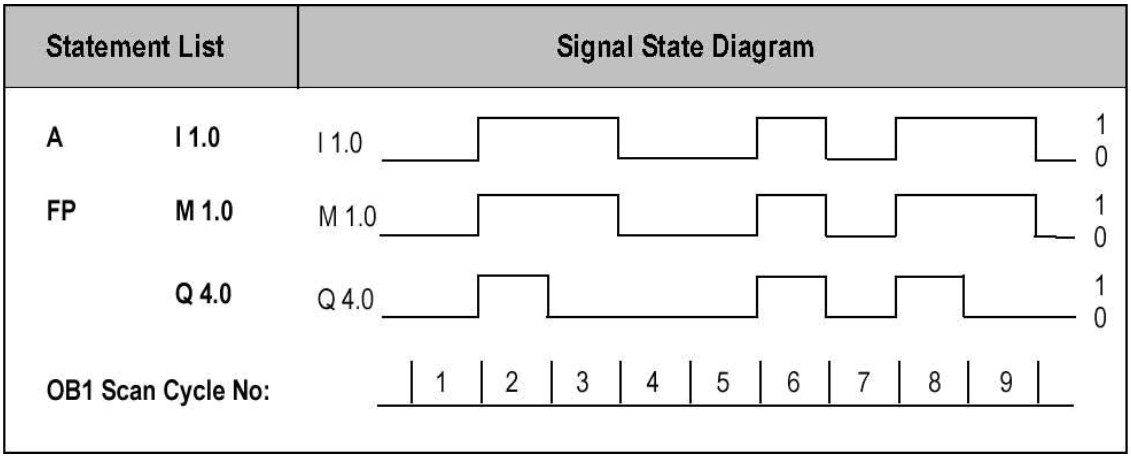
Υπάρχουν περιπτώσεις που θέλουμε να παράγουμε μια εντολή μόνο κατά την αλλαγή κατάστασης ενός σήματος (π.χ. από <<0>> σε <<1>>) και όχι κάθε όλο το επόμενο διάστημα της παραμονής του σήματος στη νέα κατάσταση. Για το σκοπό αυτό υπάρχουν εντολές που εκμεταλλεύονται αυτές τις αλλαγές αναγνωρίζοντας τις παρυφές των παλμών θετικές ή αρνητικές .



Η μεταγωγή από <<0>> σε <<1>> ονομάζεται θετική παρυφή ενώ η μεταγωγή από <<1>> σε <<0>> αρνητική.

5.6.1 Εντολή FP

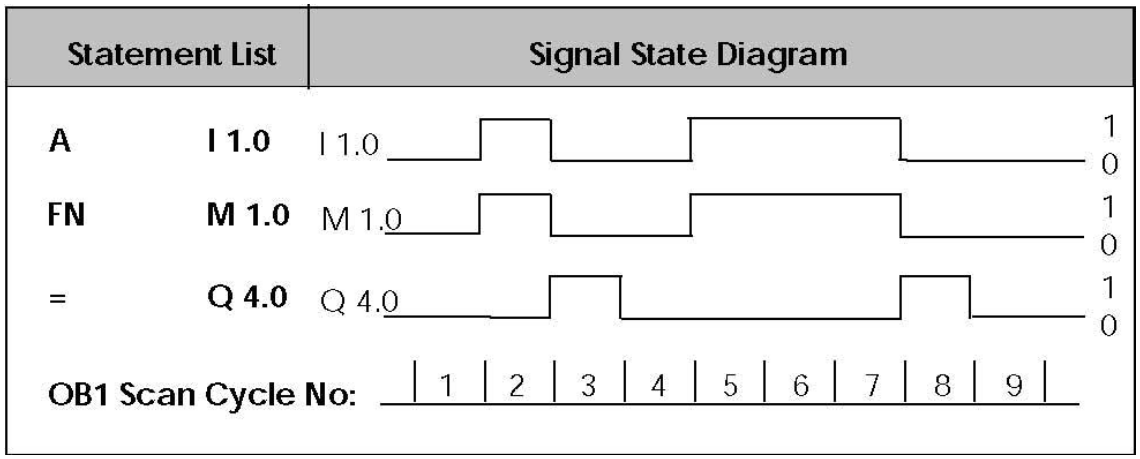
Η εντολή αυτή ελέγχει την αλλαγή του RLO από <<0>> σε <<1>> και επιδρά στην έξοδο δίνοντας σήμα <<1>> για ένα κύκλο του προγράμματος . Για να επιτραπεί στο σύστημα να αναγνωρίσει την αλλαγή , το RLO αποθηκεύεται σε ένα Memory bit ή σε ένα Data bit. Στο παράδειγμα που ακολουθεί ,για την αποθήκευση χρησιμοποιείται το M1.0



Σχήμα 5.8

5.6.2 Εντολή FN

Η εντολή αυτή ελέγχει την αλλαγή του RLO από <<1>> σε <<0>> και επιδρά στην έξοδο δίνοντας σήμα <<1>> για ένα κύκλο του προγράμματος . Για να επιτραπεί στο σύστημα να αναγνωρίσει την αλλαγή , το RLO αποθηκεύεται σε ένα Memory bit ή σε ένα Data bit. Στο παράδειγμα που ακολουθεί ,για την αποθήκευση χρησιμοποιείται το M1.0



Σχήμα 5.9

5.7 Χρονικά

Τα χρονικά ή χρονιστές όπως ονομάζονται, είναι πολύ χρήσιμα εργαλεία στον προγραμματισμό του PLC, καθώς μας δίνουν τη δυνατότητα να εισάγουμε μια συγκεκριμένη καθυστέρηση πριν την εκτέλεση κάποιας εντολής ή ακόμα μπορούμε να μετρήσουμε το χρόνο που διήρκεσε ένα γεγονός. Τα χρονικά μπορούν να μετρήσουν χρόνους μέχρι και 9990 δευτερόλεπτα ή 2 ώρες 46 λεπτά και 30 δευτερόλεπτα. Ο μικρότερος χρόνος που μπορεί να μετρηθεί είναι 10 msec. Η τιμή του χρόνου για ένα χρονικό δίνεται με την εντολή L συνοδευόμενη πάντα από την τιμή του χρόνου.

L S5T#αH_bM_cS_dMS

όπου H=Hours (ώρες), M=Minutes (λεπτά), S=Seconds (δευτερόλεπτα), MS=Millisecs (χιλιοστά δευτερολέπτου) και a,b,c,d είναι νούμερα.

Όμως, υπάρχει ένας κανόνας για την εισαγωγή του χρόνου. Αν ο χρόνος που θέλουμε να εισάγουμε είναι μέχρι 9sec και 990msec, τότε σαν μονάδα χρόνου ορίζεται τα 10msec. Αν είναι από 10sec μέχρι 1m39sec900msec (99sec900msec), τότε σαν μονάδα χρόνου τίθεται τα 100msec. Αν είναι από 1m40sec (100sec) μέχρι 16m39sec (999sec), τότε η μονάδα χρόνου είναι 1sec. Τέλος αν είναι από 16m40sec (1000sec) μέχρι και 2h46m30sec (9990sec), τότε η μονάδα χρόνου είναι τα 10sec. Δηλαδή, αν δώσετε στο χρονικό την εντολή: S5T#35S450MS, εσείς θα περιμένετε να μετρήσει 35sec και 450msec. Όμως σε αυτή την περίπτωση, μονάδα χρόνου είναι τα 100msec και έτσι το χρονικό θα μετρήσει 35sec και 400msec. Στον παρακάτω πίνακα 7.1 φαίνονται καλύτερα οι μονάδες σε κάθε πεδίο χρόνου.

Πίνακας 5.1

Μονάδα	Πεδίο
10msec	Από 10msec μέχρι 990msec
100msec	Από 1sec μέχρι 99sec900msec
1sec	Από 100sec μέχρι 999sec
10sec	Από 1000sec μέχρι 9990sec

Υπάρχουν πέντε τύποι χρονικών :

Τύπος	Περιγραφή
SP **Pulse Timer**	Το χρονικό τρέχει όσο το σήμα στην ακίδα S είναι "1". Αν γίνει"0" τότε το χρονικό θα σταματήσει και θα βγάλει στην έξοδο Q "0".
SE **Extended Pulse Timer**	Η έξοδος του χρονικού παραμένει "1" για όλο τον χρόνο που έχει προγραμματιστεί, ακόμα κι αν το σήμα στην ακίδα S έχει γίνει "0".
SD **On- delay Timer**	Η έξοδος θα γίνει "1" μόνο όταν περάσει. ο χρόνος που έχουμε θέσει στο χρονικό και το σήμα στην είσοδο είναι ακόμα "1".
SS **Retentive on- delay Timer**	Η έξοδος θα γίνει "1" μόνο όταν περάσει ο χρόνος που έχουμε θέσει, ακόμα και αν το σήμα στην είσοδο έχει γίνει "0".
SF **Off- delay Timer**	Αυτό το χρονικό φορτώνει τον προκαθορισμένο χρόνο όταν έρθει "1" στην ακίδα S αλλά, αντίθετα με τα άλλα, θέλει αρνητικό παλμό για να ξεκινήσει. Η έξοδος του θα μείνει άσος για όσο χρόνο θα μετράει το χρονικό και αν ξαναέρθει "1" στην είσοδο θα γίνει Reset.

