

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



آموزش برنامه نویسی

PLC

به زبان

STL

(Statement List)



فصل اول

مجموعه دستورات

Bit Logic

این دستورات عملیات را روی یک **بیت** انجام می دهند

$A \quad (AND)$

فرمت به کارگیری این دستور در برنامه نویسی STL به زبان به صورت زیر است

$A \quad < \text{آدرس یک بیت از حافظه} >$

مثال : $A \quad I0.0$

با اجرای دستور فوق محتوای آدرس $I0.0$ خوانده شده و با RLO عمل منطقی AND را انجام می دهد

و نتیجه را مجدداً در RLO ذخیره می کند

توجه : اگر دستور فوق در ابتدای یک $Network$ به کار رود محتوای حافظه $I0.0$ را در قرار RLO می دهد

= (Assign)

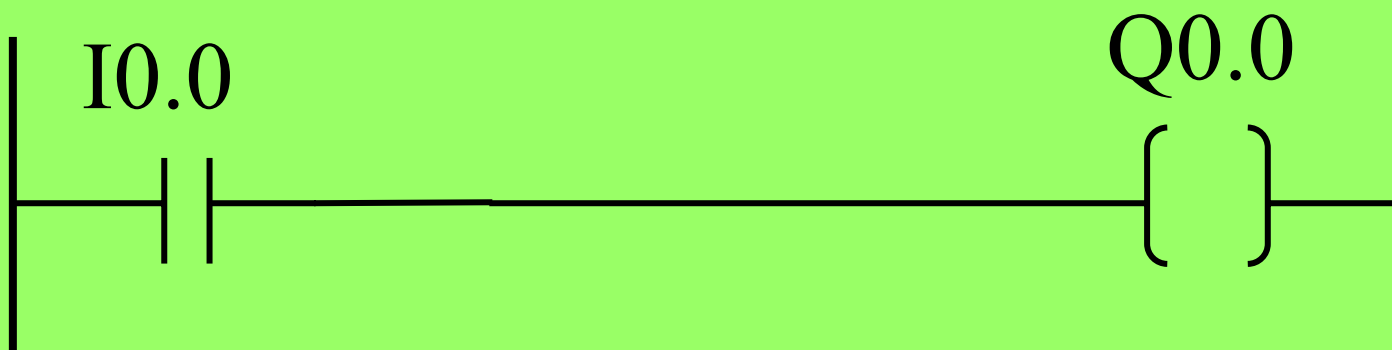
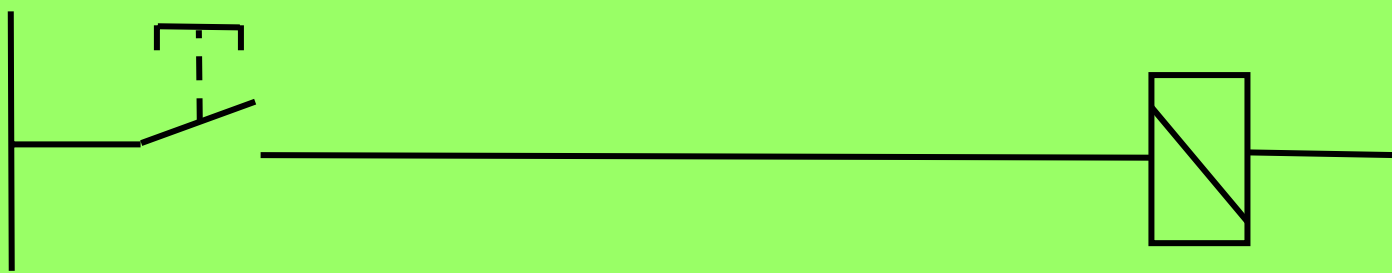
فرمت به کارگیری این دستور در برنامه نویسی *STL* به زبان به صورت زیر است

= < آدرس یک بیت از حافظه >

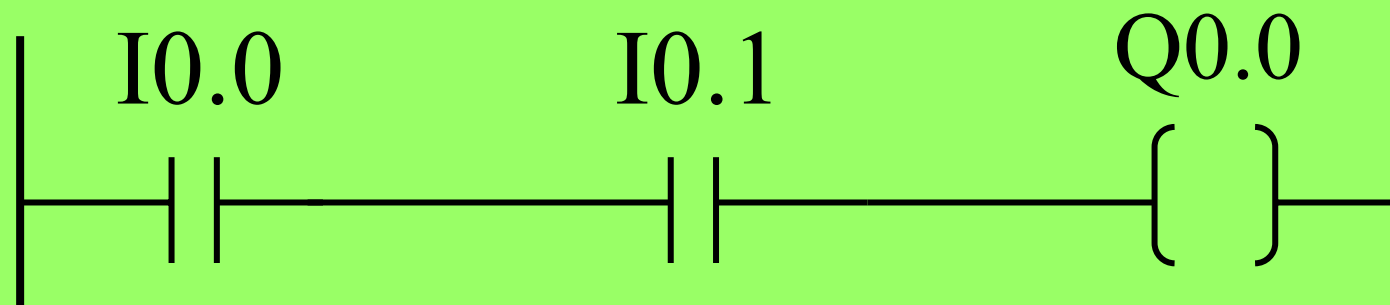
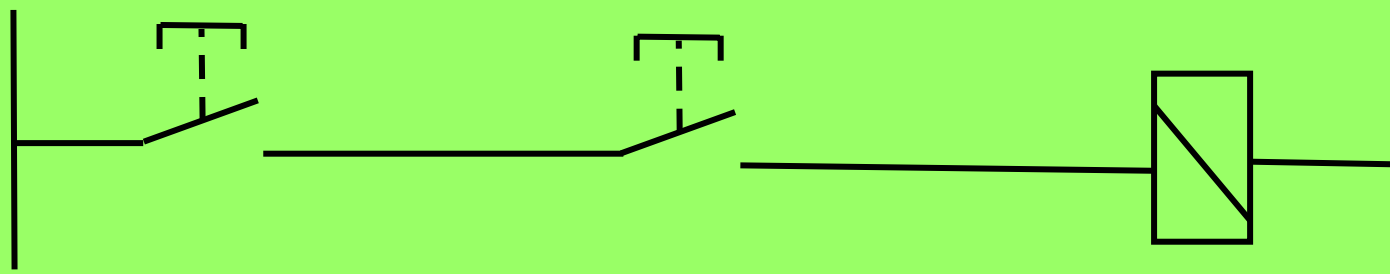
مثال : $Q0.0$

با اجرای دستور فوق محتوای *RLO* به آدرس داده شده ($Q0.0$) ارسال می گردد

توجه : برای اجرای دستور فوق یک استثناء نیز وجود دارد که بعداً راجع به آن در قسمت (*MCR*) توضیح داده خواهد شد



A I0.0
= Q0.0



A I0.0

A I0.1

= Q0.0

O (OR)

فرمت به کارگیری این دستور در برنامه نویسی STL به زبان به صورت زیر است

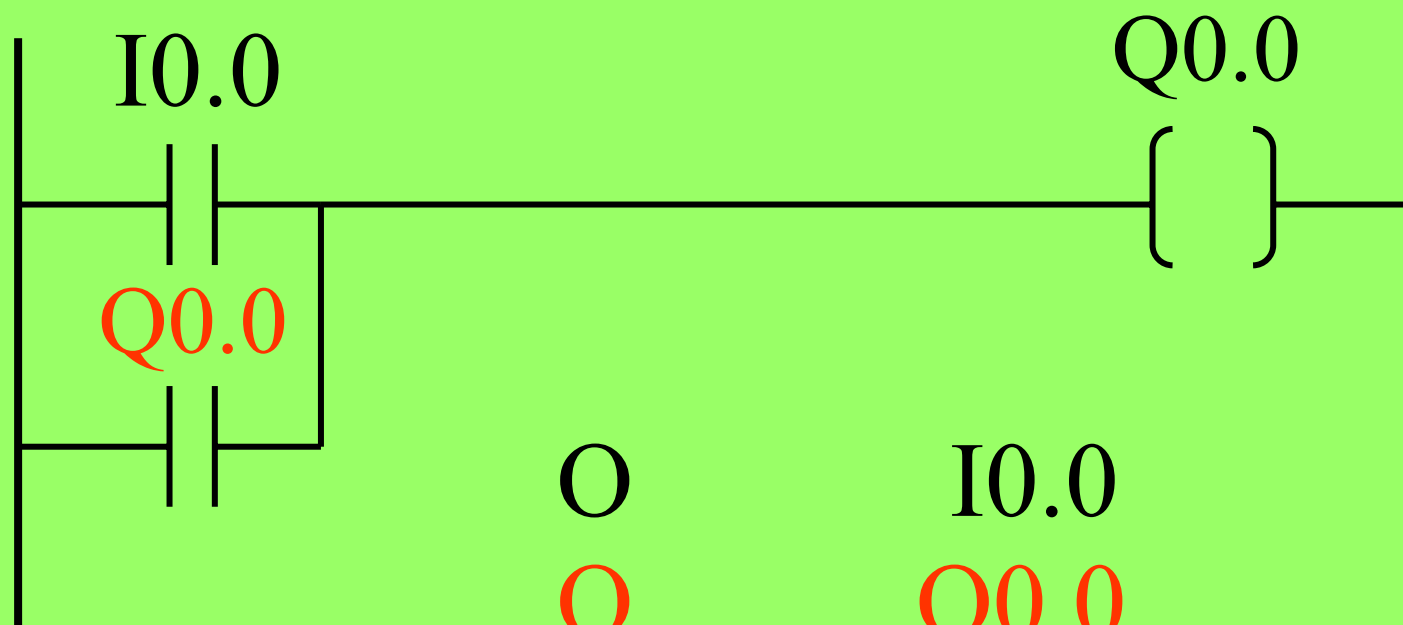
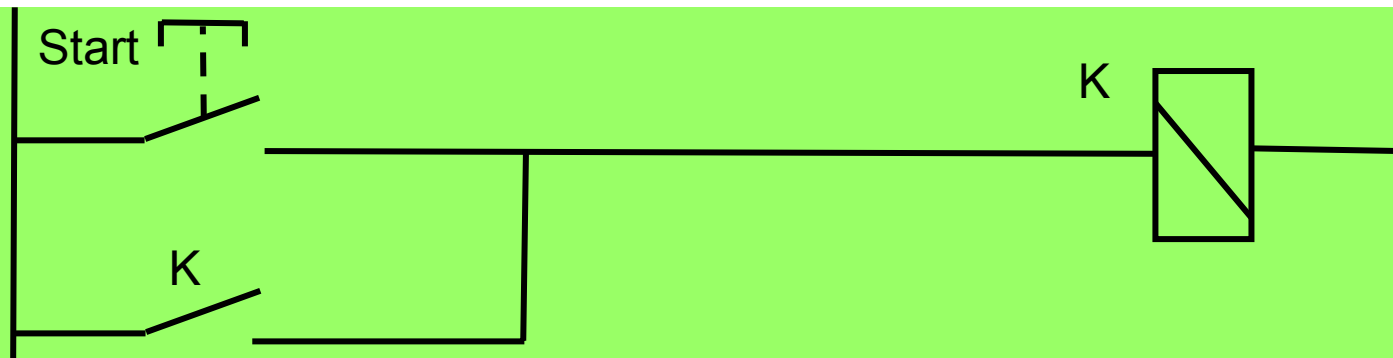
O < آدرس یک بیت از حافظه >

مثال: $O \quad M0.0$

با اجرای دستور فوق محتوای آدرس $M0.0$ خوانده شده و با RLO عمل منطقی OR را انجام می دهد

و نتیجه را مجدداً در RLO ذخیره می کند

توجه: اگر دستور فوق در ابتدای یک $Network$ به کار رود محتوای حافظه $M0.0$ را در RLO قرار می دهد



O

I0.0

O

Q0.0

=

Q0.0

AN (AND NOT)

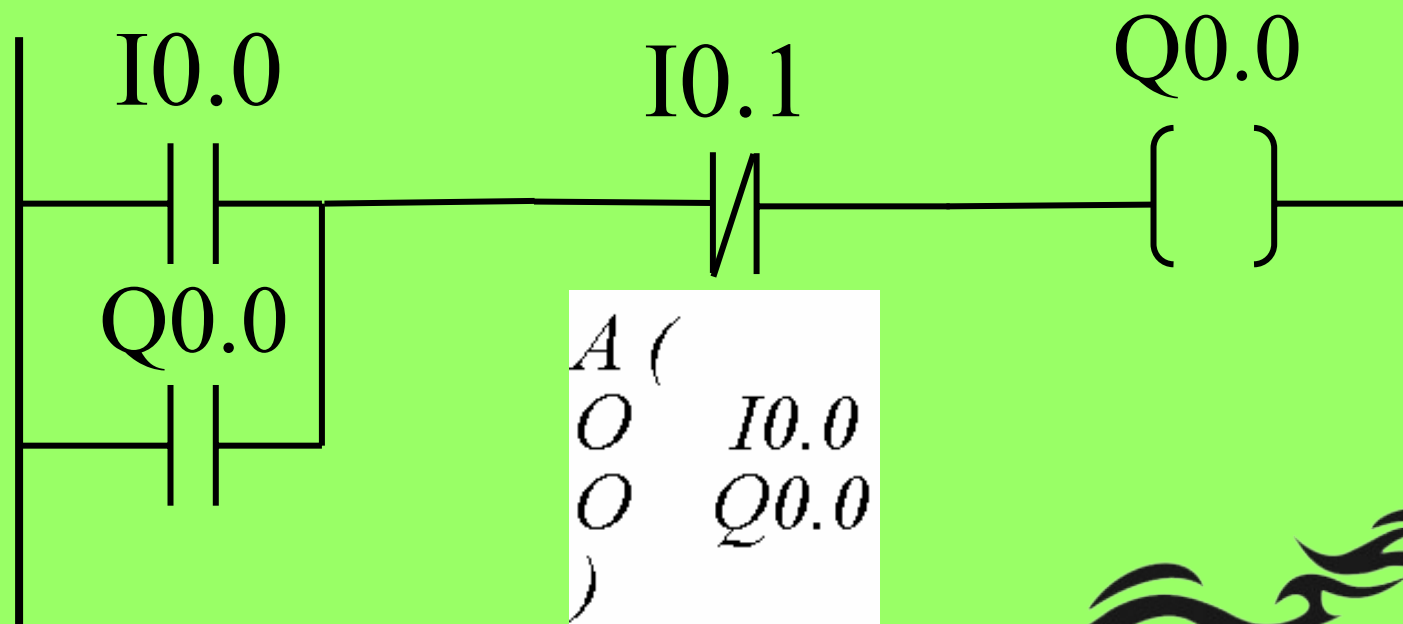
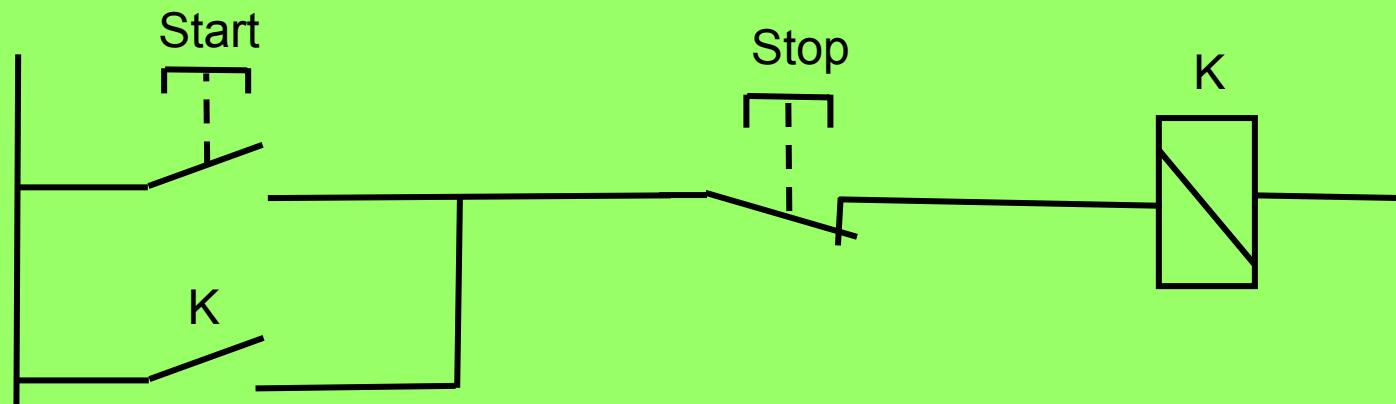
فرمت به کارگیری این دستور در برنامه نویسی *STL* به زبان به صورت زیر است

AN < آدرس یک بیت از حافظه >

مثال: *AN M3.6*

با اجرای دستور فوق محتوای آدرس *M3.6* خوانده شده و سپس نات (*NOT*) می گردد و آنگاه با *RLO* عمل منطقی *AND* را انجام می دهد و نتیجه را مجدداً در *RLO* ذخیره می کند

توجه: اگر دستور فوق در ابتدای یک *Network* به کار رود محتوای حافظه *M3.6* را خوانده شده و سپس نات (*NOT*) می کند و آنگاه آن را در *RLO* قرار می دهد



$$A ($$

$$O \quad I0.0$$

$$O \quad Q0.0$$

$$)$$

$$AN \quad I0.1$$

$$= \quad Q0.0$$



ON (OR NOT)

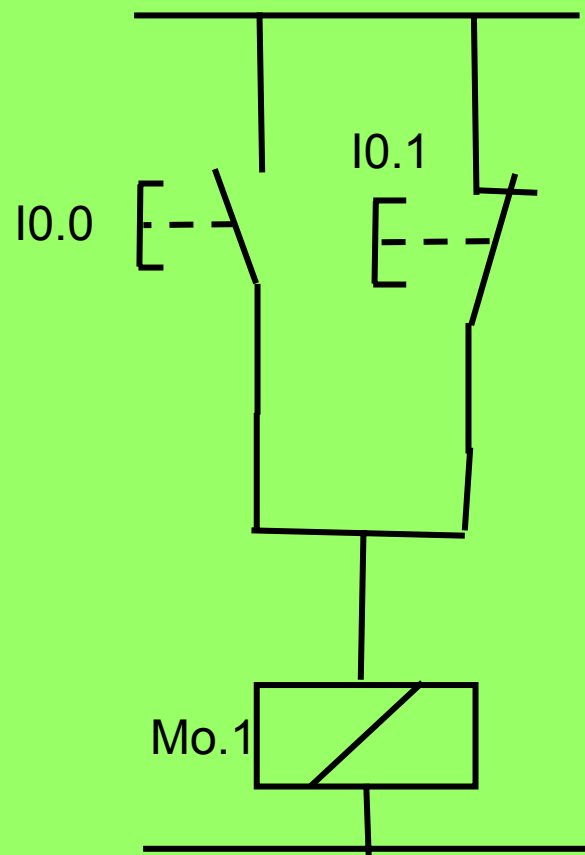
فرمت به کارگیری این دستور در برنامه نویسی *STL* به زبان به صورت زیر است

ON < آدرس یک بیت از حافظه >

مثال: *ON M3.6*

با اجرای دستور فوق محتوای آدرس *M3.6* خوانده شده و سپس نات (*NOT*) می گردد و آنگاه با *RLO* عمل منطقی *OR* را انجام می دهد و نتیجه را مجدداً در *RLO* ذخیره می کند

توجه: اگر دستور فوق در ابتدای یک *Network* به کار رود محتوای حافظه *M3.6* را خوانده شده و سپس نات (*NOT*) می کند و آنگاه آن را در *RLO* قرار می دهد



اگر بخواهیم عملکرد منطقی مدار مقابل را بنویسیم خواهیم داشت :

$$K = I0.0 + \overline{I0.1}$$

حال اگر بخواهیم عملکرد منطقی مدار مقابل را به زبان *STL* بنویسیم خواهیم داشت :

O I0.0

ON I0.1

= M0.1

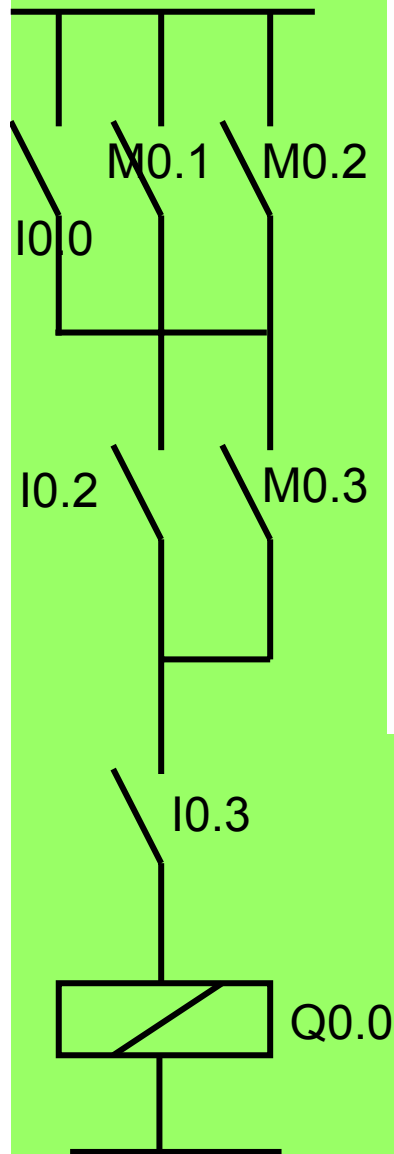
A (AND with nesting open)) Nesting closed

به مدار مقابل توجه کنید اگر رابطه منطقی آن را بنویسیم خواهیم داشت

$$Q0.0 = \underbrace{(I0.0 + M0.1 + M0.2)}_{\text{جمله ۱}} \underbrace{(I0.2 + M0.3)}_{\text{جمله ۲}} \underbrace{(I0.3)}_{\text{جمله ۳}}$$

OR *AND*

جمله های ۱ و ۲ و ۳ با یک دیگر *AND* شده اند و هر جمله شامل چندین متغیر است که با یک دیگر *OR* شده اند
لذا برای نوشتن برنامه فوق به زبان *STL* نیاز به پرانتز داریم که ابتدا متغیرها با یک دیگر *OR* شوند و سپس حاصل این *OR* ها با یک دیگر *AND* شوند
در چنین مواردی و موارد مشابه دیگر از *A* و در انتهای جمله از *)* استفاده می کنیم



```

A (
O  I0.0
O  M0.0
O  M0.2
)
A (
O  I0.2
O  M0.3
)
A  I0.3
=  Q0.0

```

O(OR with nesting open *) Nesting closed*

به مدار مقابل توجه کنید اگر رابطه منطقی آن را بنویسیم خواهیم داشت

$$Q0.0 = (I0.0 \cdot M0.1 \cdot M0.2) + (I0.2 \cdot M0.3) + (I0.3)$$

جمله ۱ جمله ۲ جمله ۳

AND OR

جمله های ۱ و ۲ و ۳ با یک دیگر OR شده اند و هر جمله شامل چندین متغیر است که با یک دیگر AND شده اند
 لذا برای نوشتن برنامه فوق به زبان STL نیاز به پرانتز داریم که ابتدا متغیرها با یک دیگر AND شوند و سپس حاصل این AND ها با یک دیگر OR شوند

در چنین مواردی و موارد مشابه دیگر از () استفاده می کنیم

```

O (
A  I0.0
A  M0.0
A  M0.2
)
O (
A  I0.2
A  M0.3
)
O  I0.3
=  Q0.0
    
```



AN(*AND NOT with nesting open*

) *Nesting closed*

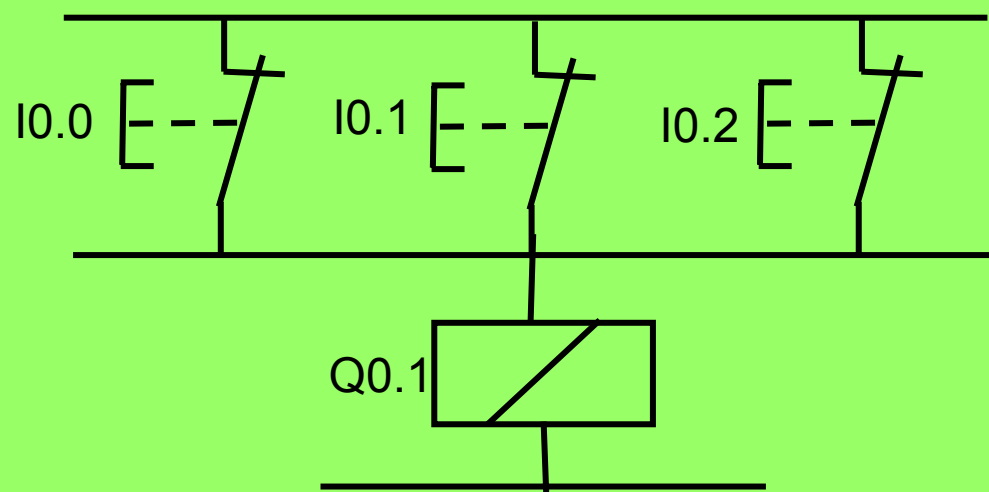
با دستور فوق ابتدا عملیات منطقی داخل پرانتز انجام شده و سپس نتیجه عملیات منطقی نات (*NOT*) شده و آنگاه این نتیجه برای عملیات منطقی *AND* با سایر نتایج قابل استفاده است .

به مدار زیر توجه کنید اگر رابطه منطقی آن را بنویسیم خواهیم داشت

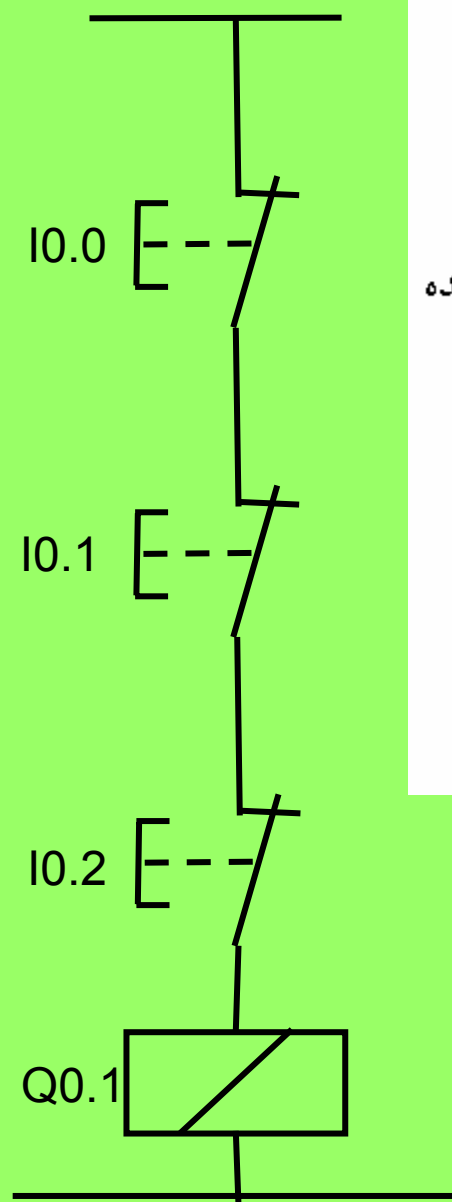
$$Q0.0 = (\overline{I0.0} + \overline{I0.1} + \overline{I0.2})$$

$$Q0.0 = \overline{(I0.0 \cdot I0.1 \cdot I0.2)}$$

طبق قضیه دمورگان



AN (
A I0.0
A I0.0
A I0.2
)



ON(OR NOT with nesting open) Nesting closed

با دستور فوق ابتدا عملیات منطقی داخل پرانتز انجام شده و سپس نتیجه عملیات منطقی نات (NOT) شده و آنگاه این نتیجه برای عملیات منطقی با سایر نتایج قابل استفاده است .

