فصل سوم

مقدمه‌ای بر زبان STEP 5

یک برنامهٔ کنترلی مجموعهٔ دستورالعمل‌هایی است که به سیستم PLC فرمانهایی جهت کنترل پروسه صادر می‌کند. بنابراین باید این برنامه به زبانی خاص و مطابق با قوانین و دستورات قابل درک برای PLC نوشته شود. زبان برنامه‌نویسی که خانوادهٔ SIEMENS از آن استفاده می‌کند STEP 5 (S5) نامیده می‌شود.

۳-۱- اشکال مختلف نمایش برنامه‌ها

در زبان برنامه‌نویسی S5 برنامه‌ها را می‌توان به صورتهای زیر نوشت:[1]

۱- نردبانی (Ladder) LAD

۲- فلوچارتی (Control System Flowchart) CSF

۳- عبارتی (Statement List) STL

---

۱ - البته روش دیگری جهت برنامه‌نویسی به زبان S5 وجود دارد که روشی گرافیکی است و GRAPH 5 نام دارد که از ذکر این روش خودداری می‌کنیم.

۳-۱-۱- روش نمایش نردبانی (LAD)

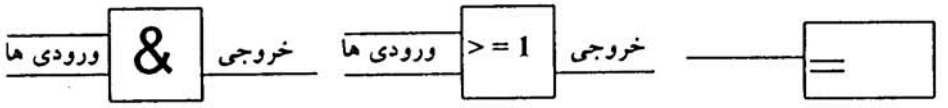
در نمایش نردبانی، هر دستور یا خط برنامه به صورت نماد اتصال و سیم‌پیچ مدارهای فرمان رله‌ای نشان داده می‌شود. در نتیجه ساختار برنامه در این روش تقریباً شبیه به شکل مدارهای فرمان رله‌ای می‌باشد. این طرز نمایش از قدیم در سیستم‌های رله‌ای متداول بوده، نقشه‌های مدار فرمان اکثراً به این روش ترسیم می‌شوند. به همین دلیل این طرز نمایش تا حد زیادی مأنوس و مورد پسند کسانی است که با سیستم‌های رله‌ای کار کرده‌اند. علاوه بر این، نمایش نردبانی به سادگی قابل درک بوده، نقشه‌ای که به این روش ترسیم شود درست مانند نقشهٔ الکتریکی مدار فرمان همان سیستم است. برخی از نمادهای مورد استفاده در این روش برنامه‌نویسی در زیر آمده است.

نماد کنتاکت در حالت عادی باز[1] — نماد سیم‌پیچ رله

شکل ۳-۱ : برخی نمادهای مورداستفاده در نمایش LAD

۳-۱-۲- روش نمایش فلوچارتی (CSF)

در نمایش فلوچارتی، برنامه به صورت مجموعه‌ای از نمادهای مستطیل شکل (بلوک) نشان داده می‌شود. این طرز نمایش بیشتر در هنگام طراحی برنامه استفاده می‌گردد. در شکل ۳-۲ چند بلوک مورد استفاده در این روش نمایش برنامه نشان داده شده است.



بلوک AND — بلوک OR — بلوک هم ارزی (نسبت دادن مقدار به خروجی)

شکل ۳-۲ : چند بلوک مورد استفاده در روش CSF

1 - NO (Normally Open)

در این روش در هر بلوک نوع عمل منطقی نشان داده می‌شود و ورودی‌ها و خروجی‌های هر بلوک نیز مشخص می‌گردند. این روش نمایش برنامه با روش ترسیم مدارهای منطقی به صورت الکترونیکی مطابقت دارد.

۳-۱-۳- روش نمایش عبارتی (STL)

قبل از ورود به بحث روش برنامه‌نویسی STL به توضیح در مورد چند مفهوم پایه در این روش برنامه‌نویسی می‌پردازیم.

عبارت یا Statement

Statement یا هر خط از برنامهٔ نوشته شده به روش STL، سطری از برنامه است که معمولاً دارای دو بخش زیر می‌باشد:

الف) عملکرد (Operation)

ب) عملوند (Operand)

الف) عملکرد (Operation)

به عمل منطقی که درعبارت صورت می‌گیرد عملکرد گفته می‌شود. عملکردهای مهم عبارتند از: AND، OR، = و ... که در جدول ۳-۱ نشان داده شده‌اند.

جدول ۳-۱ : عملکردهای مهم استفاده شده در برنامه‌نویسی به روش STL

در منطق ریاضی	در زبان محاوره‌ای		در زبان برنامه‌نویسی STL
ترکیب عطفی	"و"	(AND)	A
ترکیب فصلی	"یا"	(OR)	O
نقیض ترکیب عطفی	نقیض "و"	(AND NOT)	AN
نقیض ترکیب فصلی	نقیض "یا"	(OR NOT)	ON
ترکیب هم‌ارزی	مساوی قرار دادن	(ASSIGN TO)	=

ب) عملوند (Operand)

به قسمتی از عبارت گفته می‌شود که قرار است یک عمل منطقی (عملکرد) در مورد آن اجرا شود مانند ورودی‌ها، خروجی‌ها، فلگ‌ها و ...

در جدول ۳-۲ بخشهای Statement نشان داده شده است.

جدول ۳-۲ : بخشهای یک عبارت در برنامه‌نویسی به روش STL

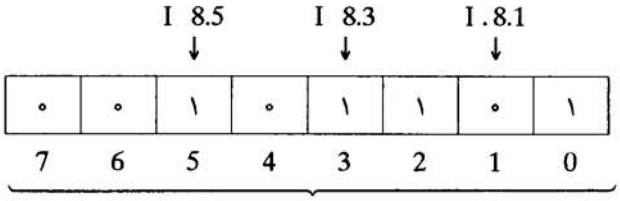
Statement	عبارت	
Operation عملکرد	*Operand*	عملوند
	Operand Identifier نوع عملوند	*Parameter* آدرس عملوند
A	I	1.0
A	I	7.1
=	Q	2.5
O	I	0.6
O	I	8.4
=	Q	1.3

همان گونه که در جدول ۳-۲ مشاهده می‌گردد عبارات موجود شامل عملکرد و عملوند می‌باشند که عملوند خود شامل دو بخش آدرس عملوند و نوع عملوند است. نوع عملوند، همان ورودی‌ها، خروجی‌ها، فلگ‌ها و ... هستند و آدرس عملوند محل عملوند را مشخص می‌نماید.

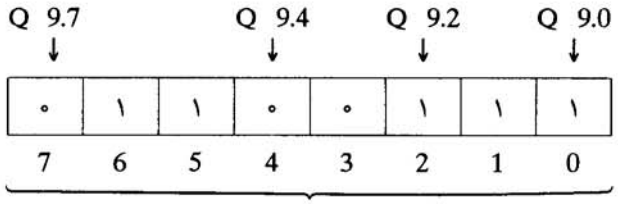
آدرس ورودی‌ها، خروجی‌ها و فلگ‌ها

از آنجایی که آدرس باس دارای ۸ خط ارتباطی است، در نتیجه ورودی‌ها، خروجی‌ها و فلگ‌ها در دسته‌های هشت بیتی (یک بایتی) سازماندهی می‌شوند. پس در آدرس‌دهی ورودی، خروجی و فلگ ابتدا باید آدرس بایت آنها مشخص وسپس موقعیت وآدرس بیت‌مربوطه درآن بایت تعیین گردد.

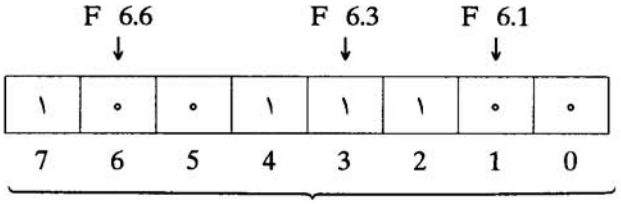
در شکل ۳-۳ نحوهٔ آدرس دهی ورودی، خروجی و فلگ نشان داده شده است.



8 IB[1] (بایت ورودی ۸)



9 QB[2] (بایت خروجی ۹)



6 FY[3] (بایت فلگ ۶)

شکل ۳-۳ : نحوهٔ آدرس دهی ورودی‌ها، خروجی‌ها و فلگ‌ها.

طریقهٔ آدرس دهی را با چند مثال بررسی می‌کنیم:

بیت سوم از بایت چهارم ورودی	(آدرس ورودی) ← 4.3	I → (ورودی)
بیت هفتم از بایت پنجم خروجی	(آدرس خروجی) ← 5.7	Q → (خروجی)
بیت دوم از بایت یازدهم فلگ	(آدرس فلگ) ← 11.2	F → (فلگ)

1 - Input Byte
2 - Output Byte
3 - Flag Byte

توضیح اینکه FB علامت اختصاری Function Block می‌باشد.

روش نمایش STL

در این روش برنامهٔ کنترل با استفاده از حروف و اعداد لاتین به صورت جملات منطقی و پشت سر هم نوشته می‌شود و هر حرف، معرف یک واژهٔ انگلیسی است. مثلاً حرف A معرف AND، O معرف OR، I معرف Input و Q معرف Output می‌باشد.

در روش STL، برنامه به صورت مجموعه‌ای از دستورات است که به هر دستور یک رشته (خط برنامه) یا Statement گفته می‌شود و هر دستور یا خط برنامه معمولاً یکی از ترکیب‌های منطقی ریاضی یعنی ترکیب‌های AND، OR، NOT، هم‌ارزی و ... را دربر دارد. علاوه بر ترکیب‌های فوق وسایل نرم‌افزاری فلگ، فلیپ‌فلاپ، شمارنده، تایمر و ... در اکثر PLC ها وجود دارند که در حقیقت بخشی از زبان برنامه‌نویسی می‌باشند.

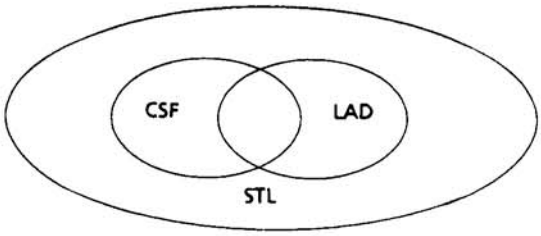
در برنامهٔ نوشته شده به روش STL به چندین سطر که عمل خاصی انجام می‌دهند یک Segment می‌گویند. یک برنامه (Program) از چندین Segment تشکیل می‌گردد. لازم به ذکر است که یک برنامه می‌تواند تنها شامل یک Segment باشد.

در زیر یک نمونهٔ کاملاً ساده از برنامهٔ نوشته شده به روش STL را می‌بینیم.

```
 PB 1

SEGMENT  1            0000
0000       :A    I     1.4
0001       :A    I     2.3
0002       :=    Q     3.2
0003       :BE
```

در این برنامه بین دو ورودی با نامهای I 1.4 و I 2.3 ترکیب عطفی (AND) صورت گرفته، حاصل این ترکیب در خروجی Q 3.2 قرار می‌گیرد. سطر انتهایی این برنامه نشان دهندهٔ پایان یافتن برنامه (Block End) است.

هر یک از روشهای برنامه‌نویسی دارای مشخصات خاصی می‌باشد. اکثر سازندگان PLC، در طراحی نرم‌افزار برنامه‌نویسی به گونه‌ای عمل کرده‌اند که می‌توان برنامهٔ نوشته شده در یک روش را به روشهای دیگر تبدیل نمود. در زبان STEP 5 لزوماً برنامهٔ نوشته شده به روش STL را نمی‌توان به LAD یا CSF تبدیل نمود اما برنامه نوشته شده به روش LAD و CSF همواره قابل تبدیل به STL می‌باشد. شکل ۳-۴ گویای این مطلب است.



شکل ۳-۴ : نمایش امکان تبدیل و ترجمهٔ برنامهٔ نوشته شده به روش‌های برنامه‌نویسی دیگر

۳-۲- سیکل زمانی اجرای برنامه (Cycle Time)

برنامه نوشته شده به روش STL را در نظر بگیرید.

```
 PB 2

SEGMENT   1              0000
0000        :A    I       1.4
0001        :A    I       1.5
0002        :=    Q       2.3
0003        :BE
```

ریزپردازنده برای اجرای این برنامه از سطر اول شروع نموده، دستورات را به ترتیب اجرا می‌کند تا به دستور BE که نشان دهندهٔ پایان اجرای برنامه است برسد. در این زمان ریزپردازنده اطلاع پیدا می‌کند که برنامه پایان یافته است بنابراین مجدداً به سطر اول باز می‌گردد و این روند همچنان ادامه پیدا می‌کند. این عمل بین تمام پردازنده‌ها مشترک بوده و به Cycle Processing معروف است.

به مدت زمانی که پردازنده از یک سطر برنامه شروع به اجرای دستورات نماید و مجدداً به همان سطر برگردد، یک Cycle Time می‌گویند.

کاملاً روشن است که هر چه سیکل زمانی یک برنامه کمتر باشد یعنی به زمان کمتری جهت پردازش نیاز داشته باشد عملکرد کلی سیستم مطلوب‌تر است. برای کاهش این سیکل زمانی می‌توان از یکی از دو روش زیر استفاده نمود:

۱- به کارگیری پردازنده‌هایی با سرعت و قابلیت بالاتر

۲- استفاده از Structured Programming در برنامه‌نویسی

۳-۳- برنامه‌نویسی سازمان‌یافته (Structured Programming)

هنگامی که PLC فرآیند پیچیده‌ای را که برنامه‌ای طولانی دارد کنترل می‌نماید، مدت زمان اجرای یک سیکل برنامه طولانی می‌شود. بنابراین باید ترتیبی اتخاذ نمود تا این مدت زمان از حد معینی تجاوز ننماید. یک راه حل آن است که از پردازنده‌های سریعتری در PLC استفاده شود، از آنجایی که سرعت پردازنده‌ها نیز محدود است راه حل مناسب‌تر استفاده از برنامه‌نویسی سازمان‌یافته می‌باشد.

فرآیندهای پیچیده معمولاً از چندین بخش که هر کدام وظیفهٔ خاصی را انجام می‌دهند و عملکرد آنها بر یکدیگر تأثیرگذار است تشکیل می‌گردند. بنابراین در نوشتن برنامهٔ آنها باید برای هر بخش، برنامهٔ جداگانه‌ای نوشته شود و پس از صحت عملکرد، آنها را که برنامه‌های فرعی نامیده می‌شوند در یک برنامهٔ اصلی فراخوانی نمود. به عبارت دیگر باید این برنامه‌های فرعی را در برنامهٔ اصلی چنان سازمان داد تا بتوان فرآیندهای پیچیده را کنترل نمود. همان گونه که ذکر شد در برنامه‌نویسی سازمان یافته، مدت زمان اجرای یک سیکل برنامه توسط PLC کاهش می‌یابد. این به معنی صرفه‌جویی در وقت، کنترل دقیق‌تر PLC بر فرآیند و استفاده بیشتر از ظرفیت PLC است.

برنامه‌نویسی سازمان یافته علاوه بر کاهش زمان سیکل انجام برنامه، نوشتن برنامه و همچنین وارد کردن برنامه را به واحد برنامه‌نویسی (PG) ساده‌تر می‌کند. برنامه‌نویسی در این روش، به گونه‌ای انجام می‌شود که قسمتهایی از برنامه که وظیفهٔ خاصی را انجام می‌دهند تحت عناوین خاصی دسته‌بندی می‌شوند. بنابراین در هنگام تغییر، تکمیل و یا عیب‌یابی می‌توان مستقیماً به سراغ قسمت مورد نظر رفت. به علاوه، درک و تحلیل برنامه برای کسانی که آن برنامه را ننوشته‌اند نیز میسر می‌شود.

جهت تشریح روش سازماندهی در برنامه‌نویسی به زبان S5، به عنوان مثال PLC مدل S5-115 U را در نظر گرفته‌ایم. روش سازماندهی در PLC های دیگر تا اندازه‌ای شبیه به هم می‌باشد.

کل برنامه‌ای که در این نوع PLC جهت کنترل پروسه و همچنین تنظیم روابط سیستم عامل PLC با برنامهٔ استفاده‌کننده (User Program) نوشته می‌شود در بلوک‌های زیر دسته‌بندی شده است.

۳-۳-۱- بلوک‌های برنامه (PB)[1]

بلوک برنامه را به اختصار PB می‌گوییم. PB‌ها با توجه به پروسهٔ تحت کنترل شامل دستوراتی هستند که استفاده کننده برای کنترل پروسه می‌نویسد.

به عبارت دیگر، PB‌ها بلوک‌های تشکیل‌دهندهٔ برنامهٔ کنترل یک فرآیند می‌باشند. به دلایل مختلفی از جمله، جلوگیری از پیچیدگی و طولانی شدن، برنامهٔ کنترل فرآیند به قسمتهای کوچکتری که همان PB‌ها هستند تفکیک می‌گردد. در این نوع PLC می‌توان از تعداد ۲۵۶ بلوک برنامه (از شمارهٔ PB 0 الی PB 255) استفاده نمود.

استفاده کننده با توجه به سلیقه و برداشت خود از فرآیند تحت کنترل می‌تواند دستوراتی را که مربوط به یکدیگر بوده و از نظر فنی عمل خاصی انجام می‌دهند در یک PB قرار دهد. در پایان هر PB یک دستور BE وجود دارد که مشخص‌کنندهٔ پایان برنامهٔ هر بلوک می‌باشد.

۳-۳-۲- بلوک‌های ترتیبی (SB)[2]

این بلوک‌ها در کنترل‌های ترتیبی مورد استفاده قرار می‌گیرند. همان گونه که می‌دانید در پروسه‌های صنعتی، کنترل فرآیند معمولاً به ۳ حالت زیر انجام می‌گیرد:

۱- کنترل دستی (Manual یا Hand)

۲- کنترل اتوماتیک (Auto)

۳- کنترل تک‌مرحله‌ای (Inching یا Single Step)

در برخی از پروسه‌ها لازم است که در ابتدای راه‌اندازی صحت عملکرد خط تولید بررسی گردد و پس از اطمینان، سیستم به صورت اتوماتیک کنترل گردد. در این پروسه‌ها هر مرحله (Sequence) به وسیلهٔ یک کلید راه‌اندازی می‌شود. بنابراین این گونه بلوک‌ها را بلوک‌های ترتیبی یا Sequence Block می‌گویند.