5.7.1 Χρονικό παλμού SP

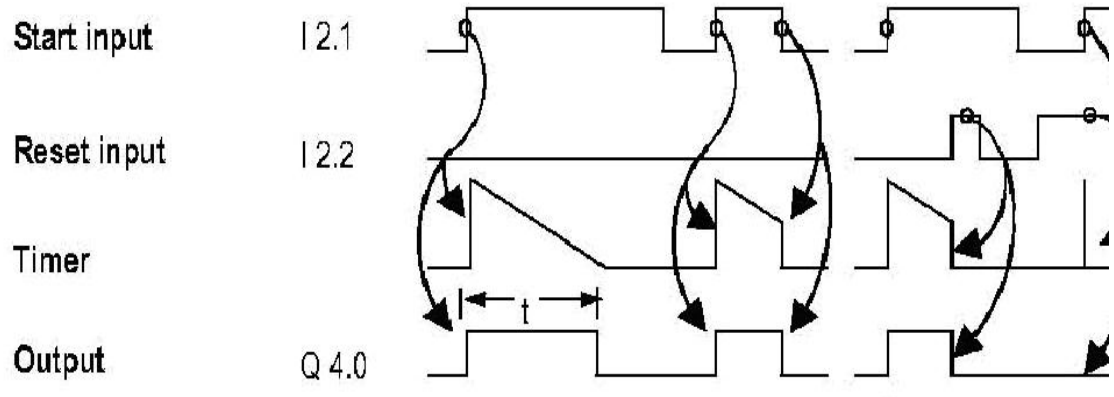
Το χρονικό αυτό είναι από τα πιο συχνά χρησιμοποιούμενα χρονικά. Εφαρμόζεται σε περιπτώσεις που θέλουμε να ανεξαρτητοποιήσουμε τη διάρκεια ενός γεγονότος από την έξοδο που θα παράγουμε.

Το χρονικό αυτό όπως και όλα τα υπόλοιπα που θα δούμε στη συνέχεια για να λειτουργήσει θα πρέπει να ακολουθήσουμε μια συγκεκριμένη διαδικασία ,σε τρία βήματα:

- Να δηλώσουμε την αιτία εκκίνησης του χρονικού.(π.χ. A I2.1)
- Χρόνος που πρέπει να καταμετρηθεί (π.χ. L S5T#10S)
- Εκκίνηση του χρονικού (π.χ. SP T1)

Παράδειγμα:

```
A    I 2.1
L    S5T#10s     //Preset 10 seconds into ACCU 1.
SP   T1          //Start timer T1 as a pulse timer.
A    I 2.2
R    T1          //Reset timer T1.
A    T1          //Check signal state of timer T1.
=    Q 4.0
```



Από το παραπάνω διάγραμμα του χρονικού διακρίνουμε τρεις περιπτώσεις όσον αφορά τη λειτουργία αυτού του χρονικού:

1. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι συνεχώς <<1>> είσοδος I2.1 . Όσο διάστημα κυλάει ο χρόνος η έξοδος είναι ενεργοποιημένη Q4.0. Μόλις μηδενίσει ο χρόνος ,παρόλο που το RLO ενεργοποίησης είναι <<1>> η έξοδος μηδενίζει.
2. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι συνεχώς <<1>> και για όλο αυτό το διάστημα η έξοδος είναι ενεργοποιημένη. Αν πριν

ολοκληρωθεί η αντίστροφη μέτρηση το RLO αυτό γίνει <<0>> σταματά η μέτρηση και απενεργοποιείται ακαριαία και η έξοδος.

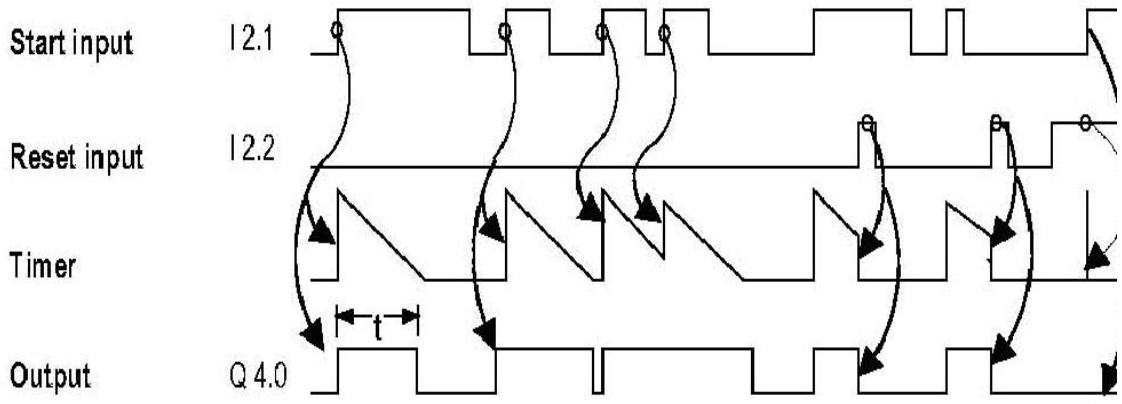
3. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι συνεχώς <<1>> και για όλο αυτό το διάστημα η έξοδος είναι ενεργοποιημένη. Αν πριν ολοκληρωθεί η αντίστροφη μέτρηση το RLO στο reset γίνει<<1>> σταματά η μέτρηση και μηδενίζεται η έξοδος.

5.7.2 Χρονικό παλμού με αυτοσυγκράτηση SE

Αυτό το χρονικό μοιάζει πάρα πολύ με το προηγούμενο με κυρίαρχη διαφορά την αυτοσυγκράτηση του RLO ενεργοποίησης. Εδώ ,αρκεί και στιγμιαία το RLO αυτό να γίνει <<1>> για να μας δώσει έξοδο για όλο το διάστημα που το χρονικό μετράει.

Παράδειγμα:

```
A     I 2.1
L     S5T#10s      //Preset 10 seconds into ACCU 1.
SE    T1           //Start timer T1 as an extended pulse timer.
A     I 2.2
R     T1           //Reset timer T1.
A     T1           //Check signal state of timer T1.
=     Q 4.0
```



Από το παραπάνω διάγραμμα του χρονικού διακρίνουμε τρεις περιπτώσεις όσον αφορά τη λειτουργία αυτού του χρονικού:

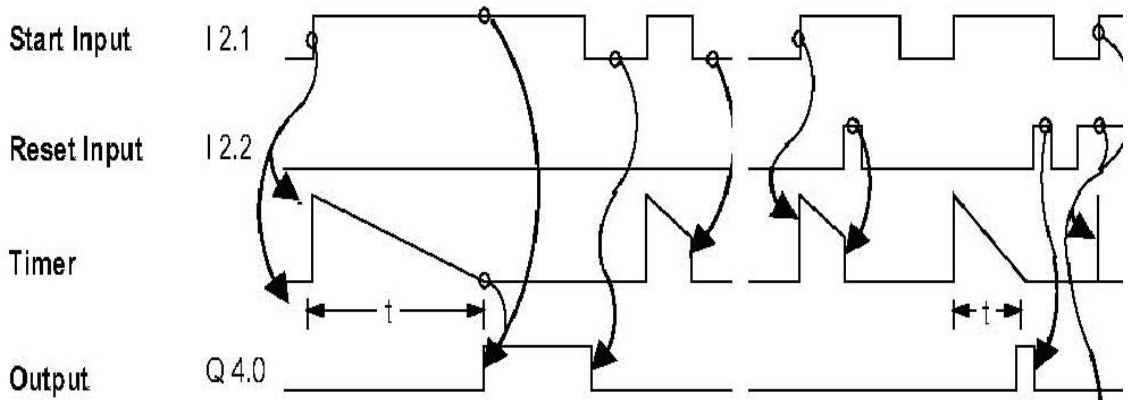
1. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι συνεχώς <<1>> είσοδος I2.1 . Όσο διάστημα κυλάει ο χρόνος η έξοδος είναι ενεργοποιημένη Q4.0. Μόλις μηδενίσει ο χρόνος ,παρόλο που το RLO ενεργοποίησης είναι <<1>> η έξοδος μηδενίζει.
2. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι έστω και στιγμιαία <<1>> .Όσο διάστημα κυλάει ο χρόνος η έξοδος είναι ενεργοποιημένη.Μόλις μηδενίσει ο χρόνος μηδενίζει και η έξοδος.
3. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι συνεχώς <<1>> .Αν το διάστημα αυτό το RLO στο reset γίνει <<1>> παρόλο που το RLO ενεργοποίησης είναι <<1>> η έξοδος μηδενίζει και παύει να μετρά αντίστροφα το χρονικό.