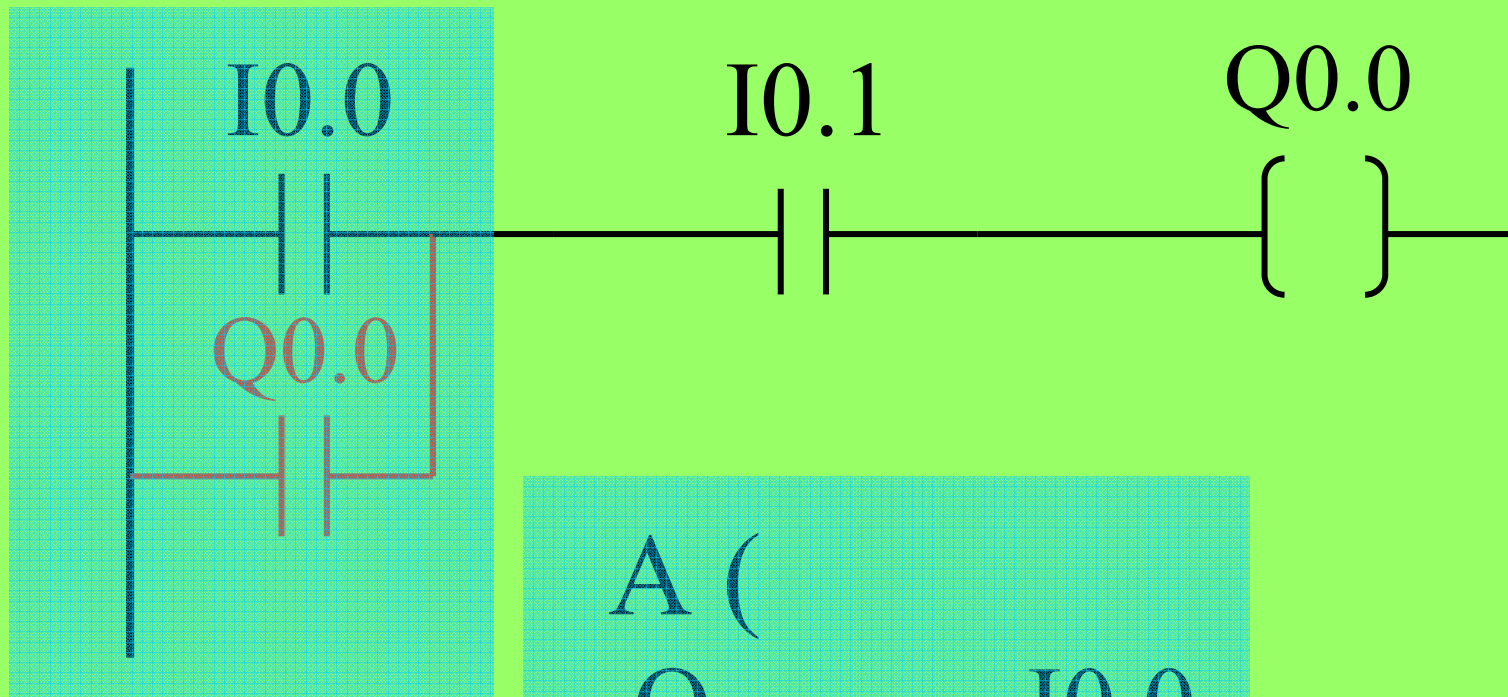
به مدار مقابل توجه کنید اگر رابطه منطقی آن را بنویسیم خواهیم داشت

$$Q0.0 = (\overline{I0.0} \cdot \overline{I0.1} \cdot \overline{I0.2})$$

$$Q0.0 = \overline{(I0.0 + I0.1 + I0.2)}$$

طبق قضیه دمورگان

ON (
 O I0.0
 O I0.0
 O I0.2
)



A (

O

O

)

$I0.0$

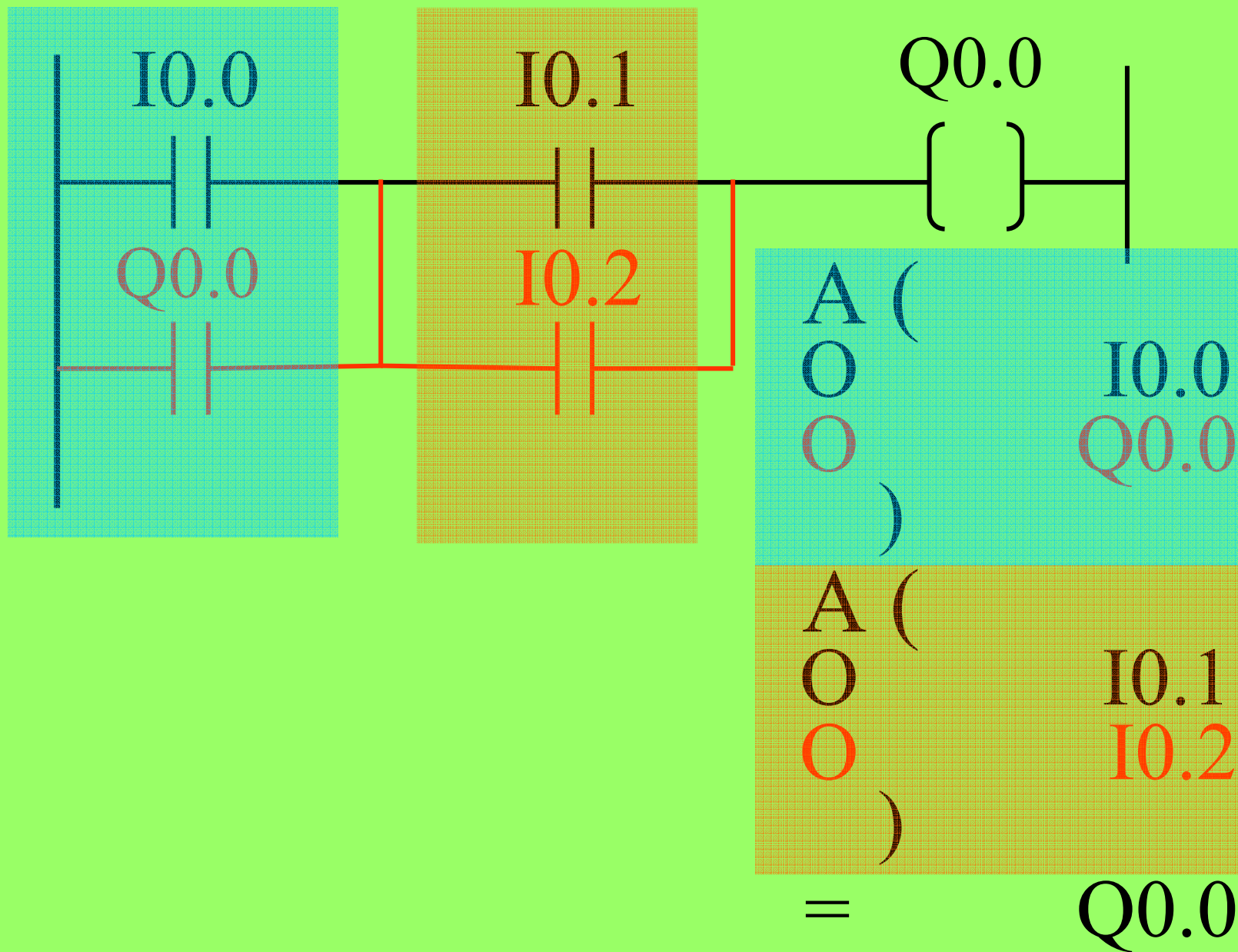
$Q0.0$

A

$I0.1$

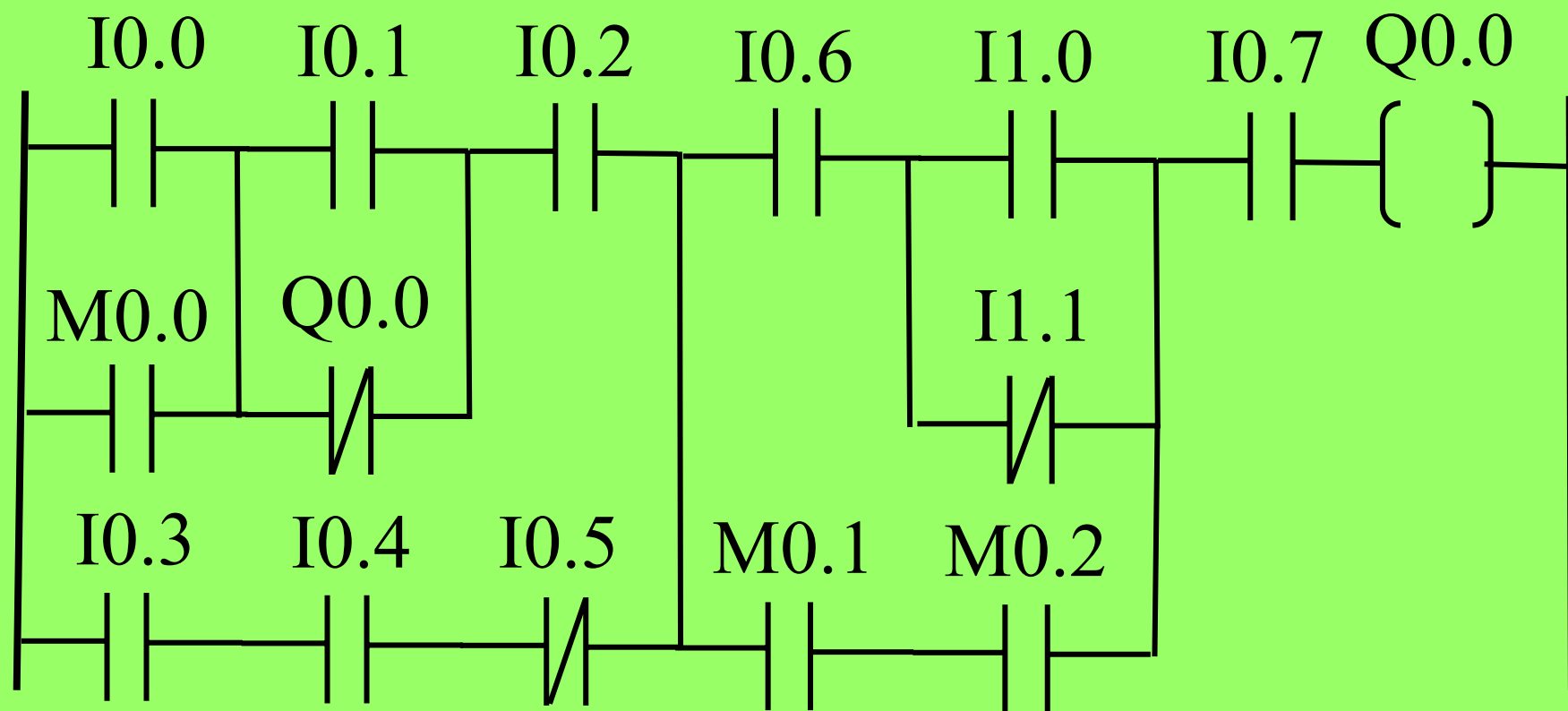
=

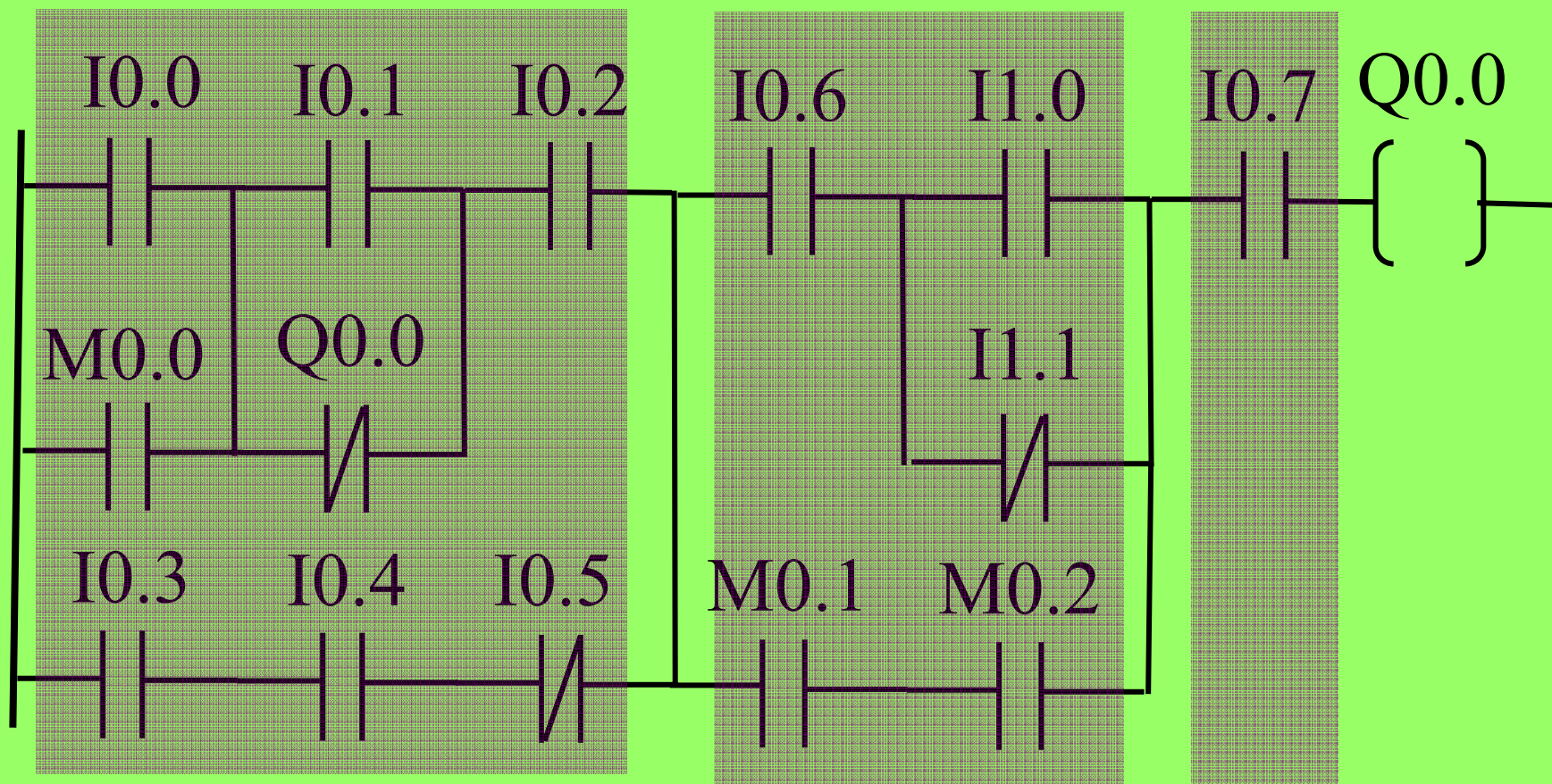
$Q0.0$

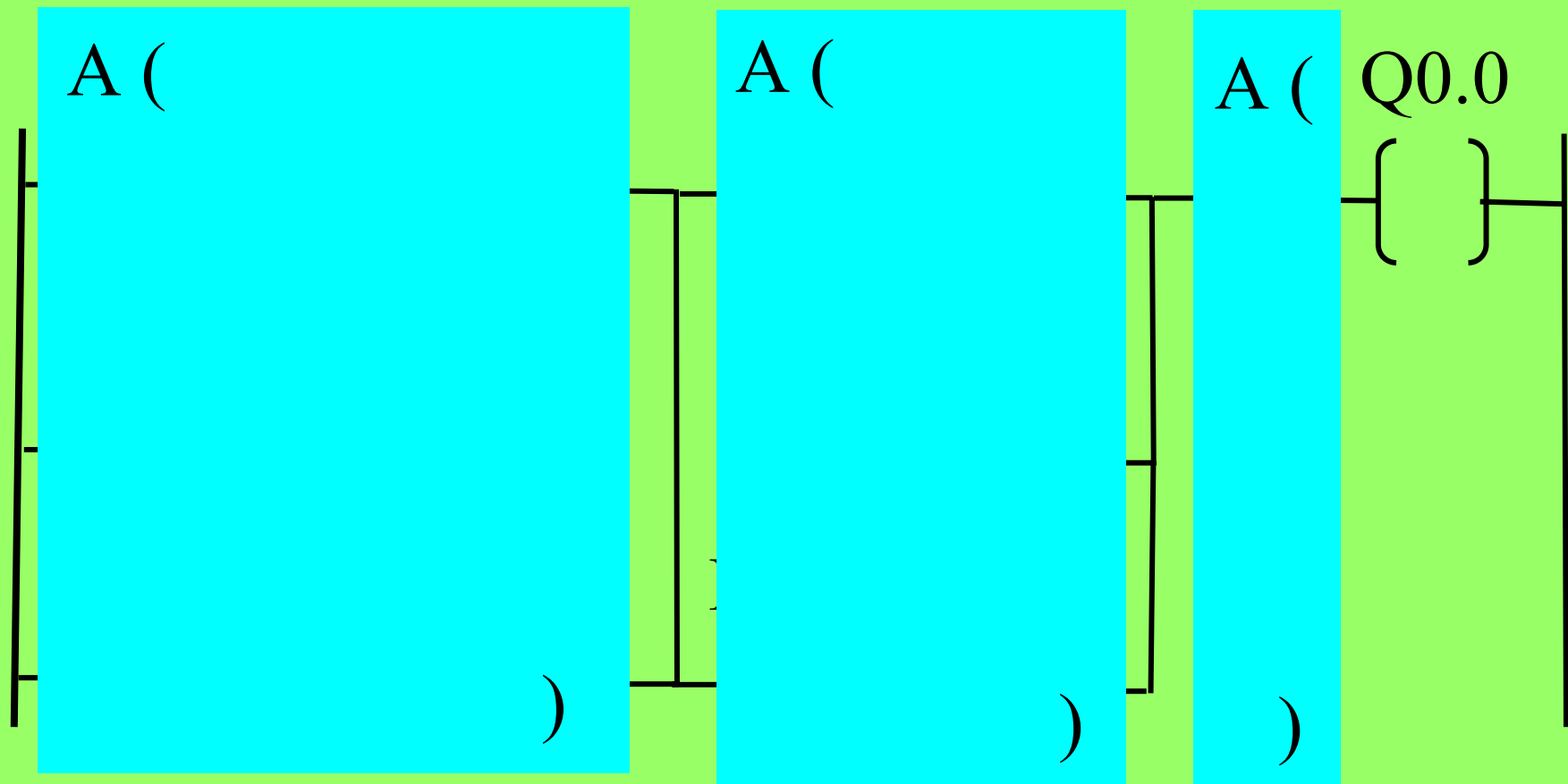


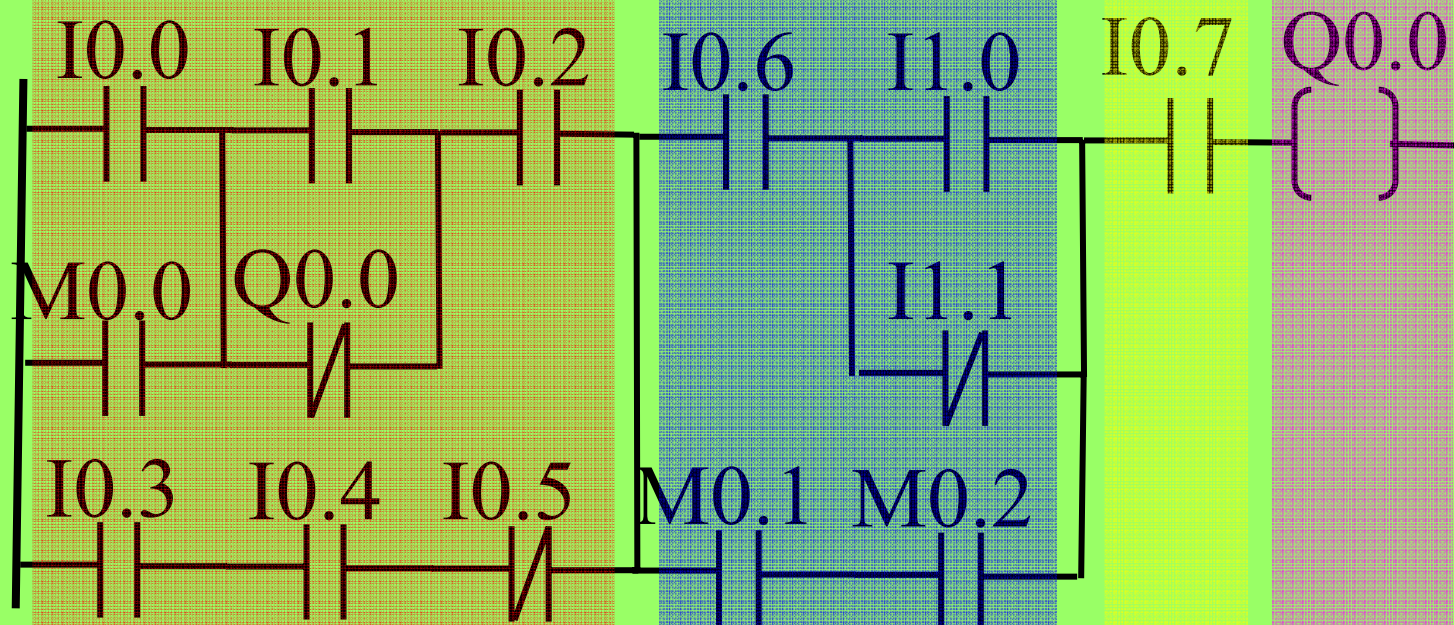
یک مثال

برنامه STL اسلاید صفحه بعد را بنویسید









```

A(
O      I      0.0
O      M      0.0
)
A(
O      I      0.1
ON     Q      0.0
)
A      I      0.2
O
A      I      0.3
A      I      0.4
AN     I      0.5
)

```

```

A(
A      I      0.6
A(
O      I      1.0
ON     I      1.1
)
O
A      M      0.1
A      M      0.2
)

```

= Q 0.0

A I 0.7

X (Exclusive OR)

فرمت به کارگیری این دستور در برنامه نویسی STL به زبان به صورت زیر است

X < آدرس یک بیت از حافظه >

مثال: $M3.6$ X

با اجرای دستور فوق محتوای آدرس $M3.6$ خوانده شده و آنگاه با RLO عمل منطقی XOR را انجام می دهد و نتیجه را مجدداً در RLO ذخیره می کند

توجه: اگر دستور فوق در ابتدای یک $Network$ به کار رود محتوای حافظه $M3.6$ را خوانده شده و سپس با $RLO=0$ - عمل منطقی XOR را انجام می دهد و نتیجه را مجدداً در RLO ذخیره می کند

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

جهت یادآوری

جدول صحت OR انحصاری

X(Exclusive OR with nesting open
) Nesting closed

با اجرای دستور فوق ابتدا عملیات منطقی داخل پرانتز انجام شده و سپس نتیجه عملیات با $RLO - OR$ انحصاری شده و نتیجه مجدداً در RLO ذخیره می شود

توجه: اگر دستور فوق در ابتدای یک *Network* به کار رود نتیجه عمل منطقی داخل پرانتز با $OR - RLO = 0$ انحصاری شده و نتیجه را مجدداً در *RLO* ذخیره می کند

XN (*Exclusive OR NOT*)

فرمت به کارگیری این دستور در برنامه نویسی STL به زبان به صورت زیر است

XN < آدرس یک بیت از حافظه >

مثال: XN $M3.6$

با اجرای دستور فوق محتوای آدرس $M3.6$ خوانده شده و سپس نات (NOT) می گردد و آنگاه

با RLO عمل منطقی XOR را انجام می دهد و نتیجه را مجدداً در RLO ذخیره می کند

توجه: اگر دستور فوق در ابتدای یک $Network$ به کار رود محتوای

حافظه $M3.6$ را خوانده و سپس نات (NOT) می کند و آنگاه

با $RLO=0$ - عمل منطقی XOR را انجام می دهد و نتیجه را

مجدداً در RLO ذخیره می کند

*XN(Exclusive OR NOT with nesting open
) Nesting closed*

با اجرای دستور فوق ابتدا عملیات منطقی داخل پرانتز انجام شده و سپس نتیجه عملیات نات (*NOT*) شده و آنگاه با *RLO - OR* انحصاری شده و نتیجه مجدداً در *RLO* ذخیره می شود

توجه: اگر دستور فوق در ابتدای یک *Network* به کار رود ابتدا عملیات منطقی داخل پرانتز انجام شده

و سپس نتیجه عملیات نات (*NOT*) شده و آنگاه

با *RLO=0 - OR* انحصاری شده و نتیجه را مجدداً در *RLO* ذخیره می کند

S Set

فرمت به کارگیری این دستور در برنامه نویسی STL به زبان به صورت زیر است

S < آدرس یک بیت از حافظه >

مثال: $S \quad M3.6$

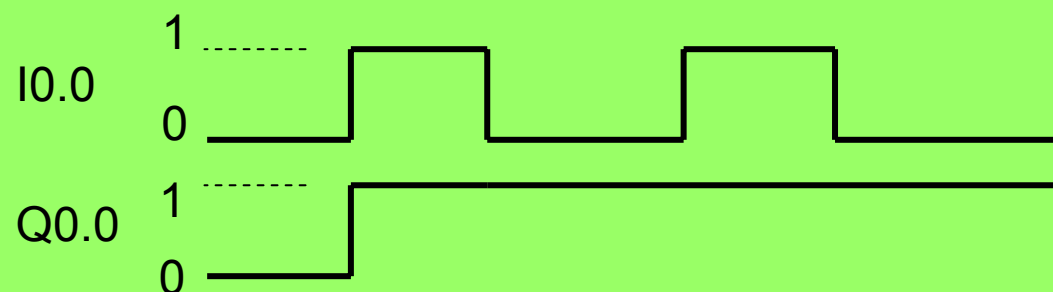
با اجرای دستور فوق اگر $RLO=1$ باشد محتوای بیت آدرس داده شده برابر یک میشود

یعنی: $M3.6 = 1$

حال اگر $RLO=0$ شد محتوای بیت آدرس داده شده برابر یک میماند اگر تحت هر شرایطی

بخواهیم محتوای بیت آدرس داده شده برابر صفر شود این کار با دستور S امکان پذیر نیست

و باید با دستور $R(Reset)$ محتوای بیت آدرس داده شده را صفر کنیم



A	I0.0
S	Q0.0

R *Reset*

فرمت به کارگیری این دستور در برنامه نویسی *STL* به زبان به صورت زیر است

R < آدرس یک بیت از حافظه >

مثال: *M3.6* *R*

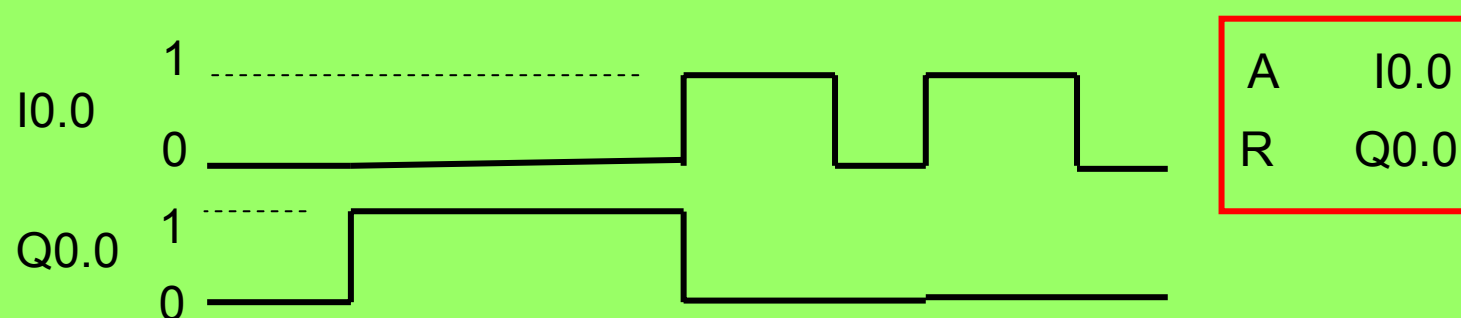
با اجرای دستور فوق اگر $RLO=1$ باشد محتوای بیت آدرس داده شده برابر صفر میشود

یعنی: $M3.6 = 0$

حال اگر $RLO=0$ شد محتوای بیت آدرس داده شده برابر صفر میماند اگر تحت هر شرایطی

بخواهیم محتوای بیت آدرس داده شده برابر یک شود این کار با دستور *R* امکان پذیر نیست

و باید با دستور *S(Set)* محتوای بیت آدرس داده شده را یک کنیم



با دستورالعمل هاي زیر می توان مقدار *RLO* را تغییر داد

NOT

(۱) با اجرای این دستورالعمل مقدار *RLO* هرچه هست معکوس می شود
(صفر را یک و یک را صفر می کند)

SET

(۲) با اجرای این دستورالعمل مقدار *RLO* را برابر یک می شود

CLR

(۳) با اجرای این دستورالعمل مقدار *RLO* را برابر صفر می شود

SAVE

(۴) با اجرای این دستورالعمل مقدار *RLO* را در رجیستر *BR* ضبط (ذخیره) می شود

مثال

SET  RLO=1

= M0.0

M0.0=1

= Q0.0

Q0.0=1

CLR  RLO=0

= M0.6

M0.6=0

= Q0.3

Q0.3=0

مثال

SET  RLO=1

= M0.5

M0.5=1



NOT  RLO=0

= M0.6

M0.6=0



NOT  RLO=1

= M0.6

M0.6=1



CLR  RLO=0

NOT  RLO=1

= Q0.0

Q0.0=1

FN (Edge Negative)

فرمت به کارگیری این دستور در برنامه نویسی به زبان *STL* به صورت زیر است

FN < آدرس یک بیت از حافظه >

مثال : *FN M0.0*

دستور فوق سبب آشکار کردن لبه نزولی (تغییر از یک به صفر) *RLO* می گردد و این مسئله باعث می شود تا *RLO=1* شود

عملکرد برنامه به صورت یک مثال توضیح داده خواهد شد

FN M0.0

= *Q0.0*

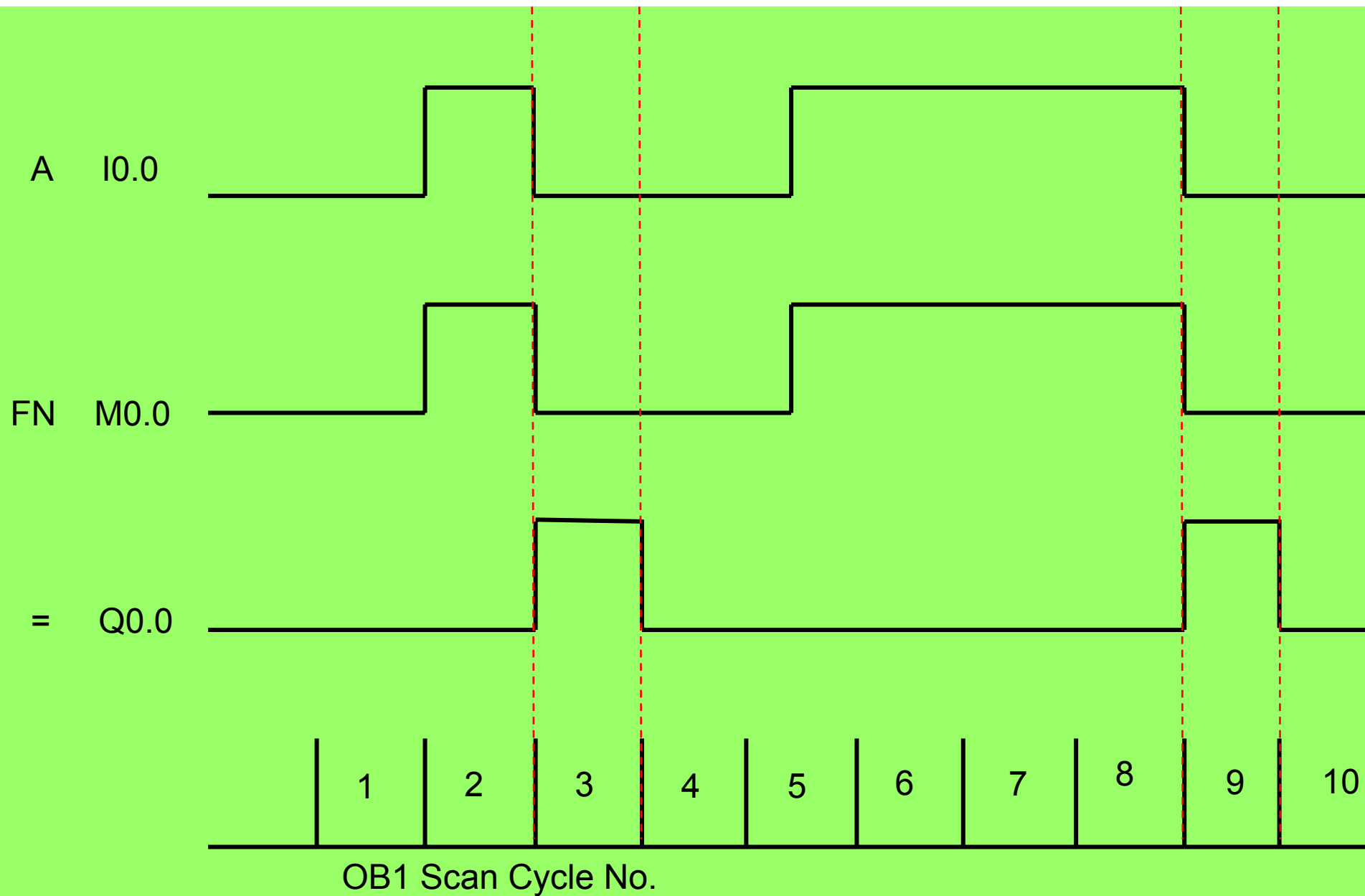
با اجرای دستور *I0.0* مقدار متغیر *A* مقدار متغیر *I0.0* با *RLO* - *AND* شده و نتیجه

مجدداً در *RLO* ذخیره می شود

و با اجرای دستور *FN M0.0* مقدار *RLO* در *M0.0* ذخیره می گردد

در ضمن در این حالت تست می شود که آیا مقدار *RLO* جدید با مقدار اسکن قبلی *RLO* تغییر

کرده است یا خیر در صورت تغییر در لبه نزولی تغییر *RLO* مقدار *RLO* برابر یک می گردد .