۳-۳-۳- بلوک‌های تابع‌ساز (FB)[3]

در کنترل برخی فرآیندها، گاه لازم است توابعی به صورت مداوم و تکراری مورد استفاده قرار

---

1 - Program Block 2 - Sequence Block 3 - Function Block

گیرند. به عنوان مثال در برخی پروسه‌ها ضرب دو عدد باینری مکرراً مورداستفاده قرار می‌گیرد. در چنین مواردی لازم نیست که برنامهٔ ضرب هر بار توسط استفاده کننده و در هر کجا که لازم باشد نوشته شود، بلکه این برنامه تحت نام یک FB تنها یک بار نوشته می‌شود، و سپس هر جا که لازم باشد فراخوانده شده، اطلاعات لازم به آن داده می‌شود و عملیات انجام می‌گیرد. در این نوع PLC تنها می‌توان تعداد ۲۵۶ بلوک تابع‌ساز تعریف نمود (FB 0 - FB 255).

هر FB از دو قسمت تشکیل شده است:

۱- سر خط بلوک (Block Header) که شامل نام و سایر مشخصات FB است.

۲- بدنهٔ بلوک (Block Body) که شامل توابع و دستوراتی است که باید در FB اجرا شود. علاوه بر دستوراتی که در زبان S5 برای نوشتن FB ها وجود دارد تعدادی دستور مخصوص با نام دستورات Supplementary نیز موجود است که تنها مخصوص استفاده در FB ها می‌باشند.

در تقسیم‌بندی کلی می‌توان FB ها را به دو دستهٔ زیر تقسیم نمود:

الف) بلوک‌های تابع‌ساز استاندارد (Standard FB) : این بلوک‌ها شامل توابعی هستند که در آنها اعمال منطقی نظیر ضرب، تقسیم و ... تعریف شده است. این بلوک‌ها به صورت بسته‌های نرم‌افزاری توسط شرکت سازنده نوشته شده و به همراه دفترچهٔ راهنما که حاوی اطلاعاتی در مورد چگونگی وارد نمودن پارامترها و دیگر اطلاعات می‌باشد در اختیار کاربر قرار می‌گیرد.

ب) بلوک‌های تابع‌ساز انتسابی (Assignable FB) : در اجرای این نوع FB می‌توان عملوندها (Operand) یعنی ورودی‌ها، خروجی‌ها و ... را در هر پروسه‌ای تعریف نمود و یا تغییر داد. به عنوان مثال برای اجرای عملیات پرس می‌توان در مرحلهٔ ۱ از عملوند I 0.0 و در مرحلهٔ ۲ از عملوند I 0.6 به عنوان ورودی استفاده نمود.

لازم به ذکر است که FBها فقط به روش STL قابل برنامه‌نویسی می‌باشند.

۳-۳-۴- بلوک‌های اطلاعاتی (DB)[1]

بلوک‌های اطلاعاتی شامل داده‌هایی هستند که هنگام اجرای برنامه توسط PLC تقاضا می‌شوند.

---

1 - Data Block

در این نوع PLC دقیقاً تعداد ۲۵۶ بلوک (DB 255 - DB 0)قابل تعریف است.

همان گونه که می‌دانید در برخی از پروسه‌ها لازم است مواردی همچون پیغام‌ها، آلارم‌ها و ... بر روی صفحهٔ نمایش ظاهر شوند.(به عنوان مثال پیام‌های HIGH TEMPERATURE، TANK LEVEL LOW و ...) محل نگهداری این پیام‌ها بلوک‌های اطلاعاتی می‌باشند. هر DB می‌تواند شامل ۲۵۶ کلمهٔ اطلاعاتی (Data Word) باشد. داده‌ها می‌توانند به صورت بیت، عدد هگز، عدد باینری و یا به صورت حروف (جهت پیام‌ها) نوشته شوند.

خواندن اطلاعات از DBبدون شرط انجام می‌گیرد. هنگامی که در اجرای یک برنامه، DB خاصی فعال یا معتبر می‌گردد برنامه با توجه به اطلاعات موجود در آن DB اجرا می‌شود و این عمل تا زمانی که DB دیگری معتبر نگردیده، انجام می‌گیرد[1] . DBها می‌توانند در تمامی بلوک‌ها فراخوانده شوند.

۳-۳-۵- بلوک‌های سازماندهی (OB)[2]

OB ها ساختار برنامهٔ استفاده کننده را مشخص می‌کنند و هر OB با یک شمارهٔ خاص مشخص می‌شود. در تعریف OB می‌توان گفت که OBها بلوک‌هایی هستند که هر یک عمل خاصی را انجام می‌دهند و در واقع ارتباط بین سیستم عامل و برنامهٔ استفاده کننده را برقرار می‌کنند. چند نمونه از این بلوک‌ها را شرح می‌دهیم:

OB 1 : در شروع هر سیکل برنامه، OS یا سیستم عامل به OB 1 مراجعه می‌کند. در واقع اولین جمله از OB 1 شروع برنامهٔ استفاده کننده و آخرین سطر آن، پایان برنامهٔ استفاده کننده است. بنابراین ساختار اجرایی برنامهٔ استفاده کننده در OB 1 تعیین می‌گردد.

OB 21 : هنگامی که PLC از حالت STOP به حالت START سوییچ می‌گردد، ابتدا برنامهٔ این بلوک اجرا می‌شود.

OB 22 : هنگامی که کلید قدرت PLC از حالت خاموش به حالت روشن (POWER ON) زده

---

۱ - در مورد اعتبار DBها در فصل آینده توضیح خواهیم داد.

2 - Organization Block

می‌شود برنامهٔ این بلوک اجرا می‌گردد. بنابراین می‌توان شرایط لازم برای در نظر گرفتن موارد ایمنی را در هنگام قطع جریان برق و وصل مجدد آن و یا ... در دو بلوک مذکور قرار داد.

OB 34 : هنگامی که باطری PLC (Battery Back up) ضعیف و یا اشکالی در آن ایجاد گردد این OB اجرا و تا زمانی که اشکال مذکور برطرف نگردد، این OB مکرراً اجرا خواهد شد. بنابراین عکس‌العمل سیستم را در برابر اشکالات باطری می‌توان به نحو دلخواهی در OB 34 برنامه‌ریزی نمود.

در مقایسه با روش برنامه‌نویسی سازمان یافته که جهت برنامه‌نویسی پروسه‌های عظیم و پیچیده استفاده می‌شود روش برنامه‌نویسی دیگری نیز با نام Linear Programming وجود دارد. این روش جهت نوشتن برنامه‌های ساده و برنامه‌هایی که تعداد سطرهای آن از ۵۰۰ سطر کمتر باشد مورد استفاده قرار می‌گیرد. این برنامه را می‌توان در OB 1 یا PB 1 نوشت.

۳-۴- عملوندهای مورد استفاده در زبان S5 (Operand Area)

در زبان برنامه‌نویسی S5 عملوندهای زیر مورد استفاده قرار می‌گیرند.

I (Inputs) ورودی‌ها، از پروسهٔ تحت کنترل به PLC.

Q (Outputs) خروجی‌ها، از PLC به پروسهٔ تحت کنترل.

F (Flags) فلگ‌ها، حافظه‌ای جهت نگهداری مقادیر میانی حاصل از عملیات باینری.

D (Data) دیتا، حافظه‌ای جهت نگهداری مقادیر میانی حاصل از عملیات دیجیتالی.

T (Timers) تایمرها، حافظه‌ای جهت تخصیص زمان‌سنج‌ها.

C (Counters) شمارنده‌ها، حافظه‌ای جهت تخصیص شمارشگرها.

K (Constants) ثابت‌ها.

۳-۵- دستورالعمل‌های زبان S5[1]

دستورات زبان S5 را در حالت کلی می‌توان به سه دستهٔ زیر تقسیم نمود:

---

۱ - دستورات زبان S5 در ضمیمهٔ ۲ آمده است.

۱- دستورالعمل‌های اصلی (Basic)

۲- دستورالعمل‌های تکمیلی (Supplementary)

۳- دستورالعمل‌های سیستم (System)

۳-۵-۱- دستورالعمل‌های اصلی (Basic)

این دستورات شامل توابعی هستند که در تمام بلوک‌ها (FB ، SB ، PB و OB) قابل اجرا و استفاده می‌باشند. به استثنای دستورات جمع (F+) و تفریق (F-) تمام دستورات اصلی می‌توانند به عنوان ورودی و خروجی در روشهای برنامه‌نویسی STL ، LAD و CSF مورد استفاده قرار گیرند.

۳-۵-۲- دستورالعمل‌های تکمیلی (Supplementary)

این دستورات مشتمل بر توابع ترکیبی نظیر دستورات جابجایی، توابع Shift و دستورات تبدیلی می‌باشند. این دستورات تنها در FBها و فقط به روش STL قابل استفاده‌اند.

۳-۵-۳- دستورالعمل‌های سیستم (System)

دستورالعمل‌های سیستم، دستوراتی هستند که مستقیماً بر سیستم عامل PLC تأثیرگذار می‌باشند و تنها یک برنامه‌نویس با تجربه مجاز است از آنها استفاده نماید.

جدول ۳-۳ حاوی اطلاعات جامعی در مورد این دستورات می‌باشد.

جدول ۳-۳ : دستورالعمل‌های زبان S5 و برخی اطلاعات لازم در مورد آنها

	دستورالعمل‌های اصلی (Basic)	*دستورالعمل‌های تکمیلی* (Supplementary)	*دستورالعمل‌های سیستم* (System)
کاربرد	در بلوک‌های FB ، SB ، PB ، OB	تنها در بلوک‌های FB	تنها در بلوک‌های FB
روش‌نمایش	STL ، LAD ، CSF	STL	STL
قابلیت‌های خاص	-	-	تنهاافرادباتجربه‌می‌توانندازاین‌دستوراتاستفاده‌نمایند.

۳-۶- خواندن صفر (Scanning for Zero)

برای درک این مطلب به ذکر یک مثال می‌پردازیم.

فرض کنید که در یکی از بایت‌های ورودی عدد $۲۰_{۱۰}$ یا ۱۰۱۰۰_{H} به صورت زیر وجود دارد.

0	0	0	1	0	1	0	0

با توجه به بیت‌های قرار داده شده در این بایت این مطلب استنباط می‌گردد که پردازنده تمام بیت‌ها را اعم از "۰"ها و "۱"ها خوانده و عدد ۰۰۰۱۰۱۰۰ را به عنوان یک بایت ورودی دریافت می‌کند. حال اگر برنامه‌نویس بخواهد با بیت "۰" حافظه، عملی انجام دهد (به عنوان مثال آن را از حافظه بخواند) باید از دستور ... AN (AND NOT) در روش STL و از نماد -|/|- در روش LAD و از -|\|- در روش CSF استفاده نماید. به این عمل در اصطلاح خواندن صفر یا Scanning for Zero می‌گویند.

۳-۷- کنتاکت در حالت عادی باز (NO)[1]

این اتصال (کنتاکت) هنگامی فعال می‌شود که مثلاً دگمهٔ پوش باتن (Push Button) فشرده یا کلیدی روشن شود. در این حالت در ورودی PLC مقدار "۱" ظاهر می‌گردد و در حالتی که دگمهٔ پوش باتن فشرده نشده یا کلید خاموش باشد در ورودی PLC ولتاژی ظاهر نشده و به اصطلاح ورودی PLC مقدار "۰" خواهد بود.

۳-۸- کنتاکت در حالت عادی بسته (NC)[2]

این اتصال بر عکس اتصال NO عمل می‌نماید، یعنی در حالتی که فعال نباشد ورودی PLC مقدار "۱" بوده و هنگامی که فعال شود در ورودی PLC مقدار "۰" را خواهیم داشت.

PLC در ورودی‌های خود تنها مطلع می‌شود که مقدار سیگنال ورودی خوانده شده "۰" یا "۱" است و هیچ مرجعی برای مطلع شدن از نوع اتصال (NO یا NC) ندارد. بنابراین نوع اتصال را باید از

---

1 - Normally Open Contact 2 - Normally Close Contact

طریق برنامه به PLC اطلاع داد. خلاصهٔ آنچه که ذکر شد و همچنین شیوهٔ نمایش اتصالات NO و NC در جدول ۳-۴ آورده شده است.

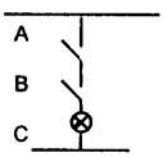
جدول ۳-۴ : نحوهٔ نمایش کنتاکت‌های NO و NC در روشهای مختلف برنامه‌نویسی

نوع اتصال	موقعیت اتصال	موقعیت سیگنال در ورودی	نگارش در برنامه		
			CSF	LAD	STL
اتصال NO	فعال	"۱"			A ...
	غیرفعال	"۰"			O ...
اتصال NC	فعال	"۰"			AN ...
	غیرفعال	"۱"			ON ...

اکنون که با روش آدرس‌دهی عملوندها و همچنین عملکردهای مهم آشنا شدیم برای روشن شدن مطلب و درک بیشتر چگونگی استفاده از دستورات S5 و همچنین روش برنامه‌نویسی به ذکر چند مثال خواهیم پرداخت. این مثالها اکثراً از موارد عملی و کاربردی که در محیط‌های صنعتی با آنها روبرو هستیم انتخاب شده‌اند.

در ضمن جهت آشنایی هر چه بیشتر خوانندگان با روشهای مختلف برنامه‌نویسی، سعی شده که حتی‌المقدور از هر سه روش یاد شده در زبان S5 استفاده گردد.

مثال ۳-۱ : مدار زیر را در نظر بگیرید، می‌خواهیم برنامه‌ای بنویسیم که شرایط روشن و خاموش بودن لامپ را مشخص نماید.



همانگونه که ملاحظه می‌شود دو کلید A و B به صورت سری قرار گرفته‌اند و لازمهٔ روشن شدن لامپ C بسته شدن هر دو کنتاکت می‌باشد. (توجه داشته باشید که هر دو این کنتاکت‌ها در حالت عادی باز هستند.)

به عبارت دیگر برای روشن شدن لامپ C لازم است تا هر دو کلید A و B در وضعیت "۱" قرار

گیرند، بنابراین از دستور AND استفاده می‌نماییم. جهت برنامه‌نویسی لازم است تا کنتاکت‌های A و B را به دو ورودی مثلاً I 1.0 و I 7.4 و لامپ C را به خروجی Q 2.5 نسبت دهیم. در PB 3 برنامهٔ مطلوب به هر سه روش نوشته شده است.

نمایش منطقی خروجی بر حسب ورودی‌ها : C = A.B

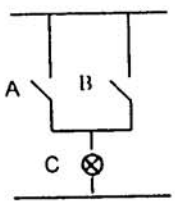
```
 PB 3

SEGMENT  1              0000
0000       :A    I      1.0
0001       :A    I      7.4
0002       :=    Q      2.5
0003       :BE
```

```
 PB 3

SEGMENT  1              0000
                +---+
 I 1.0      ---! & !     +-------+
 I 7.4      ---!   !--+-! =     ! Q 2.5
                +---+     +-------+:BE
```

```
 PB 3

SEGMENT  1              0000
!
!I 1.0      I 7.4                    Q 2.5
+---] [---+---] [---+----------+--(    )-!
!
!
!                                   :BE
```

حال فرض کنید که I 1.0 سنسور ورودی نشان دهندهٔ فعال بودن موتور الکتریکی ۱ و I 7.4 سنسور نشان دهندهٔ فعال بودن موتور الکتریکی ۲ باشد. روشن شدن خروجی Q 2.5 به منزلهٔ فعال بودن هر دو موتور الکتریکی است. این خروجی می‌تواند به صورت یک چراغ (LED) بر روی پانل کنترل نمایان شود.

مثال ۳-۲: نمایش مداری شکل زیر را در نظر بگیرید. قصد داریم با نوشتن برنامه‌ای، عملکرد این مدار را توصیف نماییم.



با دقت در این مدار ملاحظه می‌شود که دو کنتاکت A و B به صورت موازی قرار گرفته و در صورت بسته شدن هر یک از دو کنتاکت و یا بسته شدن هر دوی آنها لامپ C روشن خواهد شد. در این مدار لازم است تا از ترکیب فصلی دو ورودی استفاده گردد یعنی: C = A + B در این مثال کنتاکت‌های A و B را به ورودی‌های I 2.3 و I 5.0 و لامپ C را به خروجی Q 7.7 نسبت می‌دهیم. این برنامه در PB 4 به هر سه روش نوشته شده است.

```
 PB 4

SEGMENT  1              0000
0000       :O    I      2.3
0001       :O    I      5.0
0002       :=    Q      7.7
0003       :BE
```

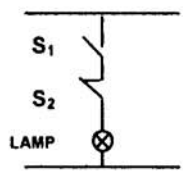
```
 PB 4

SEGMENT  1              0000
                +---+
 I 2.3      ---!>=1!     +------+
 I 5.0      ---!   !--+-! =    ! Q 7.7
                +---+    +------+:BE
```

```
 PB 4

SEGMENT  1              0000
!
!I 2.3                                  Q 7.7
+---] [---+----------+----------+--(    )-!
!         !
!I 5.0    !
+---] [---+                      :BE
!
```

در این مثال می‌توان I 2.3 را به عنوان سنسور بالابودن فشار داخل مخزن و I 5.0 را سنسور بالا بودن درجه حرارت مخزن فرض نمود. حال در صورتی که یک یا هر دو ورودی فعال شوند مقدار Q 7.7 که آن را می‌توان چراغ (LED) و یا آژیر هشدار دهنده فرض نمود "۱" شده، این خروجی فعال می‌گردد و اُپراتور را از بروز خطرات احتمالی آگاه می‌سازد.