5.7.3 Χρονικό καθυστέρησης έλξης SD

Το συγκεκριμένο χρονικό είναι το πιο συχνά χρησιμοποιούμενο από τα υπόλοιπα τέσσερα είδη και αφορά τις περιπτώσεις που χρειαζόμαστε να εισάγουμε κάποια χρονοκαθυστέρηση για να εκτελέσουμε μια διαδικασία.

Παράδειγμα:

```
A     I 2.1
L     S5T#10s     //Preset 10 seconds into ACCU 1.
SD    T1          //Start timer T1 as an on-delay timer.
A     I 2.2
R     T1          //Reset timer T1.
A     T1          //Check signal state of timer T1.
=     Q 4.0
```



Από το παραπάνω διάγραμμα χρόνου διακρίνουμε τέσσερις περιπτώσεις όσον αφορά τη λειτουργία αυτού του χρονικού:

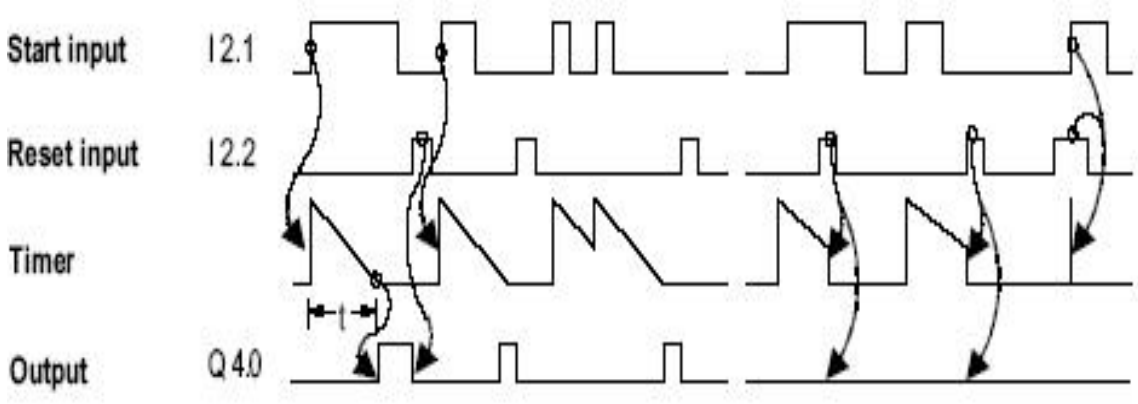
1. Το χρονικό αρχίζει να μετράει αντίστροφα από τη στιγμή που το RLO ενεργοποίησης του είναι <<1>>.Όταν κυλήσει ο χρόνος και για όσο διάστημα το RLO παραμένει <<1>> , η έξοδος είναι ενεργοποιημένη.
2. Το χρονικό αρχίζει να μετράει αντίστροφα από τη στιγμή που το RLO ενεργοποίησης του είναι <<1>>, αλλά πριν προλάβει να ολοκληρώσει τη μέτρηση ,το RLO γίνεται <<0>>. Εδώ ,η έξοδος δεν ενεργοποιείται και το χρονικό όταν το RLO γίνει και πάλι <<1>> αρχίζει να μετρά από την αρχή.
3. Το χρονικό αρχίζει να μετράει αντίστροφα από τη στιγμή που το RLO ενεργοποίησης του είναι <<1>>, αλλά πριν προλάβει να ολοκληρώσει τη μέτρηση ,το RLO στο Reset γίνεται <<1>>. Εδώ ,η έξοδος δεν ενεργοποιείται και το χρονικό μηδενίζεται.
4. Το χρονικό αρχίζει να μετράει αντίστροφα από τη στιγμή που το RLO ενεργοποίησης του είναι <<1>>.Όταν κυλήσει ο χρόνος , ενεργοποιείται η έξοδος έως ότου το RLO στο Reset γίνει <<1>>,οπότε μηδενίζεται η έξοδος και το χρονικό

5.7.4 Χρονικό καθυστέρησης έλξης με αυτοσυγκράτηση SS

Το χρονικό αυτό μοιάζει αρκετά με το προηγούμενο, μόνο που του αρκεί έστω και στιγμιαία το RLO ενεργοποίησης να γίνει <<1>>,ενώ απαιτεί και reset για να απενεργοποιηθεί, αφού αυτοσυγκρατεί την έξοδο του.

Παράδειγμά :

```
A     I 2.1
L     S5T#10s     //Preset 10 seconds into ACCU 1.
SS    T1          //Start timer T1 as an on-delay timer.
A     I 2.2
R     T1          //Reset timer T1.
A     T1          //Check signal state of timer T1.
=     Q 4.0
```



Από το παραπάνω διάγραμμα χρόνου διακρίνουμε τρεις περιπτώσεις όσον αφορά τη λειτουργία αυτού του χρονικού:

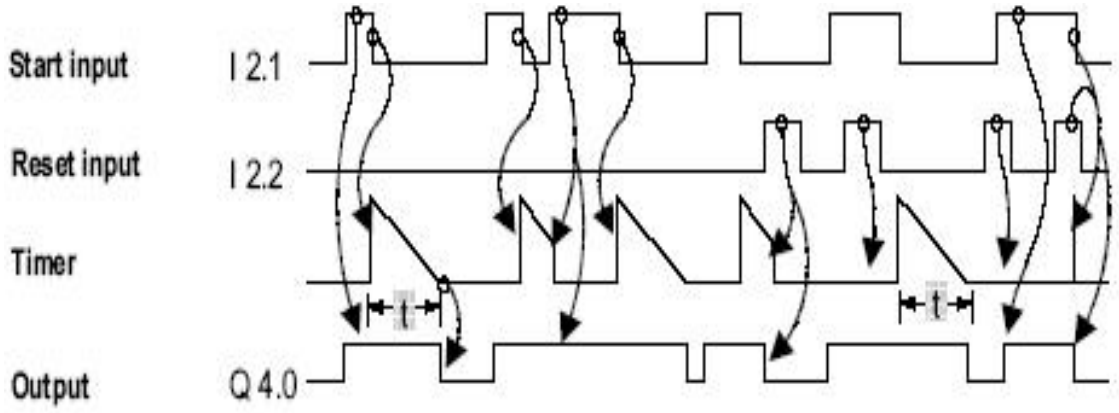
1. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι έστω και στιγμιαία <<1>>. Όταν κυλήσει ο χρόνος η έξοδος ενεργοποιείται με αυτοσυγκράτηση. Για να απενεργοποιηθεί ,πρέπει το RLO στο reset να έχει σήμα<<1>>
2. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι έστω και στιγμιαία <<1>>.Πριν ολοκληρωθεί η αντίστροφη μέτρηση το RLO στο reset έχει σήμα <<1>>, οπότε το χρονικό μηδενίζεται και δεν ενεργοποιείται η έξοδος.
3. Το χρονικό αρχίζει να μετρά αντίστροφα από την στιγμή που το RLO ενεργοποίησης του είναι έστω και στιγμιαία <<1>>.Πριν ολοκληρωθεί η αντίστροφη μέτρηση το RLO στο set γίνεται <<0>> και κατόπιν <<1>>, οπότε το χρονικό αρχίζει να μετρά από την αρχή. Όταν κυλήσει ο χρόνος η έξοδος ενεργοποιείται με αυτοσυγκράτηση. Για να απενεργοποιηθεί ,πρέπει το RLO στο reset να έχει σήμα<<1>>

5.7.5 Χρονικό καθυστέρησης πτώσης SF

Το χρονικό καθυστέρησης πτώσης χρησιμοποιείται όταν θέλουμε να έχουμε έξοδο για κάποιο διάστημα ακόμα και μετά την παρέλευση της αιτίας που το προκάλεσε την ενεργοποίησης του.