FP (Edge Positive)

فرمت به کارگیری این دستور در برنامه نویسی به زبان *STL* به صورت زیر است
FP < آدرس یک بیت از حافظه >

مثال : *FP M0.0*

دستور فوق موجب آشکار کردن لبه صعودی (تغییر از صفر به یک) *RLO* می گردد و این مسئله باعث می شود
تا *RLO=1* شود
A I0.0

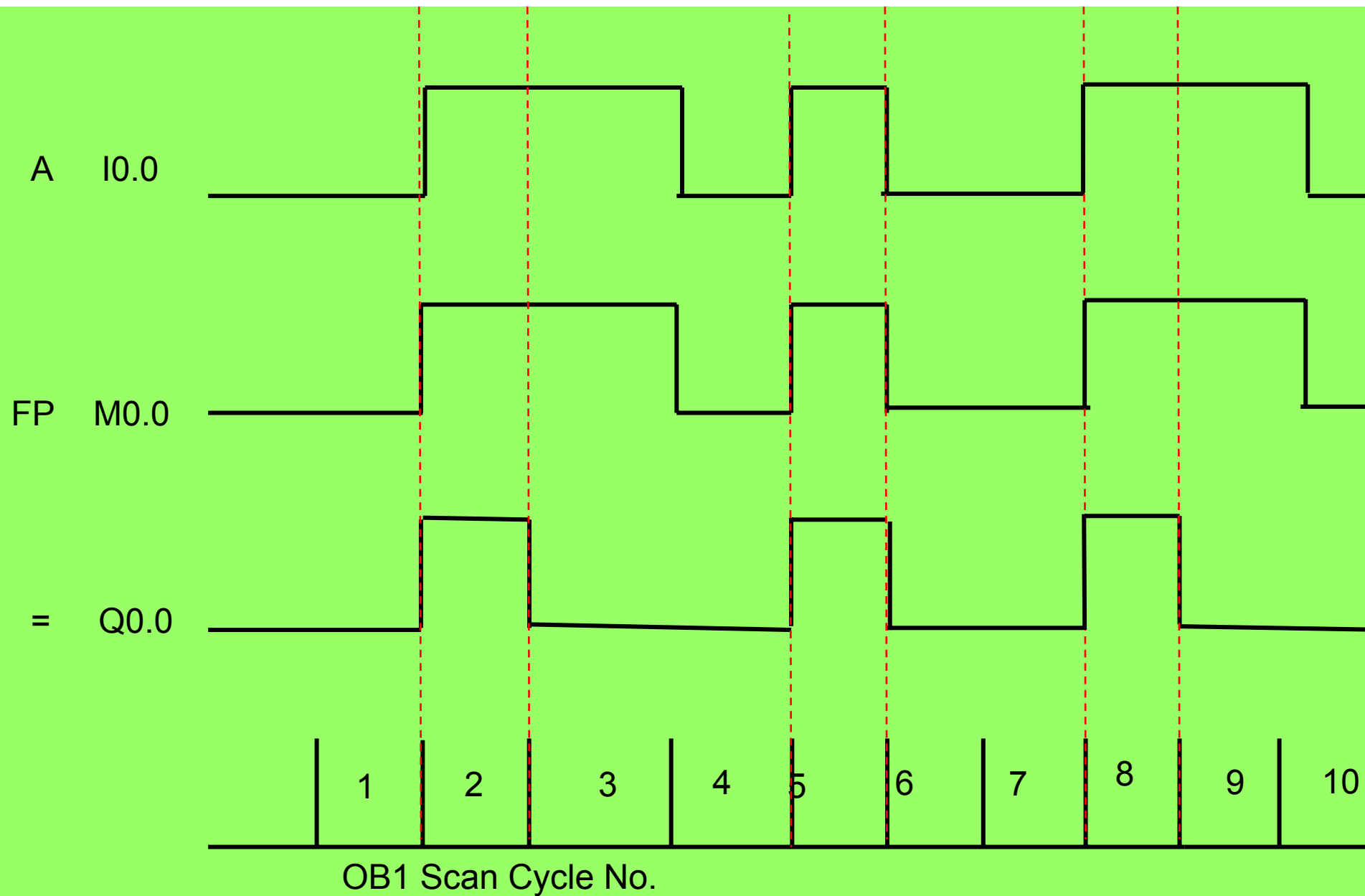
عملکرد برنامه به صورت یک مثال توضیح داده خواهد شد
FP M0.0

با اجرای دستور *I0.0* مقدار متغیر *I0.0* با *RLO* - *AND* شده و نتیجه
=*Q0.0*
مجدداً در *RLO* ذخیره می شود

و با اجرای دستور *FP M0.0* مقدار *RLO* در *M0.0* ذخیره می گردد

در ضمن در این حالت تست می شود که آیا مقدار *RLO* جدید با مقدار اسکن قبلی *RLO* تغییر

کرده است یا خیر در صورت تغییر در لبه صعودی تغییر *RLO* - مقدار *RLO* برابر یک می گردد.



پایان



فصل

اول

فصل دوم

مجموعه دستورات

Comparison Instruction

مقدمه

CPU هاي S7 داراي اکومولاتورها – رجیسترها و Stack هاي مختلفي هستند که هر کدام حسب نیاز به موقع توضیح داده خواهند شد .

در این قسمت درس – نیاز به شناخت اکومولاتورهاي یک CPU داریم .

بیشتر CPU هاي S7 داراي دو اکومولاتور هستند که با ACCU1 و

ACCU2 شناخته مي شوند . برخي CPU هاي S7 داراي ۴ اکومولاتور

مي باشند یعنی علاوه بر موارد فوق - ACCU3 و ACCU4 را نیز دارند .

ساختار داخلی یک اکومولاتور در S7

ACCU1			
ACCU1-H		ACCU1-L	
ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
HIGH WORD HIGH BYTE	HIGH WORD LOW BYTE	LOW WORD HIGH BYTE	LOW WORD LOW BYTE
3124	2316	158	70 وقتي یک بایت بار شود وارد این قسمت مي شود
		وقتي یک WORD بار شود وارد این قسمت مي شود	
وقتي یک DWORD بار شود وارد این قسمت مي شود			

دستورات مقایسه ای

دستورات مقایسه ای در S7 همیشه محتویات ACCU2 با ACCU1 مقایسه می گردد . و شش حالت مطابق جدول زیر ممکن است اتفاق بیفتد :

==	مساوی	ACCU2 is equal to ACCU1
<>	نا مساوی	ACCU2 is not equal to ACCU1
>	بزرگتر	ACCU2 is greater than to ACCU1
<	کوچکتر	ACCU2 is less than to ACCU1
>=	بزرگتر یا مساوی	ACCU2 is greater than or equal to ACCU1
<=	کوچکتر یا مساوی	ACCU2 is less than or equal to ACCU1

نوع داده هايي كه با يك ديگر مي توانند مقايسه شوند عبارتند از :

۱- داده هاي ۱۶ بيتي از نوع صحيح (Integer)

۲- داده هاي ۳۲ بيتي از نوع صحيح (Double Integer)

۳- داده هاي ۳۲ بيتي از نوع مميز شناور (Real)

۱- داده های ۱۶ بیتی از نوع صحیح (Integer)

کاراکتر هایی که در زبان برنامه نویسی STL برای عمل مقایسه بین دو مقدار

برای داده های نوع صحیح ۱۶ بیتی به کار می روند عبارتند از :

<	کوچکتر از
>=	بزرگتر از یا مساوی
<=	کوچکتر از یا مساوی

==	مساوی
<>	نا مساوی
>	بزرگتر از

۲- داده های ۳۲ بیتی از نوع صحیح (Double Integer)

کاراکتر هایی که در زبان برنامه نویسی STL برای عمل مقایسه بین دو مقدار

برای داده های نوع صحیح ۳۲ بیتی به کار می روند عبارتند از :

<D	کوچکتر از
>=D	بزرگتر از یا مساوی
<=D	کوچکتر از یا مساوی

==D	مساوی
<>D	نا مساوی
>D	بزرگتر از

۳- داده های ۳۲ بیتی از نوع ممیز شناور (Real)

کاراکتر هایی که در زبان برنامه نویسی STL برای عمل مقایسه بین دو مقدار

برای داده های نوع ممیز شناور ۳۲ بیتی به کار می روند عبارتند از :

<R	کوچکتر از
>=R	بزرگتر از یا مساوی
<=R	کوچکتر از یا مساوی

==R	مساوی
<>R	نا مساوی
>R	بزرگتر از

مثال ۱

می خواهیم محتویات MW0 را با محتویات MW1 مقایسه کنیم و ببینیم آیا MW0 از MW1 بزرگتر است یا خیر . در صورت بزرگتر بودن فلگ

M0.0 را برابر یک کند

برنامه PLC به زبان STL به صورت زیر است :

برای توضیح بیشتر به اسلاید بعدی توجه کنید

L	MW1
L	MW0
>	
=	M0.0



L

MW1



MW1	ACCU1
	ACCU2

L

MW0



MW0	ACCU1
MW1	ACCU2

>I

آیا محتویات ACCU2 بزرگتر از محتویات ACCU1 است
در صورت صحت داشتن شرط $RLO=1$ می گردد .

= M0.0



مقدار RLO را در M0.0 قرار بده

مثال ۲

می خواهیم محتویات MW0 را با عدد ۱۲۵۶ مقایسه کنیم و ببینیم آیا

MW0 از ۱۲۵۶ بزرگتر یا مساوی است یا خیر. در صورت بزرگتر بودن فلگ

M4.3 را برابر یک کند

L MW0

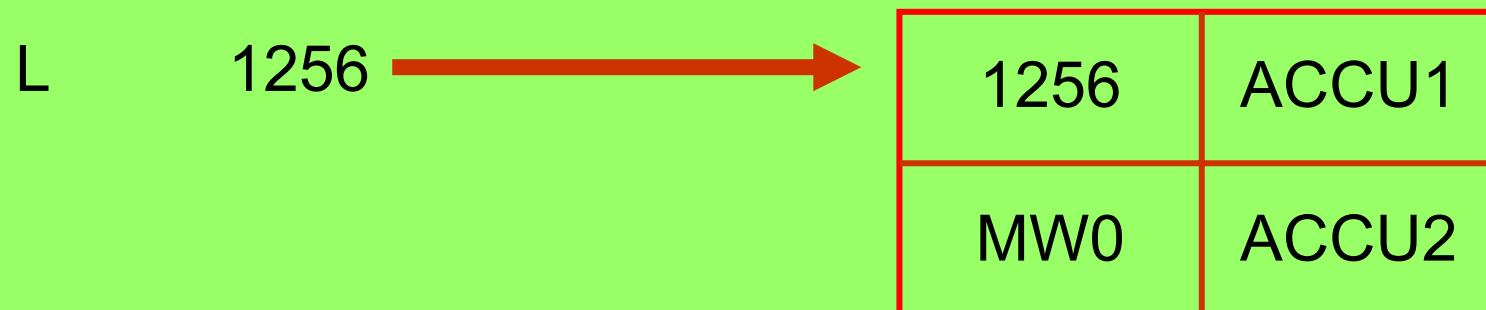
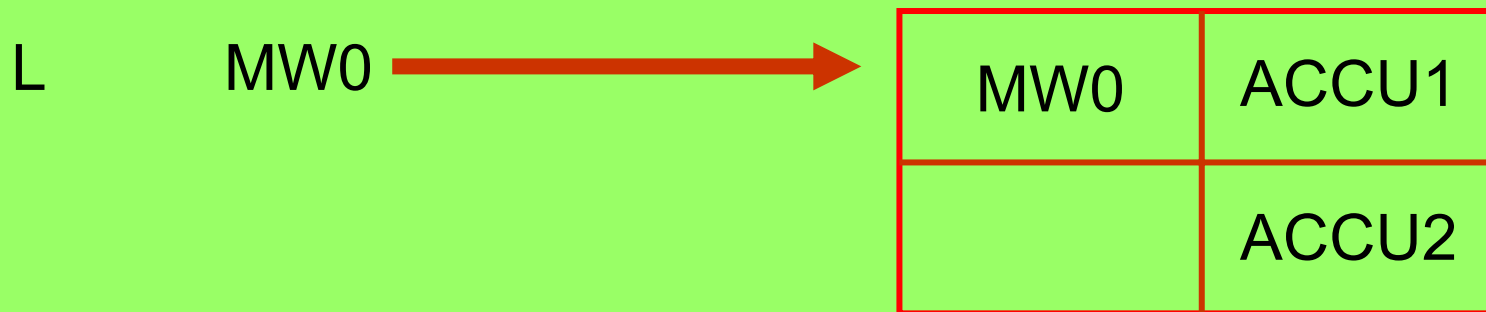
L 1256

>=I

= M4.3

برنامه PLC به زبان STL به صورت زیر است :

برای توضیح بیشتر به اسلاید بعدی توجه کنید



$\geq I$  آیا محتویات ACCU2 بزرگتر یا مساوی از محتویات ACCU1 است در صورت صحت داشتن شرط $RLO=1$ می گردد .

$= M4.3$  مقدار RLO را در M4.3 قرار بده

فصل سوم

مجموعه دستورات

Conversion Instruction

(تبدیل نوع داده ها)

در برنامه نویسی PLC گاهی نیاز به تبدیل نوع داده ها داریم .
در تمامی ادیتورهای برنامه نویسی PLC دستورالعمل ها
و فانکشن هایی را به این منظور در نظر می گیرند .
در نرم افزار S7 نیز دستورالعمل ها و فانکشن هایی
برای این منظور در نظر گرفته شده اند .

عناوين نوع تبديل

- ۱- دستور العمل هاي كه اعداد BCD ويا صحيح را به ساير فرمت هاي عددي تبديل مي كند
- ۲- دستور العمل هايي كه براي متمم ۱ و يا متمم ۲ و يا براي تغيير علامت اعداد مميز شناور
- ۳- دستور العمل هايي كه ترتيب بيت ها را در اكولاموتور تغيير مي دهند
- ۴- دستور العمل هايي كه تغيير روي اعداد اعشاري ۳۲ بيتي را انجام مي دهند .

۱ - دستور العمل هاي كه اعداد BCD ويا صحيح

را به ساير فرمت هاي عددي تبديل مي كند

BTI BCD to Integer (16 bit)

اجرای این دستورالعمل باعث تبدیل ۳ رقم عدد با کد BCD در قسمت ACCU1-L (بیت صفر تا بیت ۱۱) به عدد صحیح ۱۶ بیتی می شود (کد باینری) - در ضمن در این تبدیل در قسمت ACCU1-H و ACCU2 تغییری حاصل نخواهد شد .
عدد تبدیل یافته مجدداً در قسمت ACCU1-L ذخیره می شود
رنج اعداد از ۹۹۹+ تا ۹۹۹- است

به مثال های اسلاید های بعد توجه کنید .

مثال ۱

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به

عدد صحیح ۱۶ بیتی (Integer) کند و در حافظه MW1 ذخیره کند

قبل از اجرای

BTI

ACCU1								
ACCU1-H					ACCU1-L			
0011	0001	1000	1101	0***	0101	1000	1001	

بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفی
این سه بیت مورد استفاده قرار نمی گیرند .

بعد از اجرای

BTI

ACCU1								
ACCU1-H					ACCU1-L			
0011	0001	1000	1101	0000	0010	0100	1101	

$$(589)_{10} = (1001001101)_2$$

مثال ۲

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به

عدد صحیح ۱۶ بیتی (Integer) کند و در حافظه MW1 ذخیره کند

L **MW0**
BTI
T **MW1**

قبل از اجرای

BTI

ACCU1								
ACCU1-H				ACCU1-L				
0011	0001	1000	1101	1	*	*	*	

بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفی
این سه بیت مورد استفاده قرار نمی گیرند.

بعد از اجرای

BTI

ACCU1							
ACCU1-H				ACCU1-L			
0011	0001	1000	1101	1111	1101	1011	0011

$$(-589)_{10} = (1111110110110011)_2$$

تهیه کننده : فتح اله نظریان - فروردین ماه
۱۳۸۵

السبت، ۰۳ ربيع الأول، ۱۴۲۷

اگر از بیت ۰ تا ۱۱ عدد غیر مجاز کد BCD قرار بگیرد بعد از اجرای

BTI اعلام خطاي نرم افزاري يا خطاي سنكرون از طرف CPU خواهد شد

قبل از اجرای

BTI

ACCU1								
ACCU1-H				ACCU1-L				
0011	0001	1000	1101	0	*	*	*	1101 1000 1001

بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفي
این سه بیت مورد استفاده قرار نمی گیرند .

بعد از اجرای

BTI

اعلام خطاي نرم افزاري يا خطاي سنكرون از طرف CPU
در حالت كلي CPU به مد STOP مي رود
اما مي توان.....OB101

ITB Integer (16 bit) to BCD

اجرای این دستورالعمل باعث تبدیل عدد صحیح به کد BCD شود

در قسمت ACCU1-L (بیت صفر تا بیت ۱۱) (کد باینری) تبدیل به

۳ رقم عدد با کد BCD می شود

– در ضمن در این تبدیل در قسمت ACCU1-H و ACCU2

تغییری حاصل نخواهد شد.

عدد تبدیل یافته مجدداً در قسمت ACCU1-L ذخیره می شود

رنج اعداد از ۹۹۹+ تا ۹۹۹- است

به مثال های اسلاید های بعد توجه کنید.

مثال ۱

L **MW0**
ITB
T **MW1**

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به

کد BCD نموده و در حافظه MW1 ذخیره کند

قبل از اجرای

ITB

ACCU1							
ACCU1-H				ACCU1-L			
0011	0001	1000	1101	0000	0010	0100	1101

$$(589)_{10} = (1001001101)_2$$

بعد از اجرای

ITB

ACCU1							
ACCU1-H				ACCU1-L			
0011	0001	1000	1101	0000	0101	1000	1001

بیت های علامت - 0000 برای اعداد مثبت - 1111 برای اعداد منفی

۶۱

تهیه کننده : فتح اله نظریان - فروردین ماه
 ۱۳۸۵

السبت، ۰۳ ربيع الأول، ۱۴۲۷
 ۵ ۸ ۹

مثال ۲

L **MW0**
ITB
T **MW1**

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به

کد BCD نموده و در حافظه MW1 ذخیره کند

قبل از اجرای

ITB

ACCU1							
ACCU1-H				ACCU1-L			
0011	0001	1000	1101	1111	1101	1011	0011

$(-589)_{10}$ ، (1111110110110011)

بعد از اجرای

ITB

ACCU1							
ACCU1-H				ACCU1-L			
0011	0001	1000	1101	1111	0101	1000	1001

بیت های علامت - 0000 برای اعداد مثبت - 1111 برای اعداد منفی

۶۲

تهیه کننده : فتح اله نظریان - فروردین ماه

۱۳۸۵

5

8

9

السبت، ۰۳ ربيع الأول، ۱۴۲۷

اگر از بیت ۰ تا ۱۱ عدد غیر مجاز (بیشتر از ۹۹۹ مثبت و یا کمتر از منهای

۹۹۹ قرار بگیرد بعد از اجرای برنامه - CPU اعلام OVERFLOW

نموده و بیت های OV و OS را یک می کند

BTD BCD to Integer (32 bit)

اجرای این دستورالعمل باعث تبدیل ۷ رقم عدد با کد BCD در

ACCU1 (بیت صفر تا بیت ۲۷) به عدد صحیح ۳۲ بیتی می شود

(کد باینری) – در ضمن در این تبدیل در

ACCU2 تغییری حاصل نخواهد شد .

عدد تبدیل یافته مجدداً در ACCU1 ذخیره می شود

رنج اعداد از ۹۹۹۹۹۹۹+ تا ۹۹۹۹۹۹۹- است

به مثال های اسلاید های بعد توجه کنید .

مثال ۱

L MD0
BTD
T MD1

برنامه ای بنویسید که محتوای حافظه MD0 را تبدیل به 0011

عدد صحیح ۱۶ بیتی (Integer) کند و در حافظه MD1 ذخیره کند

قبل از اجرای

BTD

ACCU1								
ACCU1-H					ACCU1-L			
0	***	1	8	9	8	5	8	9
		0001	1000	1001	1000	0101	1000	1001

این سه بیت مورد استفاده قرار نمی گیرند .
بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفی

بعد از اجرای

BTD

ACCU1								
ACCU1-H					ACCU1-L			
0011	0001	0001	1100	1111	1000	0101	1101	

مثال ۲

L MD0
BTD
T MD1

برنامه ای بنویسید که محتوای حافظه MD0 را تبدیل به 0011

عدد صحیح ۱۶ بیتی (Integer) کند و در حافظه MD1 ذخیره کند

قبل از اجرای

BTD

ACCU1								
ACCU1-H					ACCU1-L			
1	***	1	8	9	8	5	8	9
		0001	1000	1001	1000	0101	1000	1001

این سه بیت مورد استفاده قرار نمی گیرند .
بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفی

بعد از اجرای

BTD

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1110	0011	0000	0111	1010	0011

اگر از بیت ۰ تا ۲۷ عدد غیر مجاز کد BCD قرار بگیرد بعد از اجرای

BTD اعلام خطای نرم افزاری یا خطای سنکرون از طرف CPU خواهد شد

قبل از اجرای

BTD

ACCU1								
ACCU1-H					ACCU1-L			
0	000	1	8	5	9	D	8	9
		0001	1000	0101	1001	1101	1000	1001

این سه بیت مورد استفاده قرار نمی گیرند .
 بیت علامت- 0 برای اعداد مثبت 1 برای اعداد منفی

بعد از اجرای

BTD

اعلام خطای نرم افزاری یا خطای سنکرون از طرف CPU
 در حالت کلی CPU به مد STOP می رود
 اما می توان.....OB101

DTB

Double Integer (32 bit) to BCD

اجرای این دستورالعمل باعث تبدیل عدد صحیح ۳۲ بیتی به کد BCD شود
در قسمت ACCU1 (بیت صفر تا بیت ۲۷) (کد باینری) تبدیل به
۷ رقم عدد با کد BCD می شود

بیت ۲۸ تا ۳۱ برای علامت عدد به کار می رود

۰۰۰۰ برای عدد مثبت و ۱۱۱۱ برای عدد منفی

عدد تبدیل یافته مجدداً در قسمت ACCU1 ذخیره می شود

در ضمن در این تبدیل در ACCU2 تغییری حاصل نخواهد شد .

رنج اعداد از ۹۹۹۹۹۹۹+ تا ۹۹۹۹۹۹۹- است

به مثال های اسلاید های بعد توجه کنید .

مثال ۱

L MD0
DTB
T MD1

برنامه ای بنویسید که محتوای حافظه MD0 را تبدیل به

کد BCD نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

DTB

ACCU1							
ACCU1-H				ACCU1-L			
0000	0000	0001	1100	1101	1001	0001	1101

$$(1890589)_{10} = (111001101100011101)_2$$

بعد از اجرای

DTB

ACCU1							
ACCU1-H				ACCU1-L			
0000	1	8	9	0	5	8	9
0000	0001	1000	1001	0000	0101	1000	1001

بیت های علامت- 0000 برای اعداد مثبت - 1111 برای اعداد منفی

مثال ۲

L *MD0*
DTB
T *MD1*

برنامه ای بنویسید که محتوای حافظه MD0 را تبدیل به

کد BCD نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

DTB

ACCU1							
ACCU1-H				ACCU1-L			
1111	0000	0000	0101	1111	1101	1011	0011

بعد از اجرای

DTB

ACCU1							
ACCU1-H				ACCU1-L			
-		۳	۹	۲	۶	۲	۷
1111	0000	0011	1001	0010	0110	0010	0111

بیت های علامت- 0000 برای اعداد مثبت - 1111 برای اعداد منفی

تهیه کننده : فتح اله نظریان - فروردین ماه

۱۳۸۵

السبت، ۰۳ ربيع الأول، ۱۴۲۷

اگر از بیت ۰ تا ۲۷ عدد غیر مجاز (بیشتر از ۹۹۹۹۹۹۹ مثبت و یا کمتر از منهای

۹۹۹۹۹۹۹ قرار بگیرد بعد از اجرای برنامه - CPU اعلام OVERFLOW

نموده و بیت های OV و OS را یک می کند

ITD

Integer (16 bit) to
Double Integer (32 bit)

اجرای این دستورالعمل باعث تبدیل عدد صحیح (Integer) ۱۶ بیتی

قسمت ACCU1-L

به عدد صحیح (Integer) ۳۲ بیتی می شود

عدد تبدیل یافته مجدداً در ACCU1 ذخیره می شود

– در ضمن در این تبدیل در ACCU2 تغییری حاصل نخواهد شد .

به مثال های اسلاید های بعد توجه کنید .

مثال ۱

L **MW0**
ITD
T **MD1**

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به Double Integer نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

ITD

ACCU1							
ACCU1-H				ACCU1-L			
0000	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

ITD

ACCU1							
ACCU1-H				ACCU1-L			
0000	0000	0000	0000	0101	1001	0001	1101

مثال ۲

L **MW0**
ITD
T **MD1**

برنامه ای بنویسید که محتوای حافظه MW0 را تبدیل به Double Integer نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

ITD

ACCU1							
ACCU1-H				ACCU1-L			
0000	0110	0001	1100	1101	1001	0001	1101

توجه

بعد از اجرای

ITD

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	10101	1001	0001	1101

DTR Double Integer (32 bit) to Floating - point (32 bit – IEEE-FP)

اجرای این دستورالعمل باعث تبدیل عدد صحیح (Integer) ۳۲ بیتی

واقع در ACCU1

به عدد اعشاری (Floating - point) ۳۲ بیتی می شود

عدد تبدیل یافته مجدداً در ACCU1 ذخیره می شود

– در ضمن در این تبدیل در ACCU2 تغییری حاصل نخواهد شد .

به مثال های اسلاید های بعد توجه کنید .

مثال ۱

L **MW0**
DTR
T **MD1**

برنامه ای بنویسید که محتوای حافظه MD0 را تبدیل به

Floating - point نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

DTR

ACCU1									
ACCU1-H					ACCU1-L				
0000	0000	0000	0000	0000	0000	0001	1111	0100	

+500 Integer

بعد از اجرای

DTR

ACCU1									
ACCU1-H					ACCU1-L				
0	000	0011	1	111	1010	0000	0000	0000	0000

بیت علامت

نما

مانتیس

+500 IEEE - FP

۲- دستورالعمل هایی که برای متمام ۱

و یا متمام ۲ و یا برای تغییر

علامت اعداد ممیز شناور به کار می روند

INVI

Ones Complement Integer (16 bit)

اجرای این دستورالعمل باعث می شود **متمم** ۱ عدد صحیح (Integer) ۱۶ بیتی

قسمت ACCU1-L

محاسبه شده و

عدد تبدیل یافته مجددا در ACCU1-L ذخیره می شود

به مثال اسلاید بعدی توجه کنید

مثال ۱

L MW0
INVI
T MW1

برنامه ای بنویسید که متمم ۱ محتوای حافظه MW0 را

محاسبه نموده و در حافظه MW1 ذخیره کند

قبل از اجرای

INVI

ACCU1							
ACCU1-H				ACCU1-L			
0000	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

INVI

ACCU1							
ACCU1-H				ACCU1-L			
0000	0110	0001	1100	1010	0110	1110	0010

INVD

Ones Complement Double Integer (32 bit)

اجرای این دستورالعمل باعث می شود **متمم ۱** عدد صحیح (Integer)

۳۲ بیتی موجود در

ACCU1

محاسبه شده و

عدد تبدیل یافته مجدداً در ACCU1 ذخیره می شود

به مثال اسلاید بعدی توجه کنید

مثال

L *MD0*
INVD
T *MD1*

برنامه ای بنویسید که متمم ۱ محتوای حافظه MD0 را

محاسبه نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

INVD

ACCU1							
ACCU1-H				ACCU1-L			
0110	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

INVD

ACCU1							
ACCU1-H				ACCU1-L			
1001	1001	1110	0011	1010	0110	1110	0010

NEGI

Twos Complement Integer (16 bit)

اجرای این دستورالعمل باعث می شود **متمم** ۲ عدد صحیح (Integer)

۱۶ بیتی

قسمت ACCU1-L

محاسبه شده و

عدد تبدیل یافته مجددا در ACCU1-L ذخیره می شود

به مثال اسلاید بعدی توجه کنید

مثال

L *MW0*
NEGI
T *MW1*

برنامه ای بنویسید که متمم ۲ محتوای حافظه MW0 را

محاسبه نموده و در حافظه MW1 ذخیره کند

قبل از اجرای

NEGI

ACCU1							
ACCU1-H				ACCU1-L			
0110	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

NEGI

ACCU1							
ACCU1-H				ACCU1-L			
0110	0110	0001	1100	1010	0110	1110	0011

NEGD

Twos Complement Double Integer (32 bit)

اجرای این دستورالعمل باعث می شود **متمم** ۲ عدد صحیح (Integer)

۳۲ بیتی

قسمت ACCU1

محاسبه شده و

عدد تبدیل یافته مجددا در ACCU1 ذخیره می شود

به مثال اسلاید بعدی توجه کنید

مثال

L MD0
NEGD
T MD1

برنامه ای بنویسید که متمم ۲ محتوای حافظه MD0 را

محاسبه نموده و در حافظه MD1 ذخیره کند

قبل از اجرای

NEGD

ACCU1							
ACCU1-H				ACCU1-L			
0110	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

NEGD

ACCU1							
ACCU1-H				ACCU1-L			
1001	1001	1110	0011	1010	0110	1110	0011

NEGR

Twos Complement
Double Integer (32 bit)

اجرای این دستورالعمل باعث می شود عدد اعشاری (Floatin point)

۳۲ بیتی

قسمت ACCU1

تغییر علامت داده

و عدد تغییر علامت داده مجددا در ACCU1 ذخیره شود

به مثال اسلاید بعدی توجه کنید

مثال

L MD0
NEGR
T MD1

برنامه ای بنویسید که محتوای حافظه MD0 را که یک عدد اعشاری

است تغییر علامت داده و در حافظه MD1 ذخیره کند

قبل از اجرای

NEGR

ACCU1							
ACCU1-H				ACCU1-L			
0110	0110	0001	1100	0101	1001	0001	1101

بعد از اجرای

NEGR

ACCU1							
ACCU1-H				ACCU1-L			
1001	1001	1110	0011	1010	0110	1110	0011

۳- دستور العمل هایی که ترتیب بیت ها را

در اکولاموتور تغییر می دهند

CAW Change Byte Sequence in ACCU1-L (16 bit)

اجرای این دستورالعمل باعث می شود محتویات بایت ACCU1-L-L

با محتویات بایت ACCU1-L-H **جابجا شوند**

– در ضمن در این تبدیل در قسمت ACCU1-H و ACCU2

تغییری حاصل نخواهد شد .

اطلاعات تغییر یافته مجددا در قسمت ACCU1-L ذخیره می شود

به مثال اسلاید بعدی توجه کنید

مثال

L **MW0**
CAW
T **MW1**

برنامه ای بنویسید که جای بایت های اول و دوم محتوای حافظه

MW0 را جابجا کرده و در حافظه MW1 ذخیره کند

قبل از اجرای

CAW

ACCU1							
ACCU1-H				ACCU1-L			
0 100	0001	0110	1000	1010	1100	1110	0101

بعد از اجرای

CAW

ACCU1							
ACCU1-H				ACCU1-L			
0 100	0001	0110	1000	1110	0101	1010	1100

CAD

Change Byte Sequence in ACCU1 (32 bit)

اجرای این دستورالعمل باعث می شود محتویات بایتهای زیر در ACCU1 با یکدیگر

جابجا شوند

ACCU1-L-L	→	ACCU1-H-H
ACCU1-L-H	→	ACCU1-H-L
ACCU1-H-L	→	ACCU1-L-H
ACCU1-H-H	→	ACCU1-L-L

اطلاعات تغییر یافته مجددا در قسمت ACCU1 ذخیره می شود

– در ضمن در این تبدیل در ACCU2 تغییری حاصل نخواهد شد .

به مثال اسلاید بعدی توجه کنید

تهیه کننده : فتح اله نظریان - فروردین ماه

۱۳۸۵

السبت، ۰۳ ربيع الأول، ۱۴۲۷

مثال

L MD0
CAD
T MD1

برنامه ای بنویسید که جای بایت های اول با چهارم و دوم با سوم محتوای حافظه MD0 را جابجا کرده و در حافظه MD1 ذخیره کند

قبل از اجرای

CAD

ACCU1							
ACCU1-H				ACCU1-L			
0 100	0001	0110	1000	1010	1100	1110	0101

بعد از اجرای

CAD

ACCU1							
ACCU1-H				ACCU1-L			
1110	0101	1010	1100	0110	1000	1010	1100

۴- دستورالعمل هایی که تغییر روی

اعداد اعشاری ۳۲ بیتی را انجام می دهند .