خوانندگان توجه دارند که نسبت دادن ورودی‌ها و خروجی‌ها به شرایط ورودی و فرمانهای صادرهٔ خروجی با توجه به قابلیت‌های سیستم PLC و به صورت اختیاری صورت می‌گیرد ولی در مورد آدرس‌دهی همانگونه که گفته شد باید دقت کافی مبذول داشت که آدرس‌های ورودی و خروجی تکراری کاملاً آگاهانه استفاده شوند و در صورتی که یک ورودی یا خروجی برای مقادیر دیجیتال استفاده شود نسبت دادن مقدار آنالوگ در آدرس‌دهی بعدی مجاز نیست و بالعکس.

مثال ۳-۳: فرض کنید مداری مطابق شکل زیر داریم. بدین صورت که جهت روشن شدن LAMP لازم است تا پوش باتن S1 فشرده شده، پوش باتن S2 در همان حالت اولیهٔ خود باقی بماند.(فشرده نشود) برنامه‌ای بنویسید تا عملکرد این مدار را مشخص نماید.



همانگونه که از نماد مداری استنباط می‌گردد برای کلید S1 از کنتاکت NO و برای پوش باتن S2 از کنتاکت NC استفاده می‌کنیم، چرا که برای روشن شدن LAMP لازم است تا پوش باتن S2 در حالت عادی (بدون فشرده شدن) بسته باشد. در ادامه، روشهای مختلف برنامه‌نویسی برای عملکرد این مدار آورده شده است. به کنتاکت‌های S1 و S2 دو ورودی I 1.5 و I 2.7 و به LAMP خروجی Q 4.3 را نسبت می‌دهیم.

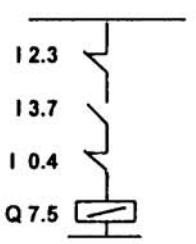
```
 PB 5

SEGMENT  1            0000
0000      :A    I     1.5
0001      :AN   I     2.7
0002      :=    Q     4.3
0003      :BE
```

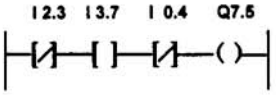
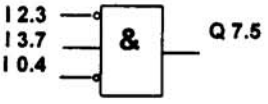
```
 PB 5

SEGMENT   1            0000
                +---+
 I 1.5      ---! & !      +-------+
 I 2.7      --O!   !--+-! =      ! Q 4.3
                +---+     +-------+:BE
```

```
 PB 5

SEGMENT   1            0000
!
!I 1.5      I 2.7                    Q 4.3
+---] [---+---]/[---+----------+--(     )-!
!                                  :BE
!
```

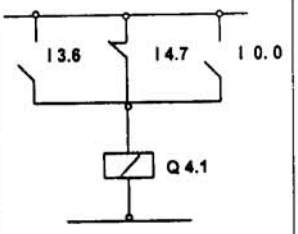
در مثالهای بعدی به پیچیدگی برنامه اضافه نموده و از ذکر جزئیات خودداری می‌کنیم.

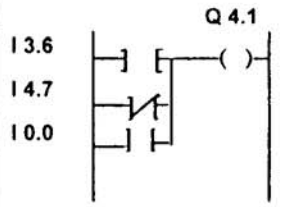
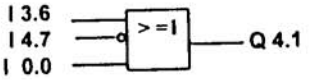
مثال ۳-۴: در این مثال دوباره قصد داریم تا چگونگی استفاده از کنتاکت‌های NC را در برنامه‌نویسی بررسی نماییم.

نماد مداری	توضیح
	خروجی Q 7.5 مقدار "۱" خواهد داشت در صورتی که ورودی‌های I 2.3 و I 0.4 "۰" بوده و I 3.7 مقدار "۱" داشته باشد. این خروجی برابر "۰" خواهد بود در صورتی که حداقل یکی از شرایط مذکور برقرار نباشد.

LAD	CSF	STL
		AN I 2.3 A I 3.7 AN I 0.4 = Q 7.5

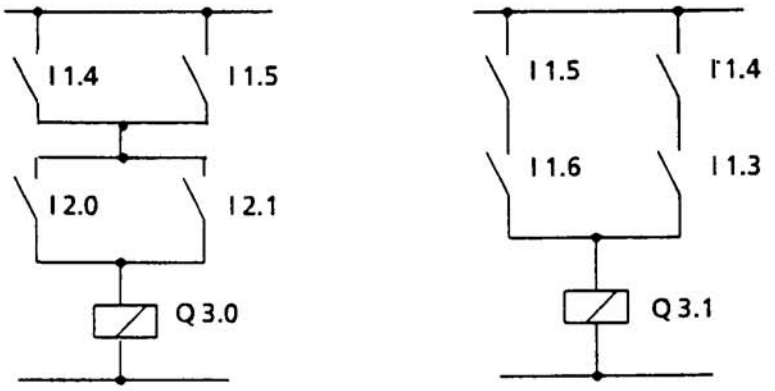
مثال ۳-۵ : در این مثال نیز ترکیب فصلی ورودی‌ها را به همراه کنتاکت‌های NC مورد بررسی قرار می‌دهیم.

نماد مداری	توضیح
	خروجی Q 4.1 برابر "۱" خواهد بود در صورتی که حداقل یکی از شرایط زیر برقرار باشد: I 3.6 = "۱" یا I 4.7 = "۰" یا I 0.0 = "۱" و مقدار این خروجی برابر "۰" خواهد بود اگر I 3.6 = "۰" و I 4.7 = "۱" و I 0.0 = "۰" باشد.

LAD	CSF	STL
		O I 3.6 ON I 4.7 O I 0.0 = Q 4.1

۳-۹- کاربرد پرانتزها در برنامه‌نویسی به روش STL

نمایش مدارهای زیر را در نظر بگیرید.



(الف) (ب)

شکل ۳-۵ : نمایش مداری دو مدار فرمان جهت بررسی در مورد استفاده از پرانتزها در روش STL

در صورتی که تابع منطقی دو خروجی مدارهای فرمان را بر حسب ورودی‌های مربوطه بنویسیم، خواهیم داشت:

الف ۳-۶ : $(I\ 1.5 \cdot I\ 1.6) + (I\ 1.4 \cdot I\ 1.3) = Q\ 3.1$

(AND قبل از OR) AND OR

ب ۳-۶ : $(I\ 1.4 + I\ 1.5) \cdot (I\ 2.0 + I\ 2.1) = Q\ 3.0$

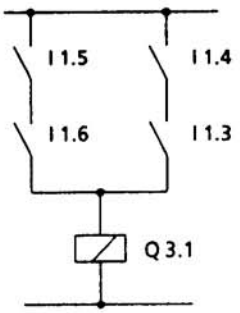
(OR قبل از AND) OR AND

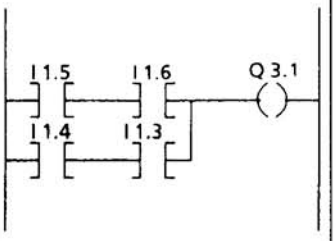
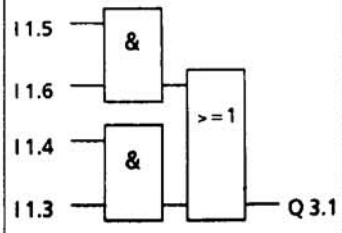
با اندکی دقت در این دو تابع در می‌یابیم که در تابع اول دستور AND قبل از دستور OR استفاده شده است در حالی که در تابع دوم دستور OR قبل از دستور AND به کار برده شده است.

بنابراین در مواردی نظیر تابع اول که دستور AND قبل از دستور OR قرار گرفته باشد نیازی به استفاده از پرانتز در برنامه‌نویسی به روش STL نیست، ولی در نمونه‌هایی نظیر تابع دوم که در آن دستور OR قبل از دستور AND قرار گرفته است استفاده از پرانتز الزامی می‌باشد.

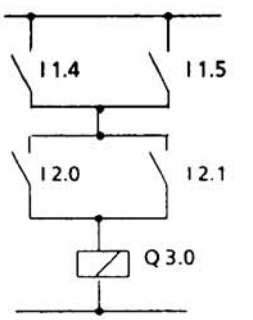
برای روشن شدن مطالب عنوان شده، در دو مثال بعدی، برنامهٔ عملکرد این دو مدار فرمان را بررسی می‌نمائیم.

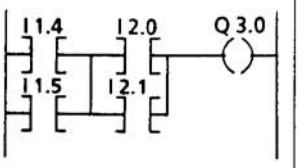
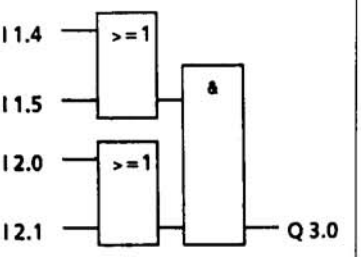
مثال ۳-۶ : در این مثال عدم استفاده از پرانتز در حالتی که دستور AND قبل از دستور OR قرار گرفته باشد مورد بررسی قرار می‌گیرد.

نماد مداری	توضیح
	خروجی Q 3.1 برابر "۱" خواهد بود هرگاه حاصل حداقل یکی از دو تابع AND برابر "۱" شود. این خروجی برابر "۰" است درصورتی که حاصل هیچ‌یک از دو تابع AND "۱" نباشد.

LAD	CSF	STL
		A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q 3.1

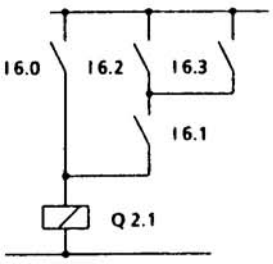
مثال ۳-۷: در این مثال چگونگی استفاده از پرانتز در حالتی که دستور OR قبل از دستور AND قرار گرفته باشد نشان داده می‌شود.

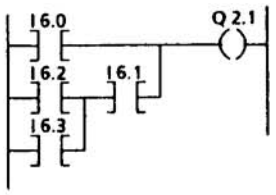
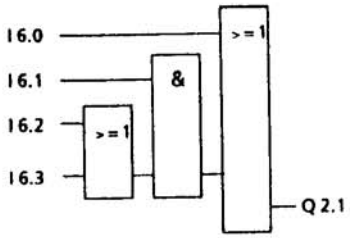
نماد مداری	توضیح
	خروجی Q 3.0 برابر "۱" خواهد بود اگر حاصل هر دو تابع OR برابر "۱" شود. این خروجی برابر "۰" خواهد بود در صورتی که حاصل حداقل یکی از دو تابع OR "۰" شود.

LAD	CSF	STL
		A(O I 1.4 O I 1.5) A(O I 2.0 O I 2.1) = Q 3.0

جهت روشن شدن مطلب و تمرین بیشتر، مثالهای دیگری ارائه می‌گردد.

مثال ۳-۸ : در این مثال نیز چگونگی کاربرد پرانتزها به همراه توابع منطقی AND و OR بررسی می‌شود.

نماد مداری	توضیح
	خروجی Q 2.1 برابر "۱" خواهد بود هر گاه حداقل یکی از شرایط زیر برقرار باشد: * ورودی I 6.0 برابر "۱" باشد. * ورودی I 6.1 و حداقل یکی از دو ورودی I 6.2 و I 6.3 برابر "۱" باشند. مقدار این خروجی برابر "۰" است اگر حاصل هیچ یک از توابع AND برابر "۱" نباشد.

LAD	CSF	STL
		O I 6.0 O A I 6.1 A(O I 6.2 O I 6.3) = Q 2.1

لازم به ذکر است که نوشتن این‌گونه برنامه‌ها بدون استفاده از پرانتز و تنها با به کارگیری فلگ‌ها نیز امکان‌پذیر است. قبل از برنامه‌نویسی این قبیل برنامه‌ها توسط فلگ‌ها، به ذکر جزئیاتی در مورد فلگ می‌پردازیم.

۳-۱۰- فلگ یا پرچم (Flag)

هر فلگ یا پرچم یک بیت از حافظهٔ PLC است که آن را می‌توان معادل رلهٔ داخلی مدار فرمان دانست. این بیت مانند هر بیت حافظه می‌تواند دو مقدار "۰" یا "۱" را بپذیرد. فلگ‌ها حافظه‌های موقتی هستند که CPU هنگام اجرای برنامه از آنها به عنوان دفترچهٔ یادداشتِ نتایج منطقی و یا حالت سیگنال‌ها استفاده می‌کند.

فلگ‌ها در برنامه‌نویسی نقش دستیار یا برگ چرک‌نویس را بازی می‌کنند یعنی مجموعه‌ای از نتایج اعمال منطقی در آنها قرار داده می‌شود. تمام فلگ‌ها در ناحیه‌ای از حافظه به نام Flag Area یا Flag Bytes قرار دارند. فلگ‌ها نیز مانند ورودی‌ها و خروجی‌ها به دسته‌های هشت بیتی (یک بایتی) تقسیم‌بندی می‌شوند.

ظرفیت محیط Flag Area نیز بستگی به نوع PLC دارد. در صورت نیاز به اطلاعات موجود در فلگ باید آن را از حافظه فرا بخوانیم.

به عنوان مثال دستور A F 6.0 در برنامه‌نویسی به روش STL مقدار بیت صفرم را از بایت ششم در محیط Flag Area فرا می‌خواند. کاربرد عمدهٔ فلگ‌ها در پاسخ به سؤال زیر مشخص می‌گردد.

- در صورتی که به یک سری اطلاعات در محل‌های مختلف و همچنین در زمانهای مختلف نیاز داشته باشیم از چه ابزاری می‌توان استفاده نمود؟

در پاسخ به این سؤال می‌توان گفت: در مدارات فرمان رله کنتاکتوری می‌توان از رله‌های رابط استفاده نمود. بدین ترتیب که مثلاً رلهٔ اول، رلهٔ دوم را فعال نموده، سپس رلهٔ دوم، رلهٔ سوم را و به همین ترتیب تا جایی که رلهٔ مورد نظر در محل و زمان مورد نظر فعال گردد. اشکال این مدار، پر هزینه و پر حجم شدن مدار فرمان می‌باشد. برای رفع این مشکل در PLCها از فلگ‌ها استفاده می‌شود. بدین ترتیب که نتیجهٔ به‌دست آمده (اطلاعات مورد نظر) توسط PLC در فلگ خاصی ذخیره شده، PLC در محل و زمان مقتضی آن را از حافظه فرا می‌خواند.

کاربرد دیگر فلگ در برنامه‌هایی است که چندین OR و AND وجود داشته و دستور OR قبل از دستور AND استفاده شده باشد. با استفاده از فلگ‌ها می‌توان پرانتزها را حذف نمود. البته با این عمل ممکن است در برخی موارد برنامهٔ مورد نظر طولانی‌تر گردد.

هنگامی که در برنامه‌ای حاصل یک دسته از اعمال منطقی باید با حاصل دستهٔ دیگری از اعمال منطقی ترکیب گردد، حاصل هر دسته را در فلگ‌های جداگانه‌ای قرار داده، سپس این فلگ‌ها را با یکدیگر ترکیب می‌نماییم. بنابراین وظیفه‌ای که پرانتزها در برنامه‌نویسی به روش STL بر عهده دارند می‌تواند با به کارگیری فلگ‌ها انجام گیرد.

همچنین هنگامی که باید قسمتی از برنامه چندین بار تکرار شود، می‌توان حاصل آن قسمت را که در حقیقت حاصل چند عمل منطقی است در یک فلگ قرار داد و با این عمل از تکرار آن بخش از برنامه و در نتیجه از طولانی شدن برنامه جلوگیری نمود.

برای روشن شدن مطالب عنوان شده و چگونگی کاربرد فلگ‌ها به جای پرانتزها مثال ۳-۷ با به کارگیری فلگ‌ها بازنویسی می‌شود.

بازنویسی مثال ۳-۷ با استفاده از فلگ‌ها : برنامهٔ نوشته شده زیر را در نظر بگیرید.

```
 PB 11

SEGMENT  1              0000
0000      :A(
0001      :O    I       1.4
0002      :O    I       1.5
0003      :)
0004      :A(
0005      :O    I       2.0
0006      :O    I       2.1
0007      :)
0008      :=    Q       3.0
0009      :BE
```

حال قصد آن داریم تا با استفاده از فلگ‌ها، پرانتزها را حذف نماییم.

```
: O   I   1.4
: O   I   1.5
: =   F   6.0
```
حاصل دستهٔ اول از اعمال منطقی در F 6.0 قرار می‌گیرد.

```
: O   I   2.0
: O   I   2.1
: =   F   6.1
```
حاصل دستهٔ دوم از اعمال منطقی در F 6.1 قرار می‌گیرد.

```
: A   F   6.0
: A   F   6.1
: =   Q   3.0
```
حاصل ترکیب عطفی دو فلگ F 6.0 و F 6.1 در خروجی Q 3.0 قرار داده می‌شود.

۳-۱۱- بیت RLO[1]

هنگامی که PLC اجرای برنامه‌ای را آغاز می‌کند مقدار عملوند یا سطر اول برنامه (مثلاً مقدار ورودی) را در بیت بخصوصی که به RLO موسوم است قرار می‌دهد و در اجرای سطر بعدی، RLO را با عملوند بعدی مطابق برنامه، ترکیب می‌کند و مجدداً حاصل را در RLO قرار می‌دهد. این عمل تا زمانی ادامه پیدا می‌کند که در سطری از برنامه به دستور هم‌ارزی (=) برسد. پس از انجام این عمل یعنی انتساب بیت RLO به عملوند موجود در سطر هم‌ارزی، RLO مقدار خود را از دست داده، پذیرای مقدار جدیدی می‌گردد. لذا مجدداً مقدار عملوند سطر بعد از عمل هم‌ارزی در RLO قرار می‌گیرد و PLC ، این روند را تا پایان برنامه ادامه می‌دهد.