Παράδειγμά :

```
A     I 2.1
L     S5T#10s     //Preset 10 seconds into ACCU 1.
SF    T1          //Start timer T1 as an on-delay timer.
A     I 2.2
R     T1          //Reset timer T1.
A     T1          //Check signal state of timer T1.
=     Q 4.0
```



Από το παραπάνω διάγραμμα χρόνου διακρίνουμε τέσσερις περιπτώσεις όσον αφορά τη λειτουργία αυτού του χρονικού:

1. Το RLO ενεργοποίησης του δίνει συνεχώς <<1>> και η έξοδος είναι ενεργοποιημένη. Όταν γίνει <<0>> , κυλάει ο χρόνος αντίστροφα και η έξοδος είναι ενεργοποιημένη μέχρις ότου μηδενίσει ο χρόνος οπότε μηδενίζει και η έξοδος
2. Το RLO ενεργοποίησης του δίνει συνεχώς <<1>> και η έξοδος είναι ενεργοποιημένη. Όταν γίνει <<0>> , κυλάει ο χρόνος αντίστροφα και η έξοδος είναι ενεργοποιημένη .Αν πριν ολοκληρωθεί η αντίστροφη μέτρηση το RLO στο reset γίνει <<1>> τότε μηδενίζει ο χρόνος και απενεργοποιείται η έξοδος.
3. Το RLO ενεργοποίησης του δίνει συνεχώς <<1>> και η έξοδος είναι ενεργοποιημένη. Αν γίνει <<0>> και κατόπιν <<1>> πριν ολοκληρωθεί η μέτρηση του χρονικού, ο χρόνος μετρά και πάλι από την αρχή.
4. Το RLO ενεργοποίησης του δίνει συνεχώς <<1>> και η έξοδος είναι ενεργοποιημένη. Αν στο μεταξύ το RLO του reset γίνει <<1>> για κάποιο διάστημα , για όλο αυτό το διάστημα η έξοδος απενεργοποιείται ενώ το χρονικό δεν μετράει.

5.8 Απαριθμητές

Ένα βασικό σετ εντολών είναι οι εντολές μετρητών - απαριθμητών που μπορούν να μετρήσουν κάποιο γεγονός που συνέβηκε από 0 ως και 999 φορές. Η συχνότητα με την οποία μπορούν να μετρήσουν εξαρτάται από το χρόνο σάρωσης της CPU και την ταχύτητα απόκρισης της κάρτας στην οποία συνδέεται το σήμα απαρίθμησης .

Εντολές απαριθμητών:

- Προτοποθέτηση απαριθμητή (**S**)
- Μηδενισμός απαριθμητή (**R**)
- Ενεργοποίηση απαριθμητή (**FR**)
- Μέτρηση προς τα επάνω (**CU**)
- Μέτρηση προς τα κάτω (**CD**)
- Φόρτωση του περιεχομένου του απαριθμητή σε :
    1. Δυαδική μορφή (**L**)
    2. BCD μορφή (**LC**)
- Έλεγχος κατάστασης απαριθμητή (A,O,X,AN,ON,XN)

5.8.1 Πρόσδωση τιμή σε απαριθμητή

Η πρόσδωση τιμής σε απαριθμητή δίνεται με την εντολή L συνοδευόμενη πάντα από την τιμή που θέλουμε να δώσουμε. Η εντολή μπορεί να δοθεί με τρεις τρόπους :

1. **L W#16#wxyz**, όπου w-είναι πάντα 0 και xyz- η τιμή σε BCD μορφή
2. **L C#xxx** ,όπου xxx είναι η τιμή
3. **L MWx**

5.8.2 Προτοποθέτηση απαριθμητή

Ένας απαριθμητής προτοποθετείται στην τιμή που έχουμε φορτώσει από πριν στον Accu 1 με την εντολή L C#xxx. Για να γίνει αυτό ,πρέπει το RLO που προηγείται της εντολής προτοποθέτησης να είναι <<1>> .

Η εντολή προτοποθέτησης είναι

S Cn

5.8.3 Μηδενισμός απαριθμητή

Η εντολή μηδενισμού είναι

R Cn

Για να εκτελεστεί η εντολή είναι απαραίτητο το RLO που προηγείται της εντολής να είναι <<1>. Μόλις γίνει αυτό ,αυτόματα το περιεχόμενο του απαριθμητή γίνεται <<0>>.

5.8.4 Αύξηση περιεχομένου απαριθμητή

Το περιεχόμενο του απαριθμητή αυξάνεται με την εντολή

CU Cn

Για να εκτελεστεί η εντολή είναι απαραίτητο το RLO που προηγείται της εντολής να μεταβληθεί από <<0>> σε <<1>>. Μόλις γίνει αυτό ,αυτόματα το περιεχόμενο του απαριθμητή αυξάνεται κατά 1. Όταν το περιεχόμενο του απαριθμητή είναι 999 και δοθεί παλμός για αύξηση ,αυτός αγνοείται και η τιμή παραμένει στο 999.

5.8.5 Μείωση περιεχομένου απαριθμητή

Το περιεχόμενο του απαριθμητή μειώνεται με την εντολή

CD Cn

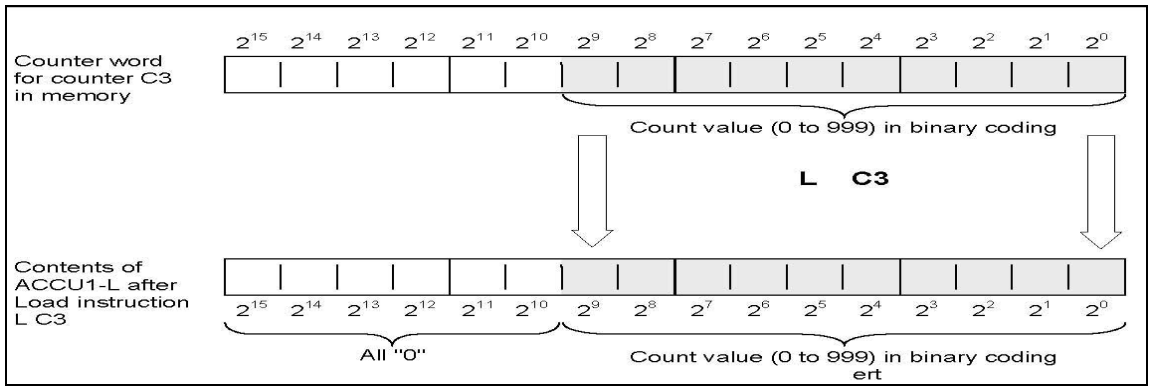
Για να εκτελεστεί η εντολή είναι απαραίτητο το RLO που προηγείται της εντολής να μεταβληθεί από <<0>> σε <<1>>. Μόλις γίνει αυτό ,αυτόματα το περιεχόμενο του απαριθμητή μειώνεται κατά 1. Όταν το περιεχόμενο του απαριθμητή είναι 0 και δοθεί παλμός για μείωση ,αυτός αγνοείται και η τιμή παραμένει στο 0

5.8.6 Φόρτωση περιεχομένου απαριθμητή

Η φόρτωση του περιεχομένου του απαριθμητή μπορεί να γίνει με δυο τρόπους :

1. Σε δυαδική μορφή (binary) με την εντολή

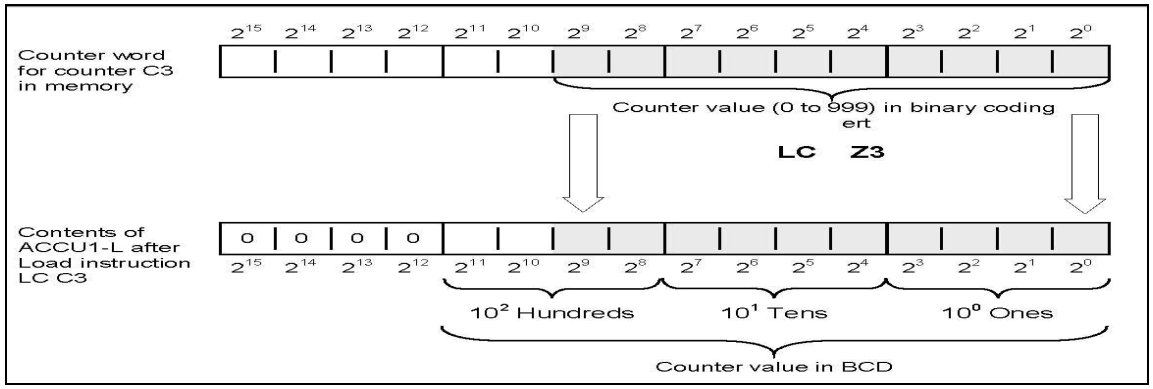
L Cn



Το περιεχόμενο του αποθηκεύεται στον Accu1 σε INT μορφή και μπορούμε να το επεξεργαστούμε περαιτέρω.