RND

Round

بعد از اجرای این دستورالعمل عدد اعشاری (Floatin point) ۳۲ بیتی

واقع در ACCU1 تبدیل به عدد صحیح ۳۲ بیتی شده و قسمت اعشار آن

به نزدیکترین عدد صحیح گرد میشود

اگر عدد دقیقاً بین دو عدد زوج و فرد با فاصله مساوی قرار گرفت دستور

فوق عدد زوج را انتخاب می کند

عدد تغییر یافته مجدداً در ACCU1 ذخیره شود

به مثال های اسلایدهای بعدی توجه کنید

قبل از اجرای RND

محتوای اکومولاتور ACCU1

13.2



بعد از اجرای RND

محتوای اکومولاتور ACCU1

13

45.8



46

18.5



18

-18.5



-18

قبل از اجرای RND

138.6

ACCU1								
ACCU1-H					ACCU1-L			
0	100	0011	0000	1010	1001	1001	1001	1010

بعد از اجرای RND

139

ACCU1								
ACCU1-H					ACCU1-L			
0000	0000	0000	0000	0000	0000	0000	1000	1011

قبل از اجرای RND

-138.6

ACCU1								
ACCU1-H					ACCU1-L			
1	100	0011	0000	1010	1001	1001	1001	1010

بعد از اجرای RND

-139

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	1111	1111	0111	0101

قبل از اجرای RND

-46.5

ACCU1								
ACCU1-H					ACCU1-L			
1	100	0010	0011	1010	0000	0000	0000	0000

بعد از اجرای RND

-46

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	1111	1111	1101	0010

TRUNC Truncate (کوتاه کردن)

بعد از اجرای این دستورالعمل عدد اعشاری (Floatin point) ۳۲ بیتی

واقع در ACCU1 تبدیل به عدد صحیح ۳۲ بیتی شده و قسمت اعشار آن

صفر می شود

اگر عدد خارج از رنج مجاز باشد بیت‌های OV و OS یک می شوند

عدد تغییر یافته مجدداً در ACCU1 ذخیره شود

به مثال های اسلایدهای بعدی توجه کنید

قبل از اجرای TRUNC
محتوای اکومولاتور ACCU1

بعد از اجرای TRUNC
محتوای اکومولاتور ACCU1

13.2		13
45.8		45
18.5		18
-18.5		-18

قبل از اجرای TRUNC

138.6

ACCU1								
ACCU1-H					ACCU1-L			
0	100	0011	0000	1010	1001	1001	1001	1010

بعد از اجرای TRUNC

138

ACCU1								
ACCU1-H					ACCU1-L			
0000	0000	0000	0000	0000	0000	0000	1000	1010

قبل از اجرای TRUNC

-46.9

ACCU1								
ACCU1-H					ACCU1-L			
1	100	0010	0011	1010	0000	0000	0000	0000

بعد از اجرای TRUNC

-46

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	1111	1111	1101	0010

RND+ Round To upper Double integer

بعد از اجرای این دستورالعمل عدد اعشاری (Floatin point) 32 بیتی

واقع در ACCU1 تبدیل به عدد صحیح 32 بیتی شده و قسمت اعشار آن

به عدد صحیح بالاتر گرد میشود

اگر عدد خارج از رنج مجاز باشد بیت‌های OV و OS یک می شوند

عدد تغییر یافته مجددا در ACCU1 ذخیره شود

به مثال های اسلایدهای بعدی توجه کنید

قبل از اجرای RND+

محتوای اکومولاتور ACCU1

13.2  **14**

45.8  **46**

18.5  **19**

-18.5  **-18**

بعد از اجرای RND+

محتوای اکومولاتور ACCU1

قبل از اجرای RND+

138.02

ACCU1								
ACCU1-H					ACCU1-L			
0	100	0011	0000	1010	0000	0101	0001	1111

بعد از اجرای RND+

139

ACCU1								
ACCU1-H					ACCU1-L			
0000	0000	0000	0000	0000	0000	0000	1000	1011

قبل از اجرای RND+

-1.9

ACCU1								
ACCU1-H					ACCU1-L			
1	011	1111	1111	0011	0011	0011	0011	0011

بعد از اجرای RND+

-1

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	1111	1111	1111	1111

RND- Round To Lower Double integer

بعد از اجرای این دستورالعمل عدد اعشاری (Floatin point) 32 بیتی

واقع در ACCU1 تبدیل به عدد صحیح 32 بیتی شده و قسمت اعشار آن

به عدد صحیح پایین تر گرد میشود

اگر عدد خارج از رنج مجاز باشد بیت های OV و OS یک می شوند

عدد تغییر یافته مجددا در ACCU1 ذخیره شود

به مثال های اسلایدهای بعدی توجه کنید

قبل از اجرای RND-

محتوای اکومولاتور ACCU1

13.2



13

45.8



45

18.5



18

-18.5



-19

بعد از اجرای RND-

محتوای اکومولاتور ACCU1

قبل از اجرای RND-

138.02

ACCU1								
ACCU1-H					ACCU1-L			
0	100	0011	0000	1010	0000	0101	0001	1111

بعد از اجرای RND-

138

ACCU1							
ACCU1-H				ACCU1-L			
0000	0000	0000	0000	0000	0000	1000	1010

قبل از اجرای RND-

-1.1

ACCU1								
ACCU1-H					ACCU1-L			
1	011	1111	1000	1100	1100	1100	1100	1101

بعد از اجرای RND-

-2

ACCU1							
ACCU1-H				ACCU1-L			
1111	1111	1111	1111	1111	1111	1111	1110

فصل چهارم

Counter Instruction



در S7 شمارنده یک فانکشن به حساب می آید که می تواند تعداد قطع و وصل ها را بشمارد و در یک مکان از حافظه که از قبل برای این منظور در نظر گرفته شده است مقدار شمارش را نگهداری می کند

بنا بر این مقدار شمارش در ACCU1 ذخیره نمی شود

تعداد شمارنده های مجاز در یک برنامه بستگی به نوع CPU دارد

معمولا ۱۲۸ یا ۲۵۶ و یا ۵۱۲

هر شمارنده برای نگهداری مقدار شمارش از دو بایت حافظه

استفاده می کند

مقدار شمارش به صورت BCD و یا باینری است

تمامی شمارنده ها از صفر تا ۹۹۹ را می توانند بشمارند

اگر شمارنده ای تا ۹۹۹ را شمرد دیگر تعداد قطع و وصل های

ورودی را لحاظ نمی کند

و اگر یک شمارنده ای به طور معکوس بشمارد وقتی مقدار شمارش

به صفر رسید دیگر به قطع و وصل ورودی اهمیت نمی دهد

هر شمارنده برای نگهداری مقدار شمارش از **دو بایت** حافظه

(۱۶ بیت - یک word) استفاده می کند

فرمت BCD

غیر قابل استفاده	0	0	0
	0 0 0 0	0 0 0 0	0 0 0 0
غیر قابل استفاده	9	9	9
	1 0 0 1	1 0 0 1	1 0 0 1

حداقل شمارش

حداکثر شمارش

هر شمارنده برای نگهداری مقدار شمارش از **دو بایت** حافظه

(۱۶ بیت – یک word) استفاده می کند

فرمت Binary

غیر قابل استفاده	0 0 0 0	0 0 0 0	0 0 0 0	← حداقل شمارش 000
غیر قابل استفاده	0 0 1 1	1 1 1 0	0 1 1 1	← حداکثر شمارش 999

در زبان STL دستورات کانترا به شرح زیر می باشند :

- FR Enable Counter (Free)**
- L Load current Counter Value into ACCU1**
- LC Load current Counter Value into ACCU1 as BCD**
- R Reset counter**
- S Set counter Preset Value**
- CU Counter Up**
- CD Counter Down**

در اسلاید های بعدی عملکرد دستورات فوق همراه با مثال توضیح داده خواهد شد

FR

Enable Counter (Free)

وقتی RLO از صفر به یک تغییر وضعیت می دهد دستور FR لبه را که برای شمارش افزایشی و یا کاهششی شمارنده به کار می رود تشخیص می دهد و شمارنده را فعال می کند . این دستور آزاد است یعنی استفاده از آن الزامی نیست بدون آن نیز با تغییر RLO از صفر به یک شمارنده فعال می گردد . به عبارت دیگر وقتی RLO از صفر به یک تغییر وضعیت می دهد آن را تشخیص داده و نتیجه را با $RLO=1$ آشکار می شود .

مثال

A 10.0

FR C10

در مثال فوق به محض این که در سیکل اسکن جدید ورودی **10.0** از صفر
به یک تغییر وضعیت داد دستور **FR** آن را تشخیص داده و کانتور **C10**
را فعال می کند بدیهی است به شمارنده فرمان شمارش بالا یا پایین
داده نشده است .

L

Load Current Counter Value into ACCU1

اجرای دستورالعمل **L** باعث می شود که ابتدا محتویات ACCU1
وارد ACCU2 شده و سپس مقداری که به صورت مستقیم و یا غیر
مستقیم به صورت آدرس داده می شود وارد ACCU1 شود .

مثال

L C10

همانطور که قبلاً نیز گفته شد مقدار شمارش شمارنده ها در مکانی از حافظه که قبلاً به این منظور در نظر گرفته شده است ذخیره می گردد .

اجرای دستور فوق باعث می شود که مقدار شمارش شمارنده C10 به صورت باینری از حافظه به ACCU1 منتقل گردد .

LC

Load Current Counter

Value into ACCU1 **as BCD**

اجرای دستورالعمل **LC** باعث می شود که ابتدا محتویات ACCU1

وارد ACCU2 شده و سپس مقداری که به صورت مستقیم و یا غیر

مستقیم به صورت آدرس داده می شود به صورت BCD وارد ACCU1

شود .

مثال C14 LC

همانطور که قبلاً نیز گفته شد مقدار شمارش شمارنده ها در مکانی از حافظه که قبلاً به این منظور در نظر گرفته شده است ذخیره می گردد .

اجرای دستور فوق باعث می شود که مقدار شمارش شمارنده C14 به صورت **BCD** از حافظه به ACCU1 منتقل گردد .

مکانی از حافظه که مقدار شمارش
در آن ضبط شده است



0000	0000	1100	1101
------	------	------	------

205

LC

C14

بعد از اجرای دستور مقابل

ACCU1			
ACCU1-H		ACCU1-L	
		0010	0000 0101

2 0 5

R

Reset Counter

اگر مقدار $RLO=1$ باشد مقدار (شمارش) شمارنده را صفر می کند

همانطور که قبلاً نیز گفته شد مقدار شمارش شمارنده ها در مکانی

از حافظه که قبلاً به این منظور در نظر گرفته شده است ذخیره می گردد .

نه در اکومولاتور $ACCU1$ لذا اجرای دستور فوق تاثیری روی

اکومولاتور ندارد

A $I0.0$
 R $C22$

مثال

در مثال فوق اگر $I0.0$ از صفر به یک تغییر وضعیت دهد مقدار

RLO نیز از صفر به یک تغییر وضعیت می دهد لذا دستورالعمل

R اجرا شده و مقدار شمارش شمارنده $C22$ صفر می شود

S

Set Counter Preset Value

S اگر مقدار RLO از صفر به یک تغییر وضعیت دهد با اجرای دستورالعمل

محتوای اکومولاتور ACCU1-L به شمارنده مورد نظر به عنوان مقدار

اولیه شمارش منتقل می شود

توجه : محتوای ACCU1 باید با فرمت BCD بوده و بین صفر

تا ۹۹۹ باشد

A **I0.0**
L **C#20**
S **C22**

مثال

در مثال فوق اگر I0.0 از صفر به یک تغییر وضعیت دهد مقدار

RLO نیز از صفر به یک تغییر وضعیت می دهد لذا دستورالعمل

S اجرا شده و مقدار شمارش شمارنده C22 را برابر ۲۰ می کند

CU

Counter Up

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

CU < Counter >

مثال : CU C44

هر گاه مقدار RLO از صفر به یک تغییر وضعیت دهد با اجرای CU

یک واحد به مقدار شمارش شمارنده C44 اضافه می شود

حداکثر مقدار شمارش ۹۹۹

A 10.0
CU C22

مثال

در مثال فوق اگر 10.0 از صفر به یک تغییر وضعیت دهد مقدار
RLO نیز از صفر به یک تغییر وضعیت می دهد لذا با اجرای

دستورالعمل **CU** مقدار شمارش شمارنده **C22** یک واحد

افزایش می یابد

حداکثر مقدار شمارش ۹۹۹

CD

Counter Down

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

CD < Counter >

مثال : **CD C45**

هر گاه مقدار **RLO** از صفر به یک تغییر وضعیت دهد با اجرای **CD**

یک واحد از مقدار شمارش شمارنده **C44** کسر می شود

حداقل مقدار شمارش 0

A **10.0**
CD **C22**

مثال ۱

در مثال فوق اگر 10.0 از صفر به یک تغییر وضعیت دهد مقدار
RLO نیز از صفر به یک تغییر وضعیت می دهد لذا با اجرای

دستورالعمل **CD** مقدار شمارش شمارنده **C22** یک واحد

کم می شود

حداقل مقدار شمارش صفر

شمارنده مقدار منفی را نمی شمارد !!!!!

L C#20
A I0.1
S C1
A I0.0
CD C1
AN C1
= Q0.0

مثال ۲

فصل پنجم

Data Block Instruction

مجموعه دستورالعمل های دیتا بلوک ها

مقدمه

اگر یک دیتا بلوک فقط و فقط به یک فانکشن بلوک اختصاص داده شود بطوری که سایر فانکشن ها - فانکشن بلوک ها-OB ها و..به آن دسترسی نداشته باشند به این نوع دیتا بلوک **Instance Data Block** گفته می شود .

اگر تمامی فانکشن ها - فانکشن بلوک ها-OB ها و.. به یک دیتا بلوک دسترسی داشته باشند به این نوع دیتا بلوک **Shared Data Block** گفته می شود .

دستورات مربوط به دیتا بلوک ها به شرح زیر هستند :

OPN *Open a Data Block*

CDB *Exchange Shared DB and Instance DB*

L DBLG *Load Length of Shared DB in ACCU1*

L DBNO *Load Number of Shared DB in ACCU1*

L DILG *Load Length of Instance DB in ACCU1*

L DINO *Load Number of Shared DB in ACCU1*

OPN

Open a Data Block

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

OPN < Data Block >

مثال : **OPN DB25**

با اجرای دستورالعمل **OPN** – یک دیتا بلوک از نوع **Shared Data Block**

و یا یک دیتا بلوک از نوع **Instance Data Block** باز می شود .

یک دیتا بلاک از نوع **Shared Data Block** و دیتا بلوک از نوع

Instance Data Block می توانند به صورت همزمان باز شوند .

OPN

L

T

OPN

L

T

DB10

DBW23

MW22

DI20

DIB12

DBB12

مثال

CDB *Exchange Shared DB and Instance DB*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

CDB

CDB

مثال :

با اجرای دستورالعمل CDB رجیستر دیتا بلوک از نوع Shared

با رجیستر دیتا بلوک از نوع Instance با یک دیگر جابجا می شوند

یعنی نوع Shared تبدیل به Instance می شود و بالعکس .

OPN
OPN
CDB

DB10
DI20

مثال

بعد از اجرای برنامه فوق دیتا بلوک **DB10** تبدیل به نوع **Instance**

می شود و دیتا بلوک **DI20** تبدیل به نوع **Shared** می گردد .

DI10

یعنی

DB20

L DBLG *Load Length of Shared DB in ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L DBLG

L DBLG

مثال :

اجرای دستورالعمل **L DBLG** باعث می شود که ابتدا محتویات

ACCU1 وارد ACCU2 شده و سپس طول دیتا بلوک Shared وارد

ACCU1 شود .

OPN

L

L

<

JC

DB10

DBLG

MD10

D

ERRO

مثال

L DBNO *Load Number of Shared DB in ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L DBNO

L DBNO

مثال :

اجرای دستورالعمل **L DBNO** باعث می شود که ابتدا محتویات

ACCU1 وارد ACCU2 شده و سپس شماره دیتا بلوک Shared وارد

ACCU1 شود .

OPN

L

T

DB10

DBNO

MW10

مثال

L DILG *Load Length of Instance DB in ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L DILG

L DILG

مثال :

اجرای دستورالعمل **L DILG** باعث می شود که ابتدا محتویات

ACCU1 وارد ACCU2 شده و سپس طول دیتا بلوک Instance وارد

ACCU1 شود .

OPN

L

L

<

JC

DI10

DILG

MW10

I

ERRO

مثال

L DINO *Load Number of Instance DB in ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L DiNO

L DiNO

مثال :

اجرای دستورالعمل **L DINO** باعث می شود که ابتدا محتویات

ACCU1 وارد **ACCU2** شده و سپس شماره دیتا بلوک **Instance** وارد

ACCU1 شود .

OPN

L

T

DI10

DINO

MW10

مثال

فصل ششم

Logic Control Instruction

مجموعه دستورالعمل های

کنترل روند اجرای برنامه

مقدمه

دستورالعمل های پرش در STL روند اجرای برنامه را می توانند

کنترل کنند . در حقیقت می توان یک شرط را آزمایش کرد و در

صورت True بودن یک مسیر را انتخاب کرده و در صورت Fale

بودن مسیر دیگری را فراروی اجرای برنامه PLC گذاشت .

دستورات پرش را می توان به صورت زیر دسته بندی کرد

۱ - دستورات پرش بدون قید و شرط

۲ - دستورات پرش مشروط به وضعیت RLO

۳ - دستورات پرش مشروط به وضعیت یک بیت از **Status Word**

۴ - دستورات پرش مشروط به نتیجه محاسبات

۵ - دستورات حلقه های تکرار Loop

۱ - دستورات پرش بدون قید و شرط

دو دستورالعمل در Step7 برای پرش بدون قید و شرط به شرح

زیر وجود دارند :

1 – JU *Jump Unconditional*

2 – JL *Jump to Labels*

JU

Jump Unconditional

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JU < Jump Lebel >

مثال : JU NAZA

دستور **JU** بدون توجه به بیت های Status Word به

محل آدرس داده شده توسط Label پرش می نماید .

توجه

قبل از Label گذاری به نکات زیر دقیقا توجه کنید :

- ۱ – نام Label نباید از ۴ کاراکتر بیشتر باشد .
- ۲ – نام Label حتما باید با حرف شروع شود .
- ۳ – نام Label باید همراه با علامت کولن (:) که به دنبال آن می آید باشد مانند Nazarian:
- ۴ – نام Label نباید تکراری باشد – یعنی باید منحصر بفرد باشد .
- ۵ – دو دستور Jump و Label حتما باید در یک بلوک باشند .
- ۶ – ماکزیمم فاصله بین دستور Jump و Label می تواند 32K_Word باشد .

مثال

A **I0.0**

A **I0.1**

JU **NAZA**

A **I0.2**

= **Q0.0**

NAZA: **O** **I0.3**

= **M0.0**

این دو دستور هیچگاه اجرا نمی شوند

JL

Jump to Labels

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JL < Jump Label >

JL NAZA

مثال :

دستور **JL** امکان برنامه ریزی چند پرش (حداکثر ۲۵۵ مورد)

را در برنامه فراهم می کند .

عملکرد آن به این صورت است که متناسب با عدد موجود در ACCU1-L-L

به JU متناسب پرش می کند یعنی اگر $ACCU1-L-L=0$ باشد به اولین **JU**

ادامه ←

پرش می کند و اگر $ACCU1-L-L=1$ باشد به دو مین **JU** پرش می کند

و اگر $ACCU1-L-L=2$ باشد به سومین **JU** و به همین صورت الی آخر

اگر تعداد **JU** ها بیش از آخرین مقدار باشد دستور **JL** به اولین دستور

بعد از انتهای لیست **JL** پرش می کند آدرس این سطر در جلوی **JL**

نوشته می شود

به اسلاید بعدی توجه کنید

مثال

L MB0
JL **LSTX**
JU SEG0
JU SEG1
JU COM
JU SEG3

LSTX: JU COM
SEG0: *

JU COM
SEG1: *

JU COM
SEG3: *

COM: *

۲ - دستورات پرش مشروط به

وضعیت RLO

JC

Jump if RLO=1

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JC < Jump Lebel >

مثال : JC NAZA

دستور **JC** در صورتیکه RLO=1 باشد به

محل آدرس داده شده توسط Label پرش می نماید .

به همین دلیل به آن دستور پرش مشروط می گویند

مثال

A I0.0

A I0.0

JC SEG0

L IW8

T MW22

SEG0: A I2.1

JCN

Jump if RLO=0

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JCN < Jump Label >

JCN NAZA

مثال :

دستور **JCN** در صورتیکه $RLO=0$ باشد به

محل آدرس داده شده توسط Label پرش می نماید .

به همین دلیل به آن دستور پرش مشروط می گویند

مثال

A I0.0

A I0.0

JCN SEG0

L IW8

T MW22

SEG0: A I2.1

JCB

Jump if RLO=1 With BR

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JCB < Jump Lebel >

مثال : JCB NAZA

با اجرای دستور **JCB** ابتدا مقدار RLO در BR کپی می شود
صرف نظر از مقدار **RLO**

و صورتیکه RLO=1 باشد به محل آدرس داده شده توسط Label

پرش می نماید .

مثال

A I0.0

A I0.0

JCB SEG0

L IW8

T MW22

SEG0: A I2.1

JNB

Jump if RLO=0 With BR

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JNB < Jump Lebel >

JNB NAZA

مثال :

با اجرای دستور **JNB** ابتدا مقدار RLO در BR کپی می شود

صرف نظر از مقدار RLO

و صورتیکه RLO=0 باشد به محل آدرس داده شده توسط Label

پرش می نماید .

مثال

A I0.0

A I0.0

JNB SEG0

L IW8

T MW22

SEG0: A I2.1

۳- دستورات پرش مشروط به وضعیت

یک بیت از *Status Word*

JB I

Jump if BR=1

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JB I < Jump Lebel >

JB I NAZA

مثال :

با اجرای دستور **JB I** در صورتیکه BR=1 باشد

به محل آدرس داده شده توسط Label پرش می نماید .

مثال

A I0.0

A I0.0

SAVE

JBI SEG0

L IW8

T MW22

SEG0: A I2.1

JNBI

Jump if BR=0

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JNBI < Jump Label >

JNBI NAZA

مثال :

با اجرای دستور **JNBI** در صورتیکه BR=0 باشد

به محل آدرس داده شده توسط Label پرش می نماید .

مثال

A I0.0

A I0.0

SAVE

JNBI SEG0

L IW8

T MW22

SEG0: A I2.1

JO

Jump if OV=1

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JO < Jump Label >

JO NAZA مثال :

با اجرای دستور **JO** در صورتیکه $OV=1$ باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

OV هنگامی در وضعیت یک قرار می گیرد که یک Overflow

در هنگام محاسبات ریاضی رخ داده باشد .

توصیه می شود بعد از انجام محاسبات ریاضی برای کنترل این که

نتیجه در رنج مجاز است یا خیر از دستورالعمل JO یا JOS استفاده

شود .

مثال

L MW0

L 3

*I

JO OVER

T MW10

*

JU NEXT

OVER: AN M4.0

S Q4.0

NEXT: NOP

0

JOS

Jump if OS=1

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JOS < Jump Lebel >

مثال : JOS NAZA

با اجرای دستور **JOS** در صورتیکه OS=1 باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

OS هنگامی در وضعیت یک قرار می گیرد که یک Overflow

در هنگام محاسبات ریاضی رخ داده باشد .

توصیه می شود بعد از انجام محاسبات ریاضی برای کنترل این که

نتیجه در رنج مجاز است یا خیر از دستورالعمل JO یا JOS استفاده

شود .

چه فرقی بین OV و OS وجود دارد ؟

اگر در نتیجه عملیات محاسباتی سرریزی (Overflow) رخ دهد بیت OV برابر یک می شود .
وقتی در همان سیکل اسکن به دستور محاسباتی جدیدی برسد این بیت ری ست شده و بر اساس نتایج جدید Update می گردد.

این بیت معرف Overflow Stored است یعنی سرریزی قبلی که در همان سیکل اتفاق افتاده را ذخیره می کند و تا پایان سیکل آن را حفظ می کند .

مثال

L IW10
L MW12
*|
L DBW25
+|
L MW14
-|

JOS OVER

A M4.0
R M4.0
JU NEXT

OVER: AN M4.0
S M4.0

NEXT: NOP 0

۴ - دستورات پرش مشروط به

نتیجه محاسبات

JZ

Jump if Zero

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JZ < Jump Lebel >

مثال : JZ NAZA

دستور **JZ** در صورتیکه نتیجه محاسبات صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات برابر صفر شود بیت های CC0 و CC1

از Status Word صفر می شوند .

BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
	0	0						

مثال

L MW10

SRW

JZ ZERO

L MW2

INC 1

T MW2

JU NEXT

ZERO: L MW4

INC 1

T MW4

NEXT: NOP 0

JN

Jump if Not Zero

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JN < Jump Lebel >

مثال : JN NAZA

دستور **JZ** در صورتیکه نتیجه محاسبات مخالف صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات مخالف صفر شود بیت های CC0 و CC1

از Status Word به شرح زیر تغییر می کنند :

BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
	0	1						

یا

BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
	1	0						

مثال

L IW8

L MW12

XOW

JN NOZE

AN M4.0

S M4.0

JU NEXT

ZERO: AN M4.1

S M4.1

NEXT: NOP 0

JP

Jump if Plus

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JP < Jump Label >

JP NAZA

مثال :

دستور **JP** در صورتیکه نتیجه محاسبات بزرگتر از صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات بزرگتر از صفر شود بیت های

CC0 و CC1

از **Status Word** به شرح زیر تغییر می کنند :

BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
	1	0						

مثال

L IW8
L MW12
-I

JP POS

AN M4.0

S M4.0

JU NEXT

POS: AN M4.1

S M4.1

NEXT: NOP 0

JM

Jump if Minus

فرمت استفاده از دستور فوق در برنامه نویسی **STL** به صورت زیر است :

JM < Jump Lebel >

JM NAZA	مثال :
----------------	--------

دستور **JM** در صورتیکه نتیجه محاسبات کوچکتر از صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات کوچکتر از صفر شود بیت های

CC0 و CC1

از **Status Word** به شرح زیر تغییر می کنند :

BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
	0	1						

مثال

L IW8
L MW12
-I

JM NEG

AN M4.0

S M4.0

JU NEXT

NEG: AN M4.1

S M4.1

NEXT: NOP 0

JPZ

Jump if Plus or Zero

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JPZ < Jump Lebel >

مثال : JPZ NAZA

دستور **JPZ** در صورتیکه نتیجه محاسبات بزرگتر یا مساوی صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات بزرگتر و یا مساوی صفر شود بیت های

CC0 و CC1

از **Status Word** به شرح زیر تغییر می کنند :

نتیجه محاسبه صفر	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
		0	0						

یا

نتیجه محاسبه مثبت	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
		1	0						

مثال

L IW8
L MW12
-I

JPZ REG0

AN M4.0

S M4.0

JU NEXT

REG0: AN M4.1

S M4.1

NEXT: NOP 0

JMZ

Jump if Minus or Zero

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JMZ < Jump Label >

مثال : JMZ NAZA

دستور **JMZ** در صورتیکه نتیجه محاسبات کوچکتر یا مساوی صفر باشد

به محل آدرس داده شده توسط Label پرش می نماید .

توجه داشته باشید که :

هنگامی که نتیجه محاسبات کوچکتر و یا مساوی صفر شود بیت های

CC0 و CC1

از Status Word به شرح زیر تغییر می کنند :

نتیجه محاسبه صفر	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
		0	0						

یا

نتیجه محاسبه منفی	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
		0	1						

مثال

L IW8
L MW12
-I

JMZ RGE0

AN M4.0

S M4.0

JU NEXT

RGE0: AN M4.1

S M4.1

NEXT: NOP 0

JUO

Jump if Unordered

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

JUO < Jump Label >

JUO NAZA مثال :

دستور **JUO** در صورتیکه $CC0=1$ و $CC1=1$ باشند

به محل آدرس داده شده توسط Label پرش می نماید .

CC0=1 و CC0=1 هنگامی اتفاق می افتد که :

۱- عمل تقسیم بر صفر انجام شود

۲- دستور غیر مجازی به کار رود

۳- فرمت غیر مجاز برای اعداد اعشاری استفاده شود

مثال

L MD10

L ID2

/D

JUO ERRO

T MD14

A M4.0

R M4.0

JU NEXT

ERRO: AN M4.0

S M4.0

NEXT: NOP 0

۵ - دستورات حلقه های تکرار

Loop

LOOP

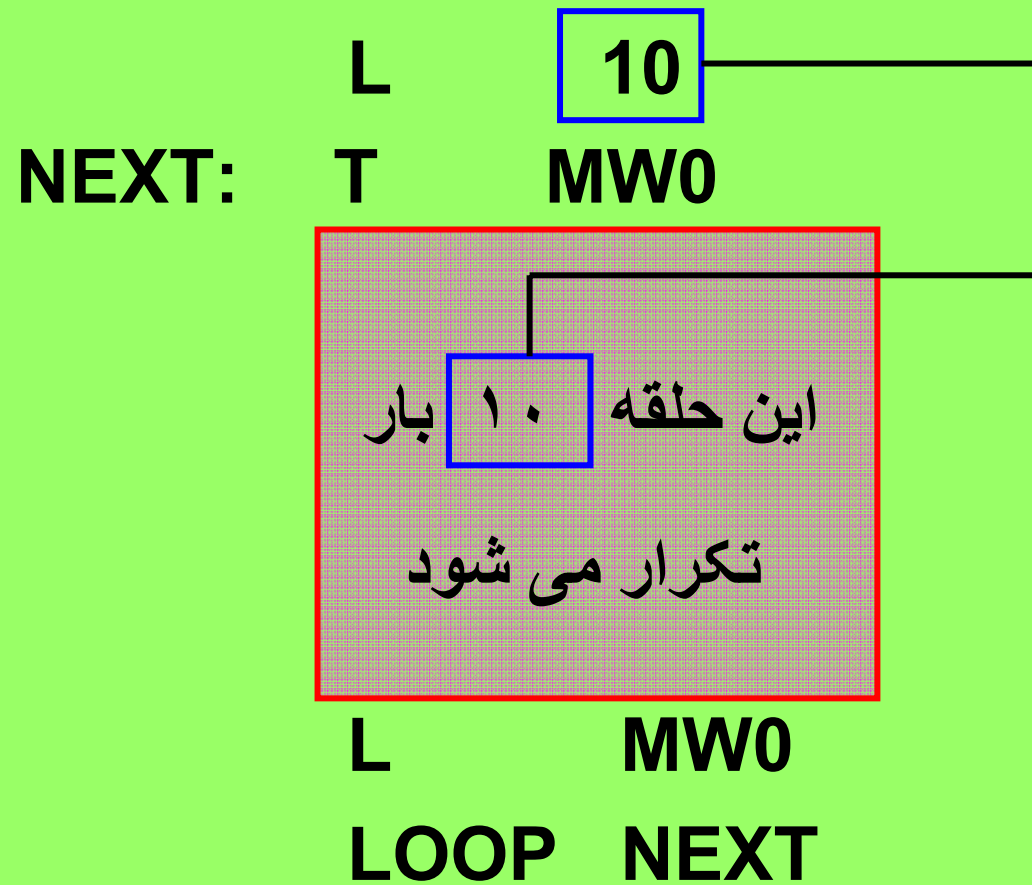
Loop

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LOOP < Jump Lebel >

مثال : LOOP NAZA

اجرای دستور **LOOP** باعث می شود از مقدار **ACCU1-L-L** یکی کم کند در صورتی که مقدار **ACCU1-L-L** مخالف صفر باشد عمل پرش را انجام می دهد . این پرش تا مادامی که **ACCU1-L-L=0** شود ادامه می یابد . عدد مربوط به حلقه باید قبل از شروع حلقه به اکومولاتور بار کرد



مثال ۱

برنامه ای بنویسید که 7! را
محاسبه کند.

NEXT:

```
L      1
T      MW3
L      7
T      MW6
```

```
L      MW3
```

```
*I
```

```
T      MW3
```

```
L      MW6
```

```
LOOP  NEXT
```

مثال ۲

برنامه ای بنویسید که عبارت زیر را محاسبه کند.

NEXT:

L 1

T MW3

L 100

T MW6

L MW3

+I

T MW3

L MW6

LOOP NEXT

$$\sum_{i=1}^{i=100} x_i$$

فصل هفتم

Integer Math Instruction

مجموعه دستورالعمل های

اعمال ریاضی روی اعداد صحیح

مقدمه :

دستورالعمل های ریاضی – اعمال ریاضی را روی اکومولاتورهای

ACCU1 و ACCU2 انجام می دهند .

هر بار که دستور بارگذاری اجرا شود ابتدا محتویات **ACCU1** وارد

ACCU2 شده و مقدار جدید وارد **ACCU1** می شود.

هنگام محاسبه اعمال ریاضی – محتوای **ACCU2** تغییری نمی کند و

نتیجه محاسبات در **ACCU1** ذخیره می گردد .