۳-۱۲- ست و ری‌ست در فلگ‌ها و خروجی‌ها

همان‌گونه که در فصل اول اشاره شد در مدارات ترتیبی از فلیپ‌فلاپ استفاده می‌گردد. فلیپ‌فلاپ دارای دو ورودی S (Set) و R (Reset) می‌باشد. در برخی شرایط کنترلی خاص به دلیل ایمنی و یا دلایل دیگر لازم است فقط در یک لحظه کلیدی فعال شده یا دگمه‌ای (Push Button) فشار داده شود تا دستگاهی شروع به کار نموده یا متوقف گردد. در این صورت باید از دستورات ست و ری‌ست در فلیپ‌فلاپ‌ها استفاده نمود. کاربرد این دستورات در برنامه‌ها بسیار زیاد است. فلیپ‌فلاپ‌های مورد استفاده در برنامه‌نویسی PLC به یکی از دو نوع زیر می‌باشند:

---

1 - Result of Logic Operation

۱- فلیپ‌فلاپ SR

۲- فلیپ‌فلاپ RS

در شکل ۳-۶ بلوک این دو نوع فلیپ‌فلاپ در روش برنامه‌نویسی CSF را می‌بینیم.

فلیپ‌فلاپ SR فلیپ‌فلاپ RS

شکل ۳-۶: طرز نمایش دو نوع فلیپ‌فلاپ در روش CSF

تفاوت این دو نوع فلیپ‌فلاپ تنها در ارجحیت ورودی‌های ست و ری‌ست می‌باشد که در این مورد به طور مفصل در همین بخش سخن خواهیم گفت. صورتهای مختلف نمایش فلیپ‌فلاپ‌ها به روش STL در ادامه نشان داده شده است.

فلیپ‌فلاپ SR

```
PB 16

SEGMENT  1              0000
0000      :A    I       1.1
0001      :S    Q       2.0
0002      :A    I       1.2
0003      :R    Q       2.0
0004      :BE
```

فلیپ‌فلاپ RS

```
PB 17

SEGMENT  1              0000
0000      :A    I       1.2
0001      :R    Q       2.0
0002      :A    I       1.1
0003      :S    Q       2.0
0004      :BE
```

طرز کار فلیپ‌فلاپ SR به صورت زیر است:

در صورتی که ورودی مربوط به ترمینال R در وضعیت "۰" باشد کافی است در یک لحظه ورودی مربوط به ترمینال S برابر "۱" شود تا خروجی Q 2.0 به طور پایدار در وضعیت "۱" قرار گیرد. خروجی Q 2.0، این وضعیت را تا تغییر وضعیت در ترمینال R حفظ خواهد نمود.

در صورتی که ورودی مربوط به ترمینال S در وضعیت "۰" باشد کافی است در یک لحظه ورودی مربوط به ترمینال R برابر "۱" شود تا خروجی Q 2.0 به صورت پایدار در وضعیت "۰" قرار گیرد. خروجی Q 2.0، این وضعیت را تا تغییر وضعیت ترمینال S حفظ خواهد نمود.

در صورتی که ورودی‌های مربوط به ترمینال‌های S و R همزمان فعال (برابر "۱") شوند دومین دستور تفوق خواهد داشت یعنی در فلیپ‌فلاپ SR ارجحیت با R و در فلیپ‌فلاپ RS ارجحیت با S خواهد بود. دلیل این امر آن است که PLC دستورات را به دنبال هم اجرا می‌کند و به محض اینکه دستور اول را اجرا نمود بلافاصله دستور دوم را که نقض کنندهٔ دستور اول است به اجرا در می‌آورد و در نتیجه دستور اول خنثی شده، دستور دوم باقی خواهد ماند. پس به عنوان یک اصل و در حالت کلی می‌پذیریم که:

هر دستوری که به خط انتهایی برنامه (BE) نزدیکتر باشد از نظر اجرایی ارجح است.

در برخی موارد انتقال یک فرمان به خروجی از طریق یک فلگ انجام می‌پذیرد. برنامهٔ نوشته شده در PB 18 این مطلب را بیان می‌کند.

```
 PB 18

SEGMENT  1              0000
0000       :A    I      0.0
0001       :S    F      3.0
0002       :A    I      0.1
0003       :R    F      3.0
0004       :A    F      3.0
0005       :=    Q      2.0
0006       :BE
```

```
 PB 18

SEGMENT  1              0000
                F 3.0
               +------+
 I 0.0      --!S     !
               !      !    +-------+
 I 0.1      --!R    Q!-+-! =     ! Q 2.0
               +------+    +-------+:BE
```

```
 PB 18

SEGMENT  1              0000
!              F 3.0
!I 0.0        +------+
+---] [---+-!S     !
!         !  !      !
!I 0.1     !  !      !              Q 2.0
+---] [---+-!R    Q!-+----------+--(    )-!
!         !  +------+
!                                 :BE
```

مبحث فلیپ‌فلاپ‌ها را با پاسخ به یک سؤال ادامه می‌دهیم.

- تفاوت بین این دو برنامه در چیست؟

```
 PB 19

SEGMENT  1              0000
0000        :A    I     1.0
0001        :S    Q     3.0
0002        :BE
```

(الف)

```
 PB 20

SEGMENT  1              0000
0000        :A    I     1.0
0001        :=    Q     3.0
0002        :BE
```

(ب)

در برنامهٔ (الف) جهت فعال شدن خروجی Q 3.0 کافی است که ورودی I 1.0 فقط یک لبهٔ بالارونده داشته باشد یا به صورت یک پالس برای یک لحظه ظاهر گردد. به محض "۱" شدن مقدار ورودی I 1.0، خروجی Q 3.0 نیز "۱" خواهد شد و در صورتی که ورودی I 1.0 به وضعیت "۰" تغییر حالت دهد خروجی باز در وضعیت "۱" باقی خواهد ماند. (البته این مطلب تا زمانی صادق

است که منبع برق PLC تأمین شده باشد و یا اینکه خروجی Q 3.0 با ورودی دیگر ری‌ست نگردد.)

در برنامهٔ (ب) جهت فعال شدن خروجی Q 3.0 لازم است تا ورودی I 1.0 نیز فعال باشد و وضعیت ورودی I 1.0 مستقیماً بر روی خروجی Q 3.0 تأثیرگذار خواهد بود و به محض این که I 1.0 به وضعیت "۰" تغییر حالت دهد خروجی Q 3.0 نیز به وضعیت "۰" تغییر مقدار می‌دهد.

خوانندگان به تفاوت بین این دو برنامه به خوبی توجه داشته باشند چرا که در برخی موارد نادیده گرفتن چنین تفاوتهایی در برنامه‌نویسی ممکن است باعث ایجاد زیانهای جبران ناپذیری در پروسهٔ تحت کنترل گردد. بسیاری از مشکلات موجود در خطوط تولید و فرآیندهای صنعتی به دلیل کم توجهی یا بی‌توجهی به برخی نکات جزیی مانند تفاوت مذکور می‌باشد.

۳-۱۳- دستور NOP 0[1]

یک فلیپ‌فلاپ SR را در نظر گرفته، برنامه‌ای جهت ست و ری‌ست نمودن آن می‌نویسیم.

```
 PB 21

SEGMENT  1            0000
0000      :A   I      1.0
0001      :S   Q      2.0
0002      :A   I      1.1
0003      :R   Q      2.0
0004      :A   Q      2.0
0005      :=   Q      2.1
0006      :BE

 PB 21

SEGMENT  1            0000
            Q 2.0
             +------+
 I 1.0     --!S     !
             !      !    +-------+
 I 1.1     --!R    Q!-+-! =     ! Q 2.1
             +------+    +-------+:BE
```

1 - No Operation Zero

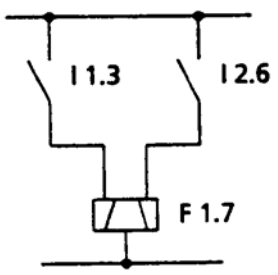
حال در صورتی که بخواهیم از خروجی، هیچ استفاده‌ای نکنیم یا به عبارت دیگر سطرهای پنجم و ششم از برنامهٔ نوشته شده به روش STL را حذف نماییم به گونه‌ای که مقادیر ورودی در ترمینال‌های R و S به خروجی منتقل نشوند از دستور NOP 0 استفاده می‌نماییم و با استفاده از این دستور برنامهٔ قبلی را بازنویسی می‌کنیم.

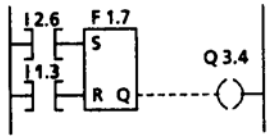
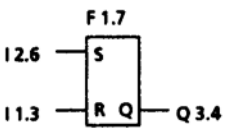
```
 PB 22

SEGMENT  1              0000
0000       :A    I      1.0
0001       :S    Q      2.0
0002       :A    I      1.1
0003       :R    Q      2.0
0004       :NOP  0
0005       :BE
```

```
 PB 22

SEGMENT  1              0000
                Q 2.0
                +-----+
 I 1.0      --!S    !
                !    !
 I 1.1      --!R   Q!-
                +-----+ :BE
```

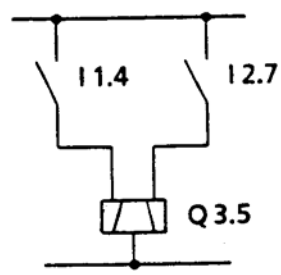
دستور NOP 0 دستوری مختص PLCهای زیمنس می‌باشد. با استفاده از این دستور در برنامه‌نویسی به روش STL (تنها در صورت عدم استفاده از بخشی از برنامه می‌توان دستور NOP 0 را جایگزین آن بخش از برنامه نمود) می‌توان برنامهٔ نوشته شده به روش STL را به LAD و CSF تبدیل نمود. در آینده از این دستور در تعریف تایمرها، شمارنده‌ها و ... مکرراً استفاده خواهیم نمود.

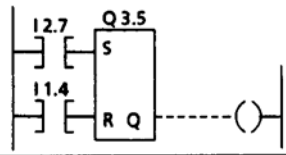
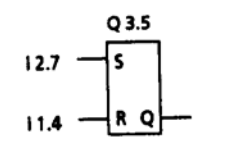
مثال ۳-۹ : در این مثال به بررسی برنامهٔ نوشته شده برای یک فلیپ فلاپ می‌پردازیم.

نماد مداری	توضیح
	وجود یک لبهٔ بالا رونده در ورودی I 2.6 ، F 1.7 را ست می‌کند. در این حالت در صورتی که سیگنال ورودی I 2.6 به "۰" تغییر وضعیت دهد مقدار F 1.7 بدون تغییر باقی خواهد ماند. در صورت وجود لبهٔ بالا رونده در ورودی I 1.3 ، F 1.7 ری‌ست خواهد شد و در صورت تغییر وضعیت I 1.3 به "۰" مقدارفلگ ثابت خواهد ماند. در هر دو حالت فوق مقدار موجود در فلگ F 1.7 به خروجی Q 3.4 نسبت داده می‌شود.

LAD	CSF	STL
		A I 2.6 S F 1.7 A I 1.3 R F 1.7 A F 1.7 = Q 3.4

مثال ۳-۱۰ : در این مثال کاربرد دستور NOP 0 در برنامه‌نویسی یک فلیپ‌فلاپ بررسی می‌گردد.

نماد مداری	توضیح
	در صورت وجود لبهٔ بالا رونده در ورودی I 2.7 فلیپ‌فلاپ Q 3.5 ست خواهد شد و در صورت تغییر وضعیت این ورودی به "۰" خروجی Q 3.5 ثابت می‌ماند. لبهٔ بالا رونده در ورودی I 1.4 فلیپ‌فلاپ را ری‌ست می‌نماید و تغییر وضعیت این ورودی به "۰" در مقدار فلیپ‌فلاپ تغییری ایجاد نمی‌کند. در اینجا از خروجی هیچ‌گونه استفاده‌ای نمی‌شود.

LAD	CSF	STL
		A I 2.7 S Q 3.5 A I 1.4 R Q 3.5 NOP 0

حال برای روشن شدن مطالب عنوان شده در مورد تقدم و تأخر اجرایی ورودی‌های S و R در فلیپ‌فلاپ‌ها و چگونگی تأثیر آنها در خروجی، مثال زیر را که انجام آن به عنوان تمرین به خوانندگان واگذار شده است در نظر بگیرید.

مثال ۳-۱۱ : بلوک برنامهٔ زیر را در نظر بگیرید. این برنامه در ۶ بخش یا Segment و به دو روش STL و LAD نوشته شده است. پس از درک عملکرد برنامه، جداول مربوط را که شامل نتایج حاصل از مقادیر خروجی و همچنین بیت RLO است تکمیل نمائید.

در این جداول شرایط اولیه‌ای برای فلیپ‌فلاپ در نظر گرفته شده است که روند عملیات را مشخص می‌نماید. در ضمن در برخی از موقعیتها (STATES) در مورد سه حالت ورودی (مثلاً ۱) مقدار خروجی خواسته شده است.

```
 PB 25

SEGMENT  1            0000
0000      :A    I     16.0
0001      :S    F     10.0
0002      :A    I     16.1
0003      :R    F     10.0
0004      :A    F     10.0
0005      :=    Q      9.5
0006      :***

SEGMENT  2            0007
0007      :A    I     16.2
0008      :S    Q      9.7
0009      :AN   I     16.3
000A      :R    Q      9.7
000B      :A    Q      9.7
000C      :=    Q      9.6
000D      :***

SEGMENT  3            000E
000E      :A    I     16.6
000F      :R    Q     10.0
0010      :AN   I     16.7
0011      :S    Q     10.0
0012      :A    Q     10.0
0013      :=    F     10.2
0014      :***
```

```
SEGMENT  4            0015
0015      :A    I     17.0
0016      :R    Q     10.1
0017      :A    I     17.1
0018      :S    Q     10.1
0019      :NOP  0
001A      :***


SEGMENT  5            001B
001B      :A    I     17.3
001C      :S    Q     10.2
001D      :***


SEGMENT  6            001E
001E      :A    I     17.4
001F      :R    Q     10.2
0020      :BE
```

PB 25

```
SEGMENT  1            0000
!           F 10.0
!I 16.0     +-----+
+---] [---+-!S    !
!         !       !
!I 16.1   !       !              Q 9.5
+---] [---+-!R   Q!-+----------+--(    )-!
!         +-----+
!
SEGMENT  2            0007
!           Q 9.7
!I 16.2     +-----+
+---] [---+-!S    !
!         !       !
!I 16.3   !       !              Q 9.6
+---]/[---+-!R   Q!-+----------+--(    )-!
!         +-----+
!
```

```
SEGMENT  3          000E
!           Q 10.0
!I 16.6     +-----+
+---] [---+-!R    !
!         !       !
!I 16.7   !       !              F 10.2
+---]/[---+-!S    Q!-+---------+--(    )-!
!         +-----+
!
SEGMENT  4          0015
!           Q 10.1
!I 17.0     +-----+
+---] [---+-!R    !
!         !       !
!I 17.1   !       !
+---] [---+-!S    Q!-
!         +-----+
!
SEGMENT  5          001B
!
!I 17.3                            Q 10.2
+---] [---+---------+---------+--(S   )-!
!
SEGMENT  6          001E
!
!I 17.4                            Q 10.2
+---] [---+---------+---------+--(R   )-!
!
!
!                                  :BE
!
```

جدول مربوط به مثال ۳-۱۱

	OPERAND	INITIAL STATE	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO
SEGMENT 1	I 16.0	۰	۱ ۰		۰		۱		۱		۰		۱		۱		۰		۰		۰	
	I 16.1	۰	۰		۰ ۱		۰		۱		۱		۱		۰		۰		۰ ۱			
	Q 9.5																					

ادامهٔ جدول مربوط به مثال ۳-۱۱

	OPERAND	INITIAL STATE	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO	STATE	RLO
SEGMENT 2	I 16.2	۰	۰/۱		۰		۱		۱		۰		۱		۱		۰		۰	
	I 16.3	۱	۱		۰/۱		۱		۰		۰		۰		۱		۱		۰/۱	
	Q 9.7																			
	Q 9.6																			
SEGMENT 3	I 16.6	۰	۰		۰/۱		۰		۱		۱		۱		۰		۰		۰/۱	
	I 16.7	۱	۰/۱		۱		۰		۰		۱		۰		۰		۱		۱	
	Q 10.0																			
	F 10.2																			
SEGMENT 4	I 17.0	۰	۰		۰/۱		۰		۱		۱		۱		۰		۰		۰/۱	
	I 17.1	۰	۰/۱		۰		۱		۱		۰		۱		۱		۰		۰	
	Q 10.1																			
SEGMENT 5	I 17.3	۰			۰		۱		۱		۰		۱		۱		۰		۰	
	Q 10.2																			
SEGMENT 6	I 17.4	۰	۰		۰/۱		۰		۱		۱		۱		۰		۰		۰	
	Q 10.2																			

۳-۱۴- کانکتور (Connector)

در برخی موارد لازم است که برقراری بعضی شرایط و یا مقدار یک خروجی را به چند خروجی دیگر نیز اعمال کنیم. برای این عمل از کانکتور یا جعبه تقسیم استفاده می‌نمائیم. در برنامه‌نویسی به روش LAD و CSF این نماد با --(#)-- نشان داده می‌شود اما در روش STL مقدار موجود در کانکتور را در یک فلگ قرار داده و در صورت احتیاج، آن را به خروجی‌های مختلف نسبت می‌دهیم. در مثال زیر کاربرد کانکتور در برنامه‌نویسی تشریح شده است.