2. Σε BCD μορφή με την εντολή

LC Cn



Το περιεχόμενο του αποθηκεύεται στον Accu1 σε BCD μορφή και μπορούμε να το επεξεργαστούμε περαιτέρω

5.9 Συγκρίσεις

Οι λειτουργίες σύγκρισης συγκρίνουν δύο ψηφιακές τιμές που βρίσκονται η μία στον Accu 1 και η άλλη στον Accu 2. Το αποτέλεσμα της σύγκρισης επηρεάζει το RLO και με αυτό μπορούμε να πάρουμε κάποιες αποφάσεις.

Ανάλογα με το είδος των αριθμών που πρόκειται να συγκρίνουμε (Integer, Double Integer, Real) υπάρχει και η αντίστοιχη εντολή.

5.9.1 Σύγκριση ακέραιου αριθμού INT (16bit)

Πίνακας με τις δίαφορες περιπτώσεις σύγκρισης ακέραιου αριθμού INT (16bit) .

Comparison instruction executed	RLO Result if ACCU 2 > ACCU 1	RLO Result if ACCU 2 = ACCU 1	RLO Result if 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

Παράδειγμα :

```
L     MW10    //Load contents of MW10 (16-bit integer).
L     IW24    //Load contents of IW24 (16-bit integer).
>I            //Compare if ACCU 2-L (MW10) is greater (>) than ACCU 1- L (IW24).
=     M 2.0   //RLO = 1 if MW10 > IW24.
```

5.9.2 Σύγκριση ακέραιου αριθμού DINT (32bit)

Πίνακας με τις δίαφορες περιπτώσεις σύγκρισης ακέραιου αριθμού DINT (32bit)

Comparison instruction executed	RLO Result if ACCU 2 > ACCU 1	RLO Result if ACCU 2 = ACCU 1	RLO Result if 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

Παράδειγμα:

```
L     MD10    //Load contents of MD10 (double integer, 32 bits).
L     ID24    //Load contents of ID24 (double integer, 32 bits).
>D            //Compare if ACCU 2 (MD10) is greater (>) than ACCU 1 (ID24).
=     M 2.0   //RLO = 1 if MD10 > ID24
```

5.9.3 Σύγκριση πραγματικων αριθμών REAL (32bit)

Πίνακας με τις διάφορες περιπτώσεις σύγκρισης πραγματικών αριθμών REAL(32bit)

Comparison instruction executed	RLO Result if ACCU 2 > ACCU 1	RLO Result if ACCU 2 = ACCU 1	RLO Result if 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

Παράδειγμα:

```
L     MD10        //Load contents of MD10 (floating-point number).
L     1.359E+02   //Load the constant 1.359E+02.
>R                //Compare if ACCU 2 (MD10) is greater (>) than ACCU 1 (1.359-E+02).
=     M 2.0       //RLO = 1 if MD10 > 1.359E+02.
```

5.10 Μετατροπείς τύπων

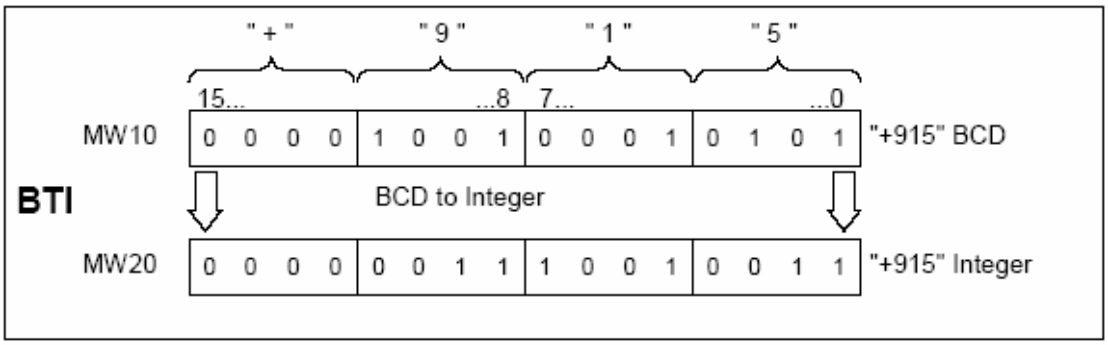
Πολλές φορές στην πράξη θέλουμε την πληροφορία που βρίσκεται σε μια μορφή να την μετατρέψουμε σε μια άλλη. Παρακάτω θα παρουσιαστούν οι σημαντικότερες εντολές μετατροπής ενός αριθμού σε κάποιον άλλο

5.10.1 Εντολή BTI

Μετατρέπει έναν BCD αριθμό σε INT (16Bit) ακέραιο

Παράδειγμα:

```
L     MW10        //Load the BCD number into ACCU 1-L.
BTI               //Convert from BCD to integer; store result in ACCU 1-L.
T     MW20        //Transfer result (integer number) to MW20
```

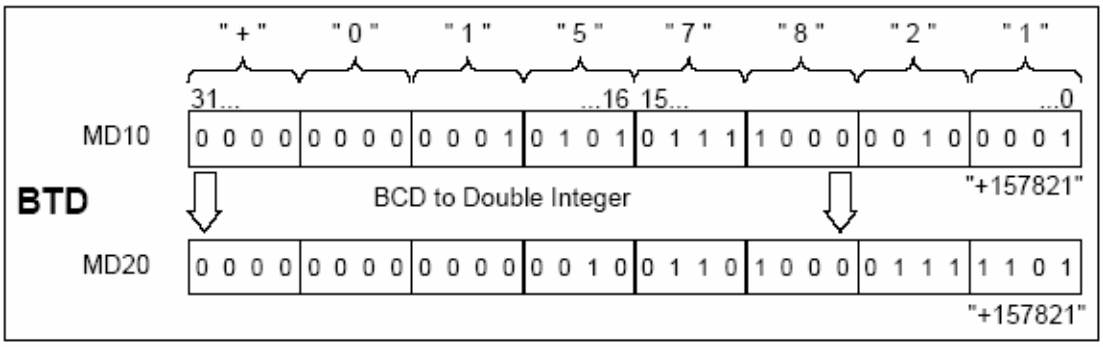


5.10.2 Εντολή BTD

Μετατρέπει έναν BCD αριθμό σε 32Bit ακέραιο.

Παράδειγμα:

```
L     MD10    //Load the BCD number into ACCU 1-L.
BTD           //Convert from BCD to integer; store result in ACCU 1-L.
T     MD20    //Transfer result (double integer number) to MD20
```

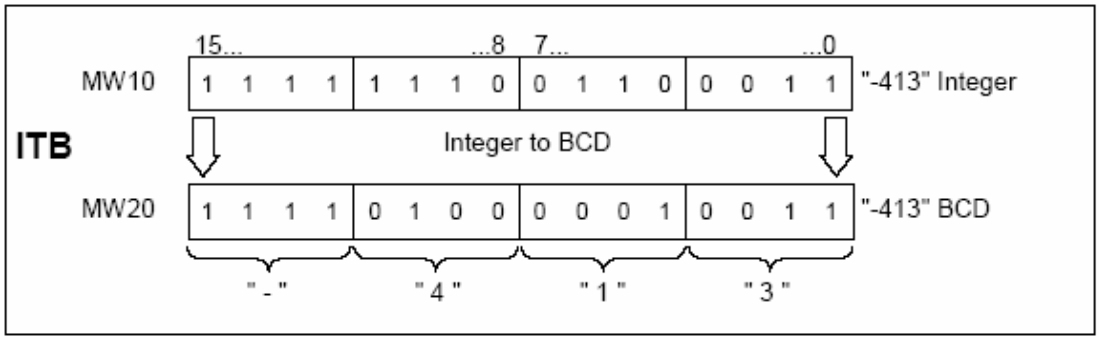


5.10.3. Εντολή ITB

Μετατρέπει έναν 16Bit ακέραιο σε BCD

Παράδειγμα:

```
L     MW10    // Load the integer number into ACCU 1-L.
ITB           //Convert from integer to BCD (16-bit); store result in ACCU 1-L.
T     MW20    //Transfer result (BCD number) to MW20.
```

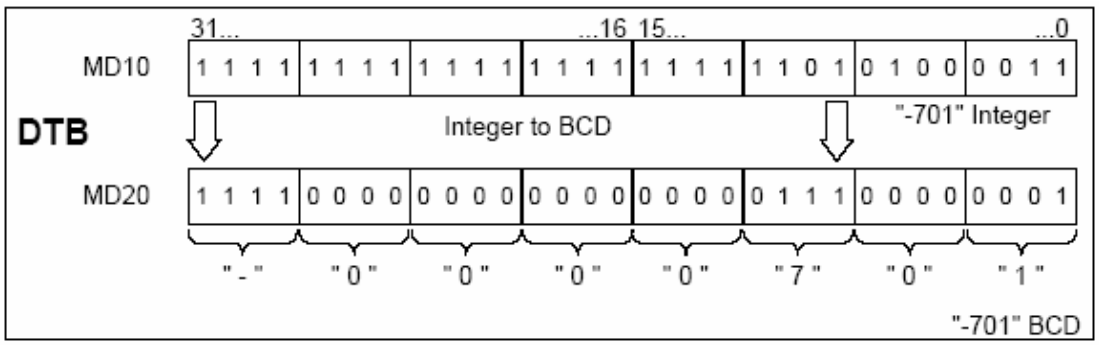


5.10.4 Εντολή DTB

Μετατρέπει έναν 32Bit ακέραιο σε BCD

Παράδειγμα:

L **MD10** //Load the 32-bit integer into ACCU 1.
DTB // Convert from integer (32-bit) to BCD, store result in ACCU 1..
T **MD20** //Transfer result (BCD number) to MD20.