اعمال ریاضی مجاز روی داده های صحیح ۱۶ بیتی و ۳۲ بیتی

- +I Add ACCU1 and ACCU2 as Integer (16 – Bit)**
- I Subtract ACCU1 from ACCU2 as Integer (16–Bit)**
- *I Multiply ACCU1 and ACCU2 as Integer (16 – Bit)**
- /I Divide ACCU1 by ACCU2 as Integer (16 – Bit)**
- + Add Integer Constant (16 OR 32 Bit)**

- +D Add ACCU1 and ACCU2 as Double Integer (32–Bit)**
- D Subtract ACCU1 from ACCU2 Double Integer (32–Bit)**
- *D Multiply ACCU1 and ACCU2 Double Integer (32–Bit)**
- /D Divide ACCU1 by ACCU2 as Double Integer (32–Bit)**
- MOD Division Remainder Double Integer (32–Bit)**

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام محاسبات ریاضی

الف – اگر نتیجه محاسبات در رنج مجاز باشد

رنج مجاز	CC1	CC0	OV	OS
0 (ZERO)	0	0	0	*
اعداد منفی $0 < \text{نتیجه} \leq -32768$ (۱۶ بیتی) $0 < \text{نتیجه} \leq -2147483648$ (۳۲ بیتی)	0	1	0	*
اعداد مثبت $0 < \text{نتیجه} \leq 32767$ (۱۶ بیتی) $0 < \text{نتیجه} \leq 2147483647$ (۳۲ بیتی)	1	0	0	*

*** بیت OS در رنج مجاز تحت تاثیر قرار نمی گیرد**

ب - اگر نتیجه محاسبات در رنج مجاز نباشد

رنج غیر مجاز	CC1	CC0	OV	OS
Underflow(Addition) نتیجه = -65536 ۱۶ بیتی نتیجه = -4294967296 ۳۲ بیتی	0	0	1	1
Underflow(multiplication) اعداد منفی نتیجه > -32768 ۱۶ بیتی نتیجه > -2147483648 ۳۲ بیتی	0	1	1	1
Overflow(Addition - subtraction) اعداد مثبت نتیجه < 32768 ۱۶ بیتی نتیجه < 2147483647 ۳۲ بیتی	0	1	1	1
Overflow(multiplication - division) اعداد مثبت نتیجه < 32768 ۱۶ بیتی نتیجه < 2147483647 ۳۲ بیتی	1	0	1	1
Underflow(Addition - subtraction) اعداد منفی نتیجه > -32768 ۱۶ بیتی نتیجه > -2147483648 ۳۲ بیتی	1	0	1	1
تقسیم بر صفر	1	1	1	1

ادامه اسلايد قبلى :

Operation	CC1	CC0	OV	OS
+D: result=-4294967296)	0	0	1	1
/D or division by 0	1	1	1	1

+I

Add ACCU1 and ACCU2 as Integer (16 – Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+I

+I

مثال :

دستور **+I** محتویات **ACCU1-L** و **ACCU2-L** را با هم جمع کرده و نتیجه را در **ACCU1-L** ذخیره می کند هر دو اکومولاتور ۱۶ بیتی و با محتوای اعداد صحیح هستند

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Sum=0	0	0	0	—
$-32768 \leq \text{sum} < 0$	0	1	0	—
$32768 \leq \text{sum} < 65536$	1	0	0	—
sum = -65536	0	0	1	1
$65536 \leq \text{sum} < 131072$	0	1	1	1
$-65536 \leq \text{sum} < -32768$	1	0	1	1

مثال

L IW10

L MW14

+I

T DB1.DBW25

-I Subtract ACCU1 from ACCU2 as Integer (16 – Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

-I

-I

مثال :

-I دستور محتویات **ACCU1-L** از **ACCU2-L** کم کرده و نتیجه را در **ACCU1-L** ذخیره می کند هر دو اکومولاتور ۱۶ بیتی و با محتوای اعداد صحیح هستند
انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Difference=0	0	0	0	—
$-32768 \leq \text{Difference} < 0$	0	1	0	—
$32767 \geq \text{Difference} > 0$	1	0	0	—
$65535 \geq \text{Difference} > 32767$	0	1	1	1
$-65535 \leq \text{Difference} < -32768$	1	0	1	1

مثال

L IW10

L MW14

-I

T DB1.DBW25

***I**

Multiply ACCU1 and ACCU2 as Integer (16 – Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

***I**

***I**

مثال :

***I** دستور محتویات **ACCU1-L** و **ACCU2-L** را با هم ضرب کرده و نتیجه را در **ACCU1-L** ذخیره می کند هر دو اکومولاتور ۱۶ بیتی و با محتوای اعداد صحیح هستند
انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Product=0	0	0	0	—
$-32768 \leq \text{Product} < 0$	0	1	0	—
$32767 \geq \text{Product} > 0$	1	0	0	—
$65535 \geq \text{Product} > 32767$	0	1	1	1
$-65535 \leq \text{Product} < -32768$	1	0	1	1

مثال

L IW10

L MW14

***I**

T DB1.DBW25

/I

Divided ACCU2 by ACCU1 as Integer (16 – Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

/I

/I

مثال :

/I دستور محتویات **ACCU2-L** را بر **ACCU1-L** تقسیم کرده و نتیجه قسمت صحیح را در **ACCU1-L** ذخیره می کند و نتیجه خارج قسمت را در **ACCU1-H** ذخیره می کند هر دو اکومولاتور ۱۶ بیتی و با محتوای اعداد صحیح هستند انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Quotient=0	0	0	0	—
$-32768 \leq \text{Quotient} < 0$	0	1	0	—
$32767 \geq \text{Quotient} > 0$	1	0	0	—
Quotient = 32768	1	0	1	1
Divided by zero	1	1	1	1

مثال

L IW10

L MW14

/I

T MD20

+

Add Integer constant (16 – 32 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+ <integer constant>

مثال :	+ 2345
--------	--------

دستور **+** محتویات **ACCU1-L** و یک عدد ثابت را با هم جمع کرده و نتیجه را در **ACCU1-L** ذخیره می کند محتوای اکومولاتور ۲ در اثر انجام این عمل ریاضی تغییری نمی کند .
انجام دستورالعمل فوق تاثیری روی **RLO** و **OS – OV – CC0 – CC1** ندارد

رنج مجاز اعداد

-32768 to +32767

۱۶ بیتی

-2147483648 to +2147483647

۳۲ بیتی

مثال

L IW10

L MW14

+I

+ 25

T DB1.DBW25

+D

Add ACCU1 and ACCU2 as Double Integer (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+D

+D

مثال :

دستور **+D** محتویات ACCU1 و ACCU2 را با هم جمع کرده و نتیجه را در ACCU1 ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد صحیح هستند
انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Sum=0	0	0	0	—
$-2147483648 \leq \text{sum} < 0$	0	1	0	—
$2147483647 \geq \text{sum} > 0$	1	0	0	—
$\text{sum} = -4294967296$	0	0	1	1
$4294967294 \geq \text{sum} > 2147483647$	0	1	1	1
$-4294967296 \leq \text{sum} < -2147483648$	1	0	1	1

مثال

L ID10

L MD14

+D

T DB1.DBD25

-D

Subtract ACCU1 from ACCU2 as Double Integer (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

-D

-D

مثال :

دستور **-D** محتویات ACCU1 را از ACCU2 کم کرده و نتیجه را در ACCU1 ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد صحیح هستند
انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Difference=0	0	0	0	—
$-2147483648 \leq \text{Difference} < 0$	0	1	0	—
$2147483648 \geq \text{Difference} > 0$	1	0	0	—
$4294967295 \geq \text{Difference} > 2147483647$	0	1	1	1
$-4294967295 \leq \text{Difference} < -2147483648$	1	0	1	1

مثال

L ID10

L MD14

-D

T DB1.DBD25

*D

Multiply ACCU1 and ACCU2 as Double Integer (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

*D

*D

مثال :

دستور ***D** محتویات ACCU1 و ACCU2 را با هم ضرب کرده و نتیجه را در ACCU1 ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد صحیح هستند
انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Product=0	0	0	0	—
$-2147483648 \leq \text{Product} < 0$	0	1	0	—
$2147483648 \geq \text{Product} > 0$	1	0	0	—
Product > 2147483647	1	0	1	1
Product < -2147483648	0	1	1	1

مثال

L ID10

L MD14

***D**

T MD20

/D

Divided ACCU2 by ACCU1 as Double Integer (32-Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

/D

/D

مثال :

دستور **/D** محتویات ACCU2 را بر ACCU1 تقسیم کرده و فقط نتیجه قسمت صحیح را محاسبه و در ACCU1 ذخیره می کند و نتیجه خارج قسمت را محاسبه نمی کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد صحیح هستند انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت های OS – OV – CC0 – CC1 را تغییر می دهد . برای داشتن خارج قسمت می توانید از دستور MOD استفاده کنید

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Quotient=0	0	0	0	—
$-2147483648 \leq \text{Quotient} < 0$	0	1	0	—
$2147483647 \geq \text{Quotient} > 0$	1	0	0	—
Quotient = 2147483648	1	0	1	1
Divided by zero	1	1	1	1

مثال

L ID10 13

L MD14 4

/D

T MD20 3

MOD

Division Remainder

Double Integer (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

MOD

MOD

مثال :

دستور **MOD** محتویات ACCU2 را بر ACCU1 تقسیم کرده و فقط نتیجه خارج قسمت را محاسبه و در ACCU1 ذخیره می کند و نتیجه قسمت صحیح را محاسبه نمی کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد صحیح هستند انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .
برای داشتن قسمت صحیح می توانید از دستور D/ استفاده کنید

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

Status bit generation	CC1	CC0	OV	OS
Remainder=0	0	0	0	—
$-2147483648 \leq \text{Remainder} < 0$	0	1	0	—
$2147483647 \geq \text{Remainder} > 0$	1	0	0	—
Divided by zero	1	1	1	1

مثال

L ID10 13

L MD14 4

MOD

T MD20 1

فصل هشتم

Floating - point Math Instruction

مجموعه دستورالعمل های

اعمال ریاضی روی اعداد با ممیز شناور

مقدمه :

دستورالعمل های ریاضی – اعمال ریاضی را روی اکومولاتورهای

ACCU1 و ACCU2 انجام می دهند .

هر بار که دستور بارگذاری اجرا شود ابتدا محتویات **ACCU1** وارد

ACCU2 شده و مقدار جدید وارد **ACCU1** می شود.

هنگام محاسبه اعمال ریاضی – محتوای **ACCU2** تغییری نمی کند و

نتیجه محاسبات در **ACCU1** ذخیره می گردد .

اعمال ریاضی مجاز روی داده های نوع Real - ۳۲ بیتی

+R Add ACCU1 and ACCU2

-R Subtract ACCU1 from ACCU2

***R Multiply ACCU1 and ACCU2**

/R Divide ACCU1 by ACCU2

← ادامه

اعمال ریاضی زیر علاوه بر موارد ذکر شده در اسلاید قبلی روی داده های نوع **Real** مجاز است .

ABS	Absolute Value
SQR	Generate the Square
SQRT	Generate the Square Root
EXP	Generate the Exponential Value
LN	Generate the Natural Logarithm
SIN	Generate the Sine of Angles
COS	Generate the Cosin of Angles
TAN	Generate the Tangent of Angles
ASIN	Generate the Arc Sine
ACOS	Generate the Arc Cosine
ATAN	Generate the Arc Tangent

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام محاسبات ریاضی

الف – اگر نتیجه محاسبات در رنج مجاز باشد

رنج مجاز	CC1	CC0	OV	OS
+0,-0	0	0	0	*
اعداد منفی -1.175494E-38 < نتیجه < -3.402823E+38	0	1	0	*
اعداد مثبت +1.175494E-38 < نتیجه < 3.402824E+38	1	0	0	*

*** بیت OS در رنج مجاز تحت تاثیر قرار نمی گیرد**

ب - اگر نتیجه محاسبات در رنج مجاز نباشد

رنج غیر مجاز	CC1	CC0	OV	OS
Underflow اعداد منفی $-1.401298E-45 < \text{نتیجه} < -1.175494E-38$	0	0	1	1
Underflow اعداد مثبت $1.175494E-38 < \text{نتیجه} < +1.40129E-45$	0	0	1	1
Overflow اعداد منفی $+1.175494E-38 < \text{نتیجه} < -3.402824E+38$	0	1	1	1
Overflow اعداد مثبت $\text{نتیجه} > 3.402823E+38$	1	0	1	1
Not a valid floating-point number or illegal instruction (input value outside the valid range)	1	1	1	1

+R

Add ACCU1 and ACCU2

as a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+R

+R

مثال :

دستور **+R** محتویات **ACCU1** و **ACCU2** را با هم جمع کرده و نتیجه را در **ACCU1** ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد با ممیز شناور هستند
انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

OPN DB10

L ID10

L MD14

+R

T DBD25



-R

Subtract ACCU1 from ACCU2

as a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

-R

-R

مثال :

دستور **-R** محتویات ACCU1 را از ACCU2 کم کرده و نتیجه را در ACCU1 ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد با ممیز شناور هستند
انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

OPN DB10

L ID10

L MD14

-R

T DBD25

*R

Multiply ACCU1 and ACCU2

as a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

*R

*R

مثال :

دستور ***R** محتویات ACCU1 و ACCU2 را در هم ضرب کرده و نتیجه را در ACCU1 ذخیره می کند هر دو اکومولاتور ۳۲ بیتی و با محتوای اعداد با ممیز شناور هستند انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

OPN DB10

L ID10

L MD14

***R**

T DBD25

/R Divide ACCU2 by ACCU1 as a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

/R

/R

مثال :

دستور **/R** محتویات ACCU2 را بر محتویات ACCU1 تقسیم کرده و نتیجه را در ACCU1 ذخیره می کند
هر دو اکومولاتور ۳۲ بیتی وبا محتوای اعداد با ممیز شناور هستند
انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

OPN DB10

L ID10

L MD14

/R

T DBD25

ABS

Absolute Value

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

ABS

ABS

مثال :

دستور **ABS** قدر مطلق محتویات ACCU1 را محاسبه کرده و نتیجه را در ACCU1 ذخیره می کند .
محتوای ACCU1 عدد با ممیز شناور است
انجام دستورالعمل فوق تاثیری روی RLO و بیت‌های CC0 – CC1
OV - OS نمی گذارد .

مثال

L ID10

ABS

T MD10

SQR

Generate the Square

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SQR

SQR

مثال :

دستور **SQR** **توان دو** محتویات **ACCU1** را محاسبه کرده و نتیجه را در **ACCU1** ذخیره می کند .
محتوای **ACCU1** عدد با ممیز شناور است

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

SQR

T MD10

SQRT

Generate the Square Root of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SQRT

SQRT

مثال :

دستور **SQRT** جذر محتویات ACCU1 را محاسبه کرده و نتیجه را در ACCU1 ذخیره می کند .
محتوای ACCU1 عدد با ممیز شناور و حتما مساوی صفر و یا بزرگتر از صفر (مثبت) باید باشد.

انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

SQRT

T MD10

EXP Generate the Exponential Value of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

EXP

مثال :	EXP
--------	-----

دستور **EXP** مقدار **e** را به توان محتویات **ACCU1** رسانده و نتیجه را در **ACCU1** ذخیره می کند .
محتوای **ACCU1** عدد با ممیز شناور باید باشد.

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

EXP

T MD10

LN Generate the Natural Logarithm of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LN

مثال : LN

دستور **LN** **لگاریتم طبیعی** محتویات **ACCU1** را محاسبه کرده و نتیجه را در **ACCU1** ذخیره می کند .
محتوای **ACCU1** عدد با ممیز شناور و **حتما بزرگتر از صفر (مثبت)** باید باشد.

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

L ID10

LN

T MD10

SIN

Generate the Sine of Angles

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SIN

SIN

مثال :

دستور **SIN** سینوس محتویات ACCU1 را محاسبه کرده و

نتیجه را در ACCU1 ذخیره می کند .

محتوای ACCU1 عدد با ممیز شناور و بعنوان زاویه بر حسب رادیان

تلقی می گردد .

انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های

OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Overflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

SIN

T MD10

COS

Generate the Cosine of Angles of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

COS

COS

مثال :

دستور **COS** کسینوس محتویات ACCU1 را محاسبه کرده و نتیجه را در ACCU1 ذخیره می کند .
محتوای ACCU1 عدد با ممیز شناور و بعنوان زاویه بر حسب رادیان تلقی می گردد .

انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Overflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

COS

T MD10

TAN

Generate the Tangent of Angles

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAN

TAN

مثال :

دستور **TAN** **تانژانت** محتویات ACCU1 را محاسبه کرده و

نتیجه را در ACCU1 ذخیره می کند .

محتوای ACCU1 عدد با ممیز شناور **وبعنوان زاویه بر حسب رادیان** تلقی می گردد .

انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های

OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+infinite	1	0	1	1	Overflow
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Underflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-infinite	0	1	1	1	Overflow
-qNaN	1	1	1	1	

مثال

L ID10

TAN

T MD10

ASIN

Generate the Arc Sine

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

ASIN

ASIN

مثال :

دستور **ASIN** آرک سینوس محتویات ACCU1 را محاسبه کرده و نتیجه را در ACCU1 ذخیره می کند .

محتوای ACCU1 عدد با ممیز شناور و در محدوده +۱ تا -۱ باید قرار داشته باشد نتیجه خروجی بر حسب رادیان در محدوده زیر قرار دارد

$$-3.14/2 \leq \text{arc sin} (\text{ACCU1}) \leq +3.14/2$$

انجام دستورالعمل فوق تاثیری روی RLO نمی گذارد ولی بیت‌های OS – OV – CC0 – CC1 را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Overflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

ASIN

T MD10

ACOS

Generate the Arc Cosine

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

ACOS

مثال :
ACOS

دستور **ACOS** آرک کسینوس محتویات **ACCU1** را محاسبه کرده و نتیجه را در **ACCU1** ذخیره می کند .

محتوای **ACCU1** عدد با ممیز شناور و در محدوده **+۱ تا -۱** باید قرار داشته باشد نتیجه خروجی بر حسب رادیان در محدوده زیر قرار دارد

$$0 \leq \text{arc cos} (\text{ACCU1}) \leq +3.141592535$$

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Overflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

ACOS

T MD10

ATAN Generate the Arc Sine

of a Floating – point number (32–Bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

ATAN

ATAN

مثال :

دستور **ATAN** آرک تانژانت محتویات **ACCU1** را محاسبه کرده و نتیجه را در **ACCU1** ذخیره می کند .
محتوای **ACCU1** عدد با ممیز شناور باید باشد.

نتیجه خروجی بر حسب رادیان در محدوده زیر قرار دارد

$$-3.14/2 \leq \text{arc tangent} (\text{ACCU1}) \leq +3.14/2$$

انجام دستورالعمل فوق تاثیری روی **RLO** نمی گذارد ولی بیت‌های **OS – OV – CC0 – CC1** را تغییر می دهد .

نحوه تاثیر پذیری بیت های (OS و OV و CC0 و CC1) بر اثر انجام این محاسبه ریاضی

The result in ACCU 1 is	CC 1	CC 0	OV	OS	Note
+qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	Overflow
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	Underflow
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

مثال

L ID10

ATAN

T MD10



فصل نهم

Load and - Transfer Instruction

مجموعه دستورالعمل های
بار گذاری و انتقال

دستورالعمل Load و Transfer امکان تبادل و جابجایی اطلاعات را
بین مدول های ورودی و خروجی و نواحی حافظه CPU و یا بین خود
نواحی حافظه CPU را فراهم می سازد .

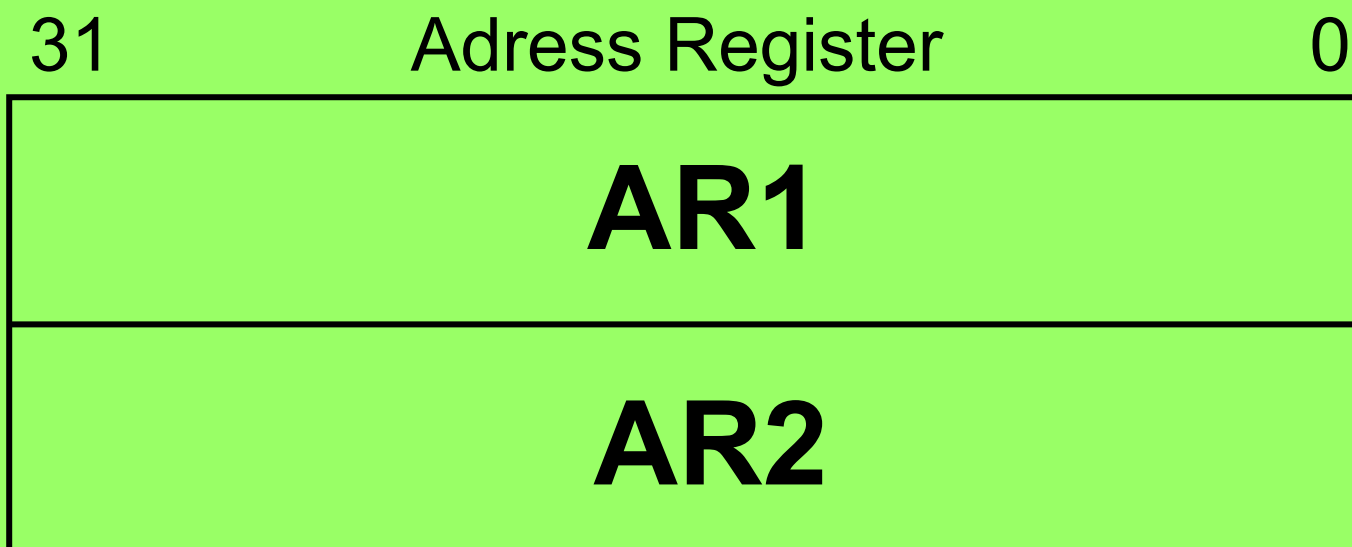
در هر سیکل اسکن **CPU** این دستورات را بدون قید و شرط و
بدون توجه به مقدار **RLO** اجرا می کند .

در ضمن اجرای این دستورات RLO تحت تاثیر قرار نمی گیرد .

به یاد داشته باشید :

در CPU های زیمنس دو رجیستر ۳۲ بیتی مخصوص نگهداری

آدرس وجود دارند



الف : دستورات بار گذاری

<i>L</i>	<i>Load</i>
<i>LSTW</i>	<i>Load Status Word into ACCU1</i>
<i>LAR1 AR2</i>	<i>Load Address Register 1 from Address Register2</i>
<i>LAR1 <D></i>	<i>Load Address Register 1 with Double Integer (32 bit pointer)</i>
<i>LAR1</i>	<i>Load Address Register 1 from ACCU1</i>
<i>LAR2 <D></i>	<i>Load Address Register 2 with Double Integer (32 bit pointer)</i>
<i>LAR2</i>	<i>Load Address Register 2 from ACCU1</i>

L

Load

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L < Address >

L MW2 مثال :

با اجرای دستور **L** ابتدا محتویات ACCU1 وارد ACCU2 میشود و سپس محتویات ACCU1 برابر صفر شده (ریست می شود) و آنگاه محتویات آدرس داده شده در ACCU1 قرار می گیرد .
نوع داده ای که میتواند درون **ACCU1** قرار بگیرد عبارتند از :

BYTE – WORD - DWORD

محتویات ACCU1 به صورت زیر تغییر می کند

Contents of ACCU 1

Contents of ACCU 1	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
before execution of load instruction	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
after execution of L MB10 (L <Byte>)	00000000	00000000	00000000	<MB10>
after execution of L MW10 (L <word>)	00000000	00000000	<MB10>	<MB11>
after execution of L MD10 (L <double word>)	<MB10>	<MB11>	<MB12>	<MB13>
after execution of L P# ANNA (in FB)	<86>	<Bit offset of ANNA relative to the FB start>. To calculate the absolute offset in the instance data block in multiple instance FBs, the contents of the AR2 register must be added to this value.		
after execution of L P# ANNA (in FC)	<An area-crossing address of the data which is transferred to ANNA>			
	X = "1" or "0"			

مثال

L	IB10
L	MB120
L	DBB12
L	DIW15
L	LD252
L	P#I8.7
L	OTTO

L STW *Load Status Word Into ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

L STW

L STW

مثال :

با اجرای دستور **L STW** محتویات Status Word وارد ACCU1 می شود.

انجام دستور العمل فوق تاثیری روی RLO و

Status Word ندارد

محتوای ACCU1 بعد از اجرای L STW

31	 10	9	8	7	6	5	4	3	2	1	0
0	 0	0	BR	cc1	cc0	OV	OS	OR	STA	RLO/FC	

مثال

L STW

LAR1 *Load Address Register 1 from ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LAR1

LAR1

مثال :

با اجرای دستور **LAR1** محتویات ACCU1(32-bit pointer) وارد Address Register 1 می شود.

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

LAR1<D>

*Load Address Register 1
With Double Integer (32- bit pointer)*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LAR1 <DWORD نوع داده با آدرس

مثال : LAR1 MD24

با اجرای دستور **LAR1<D>** محتوای حافظه آدرس داده شده وارد (AR1) Address Register1 می شود.

با اجرای دستور فوق محتویات **ACCU1 و ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

LAR1	DBD20
LAR1	DID30
LAR1	LD180
LAR1	MD24
LAR1	P#M8.7

LAR1 AR2

*Load Address Register 1
from Address Register 2*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LAR1 AR2

LAR1 AR2

مثال :

با اجرای دستور **LAR1 AR2** محتویات Address Register 2 وارد Address Register 1 می کند.

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

LAR2 *Load Address Register 2 from ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LAR2

LAR2

مثال :

با اجرای دستور **LAR2** محتویات ACCU1(32-bit pointer) وارد Address Register2 می شود.

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

LAR2<D>

*Load Address Register 2
With Double Integer (32- bit pointer)*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

LAR2 <DWORD نوع داده با آدرس

مثال : LAR2 MD24

با اجرای دستور **LAR2<D>** محتوای حافظه آدرس داده شده وارد (AR2) Address Register2 می شود.

با اجرای دستور فوق محتویات **ACCU1 و ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

LAR2	DBD20
LAR2	DID30
LAR2	LD180
LAR2	MD24
LAR2	P#M8.7

ب : دستورات انتقال

T Transfer

T STW Transfer ACCU1 INTO Status Word

TAR1 AR2 Transfer Address Register 1 to Address Register2

TAR1 <D> Transfer Address Register 1 to Destination (32- bit pointer)

TAR2 <D> Transfer Address Register 2 to Destination (32- bit pointer)

TAR1 Transfer Address Register 1 to ACCU1

TAR2 Transfer Address Register 2 to ACCU1

CAR Exchange Address Register 1 with Address Register 2

T

Transfer

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

T <Address>

T	QB10	مثال :
---	------	--------

با اجرای دستور **T** محتویات ACCU1 به آدرس داده شده انتقال می یابد . این که چه مقدار از محتویات ACCU1 انتقال می یابد بستگی به آدرس مشخص شده دارد .

با اجرای دستور فوق محتویات ACCU1 و ACCU2 تغییری نمی کنند

**انجام دستور العمل فوق تاثیری روی RLO و
Status Word ندارد**

به توضیحات اسلاید
بعدی توجه کنید

اگر در برنامه از دستور **Master Control Relay** استفاده شود
و **MCR=0** باشد با اجرای دستور **T** مقدار ۰ (صفر) به آدرس مورد
نظر ارسال می شود و ربطی به مقدار **ACCU1** ندارد .

مثال

T	QB12
T	MW120
T	DBD2

T STW

Transfer ACCU1 Into Status Word

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

T STW

T STW

مثال :

با اجرای دستور **T STW** بیت های (۰ تا ۸) - ACCU1 به رجیستر Status Word انتقال می یابند

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** ندارد

در CPU های S7-300 با اجرای دستور فوق فقط بیت های ۱ و ۴ و ۵ و ۶ و ۷ و ۸ تغییر می کنند و بیت های /FC - STA - OR تغییری نمی کنند .

CAR

Exchange Address Register 1 With Address Register 2

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

CAR

CAR

مثال :

با اجرای دستور **CAR** محتویات Address Register 1 با Address Register 2 عوض می شود .

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

TAR1 *Transfer Address Register 1 to ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAR1

TAR1

مثال :

با اجرای دستور **TAR1** محتویات Address Register 1 به ACCU1(32-bit pointer) فرستاده می شود.

با اجرای دستور فوق ابتدا محتویات **ACCU1** به **ACCU2** فرستاده می شود

انجام دستور العمل فوق تاثیری روی **RLO** و **Status Word** ندارد

TAR1<D>

*Transfer Address Register 1
Destination (32- bit pointer)*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAR1 <DWORD با داده نوع

TAR1	MD24	مثال :
-------------	-------------	--------

با اجرای دستور **TAR1<D>** محتوای Address Register1 (AR1) به حافظه آدرس داده شده فرستاده می شود.

با اجرای دستور فوق محتویات ACCU1 و ACCU2 تغییری نمی کنند

**انجام دستور العمل فوق تاثیری روی RLO و
Status Word ندارد**

TAR1 AR2

*Transfer Address Register 1
to Address Register 2*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAR1 AR2

TAR1 AR2

مثال :

با اجرای دستور **TAR1 AR2** محتویات Address Register1 وارد Address Register2 می کند.

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

TAR2 *Transfer Address Register 2 to ACCU1*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAR2

TAR2

مثال :

با اجرای دستور **TAR2** محتویات Address Register 2 وارد ACCU1(32-bit pointer) می شود.

با اجرای دستور فوق ابتدا محتویات **ACCU1** به **ACCU2** فرستاده می شود

انجام دستور العمل فوق تاثیری روی **RLO** و **Status Word** ندارد

TAR2<D>

*Transfer Address Register 2
to Destination (32- bit pointer)*

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAR2 <DWORD نوع داده با آدرس

TAR2	MD24	مثال :
------	------	--------

با اجرای دستور **TAR2<D>** محتوای Address Register2 (AR2) به حافظه آدرس داده شده ارسال می شود.

با اجرای دستور فوق محتویات **ACCU1** و **ACCU2** تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

فصل دهم

Program Control Instruction

مجموعه دستورالعمل های

کنترل برنامه

دستورات کنترل برنامه که در این فصل مورد بحث قرار می گیرند عبارتند از :

الف :

BE	Block End
BEC	Block End Conditional
BEu	Block End Unconditional
CALL	Block Call
CC	Conditional Call
UC	Unconditional Call

ب :

CALL	FB
CALL	FC
CALL	SFB
CALL	SFC
CALL	Multiple Instance
CALL	Block from a Library

ج :

MCR (Master Control Relay)

Important Notes on Using MCR Function

MCR (Save RLO in MCR Stack, Begin MCR)
MCR End MCR

MCRA Active MCR Area

MCRD Deactive MCR Area

BE

Block End

اجرای دستور BE باعث می شود که اسکن بلوک جاری متوقف شده و به بلوکی که این بلوک را فرا خوانده برگردد و به اجرای ادامه برنامه بعد از فراخوانی بلوک پردازد .

دیتاهای محلی مربوط به بلوک فعلی رها شده و دیتاهای محلی بلاک اصلی فعال می شوند . دیتا بلوک هایی که قبل از دستور CALL باز شده بودند مجدداً باز می شوند (Re – Opened) .

این دستور برای اجرا هیچ شرطی را نمی پذیرد و برنامه به محض رسیدن به آن قطع می شود . اگر از روی این دستور پرش انجام شود اجرای بلوک هیچگاه خاتمه نمی یابد .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

BE

مثال

```
A      I 1.0
JC      NEXT
L       IW4
T       IW10
A       I 6.0
A       I 6.1
S       M 12.0
```

BE

NEXT: NOP 0

BEC

Block End Conditional

اجرای دستور BEC باعث می شود که اسکن بلوک جاری متوقف شده و به بلوکی که این بلوک را فرا خوانده برگردد و به اجرای ادامه برنامه بعد از فراخوانی بلوک بپردازد .

دیتاهای محلی مربوط به بلوک فعلی رها شده و دیتاهای محلی بلاک اصلی فعال می شوند . دیتا بلوک هایی که قبل از دستور CALL باز شده بودند مجدداً باز می شوند (Re – Opened) .

برنامه به محض رسیدن به این دستور قطع می شود .