مثال ۳-۱۲ : در برخی از فرآیندهای کنترلی گاه لازم است دستگاهها و سیستم‌های مشابه به صورت همزمان با برقراری شرایطی خاص مانند شرایط ایمنی و ... فعال شوند. در این گونه برنامه‌ها برای جلوگیری از تکرار خطوط برنامه می‌توان این شرایط را در یک کانکتور قرار داد و سپس هر سیستمی که برای فعال شدن به شرایط مذکور نیاز داشته باشد می‌تواند از "۱" بودن مقدار کانکتور استفاده نماید. در این مثال عملکرد کانکتور را بررسی می‌کنیم.

```
 PB 26

SEGMENT  1          0000
!
!I 0.0     I 0.1     I 0.2     I 0.3     I 0.4     F 100.0
+---] [---+---] [---+---] [---+---] [---+---] [---+--( # )-!
!
!
!                                                    :BE
!
```

که ورودی‌ها به صورت زیر تعریف شده‌اند:

I 0.0	روشن بودن پمپ روغن
I 0.1	بالا بودن درجه حرارت روغن
I 0.2	فعال بودن فاز ۱
I 0.3	فعال بودن فاز ۲
I 0.4	فعال بودن فاز ۳

حال در صورت نیاز به استفاده از این شرایط در دستگاه می‌توان از F 100.0 برای فعال شدن خروجی آن دستگاه استفاده نمود. این روش در خلاصه‌نمودن برنامه بسیار مؤثر است.

۳-۱۵- برنامه‌نویسی یک تشخیص دهندهٔ لبهٔ پالس (Edge Detector)

در برخی موارد و در حین اجرای برنامه لازم است تا عملی در طول اجرای نرم‌افزار تنها برای مدت یک سیکل زمانی (Cycle Time) انجام شود. در این شرایط باید از تشخیص دهندهٔ پالس استفاده کنیم.

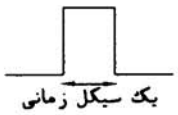
برنامهٔ زیر به روش LAD و با استفاده از یک فلیپ‌فلاپ RS و یک کانکتور نوشته شده است.

```
 PB 27

SEGMENT  1          0000
!                                        F 6.0
!I 0.0      F 6.0      F 100.0          +-----+
+---] [---+---] [---+---(#)---+-!R      !
!                                !       !
!I 0.0                           !       !
+---]/[---+---------+---------+-!S   Q!-
!                                +-----+
!
!                                              :BE
!
```

طرز عملکرد این برنامه به صورت زیر است:

فرض کنید در ابتدا I 0.0 برابر "۰" باشد بنابراین ترمینال S فعال شده، فلگ F 6.0 را ست می‌کند. در این حالت مقدار F 6.0 برابر "۱" می‌باشد. حال مقدار ورودی I 0.0 را از "۰" به "۱" تغییر وضعیت می‌دهیم بنابراین ترمینال R فعال می‌شود (زیراکه هر دو ورودی آن یعنی I 0.0 و F 6.0 برابر "۱" هستند) در این حالت فلگ F 6.0 ری‌ست می‌شود و خواهیم داشت: "۰" = F 6.0 .

در اجرای این برنامه در فلگ F 100.0 لبهٔ تیزی در یک سیکل زمانی خواهیم داشت.



لبهٔ تیز بوجود آمده در F 100.0

به دلیل بالا بودن سرعت CPU ، پالس ایجاد شده، به سرعت تغییر وضعیت می‌دهد به طوری که تغییر حالت آن از "۰" به "۱" و سپس از "۱" به "۰" برای ما قابل رویت نیست. بنابراین جهت

آشکارسازی این پالس، می‌توان F 100.0 را به ترمینال S یک فلیپ‌فلاپ RS اعمال نمائیم. در زیر این عمل انجام شده است.

```
 PB 28

SEGMENT  1              0000
!              Q 3.0
!I 1.7        +------+
+---] [---+-!R      !
!         !  !      !
!F 100.0   !  !      !              Q 3.0
+---] [---+-!S    Q!-+----------+--(    )-!
!         !  +------+
!
!                                  :BE
!
```

در این حالت در صورتی که ورودی I 0.0 از "۰" به "۱" تغییر وضعیت دهد در لحظهٔ کوتاهی فلگ F 100.0 برابر "۱" شده، خروجی Q 3.0 را ست خواهد نمود. در این برنامه از I 1.7 تنها جهت ری‌ست نمودن خروجی Q 3.0 استفاده شده است.

با اندکی دقت در طرز عملکرد این برنامه می‌توان گفت که از فلگ F 100.0 می‌توان به عنوان کلید General Reset استفاده نمود.

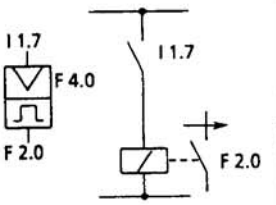
افرادی که در محیط‌های صنعتی با فرآیندهای صنعتی کوچک و بزرگ سروکار دارند به خوبی می‌دانند که در برخی از سیستم‌ها جهت ری‌ست نمودن تمامی موارد (مواردی نظیر آلارم‌ها و ...) از کلیدی با نام General Reset و یا Common Reset استفاده می‌گردد.

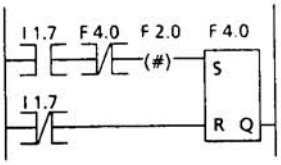
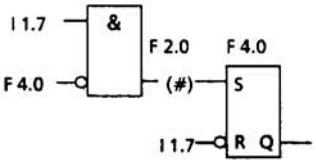
در ادامه، این دو برنامه به روش STL آورده شده‌اند.

```
 PB 27

SEGMENT  1              0000
0000      :A    I       0.0
0001      :A    F       6.0
0002      :=    F     100.0
0003      :A    F     100.0
0004      :R    F       6.0
0005      :AN   I       0.0
0006      :S    F       6.0
0007      :NOP  0
0008      :BE
```

```
PB 28

SEGMENT   1                0000
0000        :A     I         1.7
0001        :R     Q         3.0
0002        :A     F       100.0
0003        :S     Q         3.0
0004        :A     Q         3.0
0005        :=     Q         3.0
0006        :BE
```

همین برنامه یعنی برنامهٔ تشخیص لبهٔ پالس را با استفاده از فلیپ‌فلاپ SR بازنویسی می‌کنیم.

نماد مداری	توضیح
	درصورت‌وجودلبهٔ‌بالارونده‌در I 1.7 شرایط‌لازم‌برای "۱" شدن ترکیب‌عطفی I 1.7 ونقیض F 4.0 مهیا می‌شود و F 2.0 و F 4.0 ست خواهند شد. در اجرای‌سیکل‌بعدی‌حاصل‌ترکیب‌عطفی I 1.7 ونقیض F 4.0 برابر "۰" می‌باشدزیرادرسیکل‌قبلی F 4.0 ست‌شده‌بود. در این حالت F 2.0 ری‌ست می‌شود. بنابراین F 2.0 تنهادرخلال‌اجرای‌یک‌مرتبهٔ‌برنامه "۱" خواهد شد. در صورتی که "۰" = I 1.7 شود، F 4.0 ری‌ست می‌شود.

LAD	CSF	STL
		A I 1.7 AN F 4.0 = F 2.0 A F 2.0 S F 4.0 AN I 1.7 R F 4.0 **NOP 0**

روش دیگری جهت برنامه‌نویسی تشخیص دهندهٔ لبهٔ پالس وجود دارد. در روش مذکور از فلیپ‌فلاپ استفاده نمی‌شود. این برنامه کاملاً کاربردی بوده، در بسیاری از پروسه‌ها مورد استفاده

قرار می‌گیرد. برنامهٔ مذکور در PB 30 نوشته شده است.

```
 PB 30

SEGMENT  1              0000
0000       :A    I      1.0
0001       :AN   F      6.1
0002       :=    F    100.1
0003       :A    I      1.0
0004       :=    F      6.1
0005       :BE
```

در این برنامه در F 100.1 یک لبهٔ تیز خواهیم داشت.

مثال ۳-۱۳ : فرض کنید در کنترل یک پروسهٔ صنعتی یک پوش‌باتن (Push Button) وجود دارد. می‌خواهیم برنامه‌ای جهت کنترل خروجی مرتبط با این پوش‌باتن بنویسیم به گونه‌ای که:

اگر پوش‌باتن را فشار دهیم خروجی را فعال نماید و در صورتی که آن را رها کنیم خروجی در همان حالت قبلی (فعال) بماند. اگر برای بار دوم پوش‌باتن را فشار دهیم، خروجی غیرفعال شود و اگر باز آن را رها کنیم خروجی در همان حالت (غیرفعال) باقی بماند و به همین ترتیب ... .

قبل از نوشتن برنامه برای درک بیشتر عملکرد این پوش‌باتن به ذکر یک مثال در زندگی روزمرهٔ خود می‌پردازیم:

سوئیچ روشن و خاموش کردن تلویزیون را در نظر بگیرید. طرز کار این سوئیچ فشاری بدین ترتیب است که:

اگر تلویزیون خاموش باشد و این سوئیچ فشرده شود تلویزیون روشن می‌شود و در صورتی که دست را از روی سوئیچ برداریم باز تلویزیون روشن باقی می‌ماند. در صورتی که برای بار دوم سوئیچ فشرده شود تلویزیون خاموش می‌شود و در صورتی که سوئیچ را رها نمائیم تلویزیون در همان حالت خاموش باقی می‌ماند.

با اندکی دقت در مطالب عنوان شده در متن مثال در می‌یابیم که به دلیل تغییر حالت دوگانه باید از دو فلیپ‌فلاپ استفاده کنیم که این دو باید توانایی تأثیرگذاری بر فلیپ‌فلاپ دیگر را داشته باشند، به بیان دیگر بتوانند فلیپ‌فلاپ دیگر را فعال یا غیرفعال نمایند.

بنابراین استفاده از فلگ جهت ست و ری‌ست شدن فلیپ‌فلاپ‌ها الزامی است.

برنامهٔ نوشته شده به صورت زیر می‌باشد: (I 0.0 در تمامی موارد نشان دهندهٔ ورودی پوش باتن است.)

```
 PB 31

SEGMENT  1          0000
0000      :A   I     0.0
0001      :AN  F     8.1
0002      :S   F     8.0
0003      :A   I     0.0
0004      :A   F     8.1
0005      :R   F     8.0
0006      :A   F     8.0
0007      :=   Q     2.0
0008      :***

SEGMENT  2          0009
0009      :AN  I     0.0
000A      :A   F     8.0
000B      :S   F     8.1
000C      :AN  I     0.0
000D      :AN  F     8.0
000E      :R   F     8.1
000F      :NOP 0
0010      :BE
```

```
 PB 31

SEGMENT  1          0000
            +---+    F 8.0
 I 0.0    ---! & !    +-----+
 F 8.1    --O!   !----!S    !
            +---+    !      !
            +---+    !      !
 I 0.0    ---! & !    !      !   +------+
 F 8.1    ---!   !----!R   Q!-+-! =    ! Q 2.0
            +---+    +-----+   +------+

SEGMENT  2          0009
            +---+    F 8.1
 I 0.0    --O! & !    +-----+
 F 8.0    ---!   !----!S    !
            +---+    !      !
            +---+    !      !
 I 0.0    --O! & !    !      !
 F 8.0    --O!   !----!R   Q!-
            +---+    +-----+  :BE
```

```
 PB 31

SEGMENT  1          0000
!                     F 8.0
!I 0.0      F 8.1      +------+
+---] [---+---]/[---+-!S     !
!                     !      !
!I 0.0      F 8.1     !      !  Q 2.0
+---] [---+---] [---+-!R    Q!-+--(    )-!
!                     +------+
!
SEGMENT  2          0009
!                     F 8.1
!I 0.0      F 8.0      +------+
+---]/[---+---] [---+-!S     !
!                     !      !
!I 0.0      F 8.0     !      !
+---]/[---+---]/[---+-!R    Q!-
!                     +------+
!
!                                  :BE
!
```

حال به تشریح عملکرد این برنامه می‌پردازیم:

نمایش نردبانی را درنظر بگیرید. فرض کنید که در حال حاضر خروجی Q 2.0 غیرفعال است. با فشار دادن پوش باتن یا به عبارت دیگر فعال شدن I 0.0، فلیپ‌فلاپ موجود در SEGMENT 1 یعنی فلگ F 8.0 و خروجی Q 2.0 ست می‌شوند. (از آنجایی که "۰" = F 8.1 می‌باشد بنابراین حاصل ترکیب عطفی نقیضِ آن و ورودی I 0.0 که در ترمینال S فلیپ‌فلاپ قرار گرفته‌اند F 8.0 را ست می‌کند.)

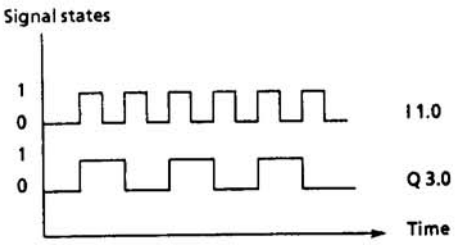
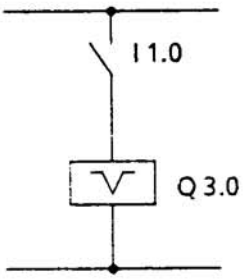
با رها کردن پوش باتن ورودی، I 0.0 "۰" خواهد شد و ترمینال S فلیپ‌فلاپ موجود در SEGMENT 2 فعال می‌شود. (زیرا ترمینال S این فلیپ‌فلاپ حاصل ترکیب عطفی F 8.0 و نقیض I 0.0 می‌باشد و از آنجایی که F 8.0 در مرحلهٔ قبل فعال شده بود حاصل ترکیب عطفی نقیض I 0.0 و F 8.1 برابر "۱" خواهد بود.) در این مرحله این فلیپ‌فلاپ ست می‌شود و خواهیم

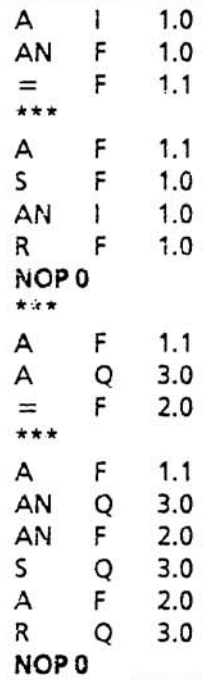
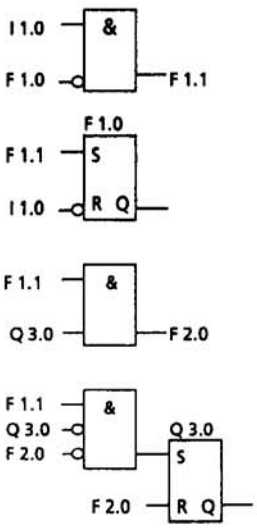
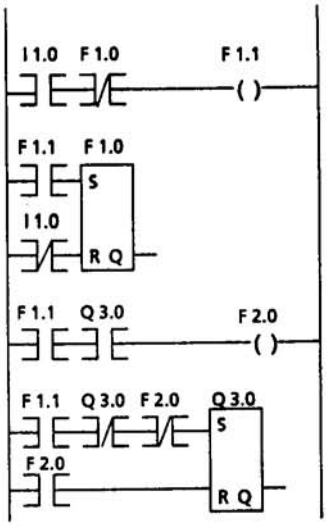
داشت: "۱" = F 8.1. در این حالت در صورتی که پوش باتن را مجدداً فشار دهیم، I 0.0 فعال شده و ترمینال R فلیپ‌فلاپ موجود در SEGMENT 1 نیز فعال می‌گردد (این ترمینال حاصل ترکیب عطفی F 8.1 و I 0.0 است که در این مرحله مقدار هر دو "۱" بوده، در نتیجه ترمینال R نیز "۱" خواهد شد) که موجب ری‌ست شدن این فلیپ‌فلاپ می‌گردد و مقدار F 8.0 و Q 2.0 برابر "۰" خواهد شد. در این حالت اگر دست را از روی پوش باتن برداریم I 0.0 برابر "۰" می‌گردد و ترمینال R فلیپ‌فلاپ موجود در SEGMENT 2 فعال شده، این فلیپ‌فلاپ را ری‌ست می‌کند. (زیرا در این مرحله F 8.0 و I 0.0 هر دو برابر "۰" بوده، ترکیب عطفی نقیض آنها برابر "۱" می‌باشد.) در این حالت فلگ F 8.1 ری‌ست شده، خواهیم داشت "۰" = F 8.1. در صورتی که مجدداً دست را بر روی پوش باتن فشار دهیم همین چرخه تکرار می‌شود.

همانگونه که عنوان شد به دلیل تغییر حالت دوگانه باید از دو فلیپ‌فلاپ استفاده کنیم. با دقت در برنامه‌ها ملاحظه می‌کنید که در هر سه روش، از دو قسمت یا SEGMENT برای برنامه‌نویسی استفاده شده است. عملوندهای استفاده شده در هر SEGMENT را می‌توان در قسمت‌های (SEGMENTهای) دیگر نیز استفاده نمود. در این گونه برنامه‌ها که معمولاً از چندین SEGMENT تشکیل می‌گردند بین هر دو قسمت از علامت *** که به مفهوم پایان یک SEGMENT و ادامهٔ برنامه در SEGMENT دیگر می‌باشد استفاده می‌شود. در این برنامه‌ها در انتهای قسمت آخر از دستور BE که به معنی پایان برنامه است استفاده می‌گردد.

در ضمن به دستور NOP 0 که در برنامه‌نویسی به روش STL استفاده شده دقت نمائید. استفاده از این دستور به دلیل عدم استفادهٔ برنامه‌نویس از خروجی فلیپ‌فلاپ موجود در SEGMENT 2 می‌باشد.

مثال ۳-۱۴ : در این مثال به برنامه‌نویسی "Binary Scaler" یا برنامهٔ نصف‌کنندهٔ فرکانس می‌پردازیم. همانگونه که در نمودار زمانی مشاهده می‌گردد فرکانس خروجی Q 3.0 نصف فرکانس ورودی I 1.0 می‌باشد. به عبارت دیگر در این برنامه، دورهٔ تناوب ورودی I 1.0 در خروجی Q 3.0 دو برابر شده است. بررسی عملکرد این برنامه را به عهدهٔ خواننده واگذار می‌کنیم.