5.10.5 Εντολή ITD

Μετατρέπει έναν 16Bit ακέραιο σε 32Bit ακέραιο

Παράδειγμα:

L **MW12** //Load the integer number into ACCU 1.
ITD //Convert from integer (16-bit) to double integer (32-bit); store result in ACCU 1.
T **MD20** //Transfer result (double integer) to MD20.

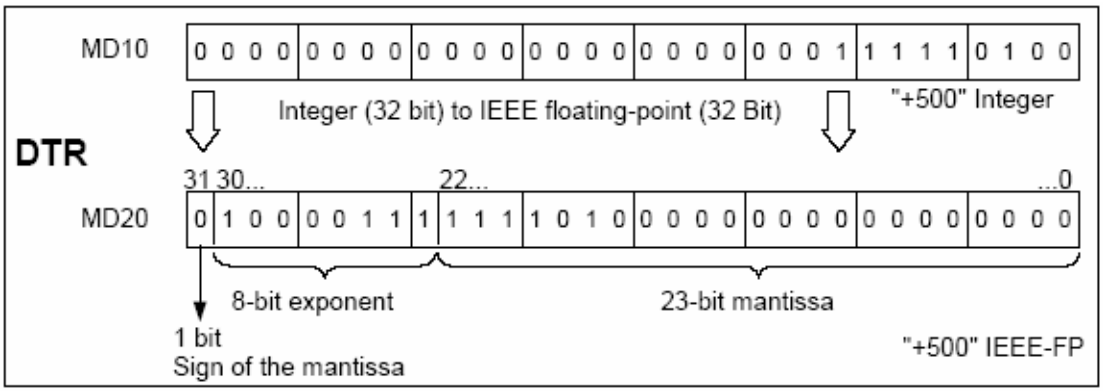
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of ITD	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
after execution of ITD	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 or 1, bits are not used for conversion)							

5.10.6 Εντολή DTR

Μετατρέπει έναν 32Bit ακέραιο σε REAL

Παράδειγμα:

```
L     MD10    //Load the 32-bit integer into ACCU 1.
DTR           // Convert from double integer to floating point (32-bit IEEE FP); store result in ACCU 1
T     MD20    //Transfer result (BCD number) to MD20.
```



5.10.7 Εντολή INVI

Συμπλήρωμα ως προς 1 πραγματικού αριθμου INT (16bit)

Παράδειγμα:

```
L     IW8     // Load value into ACCU 1-L.
INVI          // Form ones complement 16-bit
T     MW10    // Transfer result to MW10..
```

Contents	ACCU1-L			
Bit	15 . . .	. .	. .	. . . 0
before execution of **INVI**	0110	0011	1010	1110
after execution of **INVI**	1001	1100	0101	0001

5.10.8 Εντολή INVD

Συμπλήρωμα ως προς 1 πραγματικού αριθμου DINT (32bit)

Παράδειγμα:

```
L     ID8     // Load value into ACCU 1-L.
INVI          // Form ones complement 32-bit
T     MD10    // Transfer result to MD10..
```

Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **INVD**	0110	1111	1000	1100	0110	0011	1010	1110
after execution of **INVD**	1001	0000	0111	0011	1001	1100	0101	0001

5.10.9 Εντολή RND

Μετατρέπει έναν πραγματικό αριθμό σε ακέραιο με κανονική στρογγυλοποίηση.

Παράδειγμα:

```
L     MD10    //Load the floating-point number into ACCU 1-L.
RND           //Convert the floating-point number (32-bit, IEEE-FP) into an integer (32-bit)
              and round off the result.
T     MD20    //Transfer result (double integer number) to MD20.
```

Value before conversion		Value after conversion
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

5.10.10 Εντολή TRUND

Μετατρέπει έναν πραγματικό αριθμό σε ακέραιο με στρογγυλοποίηση προς τα κάτω.

Παράδειγμα:

```
L     MD10    //Load the floating-point number into ACCU 1-L.
RND           //Convert the floating-point number (32-bit, IEEE-FP) into an integer (32-bit)
              and round off the result.
T     MD20    //Transfer result (double integer number) to MD20.
```

Value before conversion		Value after conversion
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

5.11 Αριθμητικές πράξεις

Οι αριθμητικές πράξεις εφαρμόζονται επάνω σε δύο ψηφιακές τιμές που βρίσκονται η μια στον Accu 1και η άλλη στον Accu 2. Ανάλογα με το είδος των αριθμών που πρόκειται να συγκρίνουμε (Integer, Double Integer, Real) υπάρχει και η αντίστοιχη εντολή

5.11.1 Αριθμητικές πράξεις ακέραιου αριθμού INT (16bit)

- **+I** Προσθέτει δύο ακέραιους αριθμούς.
- **-I** Αφαιρεί δύο ακέραιους αριθμούς.
- ***I** Πολλαπλασιάζει δύο ακέραιους αριθμούς.
- **/I** Διαιρεί δύο ακέραιους αριθμούς.

Παράδειγμα:

```
L    IW10        //Load the value of IW10 into ACCU 1-L.
L    MW14        //Load the contents of ACCU 1-L into ACCU 2-L. Load the value of MW14
                 into ACCU 1-L.
+I               //Add ACCU 2-L and ACCU 1-L; store the result in ACCU 1-L.
T    DB1.DBW25   //The contents of ACCU 1-L (result) are transferred to DBW25 of DB1.
```

5.11.2 Αριθμητικές πράξεις ακέραιου αριθμού DINT (32bit)

- **+D** Προσθέτει δύο ακέραιους αριθμούς.
- **-D** Αφαιρεί δύο ακέραιους αριθμούς.
- ***D** Πολλαπλασιάζει δύο ακέραιους αριθμούς.
- **/D** Διαιρεί δύο ακέραιους αριθμούς.

Παράδειγμα:

```
L    ID10        //Load the value of ID10 into ACCU 1.
L    MD14        //Load contents of ACCU 1 into ACCU 2. Load contents of MD14 into
                 ACCU 1.
```

```
*D                  //Multiply ACCU 2 and ACCU 1; store the result in ACCU 1.
T    DB1.DBD25      //The contents of ACCU 1 (result) are transferred to DBD25 in DB1.
```

5.11.3 Αριθμητικές πράξεις πραγματικού αριθμού REAL (32bit)

- **+R** Προσθέτει δύο ακέραιους αριθμούς.
- **-R** Αφαιρεί δύο ακέραιους αριθμούς.
- ***R** Πολλαπλασιάζει δύο ακέραιους αριθμούς.
- **/R** Διαιρεί δύο ακέραιους αριθμούς.

Παράδειγμα:

```
OPN   DB10
L     ID10      //Load the value of ID10 into ACCU 1.
L     MD14      //Load the contents of ACCU 1 into ACCU 2. Load the value of MD14 into
                ACCU 1.
/R              //Divide ACCU 2 by ACCU 1; store the result in ACCU 1.
T     DBD20     //The content of ACCU 1 (result) is transferred to DBD20 in DB10.
```

5.12 Έλεγχος ροής προγράμματος

Με τις εντολές ελέγχου της ροής του προγράμματος μπορούμε να διακόψουμε τη γραμμική εκτέλεση του προγράμματος και να συνεχίσουμε σε κάποιο επόμενο σημείο του ιδίου μπλοκ. Αυτή η μεταπήδηση μπορεί να γίνει με ή χωρίς κάποιες προϋποθέσεις.