این دستور در صورتی اجرا می شود که RLO قبل از آن ” ۱ ” باشد
اگر از روی این دستور پرش انجام شود اجرای بلوک هیچگاه خاتمه نمی یابد .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

BEC

مثال

A I 1.0

BEC

L IW4

T MW10

BEU Block End Unconditional

اجرای دستور BEU باعث می شود که اسکن بلوک جاری متوقف شده و به بلوکی که این بلوک را فرا خوانده برگردد و به اجرای ادامه برنامه بعد از فراخوانی بلوک پردازد .

دیتاهای محلی مربوط به بلوک فعلی رها شده و دیتاهای محلی بلاک اصلی فعال می شوند . دیتا بلوک هایی که قبل از دستور CALL باز شده بودند مجدداً باز می شوند (Re – Opened) .

این دستور برای اجرا هیچ شرطی را نمی پذیرد و برنامه به محض رسیدن به آن قطع می شود . اگر از روی این دستور پرش انجام شود اجرای بلوک هیچگاه خاتمه نمی یابد .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

BE

مثال

```
A      I 1.0
JC      NEXT
L       IW4
T       IW10
A       I 6.0
A       I 6.1
S       M 12.0
```

BEU

NEXT: NOP 0

CALL

Block Call

Call < Logical block identifier >

دستورالعمل **CALL** برای فراخوانی بلوک های **FC** و **FB** و **SFC** و **SFB** یا سایر بلوک های استاندارد که از قبل توسط زیمنس برنامه نویسی شده اند به کار می رود . این دستور بلاکی که در جلوی آن به صورت مستقیم و یا سمبلیک آدرس داده شده را بدون هیچ قید و شرطی و مستقل از وضعیت **RLO** صدا می زند یعنی کنترل اجرای برنامه را به آن بلاک می برد **FB** و **SFB** باید حتما همراه با ذکر نام **DB** صدا زده شوند **DB** های فوق یا باید قبلا وجود داشته باشند و یا در همان لحظه ای که برنامه نوشته می شود با پیغامی که توسط **Step7** داده می شود ایجاد شوند .

ادامه ←

جدول زیر نحوه صدا زدن بلاک های مختلف را نشان می دهد :

Logic Block	Block Type	Absolute Address Call Syntax
FC	Function	CALL FCn
SFC	System function	CALL SFCn
FB	Function block	CALL FBn1,DBn2
SFB	System function block	CALL SFBn1,DBn2

اگر برای بلاک ها از اسامی سمبلیک استفاده شود این اسامی باید قبلا در **Symbolic Table** تعریف شده باشند

وقتی یک بلاک در برنامه **STL** با دستور **CALL** صدا زده می شود به طور اتوماتیک لیستی از متغیر ها مانند مثال زیر در زیر آن ظاهر می گردند

Formal Parameter متغیرهایی هستند که قبلا در موقع ایجاد بلاک به عنوان ورودی و خروجی بلاک تعریف شده اند

Actual Parameter متغیر یا آدرس هایی هستند که برنامه نویس باید آن ها را تعیین کند .

مثال ۱:

CALL	FC6	
	Formal parameter	Actual parameter
	NO OF TOOL	:= MW100
	TIME OUT	:= MW110
	FOUND	:= Q 0.1
	ERROR	:= Q 100.0

در عین حال ممکن است برای بلوکی ورودی و خروجی تعریف نشده باشد برای این گونه بلاک ها لیست فوق با دستور **CALL** ظاهر نمی شود

اگر **FB** صدا زده شود مقادیر و آدرس هایی که به عنوان **Actual Parameter** تعریف می شوند در موقع اجرای برنامه در دیتا بلاک **Instance** مشخص شده در دستور **CALL** ذخیره می شوند

CALL	FB99,DB2	
	Formal parameter	Akcual parameter
	MAX_RPM	:= #RPM2_MAX
	MIN_RPM	:= #RPM2
	MAX_POWER	:= #POWER2
	MAX_TEMP	:= #TEMP2

مثال ۲:

Call Multiple Instance

CALL #Variable Name

Multiple Instance به منظور دسترسی چند FB به یک DB طراحی شده است و منجر به صرفه جویی در حافظه CPU می گردد در این روش FB اصلی با یک DB لینک است و همراه با آن صدا زده می شود. مثلا FB1,DB1 اگر فرض کنیم چند FB دیگر مثلا FB2 و FB3 نیز بخواهند از DB1 همزمان استفاده کنند در این صورت در جدول Declaration مربوط به FB1 سایر FB ها را با نام دلخواه و توسط متغیری از نوع static معرفی می نماییم

ادامه ← ۳۳۳

تهیه کننده : فتح اله نظریان - فروردین ماه
۱۳۸۵

السبت، ۰۳ ربيع الأول، ۱۴۲۷

Decla.	Name	Type
Stat.	TEST2	FB2
Stat.	TEST3	FB3

مثال

پس از تکمیل این جدول می توانیم در داخل FB دو بلاک FB دیگر را به صورت

زیر صدا بزنیم و نیازی به ذکر نام DB در آن ها نیست

CALL #TEST2

CALL #TEST3

CC

Condition Call

CC < Logical block identifier >

دستور **CC** برای صدا زدن بلاک های **FC** و **SFC** به صورت شرطی به کار می رود یعنی در صورتی که **RLO=1** باشد بلاک مورد نظر صدا زده می شود در این روش پارامترهای بلاک در زیر دستور **CC** ظاهر نمی شوند به این معنی که با این روش امکان دادن دیتا به بلاک یا گرفتن دیتا از بلاک وجود ندارد . پس در مواردی که بلاک نیاز به ورودی و خروجی ندارد و قابل استفاده است .

A	I0.0
CC	FC6
A	M3.0

مثال

UC

Uncondition Call

UC < Logical block identifier >

دستور UC برای صدا زدن بلاک های FC و SFC بدون قید و شرط و بدون توجه به وضعیت RLO به کار می رود در این روش پارامترهای بلاک در زیر دستور CC ظاهر نمی شوند به این معنی که با این روش امکان دادن دیتا به بلاک یا گرفتن دیتا از بلاک وجود ندارد .
پس در مواردی که بلاک نیاز به ورودی و خروجی ندارد و قابل استفاده است .

A 10.0
UC FC6

مثال

MCR

Master Control Relay

دستورات MCR همانند یک سویچ اصلی برای قطع و وصل تغذیه عمل می کند
MCR دستورات زیر را تحت تاثیر قرار می دهد :

= <bit>

S <bit>

R <bit>

T <Byte>

<Word>

<Double Word >

اگر $MCR=0$ باشد شبیه حالت قطع کلید عمل می کند . و خروجی ها یا بیت هایی که در دستورهای = و T معرفی شده اند را صفر می کند .
یعنی این دستورات عملاً در برنامه کاری انجام نمی دهند . بعلاوه خروجی ها یا بیت هایی را که با دستورات R و S کار می کنند نیز آخرین وضعیت خود را

حفظ کرده و دیگر ست یا ری ست نمی شوند . اگر $MCR=1$ شود
 برنامه کار عادی خود را دنبال می کند یعنی مانند وصل شدن کلید
 تغذیه . نتیجه بحث فوق در جدوت زیر آمده است :

Signal State of MCR	= <bit>	S <bit>, R <bit>	T <byte>, T <word> T <double word>
0 ("OFF")	Writes 0. (Imitates a relay that falls to its quiet state when voltage is removed.)	Does not write. (Imitates a relay that remains in its current state when voltage is removed.)	Writes 0. (Imitates a component that produces a value of 0 when voltage is removed.)
1 ("ON")	Normal processing	Normal processing	Normal processing

MCR متشکل از چند دستور است که **به ترتیب زیر** نوشته می شوند

MCRA **Active MCR Area**

MCR (**Save RLO in MCR Stack, Begin MCR Area**

)MCR **End MCR Area**

MCRD **Deactive MCR Area**

MCR با **MCRA** فعال و با **MCRD** غیر فعال می شود هر دو دستور را باید به صورت جفتی به کار برد و نمی توان فقط از یکی استفاده کرد . این دو دستور بدون توجه به بیت های **Status Word** اجرا شده و تاثیری روی آن ها نمی گذارند .

برنامه بین دو دستور **MCR** و **MCR** (نوشته می شود که به آن ناحیه **MCR** می گویند .

دستور **MCR** این ناحیه را باز کرده و **RLO** را در پشته **MCR** ذخیره میکند . اگر **RLO=1** باشد در این صورت **MCR=1** یعنی **ON** است و پردازش برنامه

عادی است یعنی MCR روی آن تاثیری نمی گذارد ولی وقتی $RLO = 0$ شد در این صورت $MCR = 0$ یعنی OFF خواهد شد و خروجی ها طبق اسلاید قبلی تغییر خواهند کرد .

ناحیه MCR که با دستور (MCR باز شده با دستور MCR بسته می شود و این دو دستور با هم به کار می روند اصطلاحاً Nested هستند . می توان آن ها را تو در تو و ماکزیمم تا ۸ مرحله به کار برد ولی تعداد MCR ها باید با MCR برابر باشند در غیر این صورت خطای پشته MCR به صورت MCRF ظاهر خواهد شد .

Status Word با دستورات MCR و MCR (به صورت زیر تحت تاثیر قرار می گیرد .

OR = 0

STA=1

/FC=0

مثال

MCRA

A I 1.0

MCR(

A I 4.0

= Q 8.0

L MW20

T QW10

)MCR

MCRD

A I 1.1

= Q 8.1

MCR

Master Control Relay

اخطار : به منظور جلوگیری از بروز هرگونه حادثه هرگز
دستورات MCR را جایگزین مدارات سخت افزاری قطع
اضطراری نکنید .

فصل یازدهم

Shift and Rotate Instruction

مجموعه دستورالعمل های

شیفت و چرخش

الف : دستورات شیفت

دستورات شیفت برای جابجایی بیت به بیت محتویات ACCU1 به چپ یا راست به کار می رود .

در هر بار شیفت از راست به چپ یک صفر از سمت راست وارد آکومولاتور می شود و در هر بار شیفت چپ به راست اگر آخرین بیت (بیت MSB) صفر باشد (عدد مثبت) صفر وارد آکومولاتور می شود و اگر آخرین بیت (بیت MSB) یک باشد (عدد منفی) یک وارد آکومولاتور می شود .

در هر دو نوع شیفت آخرین بیت شیفت شده به CC1 بار می شود و بیت های CC0 و OV صفر می شوند .

می توان از بیت CC1 برای پرش (Jump) استفاده کرد .

اجرای دستورات شیفت RLO را تحت تاثیر قرار نمی دهد .

دستور العمل های شیفت

SSI	Shift Sign Integer (16 – bit)
SSD	Shift Sign Double Integer (32 – bit)
SLW	Shift Left Word (16 – bit)
SRW	Shift Right Word (16 – bit)
SLD	Shift Left Double Word (32 – bit)
SRD	Shift Right Double Word (32 – bit)

ب : دستورات چرخش

دستورات Rotate برای چرخش بیت به بیت محتویات ACCU1 به چپ یا راست به کار می رود .

فرق این دستور با شیفت این است که در شیفت صفر یا یک از یک انتها وارد آکومولاتور می شد ولی در چرخش آخرین بیت به جای اولین بیت می نشیند و از بیرون چیزی وارد آکومولاتور نمی شود

آخرین بیت چرخش شده به CC1 بار می شود و بیت های CC0 و OV صفر می شوند .

می توان از بیت CC1 برای پرش (Jump) استفاده کرد .

اجرای دستورات چرخش RLO را تحت تاثیر قرار نمی دهد .

دستورالعمل های چرخش

RLD Rotate Left Double Word (32 – bit)

RRD Rotate Right Double Word (32 – bit)

RLDA Rotate ACCU1 Left via cc1 (32 – bit)

RRDA Rotate ACCU1 Right via cc1 (32 – bit)

SSI

Shift Sign Integer (16 – bit)

دستور SSI برای شیفت راست یک عدد صحیح ۱۶ بیتی علامت دار (مثبت یا منفی) به کار می رود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت راست شیفت می دهد . محتوای ACCU1-H تغییر نمی کند .

اگر SSI به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت $SSI < Number >$ به کار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور (No Operation) NOP است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SSI <Number>

SSI

SSI 6

مثال :

L MW4

SSI 6

مثال :

T MW8

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of SSI 6	0101	1111	0110	0100	1001	1101	0011	1011
after execution of SSI 6	0101	1111	0110	0100	1111	1110	0111	0100

محتوای ACCU1-H تغییر نکرده است .

SSD

Shift Sign Double Integer (32 – bit)

دستور SSD برای شیفت راست یک عدد صحیح ۳۲ بیتی علامت دار (مثبت یا منفی) به کار می رود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست شیفت می دهد .

اگر SSD به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت **SSD < Number >** به کار رود تعداد شیفت به اندازه عدد صحیحی است که با **Number** مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور **NOP (No Operation)** است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SSD <Number>

SSD

SSD 6

مثال :

L MD4

SSD 7

T MD8

مثال :

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of SSD 7	1000	1111	0110	0100	0101	1101	0011	1011
after execution of SSD 7	1111	1111	0001	1110	1100	1000	1011	1010

SLW

Shift Left Word (16 – bit)

دستور SLW برای شیفت چپ یک WORD به کار می رود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت چپ شیفت می دهد .
محتوای ACCU1-H تغییر نمی کند .

اگر SLW به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت $SLW < Number >$ به کار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور (No Operation) NOP است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SLW <Number>

SLW

SLW 6

مثال :

L MW4

SLW 5

T MW8

مثال :

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of SLW 5	0101	1111	0110	0100	0101	1101	0011	1011
after execution of SLW 5	0101	1111	0110	0100	1010	0111	0110	0000

محتوای **ACCU1-H** تغییر نکرده است .

SRW

Shift Right Word (16 – bit)

دستور SRW برای شیفت راست یک WORD به کار می رود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت راست شیفت می دهد .
محتوای ACCU1-H تغییر نمی کند .

اگر SRW به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت $SRW < Number >$ به کار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور (No Operation) NOP است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SRW <Number>

SRW

SRW 6

مثال :

L MW4

SRW 6

مثال :

T MW8

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of SRW 6	0101	1111	0110	0100	0101	1101	0011	1011
after execution of SRW 6	0101	1111	0110	0100	0000	0001	0111	0100

محتوای **ACCU1-H** تغییر نکرده است .

SLD Shift Left Double Word (32 – bit)

دستور SLD برای شیفت چپ یک DWORD به کار می رود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت چپ شیفت می دهد .

اگر SLD به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت **SLD < Number >** به کار رود تعداد شیفت به اندازه عدد صحیحی است که با **Number** مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور **NOP (No Operation)** است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SLD <Number>

SLD

SLD 6

مثال :

L MD4

SLD 5

مثال :

T MD8

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	...	...	... 16	15 ...	...	...	... 0
before execution of SLD 5	0101	1111	0110	0100	0101	1101	0011	1011
after execution of SLD 5	1110	1100	1000	1011	1010	0111	0110	0000

SRD

Shift Right Double Word (32 – bit)

دستور SRD برای شیفت راست یک DWORD به کار می رود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست شیفت می دهد .

اگر SRD به تنهایی به کار رود تعداد شیفت به اندازه عددی است که قبلا به ACCU2-L-L بار شده است .

و اگر دستور به صورت $\text{SRD} \langle \text{Number} \rangle$ به کار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص می شود .

عدد می تواند بین ۰ تا ۱۵ باشد . اگر عدد صفر باشد عملا محتوای آکومولاتور تغییر نمی کند و معادل دستور (No Operation) NOP است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SRD <Number>

SRD

SRD 6

مثال :

L MD4

SRD 7

T MD8

مثال :

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of SRD 7	0101	1111	0110	0100	0101	1101	0011	1011
after execution of SRD 7	0000	0000	1011	1110	1100	1000	1011	1010

RLD

Rotate Left Double Word (32 – bit)

دستور RLD محتوای آکومولاتور ACCU1 را بیت به بیت به سمت چپ چرخش می دهد

اگر RLD به تنهایی به کار رود تعداد چرخش به اندازه عددی است که قبلا به ACCU2-L-L بار شده است .

و اگر دستور به صورت `RLD < Number >` به کار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص می شود .

عدد می تواند بین ۰ تا ۳۲ باشد . اگر عدد صفر باشد عملا محتوای آکومولاتور تغییر نمی کند و معادل دستور (NOP (No Operation است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

RLD <Number>

RLD

RLD 6

مثال :

L MD4

RLD 4

مثال :

T MD8

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of RLD 4	0101	1111	0110	0100	0101	1101	0011	1011
after execution of RLD 4	1111	0110	0100	0101	1101	0011	1011	0101

RRD

Rotate Right Double Word (32 – bit)

دستور RRD محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست چرخش می دهد

اگر RRD به تنهایی به کار رود تعداد چرخش به اندازه عددی است که قبلاً به ACCU2-L-L بار شده است .

و اگر دستور به صورت `RRD < Number >` به کار رود تعداد شیفت به اندازه عدد صحیحی است که با `Number` مشخص می شود .

عدد می تواند بین ۰ تا ۳۲ باشد . اگر عدد صفر باشد عملاً محتوای آکومولاتور تغییر نمی کند و معادل دستور `NOP ( No Operation )` است .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

RRD <Number>

RRD

RRD 6

مثال :

L MD4

RRD 4

مثال :

T MD8

Examples

Contents	ACCU1-H				ACCU1-L			
Bit	31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of RRD 4	0101	1111	0110	0100	0101	1101	0011	1011
after execution of RRD 4	1011	0101	1111	0110	0100	0101	1101	0011

RLDA

Rotate Left

Double Word Via CC1(32 – bit)

دستور RLDA محتوای آکومولاتور ACCU1 (Dword موجود) را فقط یکبار بیت به بیت به سمت چپ و از طریق بیت CC1 چرخش می دهد

به عبارت دیگر آخرین بیت یعنی بیت ۳۱ از ACCU1 به CC1 به اولین بیت آکومولاتور بار می شود .

این دستور بیت های CC0 و OV را صفر می کند ولی مقدار CC1 با توجه به توضیح فوق می تواند صفر یا یک باشد .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

RLDA <Number>

RLDA

RLDA

مثال :

L MD4

RLDA

مثال :

JP NEXT

Examples

Contents	CC 1	ACCU1-H				ACCU1-L			
Bit		31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of RLDA	X	0101	1111	0110	0100	0101	1101	0011	1011
after execution of RLDA	0	1011	1110	1100	1000	1011	1010	0111	011X
(X = 0 or 1, previous signal state of CC 1)									

RRDA

Rotate Right Double Word Via CC1(32 – bit)

دستور RRDA محتوای آکومولاتور ACCU1 (Dword موجود) را فقط یکبار بیت به بیت به سمت راست و از طریق بیت CC1 چرخش می دهد

به عبارت دیگر اولین بیت یعنی بیت ۰ از ACCU1 به CC1 رفته و مقدار CC1 به آخرین بیت آکومولاتور بار می شود .

این دستور بیت های CC0 و OV را صفر می کند ولی مقدار CC1 با توجه به توضیح فوق می تواند صفر یا یک باشد .

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

RRDA <Number>

RRDA

RRDA

مثال :

L MD4

RRDA

مثال :

JP NEXT

Examples

Contents	CC 1	ACCU1-H				ACCU1-L			
Bit		31 ...	..	..	... 16	15 ...	..	..	... 0
before execution of RRDA	X	0101	1111	0110	0100	0101	1101	0011	1011
after execution of RRDA	1	X010	1111	1011	0010	0010	1110	1001	1101
(X = 0 or 1, previous signal state of CC 1)									

فصل دوازدهم

Timer Instruction

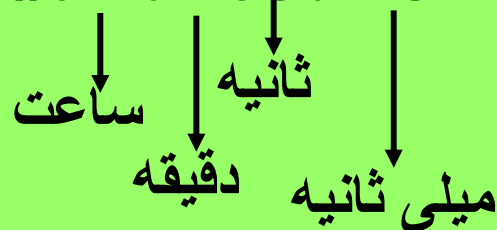
مجموعه دستورالعمل های

تایمر ها

هر CPU تعداد معینی تایمر را پشتیبانی می کند که این بستگی به نوع CPU دارد و با مراجعه به کاتالوگ CPU می توان از تعداد آن مطلع شد .
CPU ها برای هر تایمر دو بایت فضای حافظه در یک مکان خاص از قبل رزرو می کنند .

از دو بایت فضای حافظه (۱۶ بیت) مقدار زمان در بیت های صفر تا ۹ (۱۰ بیت) به صورت باینری ذخیره می شود .
قبل از راه اندازی تایمر باید مقدار زمان را به آن بار کرد .
فرمت رایج بار کردن زمان به صورت زیر است :

S5T#aHbMc\$dMS



مثال : یک ساعت و سی و شش دقیقه و پانزده ثانیه به صورت زیر نشان می دهند

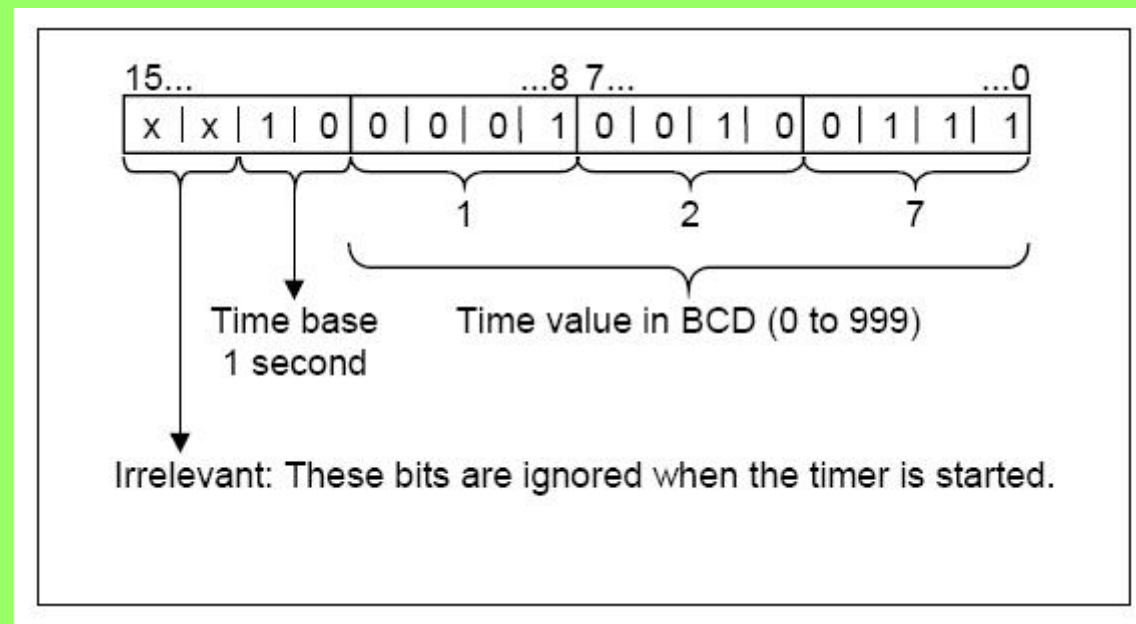
S5T#1H36M15S

پس از راه اندازی تایمر , زمان شروع به کم شدن می کند تا این که به صفر برسد .
نمایش زمان از ۰۰۰ تا ۹۹۹ است .

ماکزیم زمانی که می توان انتخاب کرد عبارت است از : ۹۹۹۰ ثانیه و یا

۲ ساعت - ۴۶ دقیقه - ۳۰ ثانیه با دقت ۱۰ ثانیه

پله های زمانی (Time Base) که زمان طبق آنها کاهش می یابد در بیت های ۱۲ و ۱۳ از Word مربوط به تایمر ذخیره می شود.
پله های زمانی حداقل ۱۰ میلی ثانیه و حداکثر ۱۰ ثانیه است .



بسته به نوع پله زمانی انتخاب شده که به آن **Resolution** هم می گویند رنج زمانی مشخص برای تایمر قابل استفاده خواهد بود طبق جدول زیر :

Resolution	Range
0.01 second	10MS to 9S_990MS
0.1 second	100MS to 1M_39S_900MS
1 second	1S to 16M_39S
10 seconds	10S to 2H_46M_30S

وقتی تایمر شروع به کار می کند محتوای آکومولاتور ۱ به عنوان مقدار زمان مورد استفاده قرار می گیرد .

مقدار زمان به صورت **BCD** در بیت های ۰ تا ۱۱ ذخیره می گردد (به شکل اسلاید قبلی توجه کنید). بیت های ۱۲ و ۱۳ پله های زمانی را به صورت باینری نمایش می دهند . مطابق جدول اسلاید بعدی 

Time Base	Binary Code for the Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

لازم به یادآوری است که :

وقتی ما مقدار زمان را تنظیم می کنیم به طور اتوماتیک Resulation

تنظیم می شود و نیازی نیست که ما آن را تنظیم کنیم

لیست دستورات تایمرها به شرح زیر است :

FR	Enable Timer
L	Load current Timer Value into ACCU1 as Integer
LC	Load current Timer Value into ACCU1 as BCD
R	Reset Timer
SD	On-Delay Timer
SE	Extended Pulse Timer
SF	Off-delay Timer
SP	Pulse Timer
SS	Retentive On-Delay Timer



FR

Enable Timer (Free)

وقتی RLO از صفر به یک تغییر وضعیت می دهد دستور FR لبه سیگنال را تشخیص داده و تایمر مربوطه را فعال می کند

این دستور آزاد است یعنی استفاده از آن الزامی نیست بدون آن نیز با تغییر RLO از صفر به یک می توان یک تایمر را فعال کرد .

فقط برای تریگر مجدد تایمری که در حال کار است بکار می رود و آن را مجدداً با مقدار زمان اولیه Restart می کند .

L

Load Current Time Value as Integer

اجرای دستورالعمل **L** باعث می شود که ابتدا محتویات ACCU1

وارد ACCU2 شده و سپس مقدار فعلی زمان تایمر را از حافظه CPU

به صورت عدد صحیح (باینری) به ACCU1-L بار کند.

لازم به یادآوری است که پله های زمانی (Time Base) که در حافظه

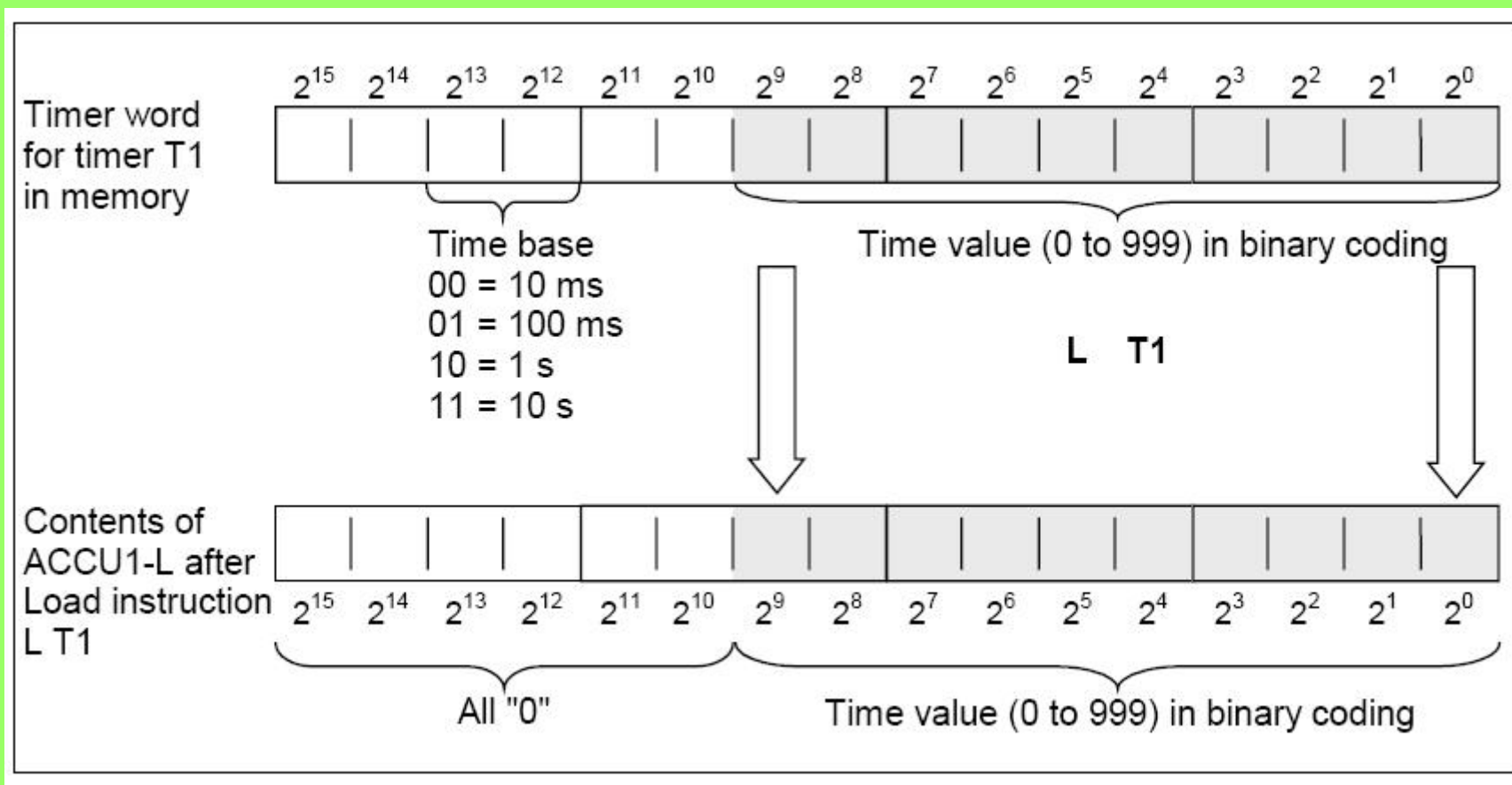
CPU موجود است به اکومولاتور بار نمی شود .

انجام دستورالعمل فوق تاثیری روی بیت های

Status Word ندارد

مثال

L T1



LC

Load Current Timer
Value into ACCU1 as BCD

اجرای دستورالعمل **L** باعث می شود که ابتدا محتویات ACCU1

وارد ACCU2 شده و سپس مقدار فعلی زمان تایمر را از حافظه CPU

به صورت BCD به ACCU1-L بار کند.