نمودار زمانی	نماد مداری
	

STL	CSF	LAD
		

۳-۱۶- دستور پرش غیرشرطی (JU)[1]

قبل از اینکه به بحث پیرامون دستور JU بپردازیم به یادآوری بیت RLO می‌پردازیم:
در صورتی که به یاد داشته باشید در فصل دوم اشاره شد که نتیجهٔ عملکرد عملیات منطقی هر سطر برنامه در بیت RLO قرار می‌گیرد.

بنابراین، ارزش بیت RLO به نتیجهٔ عملیات منطقی سطر بستگی دارد. انجام برخی از دستورات به RLO وابسته (RLO Dependent) و برخی دیگر غیر وابسته‌اند (RLO Independent).

وابستگی یک دستور به RLO بدین معنی است که جهت اجرا شدن آن باید بیت RLO سطر قبلی "۱" باشد در غیر این صورت این دستور اجرا نمی‌شود. عدم وابستگی یک دستور به RLO بدین معنی است که این دستور صرفنظر از مقدار بیت RLO، اجرا می‌شود.

یکی از دستورات غیر وابسته به بیت RLO دستور JU است. همانگونه که از نام این دستور بر می‌آید دستور پرش غیرشرطی بدین معنی است که بدون وجود هرگونه شرطی، پرش و انتقال انجام می‌گیرد. این پرش ممکن است از یک بلوک به بلوک دیگر یا از یک سطر بلوک به سطر دیگر همان بلوک انجام گیرد.

۳-۱۷- دستور پرش شرطی (JC)[2]

اجرای این دستور برخلاف دستور JU وابسته به بیت RLO است. در این دستور، پرش هنگامی صورت می‌گیرد که بیت RLO مربوط به سطر قبلی "۱" باشد. این پرش نیز ممکن است از یک بلوک به بلوک دیگر و یا از یک سطر به سطرهای دیگر همان بلوک انجام گیرد.

مثال ۳-۱۵ : برنامه‌ای بنویسید که با فشار دادن یک کلید (فعال نمودن کلید)، یک بلوک برنامه مثلاً PB 18 اجرا و در صورت غیرفعال نمودن همان کلید بلوک دیگری مثلاً PB 19 اجرا شود.

با اندکی دقت در متن مثال در می‌یابیم که این برنامه باید در بلوک OB 1 نوشته شود زیرا همانگونه که اشاره شد ساختارکلی سیستم دراین بلوک تعیین می‌گردد. بنا به تعریف دو دستور JU و JC برای پرش باید ازدستور JC به همراه شرط‌فعال‌بودن یا غیرفعال‌بودن کلیدمربوطه‌استفاده نمائیم.

---

1 - Jump Unconditional 2 - Jump Conditional

حال فرض کنید کلید مورد نظر در مثال، ورودی I 0.0 باشد برای مشاهدهٔ عملکرد این برنامه در دو بلوک PB 18 و PB 19 دو برنامهٔ متفاوت می‌نویسیم. برنامه‌های نوشته شده در بلوک‌های OB 1، PB 18 و PB 19 به روش STL در ادامه آورده شده است.

```
 OB 1

SEGMENT  1            0000
0000      :A    I      0.0
0001      :JC   PB  18
0002      :AN   I      0.0
0003      :JC   PB  19
0004      :BE


 PB 18

SEGMENT  1            0000
0000      :A    I      1.1
0001      :A    I      1.2
0002      :=    Q      2.5
0003      :BE


 PB 19

SEGMENT  1            0000
0000      :A    I      1.3
0001      :A    I      1.4
0002      :=    Q      3.5
0003      :BE
```

حال به توصیف عملکرد برنامه می‌پردازیم.

در صورتی که I 0.0 فعال یعنی "۱" باشد بیت RLO نیز برابر "۱" است و در نتیجه، دستور پرش شرطی به PB 18 انجام می‌شود. پس از اینکه پردازنده به دستور BE در PB 18 رسید به سطر بعدی در OB 1 یعنی سطر بعدیِ خطی از برنامه که از آنجا عمل پرش صورت گرفته مراجعه می‌کند. چون I 0.0 فعال است پس نقیض آن و بیت RLO برابر "۰" می‌باشند بنابراین دستور پرش شرطی

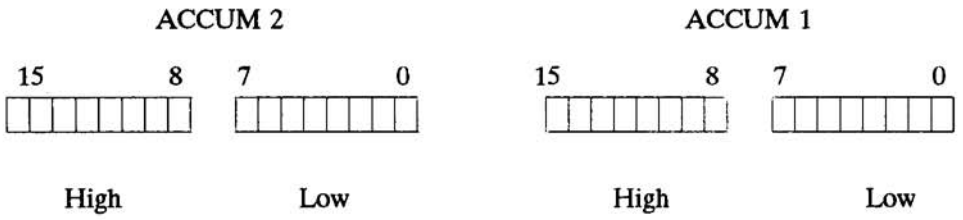
به PB 19 انجام نخواهد شد. سپس پردازنده به دستور BE در بلوک OB 1 رسیده، برنامه پایان یافته تلقی می‌شود. در اینجا مجدداً پردازنده به سطر اول بلوک OB 1 مراجعه نموده و در صورت فعال بودن I 0.0 همین روند مجدداً تکرار می‌شود. اما در صورتی که I 0.0 غیرفعال یعنی برابر "۰" باشد بیت RLO نیز "۰" است و دستور پرش مشروط به PB 18 نادیده فرض می‌شود. در این حالت نقیض ورودی I 0.0 برابر "۱" است بنابراین پرش بـه PB 19 انـجام مـی‌پذیرد. پس از پـردازش سطرهای موجوددربلوک PB 19،پردازنده به OB 1 باز می‌گردد و با مشاهدهٔ دستور BE در OB 1 ، پردازنده مجدداً به سطر اول بلوک OB 1 مراجعه می‌کند و این روند همچنان ادامه می‌یابد. برای بررسی صحت عملکرد این برنامه کافی است که تمام کلیدهای (ورودی‌های) I 1.4 ، I 1.3 ، I 1.2 و I 1.1 را فعال نمائیم. اکنون در صورتی که ورودی I 0.0 برابر "۱" باشد PB 18 انجام شده، تنها خروجی Q 2.5 فعال می‌شود و در صورتی که کلید I 0.0 غیرفعال باشد PB 19 انجام شده، یعنی تنها خروجی Q 3.5 فعال می‌شود.

با اندکی دقت در این برنامه ملاحظه می‌کنیم که این کلید یعنی ورودی I 0.0 می‌تواند نقش کلید Auto/Manual را در فرآیندهای صنعتی ایفا نماید. به عنوان مثال PB 18 شامل دستوراتی باشد که سیستم را در حالت Auto کنترل و PB 19 نیز مشتمل بر دستوراتی باشد که سیستم را در حالت Manual کنترل می‌نماید.

۳-۱۸- دستورهای بارگذاری و انتقال

سیستم‌های PLC در تایمرها، شمارنده‌ها و محاسبات، با اعداد سروکار دارند. وضعیت یک بایت ورودی، خروجی یا فلگ نیز می‌تواند معرف عددی در مبنای ۲ باشد. این اعداد می‌توانند بین قسمت‌های مختلف PLC (ورودی، خروجی، فلگ‌ها) مبادله شوند. جهت مبادلهٔ اعداد احتیاج به یک حافظهٔ واسطه می‌باشد. بنابراین قسمتی از حافظه که بـه آن آکـومولاتور یـا انـبارک (انـباره) می‌گوئیم به این عمل اختصاص داده شده است. آکومولاتورها از نوع ثبات (رجیستر) بوده و ۱۶ بیتی هستند. در بعضی از PLCها تنها از یک انبارک (ACCUM 1) و در برخی دیگر از دو انبارک (ACCUM 1 , ACCUM 2) استفاده شده است. در شکل ۳-۷ ساختار انبارک‌ها نشان داده شده است. همانگونه که مشاهده می‌کنید هر انبارک شامل ۱۶ بیت یا دو بایت با ارزش بالا (High) و

پائین (Low) می‌باشد.



شکل ۳-۷ : ساختار انبارک‌ها و بایت‌های با ارزش بالا (High) و پائین (Low)

۳-۱۸-۱- دستور L[1]

واژهٔ Load به معنی بارگذاری می‌باشد. به کمک این دستور عدد موجود در یک بایت، کلمه و یا اعداد ثابت توسط PLC خوانده شده، در انبارک قرار داده می‌شود. این دستور در موارد زیر می‌تواند به اجرا در آید:

L IB 4 ، L FY 6 ، L KD .... ، L KH .... ، L KC ....

فرض کنید در PLC مورد بحث ما دو انبارک (ACCUM 2 و ACCUM 1) استفاده شده است. حال دستور L IW 4 را در نظر بگیرید. این دستور بدان معنی است که ۱۶ بیت موجود در کلمهٔ ورودی شمارهٔ ۴ یا به عبارتی بایت‌های ۴ و ۵ به داخل انبارک ۱ یعنی ACCUM 1 فرستاده شوند. اگر در همین حالت دستور بعدی یعنی L IW 6 اجرا گردد، ابتدا اطلاعات موجود در ACCUM 1 یعنی IW 4 به ACCUM 2 منتقل شده، سپس IW 6 به ACCUM 1 انتقال می‌یابد. چنانچه در این وضعیت دستور بارگذاری دیگری نظیر L IW 12 اجرا شود. محتویات فعلی ACCUM 2 یعنی IW 4 دور ریخته شده، IW 6 در ACCUM 2 ذخیره می‌گردد. سپس IW 12 به ACCUM 1 منتقل می‌شود. بنابراین ملاحظه می‌گردد که به دلیل وجود دو انبارک و سه دستور بارگذاری، اولین دسته اطلاعات از بین می‌روند. پس در صورتی که مثلاً بخواهیم سه عدد را با هم جمع کنیم ابتدا باید دو عدد را با یکدیگر جمع نموده، سپس حاصل این دو عدد را با عدد سوم جمع نمائیم. در غیر این صورت اطلاعات بارگذاری شده مربوط به عدد اول از دست خواهند رفت. روند

---

1 - Load

انتقال و جابجایی اطلاعات در حین اجرای سه دستور بارگذاری یاد شده در زیر آمده است:

اولین دستور	L IW 4	IW 4 ⟶ ACCUM 1
دومین دستور	L IW 6	ACCUM 1 ⟶ ACCUM 2 و IW 6 ⟶ ACCUM 1
سومین دستور	L IW 12	ACCUM 1 ⟶ ACCUM 2 و IW 12 ⟶ ACCUM 1

و اطلاعات ACCUM 2 دور ریخته می‌شود.

۳-۱۸-۲- دستور T[1]

این دستور به معنی انتقال است و به کمک آن، اطلاعاتی که در انبارک قرار دارد توسط PLC به خروجی یا فلگ مورد نظر منتقل می‌گردد. نمونه‌هایی از این دستور را در زیر می‌بینید.

T QW[3] 8 ، T FW[2] 52

در اجرای دستور T QW 8، محتویات ACCUM 1 به کلمهٔ خروجی ۸ منتقل می‌گردد. لازم به ذکر است که در این جابجایی و انتقال، اطلاعات اصلی در ACCUM 1 باقی‌مانده، تنها رونوشتی از اطلاعات موجود در ACCUM 1 به QW 8 انتقال می‌یابد. با اندکی تأمل در می‌یابیم که در حقیقت رابط بین PLC و دنیای خارج همان انبارک ۱ یعنی ACCUM 1 می‌باشد.

دستورات L و T به RLO وابسته نیستند[4]، یا به عبارت دیگر این دو دستور RLO Independent می‌باشند. یعنی اجرای این دو دستور مشروط بر "۱" بودن بیت RLO مربوط به سطر قبل در برنامه نمی‌باشد و در صورتی که بیت RLO حاصل از سطر قبل "۰" یا "۱" باشد این دو دستورالعمل اجرا خواهند شد.

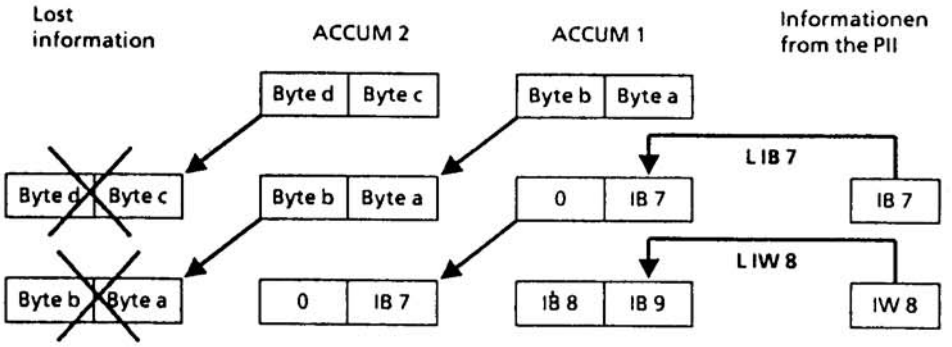
در شکل ۳-۸ چگونگی اجرای چندین دستور بارگذاری و انتقال و نحوهٔ جابجایی اطلاعات ذخیره شده در انبارک‌ها و همچنین اطلاعاتی که ممکن است در صورت وجود چندین دستور بارگذاری دور ریخته شوند نشان داده شده است.

---

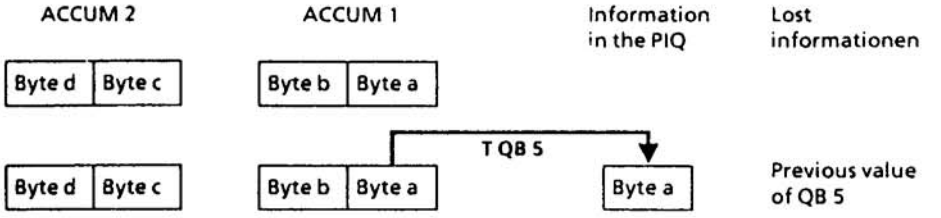
1 - Transfer 2 - FW: Flag Word 3 - QW: Output Word

۴ - در بعضی PLCها دستورات L و T مشروط (RLO dependent) نیز وجود دارند.

همان‌گونه که در شکل ۸-۳ (الف) ملاحظه می‌کنید دو دستور L IB 7 و L IB 8 اجرا شده است، و طی اجرای این دو دستور محتویات ابتدایی ACCUM 2 دور ریخته می‌شوند. ضمناً در دستور بارگذاریِ اطلاعات از PII به انبارک، اطلاعات موجود در PII ثابت باقی مانده، تنها رونوشتی از آنها به انبارک بارگذاری می‌شود.



(الف)

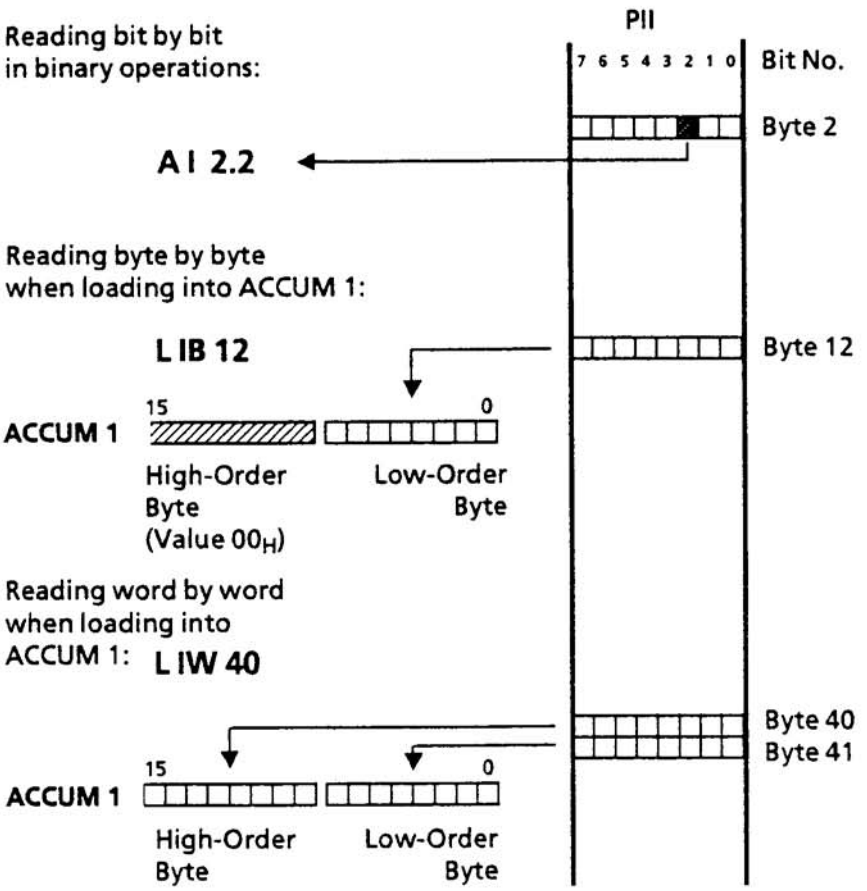


(ب)

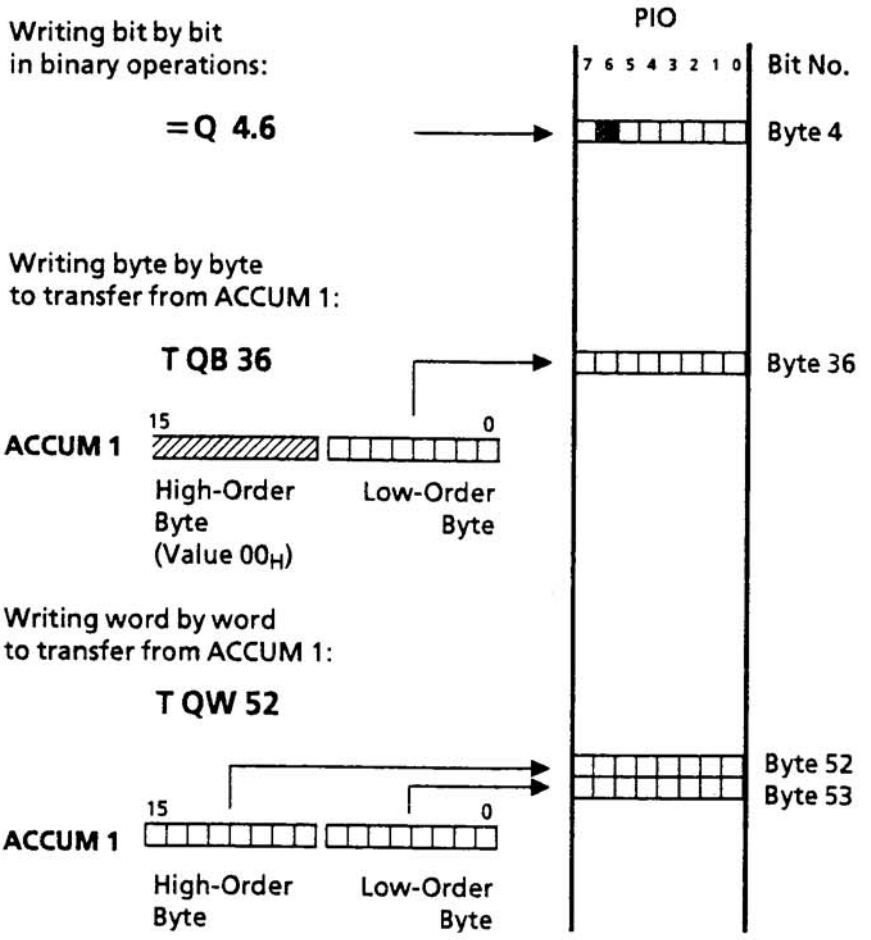
شکل ۳-۸ : چگونگی اجرای دستورات بارگذاری از PLC و انتقال به PIO

در شکل ۳-۸ (ب) ملاحظه می‌کنید که در اجرای دستور T QB 5 تنها رونوشتی از اطلاعات موجود در ACCUM 1 به بایت خروجی ۵ منتقل شده است. در این حالت پس از انتقال اطلاعات، محتویات قبلی بایت خروجی ۵ دور ریخته می‌شود. همان‌طور که می‌بینید در اجرای دستور انتقال تنها از ACCUM 1 کمک گرفته می‌شود.