5.12.1 Εντολή JU

Η εντολή JU εκτελείται πάντοτε όταν τη συναντήσει ο επεξεργαστής ,δεν επηρεάζει το RLO και δεν εξαρτάτε από αυτό.

Παράδειγμα:

```
      A   I 1.0
      A   I 1.2
     JC   DELE    //Jump if RLO=1 to jump label DELE.
      L   MB10
    INC   1
      T   MB10
     JU   FORW    //Jump unconditionally to jump label FORW.
```

```
DELE:   L   0
        T   MB10
FORW:   A   I 2.1       //Program scan resumes here after jump to jump label FORW.
```

5.12.2 Εντολή JC

Η εντολή JC εκτελείται όταν τη συναντήσει ο επεξεργαστής με την προϋπόθεση ότι το RLO είναι <<1>> ενώ εκτέλεση της δεν το επηρεάζει..

Παράδειγμα:

```
        A   I 1.0
        A   I 1.2
       JC   JOVR    //Jump if RLO=1 to jump label JOVR.
        L   IW8     //Program scan continues here if jump is not executed.
        T   MW22
JOVR:   A    I 2.1  //Program scan resumes here after jump to jump label JOVR.
```

5.12.3 Εντολή JCN

Η εντολή JCN εκτελείται όταν τη συναντήσει ο επεξεργαστής με την προϋπόθεση ότι το RLO είναι <<0>> ενώ εκτέλεση της δεν το επηρεάζει..

Παράδειγμα:

```
        A   I 1.0
        A   I 1.2
      JCN   JOVR    //Jump if RLO = 0 to jump label JOVR.
        L   IW8     //Program scan continues here if jump is not executed.
        T   MW22
JOVR:   A   I 2.1   //Program scan resumes here after jump to jump label JOVR.
```

5.13 Εντολές λογικών πράξεων μεταξύ λέξεων

Οι εντολές Word λογικής συνδέουν την τιμή του Accu 1bit προς bit με μια σταθερά ή με το περιεχόμενο του Accu 2 και αποθηκεύουν το αποτέλεσμα στον Accu 1. Η λογική αυτή λειτουργία μπορεί να εκτελεσθεί σε μια Word ή σε Double Word

5.13.1 Εντολές Word λέξεων (16bit)

1. **AW :** Με την εντολή αυτή γίνεται η λογική πράξη **AND** μεταξύ των λέξεων
2. **OW:** Είναι ίδια με την παραπάνω με τη διαφορά ότι εδώ η λογική πράξη είναι η OR και όχι η AND.
3. **XOW** : Ομοίως η εντολή αυτή εκτελεί τη λογική πράξη **XOR** μεταξύ των δύο αριθμών.

Παράδειγμα:

```
L     IW20   //Load contents of IW20 into ACCU 1-L.
L     IW22   //Load contents of ACCU 1 into ACCU 2. Load contents of ID24 into ACCU 1-
             L.
XOW          //Combine bits of ACCU 1-L with ACCU 2-L bits by XOR, store result in
             ACCU 1-L.
T     MW8    //Transfer result to MW8.
```

Bit	15 . . .	. .	. .	. . . 0
ACCU 1 before execution of **XOW**	0101	0101	0011	1011
ACCU 2-L or 16-bit constant:	1111	0110	1011	0101
Result (ACCU 1) after execution of **XOW**	1010	0011	1000	1110

5.13.2 Εντολές Double Word λέξεων(32bit)

1. **AD :** Με την εντολή αυτή γίνεται η λογική πράξη **AND** μεταξύ των λέξεων
2. **OD:** Είναι ίδια με την παραπάνω με τη διαφορά ότι εδώ η λογική πράξη είναι η OR και όχι η AND.
3. **XOD** : Ομοίως η εντολή αυτή εκτελεί τη λογική πράξη **XOR** μεταξύ των δύο αριθμών.

Παράδειγμα:

```
L     ID20   //Load contents of ID20 into ACCU 1.
L     ID24   //Load contents of ACCU 1 into ACCU 2. Load contents of ID24 into ACCU 1.
OD           //Combine bits from ACCU 1 with ACCU 2 bits by OR; store result in ACCU 1
T     MD8    //Transfer result to MD8.
```

Bit	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
ACCU 1 before execution of OD	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 or 32-bit constant:	1111	0011	1000	0101	0111	0110	1011	0101
Result (ACCU 1) after execution of OD	1111	0011	1111	1101	1111	0111	1011	1111

5.14 Εντολές μαζικής μεταφοράς

Όταν λέμε μαζική μεταφορά πληροφοριών εννοούμε την μεταφορά ολόκληρων byte, word ή double word από μια περιοχή της μνήμης σε μια άλλη.

Η μεταφορά αυτή γίνεται πάντα έμμεσα , περνώντας και αποθηκεύοντας την πληροφορία σε ενδιάμεσους σταθμούς , τους accumulators.

Οι accumulators είναι θέσεις μνήμης μήκους 32bits και χαρακτηρίζονται ως accumulator 1(Accu 1), Accu 2, Accu 3 και Accu 4.

5.14.1 Εντολή Load

Κατά την εκτέλεση της εντολής **L** αντιγράφονται τα δεδομένα, από την περιοχή μνήμης που ορίζεται στον ACCU1. Το περιεχόμενο του ACCU1 μεταφέρεται στον ACCU2. Το αρχικό περιεχόμενο του ACCU2 χάνεται..

Παράδειγμα:

```
L     IB10     //Load input byte IB10 into ACCU 1-L-L
L     MB120    //Load memory byte MB120 into ACCU 1-L-L.
L     DBB12    //Load data byte DBB12 into ACCU 1-L-L.
L     DIW15    //Load instance data word DIW15 into ACCU 1-L.
```

5.14.2 Εντολή Transfer

Κατά την εκτέλεση της εντολής **T** αντιγράφονται τα δεδομένα, από τον ACCU1 στην περιοχή μνήμης που ορίζεται

Παράδειγμα:

```
T     QB10     //Transfers contents of ACCU 1-L-L to output byte QB10.
T     MW14     //Transfers contents of ACCU 1-L to memory word MW14.
T     DBD2     //Transfers contents of ACCU 1 to data double word DBD2.
```

5.15 Πίνακας εντολών στην STL

English Mnemonics	Program Elements Catalog	Description
+	Integer math Instruction	Add Integer Constant (16, 32-bit)
=	Bit logic Instruction	Assign
)	Bit logic Instruction	Nesting Closed
0	Accumulator	AR1 Add ACCU 1 to Address Register 1
0	Accumulator	AR2 Add ACCU 1 to Address Register 2
+D	Integer math Instruction	Add ACCU 1 and ACCU 2 as Double Integer (32-bit)
–D	Integer math Instruction	Subtract ACCU 1 from ACCU 2 as Double Integer (32-bit)
*D	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Double Integer (32-bit)
/D	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Double Integer (32-bit)
? D	Compare	Compare Double Integer (32-bit) ==, <>, >, <, >=, <=
+I	Integer math Instruction	Add ACCU 1 and ACCU 2 as Integer (16-bit)
–I	Integer math Instruction	Subtract ACCU 1 from ACCU 2 as Integer (16-bit)
*I	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Integer (16-bit)
/I	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Integer (16-bit)
? I	Compare	Compare Integer (16-bit) ==, <>, >, <, >=, <=
+R	Floating point Instruction	Add ACCU 1 and ACCU 2 as a Floating-point Number (32-bit IEEE-FP)
–R	Floating point Instruction	Subtract ACCU 1 from ACCU 2 as a Floating-point Number (32-bit IEEE-FP)
*R	Floating point Instruction	Multiply ACCU 1 and ACCU 2 as Floating-point Numbers (32-bit IEEE-FP)
/R	Floating point Instruction	Divide ACCU 2 by ACCU 1 as a Floating-point Number (32-bit IEEE-FP)
? R	Compare	Compare Floating-point Number (32-bit) ==, <>, >, <, >=, <=
A	Bit logic Instruction	And
A(	Bit logic Instruction	And with Nesting Open