لازم به یادآوری است که پله های زمانی (Time Base) که در حافظه

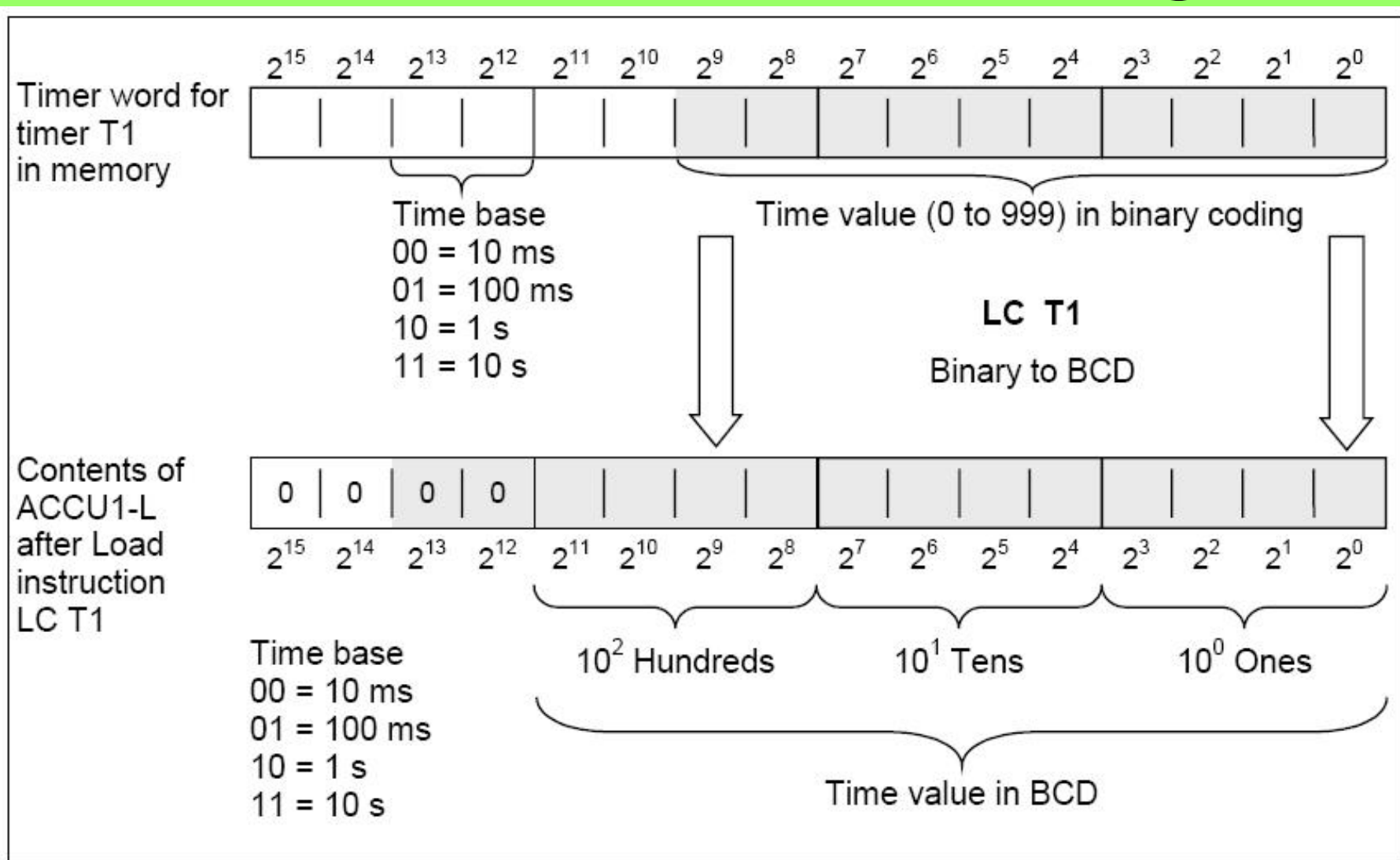
CPU موجود است به اکومولاتور بار نمی شود .

انجام دستورالعمل فوق تاثیری روی بیت های

Status Word ندارد

LC T1

مثال



R

Reset Timer

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

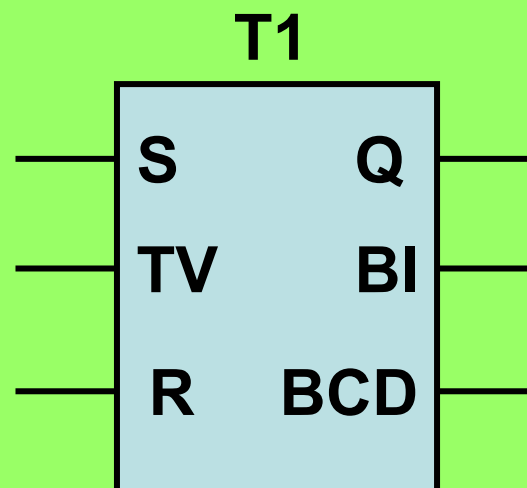
R <Timer>

اگر مقدار RLO از صفر به یک تغییر وضعیت دهد دستور فوق اجرا می شود .

با اجرای دستور فوق تایم گیری متوقف و مقدار زمان اندازه گیری شده را در حافظه CPU صفر می کند .

همانطور که قبلاً نیز گفته شد مقدار زمان تایمر ها در مکانی از حافظه که قبلاً به این منظور در نظر گرفته شده است ذخیره می گردد . نه در اکومولاتور ACCU1 لذا اجرای دستور فوق تاثیری روی اکومولاتور ندارد

قبل از نحوه معرفی تایمرها به نکات زیر توجه نمایید :
یک تایمر را به صورت یک بلوک مطابق شکل زیر در نظر بگیرید



اگر پایه **S** از صفر به یک تغییر وضعیت دهد تایمر شروع به کار می کند و مقدار زمان تنظیمی به صورت باینری داخل **BI** و به صورت **BCD** داخل **BCD** قرار خواهد گرفت

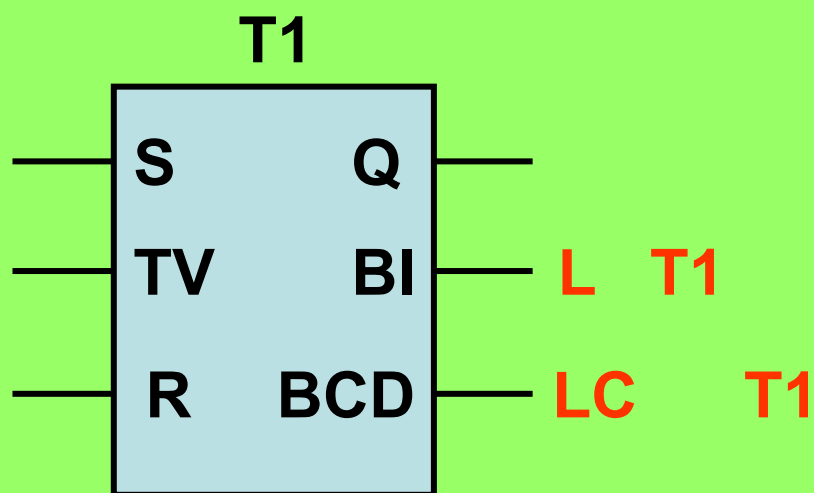
اگر پایه **R** از صفر به یک تغییر وضعیت دهد تایمر **Reset** می شود و مقدار زمان باقیمانده داخل **BI** و یا **BCD** صفر می شود

مقدار زمان تنظیمی به پایه **TV** مطابق فرمت گفته شده داده خواهد شد .

ادامه ←

وضعیت فعال بودن خروجی تایمر بستگی به وضعیت ورودی ها و نوع تایمر دارد .
مقدار زمان باقیمانده در هر لحظه به صورت باینری از پایه BI و به صورت BCD
از پایه BCD قابل دریافت است.

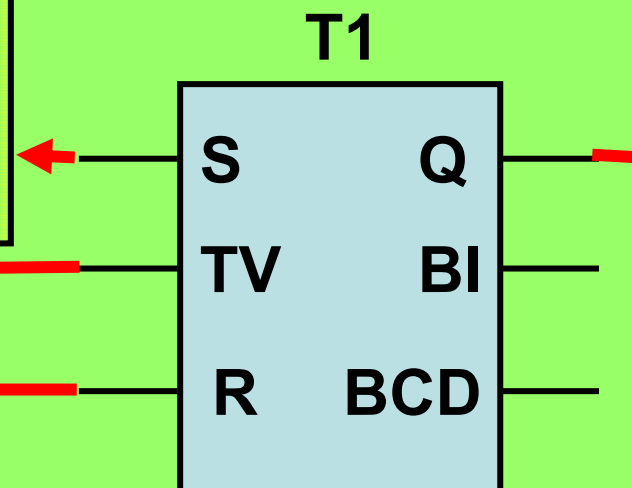
با توجه به موارد فوق در زیر بررسی میکنیم ببینیم نحوه معرفی تایمر و همچنین
نحوه فرمان به پایه ها در برنامه نویسی به زبان STL چگونه است



با این دستورها مقدار زمان باقیمانده تایمر به
فرم باینری و یا BCD وارد ACCU1-L
می شود و از اکومولاتور می توان به هر مکان
دلخواه از حافظه انتقال داد (با دستورهای
انتقال)

ادامه ←

FR T1 اگر قبل از دستور مقابل مقدار RLO از صفر به یک تغییر وضعیت دهد دستور فوق اجرا شده و تایمر را فعال می کند . اگر قبل از دستور مقابل مقدار **SP T1** RLO از صفر به یک تغییر وضعیت دهد دستور فوق اجرا شده و تایمر تایم گیری را شروع می کند.



این زمان **Preset Time Value** را تایمر از اکومولاتور شماره ۱ می خواند با دستور زیر می توانیم زمان مورد نظر را به اکومولاتور بار کنیم

L S5T#23S

R T1 اگر قبل از دستور فوق مقدار RLO از صفر به یک تغییر وضعیت دهد دستور فوق اجرا شده و تایمر را ری ست می کند .

با دستوراتی شبیه **T1 A** و یا دستورات مشابه می توان وضعیت خروجی تایمر را برای استفاده در قسمت های مختلف برنامه استفاده کرد

توصیه می شود به مثال های مربوط به

تایمرها در اسلایدهای بعدی بیشتر توجه کنید

SP

Pulse Timer

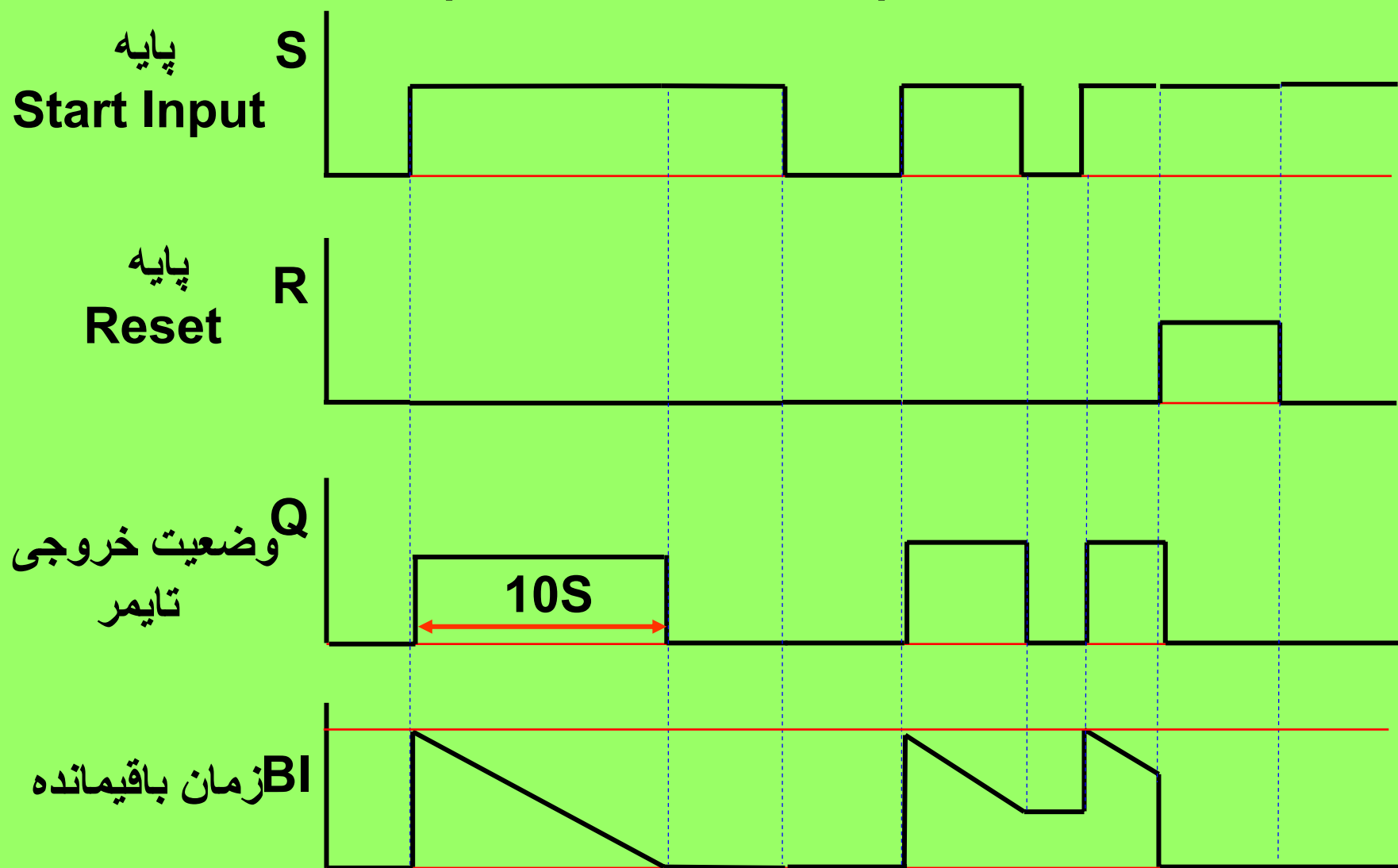
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SP <Timer>

مثال :	SP T5
--------	-------

وقتی RLO از صفر به یک تغییر وضعیت دهد تایمر آدرس داده شده با زمان تعیین شده راه اندازی می شود
اگر RLO از یک به صفر تغییر وضعیت دهد تایمر قطع می گردد هر چند زمان آن به پایان نرسیده باشد
برای پی بردن به طرز کار دقیق تایمر به عملکرد آن در اسلاید بعدی که به صورت انیمیشن است مراجعه کنید .
به تایمر فوق One Shot نیز می گویند .

عملکرد تایمر SP(Pulse Timer)



Example

STL	Explanation
A I 2.0	
FR T1	//Enable timer T1.
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	
L T1	//Load current time value of timer T1 as binary.
T MW10	
LC T1	//Load current time value of timer T1 as BCD.
T MW12	

SE Extended Pulse Timer

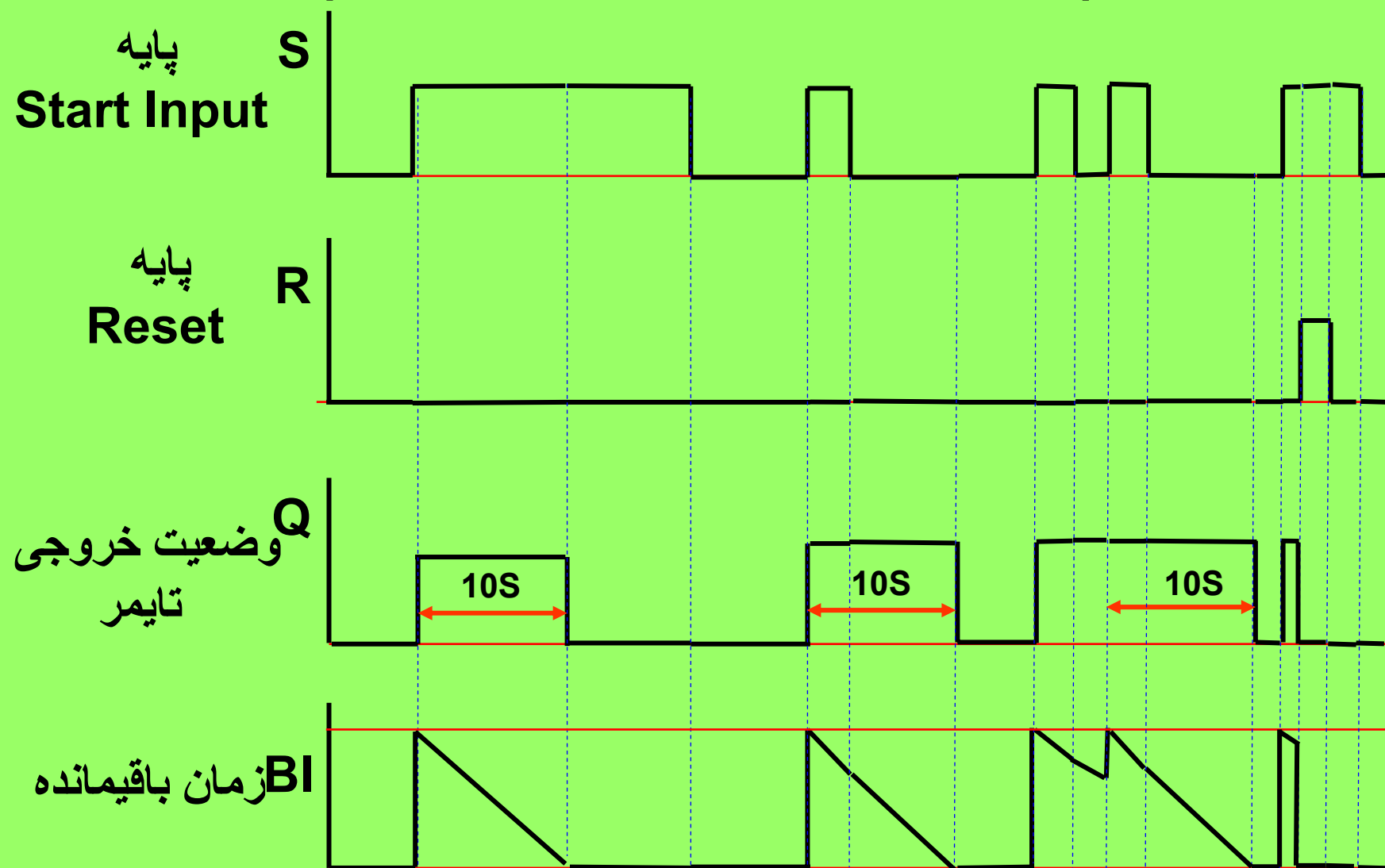
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SE <Timer>

مثال :	SE T5
--------	-------

وقتی RLO از صفر به یک تغییر وضعیت دهد تایمر آدرس داده شده با زمان تعیین شده راه اندازی می شود
اگر RLO از یک به صفر تغییر وضعیت دهد تایمر تا زمان تعیین شده برای آن به پایان نرسیده باشد قطع نمی گردد.
برای پی بردن به طرز کار دقیق تایمر به عملکرد آن در اسلاید بعدی که به صورت انیمیشن است مراجعه کنید .

عملکرد تایمر SE (Extended Pulse Timer)



Example

STL	Explanation
A I 2.0	
FR T1	//Enable timer T1.
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	
L T1	//Load current timer value of timer T1 as binary.
T MW10	
LC T1	//Load current timer value of timer T1 as BCD.
T MW12	

SD On - Delay Timer

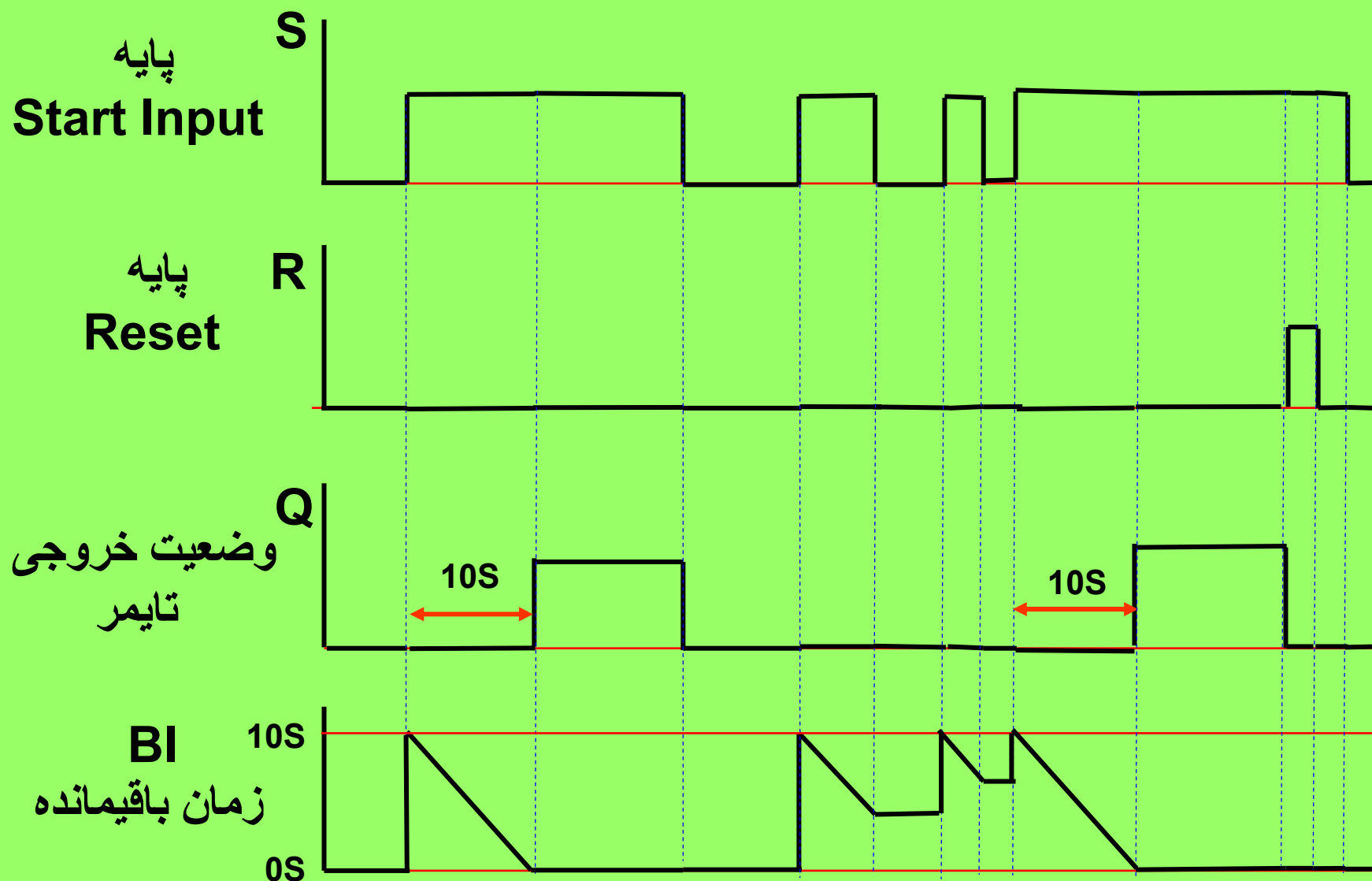
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SD <Timer>

مثال : SD T5

وقتی RLO از صفر به یک تغییر وضعیت دهد تایمر آدرس داده شده با زمان تعیین شده راه اندازی می شود
خروجی آن ابتدا صفر و پس از زمان تعیین شده به شرط این که ورودی هنوز یک باشد یک می شود و با صفر شدن خروجی صفر می شود .
برای پی بردن به طرز کار دقیق تایمر به عملکرد آن در اسلاید بعدی که به صورت انیمیشن است مراجعه کنید .

عملکرد تایمر SD (On Delay Timer)



Example

STL	Explanation
A I 2.0	
FR T1	//Enable timer T1.
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	
L T1	//Load current timer value of timer T1 as binary.
T MW10	
LC T1	//Load current timer value of timer T1 as BCD.
T MW12	

SS Retentive On - Delay Timer

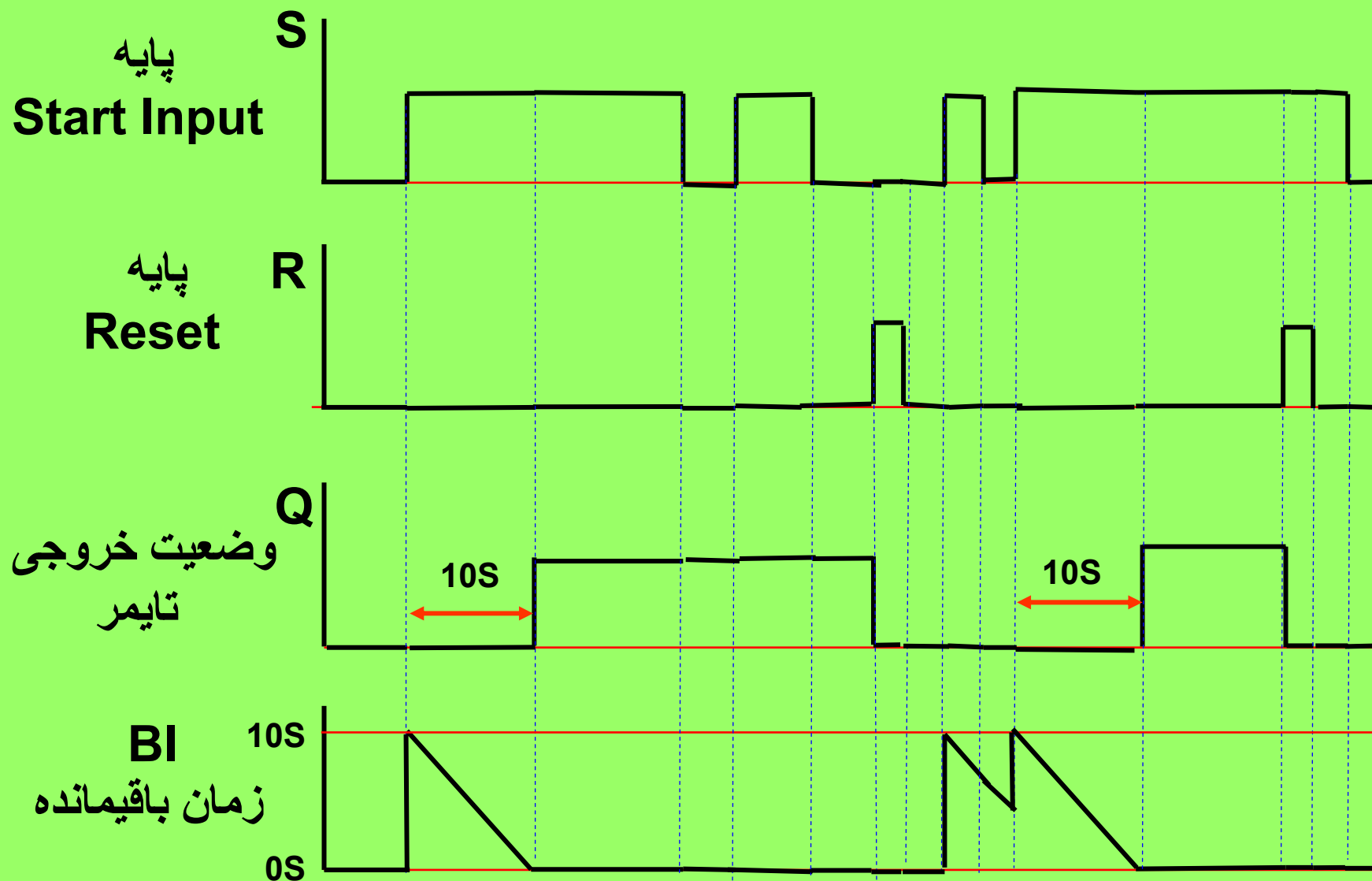
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SS <Timer>

مثال :	SS T5
--------	-------

وقتی RLO از صفر به یک تغییر وضعیت دهد تایمر آدرس داده شده با زمان تعیین شده راه اندازی می شود
خروجی آن ابتدا صفر و پس از زمان تعیین شده حتی اگر ورودی آن هم صفر باشد یک می شود و یک باقی می ماند و فقط با پایه ری ست تایمر قطع می شود
برای پی بردن به طرز کار دقیق تایمر به عملکرد آن در اسلاید بعدی که به صورت انیمیشن است مراجعه کنید .

عملکرد تایمر SS (Retentive On Delay Timer)



Example

STL	Explanation
A I 2.0	
FR T1	//Enable timer T1.
A I 2.1	
L S5T#10s	//Preset 10 seconds into ACCU 1.
SS T1	//Start timer T1 as a retentive on-delay timer.
A I 2.2	
R T1	//Reset timer T1.
A T1	//Check signal state of timer T1.
= Q 4.0	
L T1	//Load current time value of timer T1 as binary.
T MW10	
LC T1	//Load current time value of timer T1 as BCD.
T MW12	

SF Off - Delay Timer

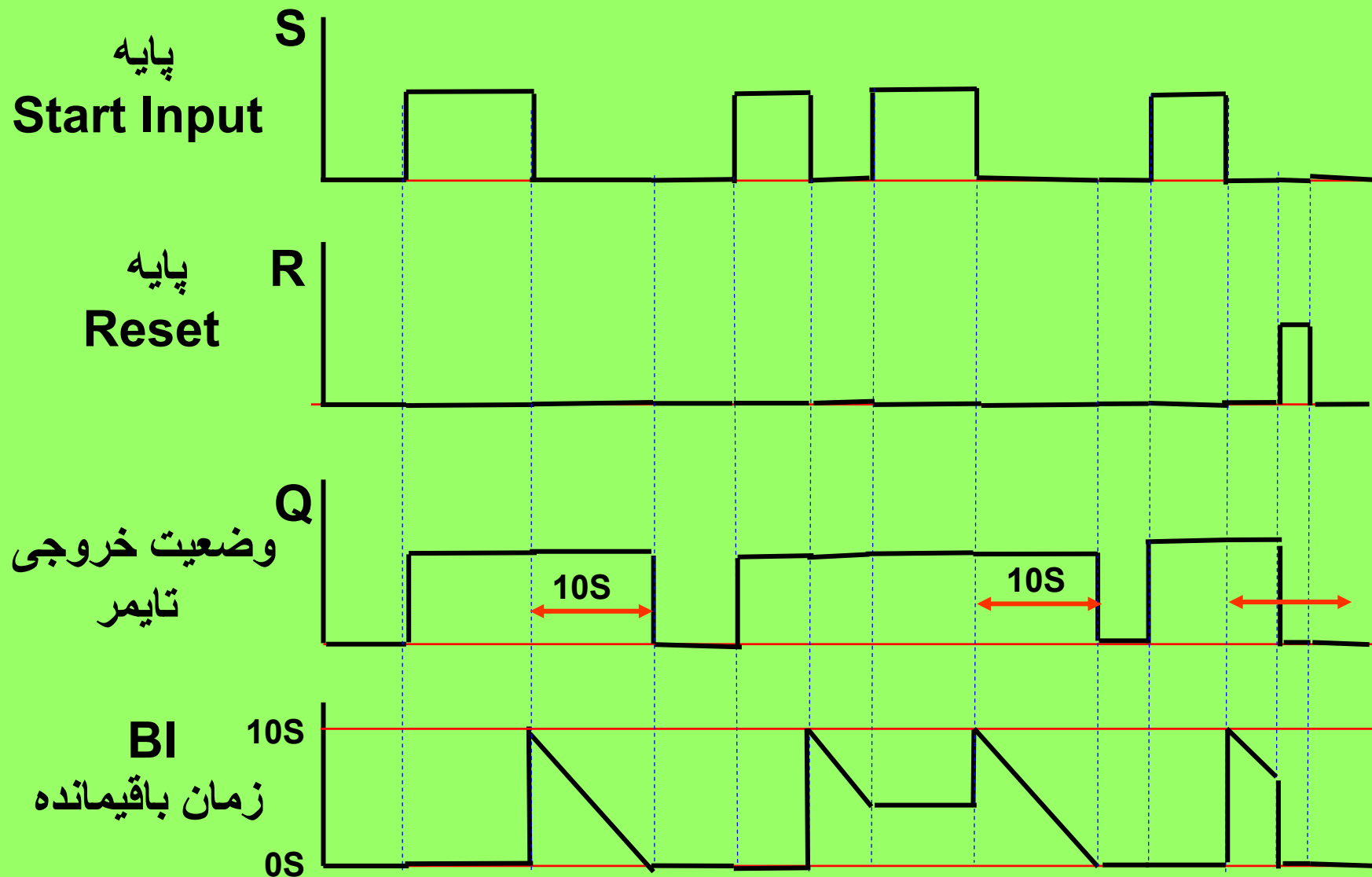
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

SF <Timer>

SF	T5	مثال :
----	----	--------

وقتی RLO از صفر به یک تغییر وضعیت دهد تایمر با آدرس داده شده با زمان تعیین شده راه اندازی می شود
تایم گیری آن از وقتی شروع می شود که ورودی آن صفر شود پس از آن به اندازه زمانی که تنظیم شده است در وضعیت یک باقی می ماند. (تاخیر در قطع)
برای پی بردن به طرز کار دقیق تایمر به عملکرد آن در اسلاید بعدی که به صورت انیمیشن است مراجعه کنید.

عملکرد تایمر SF (Off Delay Timer)



Example

STL	Explanation
A I 2.0	
FR T1	//Enable timer T1.
A I 2.1	
L S5T#10s	//Preset 10 seconds into ACCU 1.
SF T1	//Start timer T1 as an off-delay timer.
A I 2.2	
R T1	//Reset timer T1.
A T1	//Check signal state of timer T1.
= Q 4.0	
L T1	//Load current timer value of timer T1 as binary.
T MW10	
LC T1	//Load current timer value of timer T1 as BCD.
T MW12	

فصل سیزدهم

Word Logic

Instruction

مجموعه دستورالعمل های

اعمال منطقی روی داده ها

دستورات **Word Logic** یک جفت **Word** و یا **Dword** را بیت به بیت (بیت های متناظر) مطابق دستور ذکر شده به صورت منطق بولی با هم ترکیب می کند.