همان‌گونه که در ابتدای بحث ذکر شد از دستور L جهت بارگذاری اطلاعات به صورت بایتی یا کلمه‌ای استفاده می‌گردد. برای بارگذاری یا فراخوانی یک بیت از ورودی (PII) ، از دستور AND استفاده می‌شود. همین مطلب در مورد دستور T نیز صادق است. یعنی دستور T ، اطلاعات را به صورت بایتی یا کلمه‌ای منتقل می‌کند و برای انتقال یا نسبت دادن تنها یک بیت به خروجی (PIO) از دستور هم‌ارزی (=) استفاده می‌کنیم. در دو شکل ۳-۹ و ۳-۱۰ نحوهٔ بارگذاری از PII و انتقال به PIO به صورت بیتی، بایتی و کلمه‌ای نشان داده شده است.



شکل ۳-۹ : نحوهٔ بارگذاری اطلاعات به صورت بیتی، بایتی و کلمه‌ای از PII



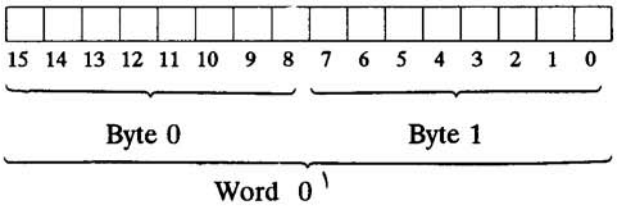
شکل ۳-۱۰ : نحوهٔ انتقال اطلاعات به صورت بیتی، بایتی و کلمه‌ای به PIO

۳-۱۹- نکاتی در مورد انتقال و بارگذاری اطلاعات به صورت کلمه‌ای

از آنجایی که هر کلمه شامل ۱۶ بیت یا ۲ بایت می‌باشد باید بدانیم که کدام یک از بایت‌ها در اجرای دستورات L و T بارگذاری و منتقل می‌شوند. برای درک بیشتر مطلب به ذکر دو مثال می‌پردازیم:

مثال ۳-۱۶ : دستور L IW 0 را در نظر بگیرید. چگونگی بارگذاری بایت‌های این کلمه یعنی کلمهٔ ورودی صفر را بیان کنید.

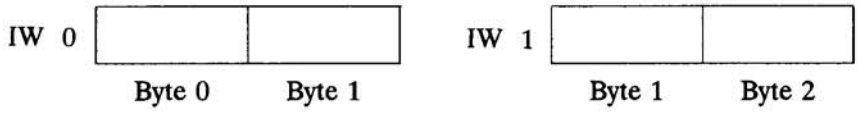
در صورت اجرای این دستور، بایت‌های ۰ و ۱ به داخل انبارک ۱ بارگذاری می‌شوند. بدین ترتیب که بایت ۰ در بایت باارزش بالا و بایت ۱ در بایت باارزش پائین جای می‌گیرد. در شکل زیر نحوهٔ بارگذاری اطلاعات به داخل انبارک ۱ نشان داده شده است.



[1]

مثال ۳-۱۷ : در صورتی که پس از اجرای دستور L IW 0 دستور L IW 1 را اجرا نمائیم چگونگی انجام این بارگذاری را توضیح دهید.

در صورتی که دستور L IW 1 اجرا شود بایتهای ۱ و ۲ ورودی در انبارک بارگذاری می‌شوند. در شکل ۳-۱۱ چگونگی اجرای دو دستور L IW 0 و L IW 1 نشان داده شده است.



شکل ۳-۱۱ : چگونگی اجرای دو دستور L IW 0 و L IW 1

اشکالی که در اجرای این دو دستور مشاهده می‌گردد آن است که بایت ورودی ۱ در دستور L IW 0 در بایت Low (با ارزش پائین) جای می‌گیرد در حالی که در دستور L IW 1، این بایت در بایت High (با ارزش بالا) قرار می‌گیرد. با کمی توجه در این دو مثال در می‌یابیم که تمام بایت‌های با شمارهٔ فرد همیشه در هنگام اجرای این گونه دستورات هم‌پوشانی[2] می‌شوند یعنی یک بار در موقعیت بایت Low و بار دیگر در موقعیت بایت High قرار می‌گیرند.

بنابراین به عنوان یک قرارداد در استفاده از کلمات همواره شماره‌های زوج یا فرد را به کار

---

۱ - در برخی از PLCها شماره‌گذاری بایت‌ها برعکس روش مذکور در PLCهای SIEMENS انجام می‌شود.

2 - Overlap

می‌بریم تا از بروز چنین مواردی جلوگیری شود. با به کارگیری این قاعده در استفاده از کلمات با شمارهٔ زوج یا فرد همواره شمارهٔ کلمه مثلاً صفر در IW 0 با شمارهٔ بایت High مطابقت دارد.

کلمات با شماره‌های زوج:
- L IW 20 — بایت‌های ۲۰ و ۲۱ بارگذاری می‌شوند.
- L IW 22 — بایت‌های ۲۲ و ۲۳ بارگذاری می‌شوند.

کلمات با شماره‌های فرد:
- L IW 31 — بایت‌های ۳۲ و ۳۱ بارگذاری می‌شوند.
- L IW 57 — بایت‌های ۵۸ و ۵۷ بارگذاری می‌شوند.

همان‌گونه که ملاحظه می‌شود در دستور L IW 20 بایت‌های ورودی ۲۱ و ۲۰ بارگذاری شده که شمارهٔ کلمه با شمارهٔ بایت High یکسان است. این قاعده را نه تنها در مورد ورودی‌ها بلکه در مورد خروجی‌ها و فلگ‌ها نیز رعایت می‌کنیم.

۳-۲۰- موارد استفادهٔ انبارک‌ها

انبارک علاوه بر استفاده در اجرای دستورات L و T، جهت انجام اعمال محاسباتی نظیر جمع، تفریق و ... نیز مورد استفاده قرار می‌گیرد که در ادامه به شرح هر یک از دستورات محاسباتی و چگونگی استفاده از انبارک‌ها در این گونه دستورات می‌پردازیم.

۳-۲۰-۱- دستور جمع دو عدد (+ F)

این دستور، دو عدد بارگذاری شده در انبارک‌ها را با یکدیگر جمع می‌کند. در برنامهٔ زیر که به روش STL نوشته شده است حاصل جمع دو کلمهٔ ورودی ۰ و ۲ به کلمهٔ خروجی ۴ انتقال می‌یابد. این برنامه تنها شامل یک Segment بوده، تحت عنوان PB 20 نوشته شده است.

```
 PB 20

SEGMENT  1            0000
0000       :L    IW    0
0001       :L    IW    2
0002       :+F
0003       :T    QW    4
0004       :BE
```

مثال ۳-۱۸ : با استفاده از برنامهٔ PB 20 حاصل جمع دو عدد دهدهی ۱۴۵۶ و ۷۶۵۹ را به دست آورید.

قبل از هر چیز ابتدا باید اعداد دهدهی داده شده را به مبنای ۲ تبدیل نمود و آنها را در دو کلمهٔ ورودی قرار داد.

$۱۴۵۶_{(۱۰)} = ۰۰۰۰۰۱۰۱\ \ ۱۰۱۱۰۰۰۰_{(۲)}$ + ⟶ IW 0

$۷۶۵۹_{(۱۰)} = ۰۰۰۱۱۱۰۱\ \ ۱۱۱۰۱۰۱۱_{(۲)}$ ⟶ IW 2

$= ۰۰۱۰۰۰۱۱\ \ ۱۰۰۱۱۰۱۱_{(۲)}$ ⟶ QW 4

$= ۰\times۲^{۱۵} + ۰\times۲^{۱۴} + ۱\times۲^{۱۳} + ۰\times۲^{۱۲} + ۰\times۲^{۱۱} + ۰\times۲^{۱۰} + ۱\times۲^{۹} + ۱\times۲^{۸} + ۱\times۲^{۷} + ۰\times۲^{۶}$

$+ ۰\times۲^{۵} + ۱\times۲^{۴} + ۱\times۲^{۳} + ۰\times۲^{۲} + ۱\times۲^{۱} + ۱\times۲^{۰} = ۹۱۱۵_{(۱۰)}$

بنابراین در خروجی (QW 4) عددودوئی $_{۲}$(۰۰۱۰۰۰۱۱۱۰۰۱۱۰۱۱)راخواهیم‌داشت‌که‌پس از تبدیل مبنا عدد $۹۱۱۵_{(۱۰)}$ به دست می‌آید که برابر حاصل جمع دو عدد ۱۴۵۶ و ۷۶۵۹ است.

همان‌گونه که گفته شد در دستورات L و T می‌توان از بایت به جای کلمه نیز استفاده نمود. در برنامهٔ زیر این مطلب نشان داده شده است.

```
 PB 10

SEGMENT   1              0000
0000        :L    IB     4
0001        :L    IB    12
0002        :+F
0003        :T    QB     0
0004        :BE
```

در صورتی که دو عدد $۲_{(۱۰)}$ و $۵_{(۱۰)}$ را در بایتهای ورودی ۴ و ۱۲ داشته باشیم حاصل جمع این دو عدد که همان $۷_{(۱۰)}$ می‌باشد به بایت خروجی صفر انتقال می‌یابد.

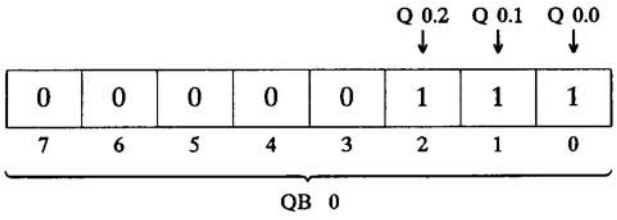
$۲_{(۱۰)} = ۰۰۰۰۰۰۱۰$ + ⟶ IB 4

$۵_{(۱۰)} = ۰۰۰۰۰۱۰۱$ ⟶ IB 12

$= ۰۰۰۰۰۱۱۱$ ⟶ QB 0

$= ۱\times۲^{۲} + ۱\times۲^{۱} + ۱\times۲^{۰} = ۷_{(۱۰)}$

عدد $۷_{(۱۰)}$ در QB 0 به صورت زیر انتقال می‌یابد.



بنابراین داریم:

Q 0.0 = "۱" و Q 0.1 = "۱" و Q 0.2 = "۱"

سایر بیت‌های این بایت خروجی دارای مقدار "۰" می‌باشند. پس توانسته‌ایم با جمع نمودن دو عدد و فرستادن حاصل جمع آن دو به خروجی، تعدادی از بیت‌های خروجی را فعال و بعضی دیگر را غیرفعال نمائیم.

نتیجه‌ای که از این مثال به‌دست می‌آید این است که:

با انجام عملیات محاسباتی بر روی ورودی‌ها می‌توان به خروجی‌ها فرمان داد.

همان‌گونه که در فصل اول بیان شد به دلیل مشکلات موجود در تبدیل اعداد باینری به دهدهی و بالعکس، از مبناهای بالاتر از مبنای دو یعنی مبناهای ۸ و ۱۶ استفاده می‌کنیم. در صورتی که از مبنای ۱۶ استفاده کنیم کلمهٔ ۱۶ بیتی را به ۴ گروه ۴ بیتی تقسیم می‌نمائیم. بنابراین هر کلمه در مبنای ۱۶ با ۴ کاراکتر (اعداد ۰ الی ۹ و حروف A الی F) قابل نمایش است.

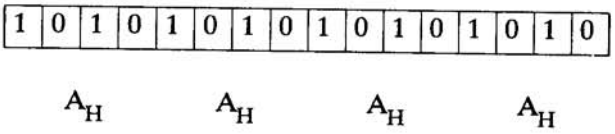
جهت بارگذاری انبارک با اعداد ثابت در مبنای دهدهی از دستور ... L KD استفاده می‌کنیم در حالی که جهت بارگذاری انبارک با عدد ثابتی در مبنای ۱۶ از دستور ... L KH استفاده می‌نمائیم. (به جای نقطه‌چین عدد بارگذاری شده قرار می‌گیرد.)

مثال ۳-۱۹ : برنامه‌ای بنویسید که بیت‌های کلمه خروجی ۲ (QW 2) را یک در میان فعال و غیرفعال نماید.

با دقت در متن مثال در می‌یابیم که بیت‌های خروجی ۲ باید به یکی از دو صورت زیر باشند.

0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

5_H 5_H 5_H 5_H



بنابراین می‌توان یکی از دو عدد 5555_H و یا $AAAA_H$ را در QW 2 قرار داد. برنامه‌های نوشته شده به روش STL جهت بارگذاری هر دو عدد مذکور در زیر آمده است.

```
 PB 5

SEGMENT  1          0000
0000      :L   KH 5555
0002      :T   QW    2
0003      :BE
```

```
 PB 8

SEGMENT  1          0000
0000      :L   KH AAAA
0002      :T   QW    2
0003      :BE
```

۳-۲۰-۲- دستور تفریق دو عدد (F -)

با استفاده از این دستور می‌توان دو عدد را تفریق نمود. نمونهٔ برنامه تفریق دو عدد در PB 15 آمده است.

```
 PB 15

SEGMENT  1          0000
0000      :L   IB    1
0001      :L   IB    2
0002      :-F
0003      :T   QB    3
0004      :BE
```

در این برنامه محتویات IB 2 از محتویات IB 1 تفریق می‌گردد. روش استفاده شده در کامپیوتر، PLC و سیستم‌های دیجیتالی جهت تفریق دو عدد، جمع عدد اول (مفروق منه) با مکمل ۲، عدد دوم (مفروق) است.

مثال ۳-۲۰: حاصل تفریق دو عدد $۳۶_{(۱۰)}$ و $۵۴_{(۱۰)}$ را با استفاده از PB 15 به‌دست آورید.

ابتدا دو عدد داده شده را به مبنای ۲ تبدیل می‌کنیم.

$$۵۴_{(۱۰)} = ۰۰۱۱۰۱۱۰_{(۲)} - \longrightarrow \text{IB } 1$$

$$۳۶_{(۱۰)} = ۰۰۱۰۰۱۰۰_{(۲)} \longrightarrow \text{IB } 2$$

$$= ۰۰۰۱۰۰۱۰_{(۲)} \longrightarrow \text{QB } 3$$

$$= ۱\times۲^{۴} + ۱\times۲^{۱}$$

$$= ۱۶+۲ = ۱۸$$

همان‌گونه که انتظار داشتیم حاصل تفریق دو عدد مذکور، عدد $۱۸_{(۱۰)}$ می‌باشد، که معادل با $۰۰۰۱۰۰۱۰_{(۲)}$ است. حال اگر عدد دوم بیش از عدد اول باشد حاصل تفریق چه خواهد شد؟ برای روشن شدن این مطلب عملیات تفریق روبرو را درنظر می‌گیریم: $۰_{(۱۰)} - ۱_{(۱۰)} = ?$

فرض کنید IB 1 حاوی $۰۰۰۰۰۰۰۰_{(۲)}$ و IB 2 نیز حاوی $۰۰۰۰۰۰۰۱_{(۲)}$ باشد، طبق عملیات تفریق (جمع مفروق منه با مکمل ۲ مفروق) خواهیم داشت:

IB 1 : ۰۰۰۰۰۰۰۰ −

IB 2 : ۰۰۰۰۰۰۰۱

?

جمع با مکمل ۲ ←

۰۰۰۰۰۰۰۰ +

۱۱۱۱۱۱۱۰ ← مکمل ۱

۱۱۱۱۱۱۱۰ +

۱

۱۱۱۱۱۱۱۱ ⟶ QB 3

بنابراین عدد $۱۱۱۱۱۱۱۱_{(۲)}$ به بایت خروجی ۳ منتقل می‌گردد. توجه داشته باشید که با انجام این عمل توانسته‌ایم تمام بیت‌های بایت خروجی ۳ را فعال نمائیم. پس تفریق دو عدد نیز همانند جمع آنها می‌تواند به عنوان روشی جهت فعال یا غیرفعال نمودن بیت‌های خروجی مورد استفاده قرار گیرد.