ABS	Floating point Instruction	Absolute Value of a Floating-point Number (32-bit IEEE-FP)
ACOS	Floating point Instruction	Generate the Arc Cosine of a Floating-point Number (32-bit)
AD	Word logic Instruction	AND Double Word (32-bit)
AN	Bit logic Instruction	And Not
AN(	Bit logic Instruction	And Not with Nesting Open
ASIN	Floating point Instruction	Generate the Arc Sine of a Floating-point Number (32-bit)
ATAN	Floating point Instruction	Generate the Arc Tangent of a Floating-point Number (32-bit)
AW	Word logic Instruction	AND Word (16-bit)
BE	Program control	Block End
BEC	Program control	Block End Conditional
BEU	Program control	Block End Unconditional
BLD	Program control	Program Display Instruction (Null)
BTD	Convert	BCD to Integer (32-bit)
BTI	Convert	BCD to Integer (16-bit)
CAD	Convert	Change Byte Sequence in ACCU 1 (32-bit)
CALL	Program control	Block Call
CALL	Program control	Call Multiple Instance
CALL	Program control	Call Block from a Library
CAR	Load/Transfer	Exchange Address Register 1 with Address Register 2
CAW	Convert	Change Byte Sequence in ACCU 1-L (16-bit)
CC	Program control	Conditional Call
CD	Counters	Counter Down
CDB	Convert	Exchange Shared DB and Instance DB
CLR	Bit logic Instruction	Clear RLO (=0)
COS	Floating point Instruction	Generate the Cosine of Angles as Floating-point Numbers (32-bit)
CU	Counters	Counter Up
DEC	Accumulator	Decrement ACCU 1-L-L
DTB	Convert	Double Integer (32-bit) to BCD
DTR	Convert	Double Integer (32-bit) to Floating-point (32-bit IEEE-FP)
ENT	Accumulator	Enter ACCU Stack
EXP	Floating point Instruction	Generate the Exponential Value of a Floating-point Number (32-bit)
FN	Bit logic Instruction	Edge Negative
FP	Bit logic Instruction	Edge Positive
FR	Counters	Enable Counter (Free) (free, FR C 0 to C 255)
FR	Timers	Enable Timer (Free)
INC	Accumulator	Increment ACCU 1-L-L
INVD	Convert	Ones Complement Double Integer (32-bit)
INVI	Convert	Ones Complement Integer (16-bit)
ITB	Convert	Integer (16-bit) to BCD
ITD	Convert	Integer (16-bit) to Double Integer (32-bit)
JBI	Jumps	Jump if BR = 1

JC	Jumps	Jump if RLO = 1
JCB	Jumps	Jump if RLO = 1 with BR
JCN	Jumps	Jump if RLO = 0
JL	Jumps	Jump to Labels
JM	Jumps	Jump if Minus
JMZ	Jumps	Jump if Minus or Zero
JN	Jumps	Jump if Not Zero
JNB	Jumps	Jump if RLO = 0 with BR
JNBI	Jumps	Jump if BR = 0
JO	Jumps	Jump if OV = 1
JOS	Jumps	Jump if OS = 1
JP	Jumps	Jump if Plus
JPZ	Jumps	Jump if Plus or Zero
JU	Jumps	Jump Unconditional
JUO	Jumps	Jump if Unordered
JZ	Jumps	Jump if Zero
L	Load/Transfer	Load
L DBLG	Load/Transfer	Load Length of Shared DB in ACCU 1
L DBNO	Load/Transfer	Load Number of Shared DB in ACCU 1
L DILG	Load/Transfer	Load Length of Instance DB in ACCU 1
L DINO	Load/Transfer	Load Number of Instance DB in ACCU 1
L STW	Load/Transfer	Load Status Word into ACCU 1
L	Timers	Load Current Timer Value into ACCU 1 as Integer (the current timer value can be a number from 0 to 255, for example, L T 32)
L	Counters	Load Current Counter Value into ACCU 1 (the current counter value can be a number from 0 to 255, for example, L C 15)
LAR1	Load/Transfer	Load Address Register 1 from ACCU 1
LAR1 <D>	Load/Transfer	Load Address Register 1 with Double Integer (32-bit Pointer)
LAR1 AR2	Load/Transfer	Load Address Register 1 from Address Register 2
LAR2	Load/Transfer	Load Address Register 2 from ACCU 1
LAR2 <D>	Load/Transfer	Load Address Register 2 with Double Integer (32-bit Pointer)
LC	Counters	Load Current Counter Value into ACCU 1 as BCD (the current timer value can be a number from 0 to 255, for example, LC C 15)
LC	Timers	Load Current Timer Value into ACCU 1 as BCD (the current counter value can be a number from 0 to 255, for example, LC T 32)
LEAVE	Accumulator	Leave ACCU Stack
LN	Floating point Instruction	Generate the Natural Logarithm of a Floating-point Number (32-bit)
LOOP	Jumps	Loop
MCR(	Program control	Save RLO in MCR Stack, Begin MCR
)MCR	Program control	End MCR

MCRA	Program control	Activate MCR Area
MCRD	Program control	Deactivate MCR Area
MOD	Integer math Instruction	Division Remainder Double Integer (32-bit)
NEGD	Convert	Twos Complement Double Integer (32-bit)
NEGI	Convert	Twos Complement Integer (16-bit)
NEGR	Convert	Negate Floating-point Number (32-bit, IEEE-FP)
NOP 0	Accumulator	Null Instruction
NOP 1	Accumulator	Null Instruction
NOT	Bit logic Instruction	Negate RLO
O	Bit logic Instruction	Or
O(	Bit logic Instruction	Or with Nesting Open
OD	Word logic Instruction	OR Double Word (32-bit)
ON	Bit logic Instruction	Or Not
ON(	Bit logic Instruction	Or Not with Nesting Open
OPN	DB call	Open a Data Block
OW	Word logic Instruction	OR Word (16-bit)
POP	Accumulator	POP
POP	Accumulator	CPU with Two ACCUs
POP	Accumulator	CPU with Four ACCUs
PUSH	Accumulator	CPU with Two ACCUs
PUSH	Accumulator	CPU with Four ACCUs
R	Bit logic Instruction	Reset
R	Counters	Reset Counter (the current counter can be a number from 0 to 255, for example, R C 15)
R	Timers	Reset Timer (the current timer can be a number from 0 to 255, for example, R T 32)
RLD	Shift/Rotate	Rotate Left Double Word (32-bit)
RLDA	Shift/Rotate	Rotate ACCU 1 Left via CC 1 (32-bit)
RND	Convert	Round
RND–	Convert	Round to Lower Double Integer
RND+	Convert	Round to Upper Double Integer
RRD	Shift/Rotate	Rotate Right Double Word (32-bit)
RRDA	Shift/Rotate	Rotate ACCU 1 Right via CC 1 (32-bit)
S	Bit logic Instruction	Set
S	Counters	Set Counter Preset Value (the current counter can be a number from 0 to 255, for example, S C 15)
SAVE	Bit logic Instruction	Save RLO in BR Register
SD	Timers	On-Delay Timer
SE	Timers	Extended Pulse Timer
SET	Bit logic Instruction	Set
SF	Timers	Off-Delay Timer
SIN	Floating point Instruction	Generate the Sine of Angles as Floating-point Numbers (32-bit)
SLD	Shift/Rotate	Shift Left Double Word (32-bit)
SLW	Shift/Rotate	Shift Left Word (16-bit)
SP	Timers	Pulse Timer

SQR	Floating point Instruction	Generate the Square of a Floating-point Number (32-bit)
SQRT	Floating point Instruction	Generate the Square Root of a Floating-point Number (32-bit)
SRD	Shift/Rotate	Shift Right Double Word (32-bit)
SRW	Shift/Rotate	Shift Right Word (16-bit)
SS	Timers	Retentive On-Delay Timer
SSD	Shift/Rotate	Shift Sign Double Integer (32-bit)
SSI	Shift/Rotate	Shift Sign Integer (16-bit)
T	Load/Transfer	Transfer
T STW	Load/Transfer	Transfer ACCU 1 into Status Word
TAK	Accumulator	Toggle ACCU 1 with ACCU 2
TAN	Floating point Instruction	Generate the Tangent of Angles as Floating-point Numbers (32-bit)
TAR1	Load/Transfer	Transfer Address Register 1 to ACCU 1
TAR1	Load/Transfer	Transfer Address Register 1 to Destination (32-bit Pointer)
TAR1	Load/Transfer	Transfer Address Register 1 to Address Register 2
TAR2	Load/Transfer	Transfer Address Register 2 to ACCU 1
TAR2	Load/Transfer	Transfer Address Register 2 to Destination (32-bit Pointer)
TRUNC	Convert	Truncate
UC	Program control	Unconditional Call
X	Bit logic Instruction	Exclusive Or
X(	Bit logic Instruction	Exclusive Or with Nesting Open
XN	Bit logic Instruction	Exclusive Or Not
XN(	Bit logic Instruction	Exclusive Or Not with Nesting Open
XOD	Word logic Instruction	Exclusive OR Double Word (32-bit)
XOW	Word logic Instruction	Exclusive OR Word (16-bit)