بدیهی است هر کدام از جفت های فوق باید در یکی از دو اکومولاتور موجود باشند برای **Word** فقط بیت های بخش **Low Word** اکومولاتورها با هم ترکیب می شوند و برای **Dword** تمام بیت های آکولاموتور با هم ترکیب می شوند. در هر دو حالت نتیجه عمل منطقی در اکومولاتور ۱ ذخیره شده و مقدار قبلی آن از بین می رود.

این دستورات بیت های **OV** و **CC0** از بیت های **Status Word** را صفر کرده ولی وضعیت بیت **CC1** بستگی به نتیجه عملیات دارد اگر نتیجه مخالف صفر باشد **CC1=1** و اگر نتیجه برابر صفر بود **CC1=0** می شود.

دستورالعمل های Word Logic به شرح زیر هستند :

AW	AND Word (16 bit)
OW	OR Word (16 bit)
XOW	Exclusive OR Word (16 bit)
AD	AND Double Word (32 bit)
OD	OR Double Word (32 bit)
XOD	Exclusive OR Double Word (32 bit)

جهت یادآوری

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

جدول صحت AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

جدول صحت OR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

جدول صحت OR انحصاري

AW

AND Word (16 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

AW

AW <Constant>

AW

مثال

با اجرای دستور **AW** محتویات ACCU1-L با ACCU2-L بیت به بیت (بیت های متناظر) با یک دیگر AND می شوند . و نتیجه AND در ACCU1-L ذخیره می گردد. آکومولاتورهای ACCU1-H و ACCU2 بدون تغییر باقی می مانند

دستور **<Constant> AW** محتوای **ACCU1-L** با یک عدد ثابت ۱۶ بیتی عمل منطقی **AND** را انجام داده و نتیجه را در **ACCU1-L** ذخیره می کند

Examples

Bit	15 ...	..	..	... 0
ACCU 1-L before execution of AW	0101	1001	0011	1011
ACCU 2-L or 16-bit constant:	1111	0110	1011	0101
Result (ACCU 1-L) after execution of AW	0101	0000	0011	0001

Example 1

STL	Explanation
L IW20	//Load contents of IW20 into ACCU 1-L.
L IW22	//Load contents of ACCU 1 into ACCU 2. Load contents of IW22 into ACCU 1-L.
AW	//Combine bits from ACCU 1-L with ACCU 2-L bits by AND; store result in ACCU 1-L.
T MW 8	//Transfer result to MW8.

Example 2

STL	Explanation
L IW20	//Load contents of IW20 into ACCU 1-L.
AW W#16#0FFF	//Combine bits of ACCU 1-L with bit pattern of 16-bit constant (0000_1111_1111_1111) by AND; store result in ACCU 1-L.
JP NEXT	//Jump to NEXT jump label if result is unequal to zero, (CC 1 = 1).

OW

OR Word (16 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

OW

OW <Constant>

OW

مثال :

با اجرای دستور **OW** محتویات ACCU1-L با ACCU2-L بیت به بیت (بیت های متناظر) با یک دیگر OR می شوند . و نتیجه OR در ACCU1-L ذخیره می گردد. آکومولاتورهای ACCU1-H و ACCU2 بدون تغییر باقی می مانند

دستور **<Constant> OW** محتوای **ACCU1-L** با یک عدد ثابت ۱۶ بیتی عمل منطقی **OR** را انجام داده و نتیجه را در **ACCU1-L** ذخیره می کند

Examples

Bit	15 ...	..	..	... 0
ACCU 1-L before execution of OW	0101	0101	0011	1011
ACCU 2-L or 16 bit constant:	1111	0110	1011	0101
Result (ACCU 1-L) after execution of OW	1111	0111	1011	1111

Example 1

STL	Explanation
L IW20	//Load contents of IW20 into ACCU 1-L.
L IW22	//Load contents of ACCU 1 into ACCU 2. Load contents of IW22 into ACCU 1-L.
OW	//Combine bits from ACCU 1-L with ACCU 2-L by OR, store result in ACCU 1-L.
T MW8	//Transfer result to MW8.

Example 2

STL	Explanation
L IW20	//Load contents of IW 20 into ACCU 1-L.
OW W#16#0FFF	//Combine bits of ACCU 1-L with bit pattern of 16-bit constant (0000_1111_1111_1111) by OR; store result in ACCU 1-L.
JP NEXT	//Jump to NEXT jump label if result is unequal to zero (CC 1 = 1).

XOW

Exclusive OR Word (16 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

OWX

XOW <Constant>

XOW

مثال :

با اجرای دستور **XOW** محتویات ACCU1-L با ACCU2-L
بیت به بیت (بیت های متناظر) با یک دیگر XOR می شوند . و نتیجه
XOR در ACCU1-L ذخیره می گردد. آکومولاتورهای ACCU1-H و
ACCU2 بدون تغییر باقی می مانند

دستور **<Constant> XOW** محتوای **ACCU1-L** با یک عدد ثابت ۱۶
بیتی عمل منطقی XOR را انجام داده و نتیجه را در **ACCU1-L** ذخیره می کند

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

Example 1

STL	Explanation
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.

Example 2

STL	Explanation
L IW20	//Load contents of IW20 into ACCU 1-L.
XOW 16#0FFF	//Combine bits of ACCU 1-L with bit pattern of 16-bit constant (0000_1111_1111_1111) by XOR, store result in ACCU 1-L.
JP NEXT	//Jump to NEXT jump label if result is unequal to zero, (CC 1 = 1).

AD

AND Double Word (32 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

AD

AD <Constant>

AD

مثال :

با اجرای دستور **AD** محتویات ACCU1 با ACCU2 بیت به بیت (بیت های متناظر) با یک دیگر AND می شوند . و نتیجه AND در ACCU1 ذخیره می گردد. آکومولاتور ACCU2 بدون تغییر باقی می ماند

دستور **<Constant> AD** محتوای **ACCU1** با یک عدد ثابت **۳۲** بیتی عمل منطقی **AND** را انجام داده و نتیجه را در **ACCU1** ذخیره می کند

Examples

Bit	31 ..	..	..	..	..	..	..	... 0
ACCU 1 before execution of UD	0101	0000	1111	1100	1000	1001	0011	1011
ACCU 2 or 32-bit constante	1111	0011	1000	0101	0111	0110	1011	0101
Result (ACCU 1) after execution of UD	0101	0000	1000	0100	0000	0000	0011	0001

Example 1

STL	Explanation	
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.	
AD	//Combine bits from ACCU 1 with ACCU 2 by AND, store result in ACCU 1.	
T MD8	//Transfer result to MD8.	

Example 2

STL	Explanation	
L ID 20	//Load contents of ID20 into ACCU 1.	
AD DW#16#0FFF_EF21	//Combine bits of ACCU 1 with bit pattern of 32-bit constant (0000_1111_1111_1111_1110_1111_0010_0001) by AND; store result in ACCU 1.	
JP NEXT	//Jump to NEXT jump label if result is unequal to zero, (CC 1 = 1) .	

OD

OR Double Word (32 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

OD

OD <Constant>

OD

مثال :

با اجرای دستور **OD** محتویات ACCU1 با ACCU2 بیت به بیت (بیت های متناظر) با یک دیگر OR می شوند . و نتیجه OR در ACCU1 ذخیره می گردد. آکومولاتور ACCU2 بدون تغییر باقی می ماند

دستور **<Constant> OD محتوای ACCU1 با یک عدد ثابت ۳۲** **بیتی عمل منطقی OR را انجام داده و نتیجه را در ACCU1 ذخیره می کند**

Examples

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

Example 1

STL	Explanation
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.

Example 2

STL	Explanation
L ID20	//Load contents of ID20 into ACCU 1.
OD DW#16#0FFF_EF21	//Combine bits of ACCU 1 with bit pattern of 32-bit constant (0000_1111_1111_1111_1110_1111_0010_0001) by OR, store result in ACCU 1.
JP NEXT	//Jump to NEXT jump label if result is not equal to zero, (CC 1 = 1) .

XOD

XOR Double Word (32 bit)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

XOD

XOD <Constant>

XOD

مثال :

با اجرای دستور **XOD** محتویات ACCU1 با ACCU2 بیت به

بیت (بیت های متناظر) با یک دیگر XOR می شوند . و نتیجه XOR در ACCU1 ذخیره می گردد. آکومولاتور ACCU2 بدون تغییر باقی می ماند
دستور **<Constant> XOD محتوای ACCU1 با یک عدد ثابت ۳۲**
بیتی عمل منطقی XOR را انجام داده و نتیجه را در ACCU1 ذخیره می کند

Examples

Bit	31 ..	..	..	..	..	..	..	... 0
ACCU 1 before execution of XOD	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 XOD	1010	0011	0111	1001	1111	0011	1000	1110

Example 1

STL	Explanation
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.
XOD	//Combine bits from ACCU 1 with ACCU 2 by XOR; store result in ACCU 1.
T MD8	//Transfer result to MD8.

Example 2

STL	Explanation
L ID20	//Load contents of ID20 into ACCU 1.
XOD DW#16#0FFF_EF21	//Combine bits from ACCU 1 with bit pattern of 32-bit constant (0000_1111_1111_1111_1111_1110_0010_0001) by XOR, store result in ACCU 1.
JP NEXT	//Jump to NEXT jump label if result is unequal to zero, (CC 1 = 1).

فصل چهاردهم

Accumulator Instruction

مجموعه دستورالعمل های

اکومولاتوری

- اکثر CPU های S7 دارای دو اکومولاتور و بعضی دارای چهار اکومولاتور هستند

- هر اکومولاتور ۳۲ بیتی است .

- ساختار هر اکومولاتور مانند شکل زیر است .

ساختار داخلی یک اکومولاتور در S7

ACCU1			
ACCU1-H		ACCU1-L	
ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
HIGH WORD HIGH BYTE	HIGH WORD LOW BYTE	LOW WORD HIGH BYTE	LOW WORD LOW BYTE
31 24	23 16	15 8	7 0

دستورات اکومولاتوری به شرح زیر هستند :

TAK	Toggle ACCU1 with ACCU2
PUSH	CPU with Two ACCUs
PUSH	CPU with Four ACCUs
POP	CPU with Two ACCUs
POP	CPU with Four ACCUs
ENT	Enter ACCU Stack
LEAVE	Leave ACCU Stack
INC	Increment ACCU1-L-L
DEC	Decrement ACCU1-L-L
+AR1	Add ACCU1 to Address Register1
+AR2	Add ACCU1 to Address Register2
BLD	Program Display Instructio (Null)
NOP 0	Null Instruction
NOP 1	Null Instruction

TAK

Toggle ACCU1 with ACCU2

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

TAK

TAK

مثال :

با اجرای دستور **TAK** محتوای ACCU1 با ACCU2 جابجا می شوند

در CPU هایی که دارای چهار اکومولاتور دارند اکومولاتور ۳ و ۴ تغییری نمی کنند

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال – برنامه زیر همیشه مقدار کوچکتر را از مقدار بزرگتر کم می کند :

Example: Subtract smaller value from greater value

STL	Explanation		
L	MW10	//Load contents of MW10 into ACCU 1-L.	
L	MW12	//Load contents of ACCU 1-L into ACCU 2-L. Load contents of MW12 into ACCU 1-L.	
>I		//Check if ACCU 2-L (MW10) greater than ACCU 1-L (MW12) .	
SPB	NEXT	//Jump to NEXT jump label if ACCU 2 (MW10) is greater than ACCU 1 (MW12) .	
	TAK	//Swap contents ACCU 1 and ACCU 2	
NEXT:	-I	//Subtract contents of ACCU 2-L from contents of ACCU 1-L.	
	T	MW14	//Transfer result (= greater value minus smaller value) to MW14.

Contents	ACCU 1	ACCU 2
before executing TAK instruction	<MW12>	<MW10>
after executing TAK instruction	<MW10>	<MW12>

POP CPU With Two ACCUs

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

POP

POP

مثال :

با اجرای دستور **POP** محتوای ACCU2 در ACCU1 کپی می شود

و مقدار ACCU2 بدون تغییر می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد



مثال

Example

STL	Explanation
T MD10	//Transfer contents of ACCU 1 (= value A) to MD10
POP	//Copy entire contents of ACCU 2 to ACCU 1
T MD14	//Transfer contents of ACCU 1 (= value B) to MD14

Contents	ACCU 1	ACCU 2
before executing POP instruction	value A	value B
after executing POP instruction	value B	value B

POP

CPU With Four ACCUs

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

POP

POP

مثال :

با اجرای دستور **POP** محتوای ACCU2 در ACCU1 کپی می شود
و محتوای ACCU3 در ACCU2 و محتوای ACCU4 در ACCU3 کپی می شود
و مقدار ACCU4 بدون تغییر می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

Example

STL	Explanation
T MD10	//Transfer contents of ACCU 1 (= value A) to MD10
POP	//Copy entire contents of ACCU 2 to ACCU 1
T MD14	//Transfer contents of ACCU 1 (= value B) to MD14

Contents	ACCU 1	ACCU 2	ACCU 3	ACCU 4
before executing POP instruction	value A	value B	value C	value D
after executing POP instruction	value B	value C	value D	value D

PUSH

CPU With Two ACCUs

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

PUSH

PUSH

مثال :

با اجرای دستور **PUSH** محتوای ACCU1 در ACCU2 کپی می شود
و مقدار ACCU1 بدون تغییر می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

Example

STL	Explanation
L MW10	//Load the contents of MW10 into ACCU 1.
PUSH	//Copy entire contents of ACCU 1 into ACCU 2.

Contents	ACCU 1	ACCU 2
before executing PUSH instruction	<MW10>	<X>
after executing PUSH instruction	<MW10>	<MW10>

PUSH

CPU With Four ACCUs

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

PUSH

PUSH

مثال :

با اجرای دستور **PUSH** محتوای ACCU1 در ACCU2 کپی می شود
و محتوای ACCU2 در ACCU3 و محتوای ACCU3 در ACCU4 کپی می شود
و مقدار ACCU1 بدون تغییر می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

Example

STL	Explanation
L MW10	//Load the contents of MW10 into ACCU 1.
PUSH	//Copy the entire contents of ACCU 1 to ACCU 2, the contents of ACCU 2 to ACCU 3, and the contents of ACCU 3 to ACCU 4.

Contents	ACCU 1	ACCU 2	ACCU 3	ACCU 4
before executing PUSH instruction	value A	value B	value C	value D
after executing PUSH instruction	value A	value A	value B	value C

ENT Enter ACCU Stack

این دستور مخصوص **CPU** هایی است که دارای ۴ اکومولاتور هستند
فرمت استفاده از دستور فوق در برنامه نویسی **STL** به صورت زیر است :

ENT

ENT

مثال :

با اجرای دستور **ENT** محتوای ACCU3 در ACCU4

و محتوای ACCU2 در ACCU3 کپی می شود

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

در مورد کاربرد این دستور به اسلاید بعدی با دقت توجه کنید

مثال

Example

STL	Explanation
L DBD0	//Load the value from data double word DBD0 into ACCU 1. (This value must be in the floating point format).
L DBD4	//Copy the value from ACCU 1 into ACCU 2. Load the value from data double word DBD4 into ACCU 1. (This value must be in the floating point format).
+R	//Add the contents of ACCU 1 and ACCU 2 as floating point numbers (32 bit, IEEE-FP) and save the result in ACCU 1.
L DBD8	//Copy the value from ACCU 1 into ACCU 2 load the value from data double word DBD8 into ACCU 1.
ENT	//Copy the contents of ACCU 3 into ACCU 4. Copy the contents of ACCU 2 (intermediate result) into ACCU 3.
L DBD12	//Load the value from data double word DBD12 into ACCU 1.
-R	//Subtract the contents of ACCU 1 from the contents of ACCU 2 and store the result in ACCU 1. Copy the contents of ACCU 3 into ACCU 2. Copy the contents of ACCU 4 into ACCU 3.
/R	//Divide the contents of ACCU 2 (DBD0 + DBD4) by the contents of ACCU 1 (DBD8 - DBD12). Save the result in ACCU 1.
T DBD16	//Transfer the results (ACCU 1) to data double word DBD16.

LEAVE

Leave ACCU Stack

این دستور مخصوص **CPU** هایی است که دارای ۴ اکومولاتور هستند
فرمت استفاده از دستور فوق در برنامه نویسی **STL** به صورت زیر است :

LEAVE

مثال :
LEAVE

با اجرای دستور **LEAVE** محتوای ACCU3 در ACCU2
و محتوای ACCU4 در ACCU3 کپی می شود

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

INC

Increment ACCU1-L-L

این دستور فقط برای مقادیر ۸ بیتی به کار می رود
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

INC <8- bit Integer>

مثال :	INC 1
--------	-------

با اجرای دستور **INC** به مقدار ACCU1-L-L یکی اضافه می شود و
نتیجه مجددا در ACCU1-L-L ذخیره می گردد .
و محتوای ACCU1-L-H و ACCU2 بدون تغییر باقی می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

L	MB22
INC	1
T	MB22

DEC

Decrement ACCU1-L-L

این دستور فقط برای مقادیر ۸ بیتی به کار می رود
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

DEC <8- bit Integer>

مثال : **DEC 1**

با اجرای دستور **DEC** از مقدار ACCU1-L-L یکی کم می شود و
نتیجه مجددا در ACCU1-L-L ذخیره می گردد .
و محتوای ACCU1-L-H و ACCU2 بدون تغییر باقی می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مثال

L	MB22
DEC	1
T	MB22

DEC

Decrement ACCU1-L-L

این دستور فقط برای مقادیر ۸ بیتی به کار می رود
فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

DEC <8- bit Integer>

مثال :
DEC 1

با اجرای دستور **DEC** از مقدار ACCU1-L-L یکی کم می شود و
نتیجه مجددا در ACCU1-L-L ذخیره می گردد .
و محتوای ACCU1-L-H و ACCU2 بدون تغییر باقی می ماند .

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

+AR1

Add accu1 to Address Register1

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+AR1

+AR1 <P#Byte.Bit>

مثال : P#300.0 +AR1 +AR1

با اجرای دستور **+AR1** محتوای ACCU1-L (مقدار صحیح ۱۶ بیتی) به محتوای AR1 اضافه می شود.
+AR1<P#Byte.Bit> مقدار آفستی که باید با AR1 جمع شود را مشخص می نماید

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مقادیر مجاز ۳۲۷۶۷ تا ۳۲۷۶۸-

L +300
+AR1

مثال ۱

+AR1 P300.0

مثال ۲

+AR2

Add accu1 to Address Register2

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

+AR2

+AR2 <P#Byte.Bit>

مثال : P#300.0 +AR2 +AR2

با اجرای دستور **+AR2** محتوای ACCU1-L (مقدار صحیح ۱۶ بیتی) به محتوای AR2 اضافه می شود.
+AR2<P#Byte.Bit> مقدار آفستی که باید با AR2 جمع شود را مشخص می نماید

انجام دستور العمل فوق تاثیری روی **RLO** و
Status Word ندارد

مقادیر مجاز ۳۲۷۶۷ تا ۳۲۷۶۸-

L +300
+AR2

مثال ۱

+AR2 P300.0

مثال ۲

BLD

Program Display Instruction (Null)

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

BLD <NUMBER>

BLD 23

مثال :

دستور **BLD** کار خاصی انجام نمی دهد و فقط برای نمایش توسط PG به کار می رود .

وقتی برنامه LAD یا FBD به صورت STL نمایش داده می شود این دستور به طور اتوماتیک با عددهای تعیین شده توسط سیستم (بین BLD0 تا BLD255) روی صفحه نمایش داده می شود .

NOP0

Null Instruction

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

NOP0

NOP0

مثال :

دستور **NOP0** کار خاصی انجام نمی دهد و در برنامه نویسی در

صورت لزوم به کار می رود .

این دستور دارای Bit Pattern با شانزده صفر است .

NOP1

Null Instruction

فرمت استفاده از دستور فوق در برنامه نویسی STL به صورت زیر است :

NOP1

NOP1

مثال :

دستور **NOP1** کار خاصی انجام نمی دهد و در برنامه نویسی در

صورت لزوم به کار می رود .

این دستور دارای Bit Pattern با شانزده یک است .

ضمیمہ ۱

لیست کامل دستورات

STL

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
+	+	Integer math Instruction	Add Integer Constant (16, 32-bit)
=	=	Bit logic Instruction	Assign
)	)	Bit logic Instruction	Nesting Closed
+AR1	+AR1	Accumulator	AR1 Add ACCU 1 to Address Register 1
+AR2	+AR2	Accumulator	AR2 Add ACCU 1 to Address Register 2
+D	+D	Integer math Instruction	Add ACCU 1 and ACCU 2 as Double Integer (32-bit)
-D	-D	Integer math Instruction	Subtract ACCU 1 from ACCU 2 as Double Integer (32-bit)
*D	*D	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Double Integer (32-bit)
/D	/D	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Double Integer (32-bit)
? D	? D	Compare	Compare Double Integer (32-bit) ==, <>, >, <, >=, <=
+I	+I	Integer math Instruction	Add ACCU 1 and ACCU 2 as Integer (16-bit)
-I	-I	Integer math Instruction	Subtract ACCU 1 from ACCU 2 as Integer (16-bit)

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
*I	*I	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Integer (16-bit)
/I	/I	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Integer (16-bit)
? I	? I	Compare	Compare Integer (16-bit) ==, <>, >, <, >=, <=
+R	+R	Floating point Instruction	Add ACCU 1 and ACCU 2 as a Floating-point Number (32-bit IEEE-FP)
-R	-R	Floating point Instruction	Subtract ACCU 1 from ACCU 2 as a Floating-point Number (32-bit IEEE-FP)
*R	*R	Floating point Instruction	Multiply ACCU 1 and ACCU 2 as Floating-point Numbers (32-bit IEEE-FP)
/R	/R	Floating point Instruction	Divide ACCU 2 by ACCU 1 as a Floating-point Number (32-bit IEEE-FP)
? R	? R	Compare	Compare Floating-point Number (32-bit) ==, <>, >, <, >=, <=
A	U	Bit logic Instruction	And
A(	U(	Bit logic Instruction	And with Nesting Open
ABS	ABS	Floating point Instruction	Absolute Value of a Floating-point Number (32-bit IEEE-FP)
ACOS	ACOS	Floating point Instruction	Generate the Arc Cosine of a Floating-point Number (32-bit)

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
AD	UD	Word logic Instruction	AND Double Word (32-bit)
AN	UN	Bit logic Instruction	And Not
AN(	UN(	Bit logic Instruction	And Not with Nesting Open
ASIN	ASIN	Floating point Instruction	Generate the Arc Sine of a Floating-point Number (32-bit)
ATAN	ATAN	Floating point Instruction	Generate the Arc Tangent of a Floating-point Number (32-bit)
AW	UW	Word logic Instruction	AND Word (16-bit)
BE	BE	Program control	Block End
BEC	BEB	Program control	Block End Conditional
BEU	BEA	Program control	Block End Unconditional
BLD	BLD	Program control	Program Display Instruction (Null)
BTB	BTB	Convert	BCD to Integer (32-bit)
BTI	BTI	Convert	BCD to Integer (16-bit)
CAD	TAD	Convert	Change Byte Sequence in ACCU 1 (32-bit)
CALL	CALL	Program control	Block Call
CALL	CALL	Program control	Call Multiple Instance
CALL	CALL	Program control	Call Block from a Library

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
CAR	TAR	Load/Transfer	Exchange Address Register 1 with Address Register 2
CAW	TAW	Convert	Change Byte Sequence in ACCU 1-L (16-bit)
CC	CC	Program control	Conditional Call
CD	ZR	Counters	Counter Down
CDB	TDB	Convert	Exchange Shared DB and Instance DB
CLR	CLR	Bit logic Instruction	Clear RLO (=0)
COS	COS	Floating point Instruction	Generate the Cosine of Angles as Floating-point Numbers (32-bit)
CU	ZV	Counters	Counter Up
DEC	DEC	Accumulator	Decrement ACCU 1-L-L
DTB	DTB	Convert	Double Integer (32-bit) to BCD
DTR	DTR	Convert	Double Integer (32-bit) to Floating-point (32-bit IEEE-FP)
ENT	ENT	Accumulator	Enter ACCU Stack
EXP	EXP	Floating point Instruction	Generate the Exponential Value of a Floating-point Number (32-bit)
FN	FN	Bit logic Instruction	Edge Negative
FP	FP	Bit logic Instruction	Edge Positive

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
FR	FR	Counters	Enable Counter (Free) (free, FR C 0 to C 255)
FR	FR	Timers	Enable Timer (Free)
INC	INC	Accumulator	Increment ACCU 1-L-L
INVD	INVD	Convert	Ones Complement Double Integer (32-bit)
INVI	INVI	Convert	Ones Complement Integer (16-bit)
ITB	ITB	Convert	Integer (16-bit) to BCD
ITD	ITD	Convert	Integer (16-bit) to Double Integer (32-bit)
JBI	SPBI	Jumps	Jump if BR = 1
JC	SPB	Jumps	Jump if RLO = 1
JCB	SPBB	Jumps	Jump if RLO = 1 with BR
JCN	SPBN	Jumps	Jump if RLO = 0
JL	SPL	Jumps	Jump to Labels
JM	SPM	Jumps	Jump if Minus
JMZ	SPMZ	Jumps	Jump if Minus or Zero
JN	SPN	Jumps	Jump if Not Zero
JNB	SPBNB	Jumps	Jump if RLO = 0 with BR
JNBI	SPBIN	Jumps	Jump if BR = 0
JO	SPO	Jumps	Jump if OV = 1

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
JOS	SPS	Jumps	Jump if OS = 1
JP	SPP	Jumps	Jump if Plus
JPZ	SPPZ	Jumps	Jump if Plus or Zero
JU	SPA	Jumps	Jump Unconditional
JUO	SPU	Jumps	Jump if Unordered
JZ	SPZ	Jumps	Jump if Zero
L	L	Load/Transfer	Load
L DBLG	L DBLG	Load/Transfer	Load Length of Shared DB in ACCU 1
L DBNO	L DBNO	Load/Transfer	Load Number of Shared DB in ACCU 1
L DILG	L DILG	Load/Transfer	Load Length of Instance DB in ACCU 1
L DINO	L DINO	Load/Transfer	Load Number of Instance DB in ACCU 1
L STW	L STW	Load/Transfer	Load Status Word into ACCU 1
L	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	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	LAR1	Load/Transfer	Load Address Register 1 from ACCU 1

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
LAR1 <D>	LAR1<D>	Load/Transfer	Load Address Register 1 with Double Integer (32-bit Pointer)
LAR1 AR2	LAR1 AR2	Load/Transfer	Load Address Register 1 from Address Register 2
LAR2	LAR2	Load/Transfer	Load Address Register 2 from ACCU 1
LAR2 <D>	LAR2 <D>	Load/Transfer	Load Address Register 2 with Double Integer (32-bit Pointer)
LC	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	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	LEAVE	Accumulator	Leave ACCU Stack
LN	LN	Floating point Instruction	Generate the Natural Logarithm of a Floating-point Number (32-bit)
LOOP	LOOP	Jumps	Loop
MCR(	MCR(	Program control	Save RLO in MCR Stack, Begin MCR
)MCR	)MCR	Program control	End MCR
MCRA	MCRA	Program control	Activate MCR Area
MCRD	MCRD	Program control	Deactivate MCR Area

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
MOD	MOD	Integer math Instruction	Division Remainder Double Integer (32-bit)
NEGD	NEGD	Convert	Twos Complement Double Integer (32-bit)
NEGI	NEGI	Convert	Twos Complement Integer (16-bit)
NEGR	NEGR	Convert	Negate Floating-point Number (32-bit, IEEE-FP)
NOP 0	NOP 0	Accumulator	Null Instruction
NOP 1	NOP 1	Accumulator	Null Instruction
NOT	NOT	Bit logic Instruction	Negate RLO
O	O	Bit logic Instruction	Or
O(	O(	Bit logic Instruction	Or with Nesting Open
OD	OD	Word logic Instruction	OR Double Word (32-bit)
ON	ON	Bit logic Instruction	Or Not
ON(	ON(	Bit logic Instruction	Or Not with Nesting Open
OPN	AUF	DB call	Open a Data Block
OW	OW	Word logic Instruction	OR Word (16-bit)
POP	POP	Accumulator	POP
POP	POP	Accumulator	CPU with Two ACCUs
POP	POP	Accumulator	CPU with Four ACCUs

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
PUSH	PUSH	Accumulator	CPU with Two ACCUs
PUSH	PUSH	Accumulator	CPU with Four ACCUs
R	R	Bit logic Instruction	Reset
R	R	Counters	Reset Counter (the current counter can be a number from 0 to 255, for example, R C 15)
R	R	Timers	Reset Timer (the current timer can be a number from 0 to 255, for example, R T 32)
RLD	RLD	Shift/Rotate	Rotate Left Double Word (32-bit)
RLDA	RLDA	Shift/Rotate	Rotate ACCU 1 Left via CC 1 (32-bit)
RND	RND	Convert	Round
RND-	RND-	Convert	Round to Lower Double Integer
RND+	RND+	Convert	Round to Upper Double Integer
RRD	RRD	Shift/Rotate	Rotate Right Double Word (32-bit)
RRDA	RRDA	Shift/Rotate	Rotate ACCU 1 Right via CC 1 (32-bit)
S	S	Bit logic Instruction	Set
S	S	Counters	Set Counter Preset Value (the current counter can be a number from 0 to 255, for example, S C 15)
SAVE	SAVE	Bit logic Instruction	Save RLO in BR Register
SD	SE	Timers	On-Delay Timer
SE	SV	Timers	Extended Pulse Timer

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
SET	SET	Bit logic Instruction	Set
SF	SA	Timers	Off-Delay Timer
SIN	SIN	Floating point Instruction	Generate the Sine of Angles as Floating-point Numbers (32-bit)
SLD	SLD	Shift/Rotate	Shift Left Double Word (32-bit)
SLW	SLW	Shift/Rotate	Shift Left Word (16-bit)
SP	SI	Timers	Pulse Timer
SQR	SQR	Floating point Instruction	Generate the Square of a Floating-point Number (32-bit)
SQRT	SQRT	Floating point Instruction	Generate the Square Root of a Floating-point Number (32-bit)
SRD	SRD	Shift/Rotate	Shift Right Double Word (32-bit)
SRW	SRW	Shift/Rotate	Shift Right Word (16-bit)
SS	SS	Timers	Retentive On-Delay Timer
SSD	SSD	Shift/Rotate	Shift Sign Double Integer (32-bit)
SSI	SSI	Shift/Rotate	Shift Sign Integer (16-bit)
T	T	Load/Transfer	Transfer
T STW	T STW	Load/Transfer	Transfer ACCU 1 into Status Word
TAK	TAK	Accumulator	Toggle ACCU 1 with ACCU 2

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
TAN	TAN	Floating point Instruction	Generate the Tangent of Angles as Floating-point Numbers (32-bit)
TAR1	TAR1	Load/Transfer	Transfer Address Register 1 to ACCU 1
TAR1	TAR1	Load/Transfer	Transfer Address Register 1 to Destination (32-bit Pointer)
TAR1	TAR1	Load/Transfer	Transfer Address Register 1 to Address Register 2
TAR2	TAR2	Load/Transfer	Transfer Address Register 2 to ACCU 1
TAR2	TAR2	Load/Transfer	Transfer Address Register 2 to Destination (32-bit Pointer)
TRUNC	TRUNC	Convert	Truncate
UC	UC	Program control	Unconditional Call
X	X	Bit logic Instruction	Exclusive Or
X(	X(	Bit logic Instruction	Exclusive Or with Nesting Open
XN	XN	Bit logic Instruction	Exclusive Or Not
XN(	XN(	Bit logic Instruction	Exclusive Or Not with Nesting Open
XOD	XOD	Word logic Instruction	Exclusive OR Double Word (32-bit)
XOW	XOW	Word logic Instruction	Exclusive OR Word (16-bit)