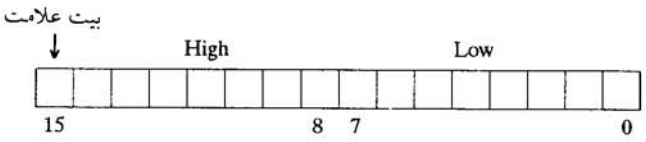
مثال عملی متناظر با تفریق عدد ۱ از ۰ را می‌توان در شمارنده‌های (کانترهای) مکانیکی ضبط‌صوت یافت. (فرض کنید که این شمارنده ۴ رقمی است یعنی از عدد 0000 شروع شده، تا

9999 ادامه می‌یابد و پس از 9999 مجدداً به 0000 باز می‌گردد و ...) اگر شمارندهٔ مکانیکی ضبط صوت بر روی 0000 قرار گرفته باشد و شما توسط کم ارزش‌ترین رقم (یعنی رقم سمت راست)، شمارنده را به اندازهٔ یک واحد در خلاف جهت افزایش شماره‌ها بچرخانید، با این عمل در حقیقت از عدد 0000 ، یک واحد کم کرده‌اید. عددی که بر روی شمارنده ظاهر می‌شود عدد 9999 خواهد بود.

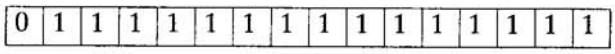
در مورد مثال تفریق عدد ۱ از عدد ۰ نیز حاصل تفریق، تقریباً مشابه این عدد یعنی ۱۱۱۱۱۱۱۱ به‌دست آمد، عدد حاصل منفی است. حال این سؤال مطرح می‌شود که:

طرز تشخیص علامت اعداد (مثبت یا منفی بودن آنها) چگونه است؟

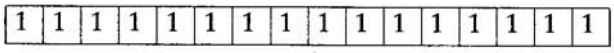
در جواب به این سؤال باید گفت که آخرین بیت (پر ارزش‌ترین بیت) در انبارک به عنوان بیت علامت یا Sign bit در نظر گرفته می‌شود. چنانچه این بیت "۰" باشد عدد موجود در انبارک مثبت و در غیر این صورت عدد مذکور منفی است. در شکل زیر بیت علامت در انبارک‌ها نشان داده شده است.



با توضیحات فوق می‌توان گفت که جهت بارگذاری در انبارک‌ها اعداد $FFFF_H$ و $7FFF_H$ به ترتیب کوچکترین عدد منفی و بزرگترین عدد مثبت در مبنای ۱۶ می‌باشند.



7 F F F (بزرگترین عدد مثبت مبنای ۱۶)



F F F F (کوچکترین عدد منفی مبنای ۱۶)

با توجه به تخصیص بیت آخر به عنوان بیت علامت باید دقت شود که در جمع دو عدد مثبت، مقدار این بیت "۱" شود و گرنه عدد به‌دست آمده به عنوان عددی منفی تلقی می‌شود. در ضمن در PLCها برای حاصل جمع دو عدد، رقم نقلی یا carry bit وجود ندارد. (به غیر از موارد خاص)

برای تعریف اعداد در مبنای ۱۰ و پرهیز از استفادهٔ مبنای ۱۶، از فرم ...KF استفاده می‌کنیم. در این حالت می‌توان از اعداد مثبت، منفی و صفر استفاده نمود. محدودهٔ تغییرات اعداد دهدهی در فاصلهٔ [+۳۲۷۶۸ و -۳۲۷۶۸] می‌باشد.

جهت بارگذاری انبارک می‌توان از اعداد BCD نیز استفاده نمود. برای بارگذاری اعداد بـه فـرم BCD از فرمت ...KC یا ...KT استفاده می‌شود.

از آنجایی که اعداد BCD، اعداد باینری کد شده در مبنای ۱۰ هستند بنابراین کوچکترین عددی که می‌توان در انبارک بارگذاری نمود عدد 9999- است که چگونگی بارگذاری آن در انبارک در زیر نشان داده شده است.

1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
	9				9				9				9		

بنابراین محدودهٔ این اعداد در فاصلهٔ [7999, -9999] می‌باشد.

با استفاده از دستورات L و T و بارگذاری اعداد مختلف در مبناهای متفاوت می‌توان صحت ارتباط بین ورودی‌ها و خروجی‌ها[1] را در یک سیستم بررسی نمود. بسیاری از خـوانـندگانی کـه محیط‌های صنعتی را تجربه نموده‌اند به خوبی آگاهند که قبل از راه‌اندازی هر فرآیـند، بـایستی از صحت ارتباط ورودی‌ها و خروجی‌ها (در لوپها و مدارات) اطمینان حاصل کرد. این عمل توسط روشهای گوناگونی از جمله لوپ چک (Loop Check) و یا تـلفن چک (Telephone Check) انجام می‌گیرد و همانگونه که اشاره شد با استفاده از دستورات L و T می‌توان این عمل را انجام داد.

برای این کار می‌توان ورودی‌ها را به دسته‌های ۱۶ تایی (Word) تقسیم نمود و برنامهٔ زیر را در مورد هر دسته جداگانه اجرا کرد، تا اینکه تمام ارتباطات بررسی شوند.

```
 PB 14

SEGMENT  1            0000
0000       :L   IW    0
0001       :T   QW    2
0002       :BE
```

۱ - منظور از ورودی و خروجی در این حالت، ورودی و خروجی PLC نمی‌باشد. بلکه منظور ارتباط بین دو قسمت مدار است، که یکی را ورودی و دیگری را خروجی نامیده‌ایم.

نحوهٔ عملکرد این برنامه و چگونگی بررسی صحت ارتباطات در قسمتهای مختلف مدار به صورت زیر است:

فرض کنید یک دستهٔ ۱۶ تایی از ورودی‌ها را در اختیار داریم که در ورودی تمامی آنها (قبل از کلیدها) ولتاژ "۱" برقرار نموده‌ایم. پس از اجرای این برنامه و تغییر حالت تمامی کلیدها از OFF به ON، بایستی خروجی متناظر هر کلید فعال شود. در غیر این صورت ارتباط بین این دو قسمت مدار (ورودی و خروجی) صحیح نبوده، بررسی مدار جهت یافتن اشکال موجود الزامی است. پس از اتمام این دستهٔ ۱۶ تایی، در مورد دستهٔ دوم نیز همین عمل به همراه همین برنامه اجرا می‌شود تا اینکه صحت ارتباط تمامی قسمت‌های مدار بررسی گردد.

۳-۲۱- مقایسه کننده‌ها (Comparators)

یکی دیگر از موارد استفادهٔ انبارک در مقایسه‌کننده است. یک مقایسه‌کننده مقدار دو ورودی را دریافت نموده، با توجه به نوع و نتیجهٔ مقایسه، خروجی مقایسه‌کننده را فعال و یا غیرفعال می‌کند. اعدادی که می‌توان در مقایسه‌کننده استفاده نمود به صورت بایتی بوده، شامل IB ها، QB ها و FY ها می‌باشند. علاوه بر موارد مذکور اعداد ثابت نیز می‌توانندبا فرمت ...KF[1] برای مقایسه در برنامه وارد شوند.

در جدول ۳-۵ انواع مقایسه، نمادهای ریاضی و نمادهای برنامه‌نویسی[2] متناظر با هر یک نشان داده شده است.

---

۱ - فرمت KF برای وارد نمودن اعدادی ثابت در مبنای ۱۰ استفاده می‌شود.

۲ - این نمادها در برنامه‌نویسی به روش STL مورد استفاده قرار می‌گیرند. در روشهای دیگر برنامه‌نویسی، نمادهای بلوکی و ... استفاده می‌شود. تعداد این نمادها در انواع مختلف یکسان است و تنها تفاوت آنها در طرز نمایش آنها می‌باشد.

جدول ۳-۵ : عملکرد و حالت مقایسه‌ای برای موارد گوناگون مقایسه

عملکرد یا حالت مقایسه	نماد ریاضی	نماد برنامه‌نویسی
مساوی بودن دو عدد	=	! = F
نامساوی بودن دو عدد	≠	> < F
عدد اول بزرگتر از عدد دوم	>	> F
عدد اول بزرگتر یا مساوی با عدد دوم	≥	> = F
عدد اول کوچکتر از عدد دوم	<	< F
عدد اول کوچکتر یا مساوی عدد دوم	≤	< = F

حاصل مقایسهٔ دو عدد در مقایسه‌کننده به صورت بیت RLO می‌باشد. در ضمن حاصل عمل مقایسه را می‌توان به یک خروجی یا یک فلگ نسبت داد. در نوشتن برنامهٔ مقایسهٔ دو عدد، باید حالت مقایسه و دو عدد مورد نظر در برنامه وارد شوند. مثالهای زیر این مطلب را روشن می‌کند.

مثال ۳-۲۱ :به برنامه PB 25 توجه کنید.

```
 PB 25

SEGMENT  1            0000
0000      :L    IB    5
0001      :L    KF +100
0003      :!=F
0004      :=    Q     1.4
0005      :BE
```

در این مثال، در سطر اول برنامه به کمک دستور بارگذاری، بایت ورودی ۵ وارد انبارک می‌شود. سپس با استفاده از دستور L KF 100 در سطر دوم محتویات ACCUM 1 یعنی IB 5 به ACCUM 2 انتقال یافته، عدد دهدهی ۱۰۰ در ACCUM 1 قرار می‌گیرد. در این مثال، عملکرد مقایسه، بررسی تساوی دو عدد است. چنانچه دو عدد مساوی باشند بیت RLO "۱" شده، در سطر چهارم برنامه، خروجی Q 1.4 نیز فعال می‌شود. در غیر این صورت خروجی مذکور "۰" خواهد شد.

مثال ۳-۲۲: در این مثال حالت دیگری از مقایسهٔ دو عدد بررسی می‌گردد.

```
 PB 36

SEGMENT   1            0000
0000        :L    QB   12
0001        :L    FY   20
0002        :><F
0003        :=    F    100.0
0004        :BE
```

عملکرد این برنامه به شرح زیر است:

در سطر اول: عدد موجود در بایت خروجی ۱۲ در انبارک ۱ بارگذاری می‌شود.

در سطر دوم: عدد موجود در بایت فلگ ۲۰ به انبارک ۱ و محتویات انبارک ۱ به انبارک ۲ منتقل می‌گردد و در صورتی که دو عدد نامساوی باشند بیت فلگ 100.0 F فعال می‌شود.

مثال ۳-۲۳: برای درک بیشتر حالات مختلف مقایسه، برنامهٔ کاملی شامل تمام حالات مقایسه نوشته شده است. با تکمیل جدول ۳-۶ در هر حالت چگونگی عملکرد را توضیح دهید. بلوک برنامهٔ نوشته شده شامل ۶ بخش یا Segment بوده که در هر قسمت یکی از حالات مقایسه بررسی شده است. توجه داشته باشید که هنگام اجرای یک PB تمام Segmentهای آن بلوک در اجرای برنامه نقش دارند. لازم به ذکر است که اعداد موجود در جدول (مقادیر ورودی اول و دوم) اعداد دهدهی می‌باشند.

```
 PB 1

SEGMENT   1            0000
0000        :L    IB   16
0001        :L    IB   17
0002        :!=F
0003        :=    Q     8.0
0004        :***


SEGMENT   2            0005
0005        :L    IB   16
0006        :L    IB   17
0007        :><F
0008        :=    Q     8.1
0009        :***
```

```
SEGMENT  3              000A
000A      :L    IB  16
000B      :L    IB  17
000C      :>=F
000D      :=    Q    8.2
000E      :***

SEGMENT  4              000F
000F      :L    IB  16
0010      :L    IB  17
0011      :>F
0012      :=    Q    8.3
0013      :***

SEGMENT  5              0014
0014      :L    IB  16
0015      :L    IB  17
0016      :<=F
0017      :=    Q    8.4
0018      :***

SEGMENT  6              0019
0019      :L    IB  16
001A      :L    IB  17
001B      :<F
001C      :=    Q    8.5
001D      :BE


 PB 1

SEGMENT  1              0000
!
!              +-----+
!IB 16      --!V1   F!
!               !!=   !              Q 8.0
!IB 17      --!V2   Q!-+----------+--(    )-!
!              +-----+
!
```

ادامهٔ بلوک PB 1:

```
SEGMENT  2          0005
!
!                +-----+
!IB 16     --!V1  F!
!                !><   !                Q 8.1
!IB 17     --!V2  Q!-+---------+--(    )-!
!                +-----+
!
SEGMENT  3          000A
!
!                +-----+
!IB 16     --!V1  F!
!                !>=   !                Q 8.2
!IB 17     --!V2  Q!-+---------+--(    )-!
!                +-----+
!
SEGMENT  4          000F
!
!                +-----+
!IB 16     --!V1  F!
!                !>    !                Q 8.3
!IB 17     --!V2  Q!-+---------+--(    )-!
!                +-----+
!
SEGMENT  5          0014
!
!                +-----+
!IB 16     --!V1  F!
!                !<=   !                Q 8.4
!IB 17     --!V2  Q!-+---------+--(    )-!
!                +-----+
!
SEGMENT  6          0019
!
!                +-----+
!IB 16     --!V1  F!
!                !<    !                Q 8.5
!IB 17     --!V2  Q!-+---------+--(    )-!
!                +-----+
!
!                                    :BE
```

جدول ۳-۶ : مربوط به مثال ۳-۲۳

Val. 1 IB 16	Val. 2 IB 17	Q 8.5 < F	Q 8.4 < = F	Q 8.3 > F	Q 8.2 > = F	Q 8.1 > < F	Q 80 ! = F
10	10						
9	10						
10	9						

(راهنمایی: برای هر سه حالت مفروض مقادیر ورودی را اعلام کرده، نتایج را در خروجی به‌دست آورید. در هر یک از حالات مذکور سه بیت از ۶ بیت استفاده شده از بایت هشتم فعال می‌شوند.)

موارد استفادهٔ مقایسه‌کننده‌ها در کنترل پروسه‌های صنعتی بسیار زیاد است. مثلاً می‌توان به یکی از ورودی‌ها مقادیر ثابت (Set Point) و به ورودی دیگر مقادیر متغیر ورودی پروسه، مثلاً درجه حرارت، فشار یا ... را وارد کرد و با استفاده از یکی از حالات مقایسه و با توجه به تعریف کنترل متغیر ورودی، خروجی را فعال یا غیرفعال نمود. در مثال زیر از این مطلب استفاده شده است.

مثال ۳-۲۴ : قصد داریم در یک فرآیند صنعتی درجه حرارت روغن خنک‌کنندهٔ سیستم را کنترل نمائیم. برنامه‌ای بنویسید که اگر درجه حرارت روغن به کمتر از ۲۰°C برسد گرم کنندهٔ روغن (Heater) موجود در مخزن روغن روشن و در صورتی که درجه حرارت روغن به بیش از ۳۰°C برسد Heater خاموش شود.

برای درک بهتر این مثال، ابتدا برنامه را به روش CSF می‌نویسیم. همان‌گونه که حدس زده‌اید در این برنامه باید از دو مقایسه‌کننده همراه با دو حالت مقایسهٔ جداگانه و در دو Segment استفاده کنیم.

در برنامه PB 29 فرض بر این است که درجه حرارت روغن توسط بایت ورودی ۵ (IB 5) به مقایسه‌کننده وارد می‌شود. در قسمت اول، این عدد با عدد ۲۰ مقایسه شده و در صورتی که درجه حرارت روغن از ۲۰°C کمتر باشد خروجی مقایسه‌کنندهٔ اول فعال می‌شود و Q 0.7 را ست می‌کند بنابراین خواهیم داشت: "۱" = Q 0.7 . این خروجی می‌تواند فرمان روشن شدن گرم کنندهٔ

```
 PB 29

SEGMENT  1          0000
!
!            +-----+
!IB 5      --!V1  F!
!            !<    !                Q 0.7
!KF +20    --!V2  Q!-+----------+--(S   )-!
!            +-----+
!
SEGMENT  2          0006
!
!            +-----+
!IB 5      --!V1  F!
!            !>    !                Q 0.7
!KF +30    --!V2  Q!-+----------+--(R   )-!
!            +-----+
!
!                                   :BE
```

الکتریکی (Heater) باشد. در قسمت دوم، درجه حرارت روغن با عدد ۳۰ مقایسه می‌شود و در صورتی که درجه حرارت روغن از ۳۰°C تجاوز نماید خروجی این مقایسه‌کننده فعال می‌شود. یعنی Q 0.7 را ری‌ست نموده "۰" = Q 0.7 می‌شود. به عبارت دیگر Heater خاموش می‌شود.

در ادامه، همین برنامه به روش STL آمده است.

```
 PB 29

SEGMENT  1          0000
0000      :L   IB   5
0001      :L   KF +20
0003      :<F
0004      :S   Q    0.7
0005      :***


SEGMENT  2          0006
0006      :L   IB   5
0007      :L   KF +30
0009      :>F
000A      :R   Q    0.7
000B      :BE
```

۳-۲۲- شمارنده‌ها (Counters)

یکی از مواردی که در کنترل فرآیندهای صنعتی کاربرد فراوانی دارند شمارنده‌ها هستند. در برخی از پروسه‌ها و خطوط تولید نیاز به شمارش به وفور دیده می‌شود. مثلاً شمارش قطعات گذشته از خط تولید و یا تعداد عناصری که بایستی در یک جعبه، بسته‌بندی شوند و ... . علاوه بر این شمارنده‌ها در برنامه‌نویسی نیز کاربرد قابل ملاحظه‌ای دارند. در ادامه، طرق مختلف نمایش شمارنده در روشهای مختلف برنامه‌نویسی نشان داده شده است. در PLC های مختلف تعداد شمارنده‌هایی که می‌توان استفاده نمود متفاوت است. برای استفاده از هر شمارنده باید شمارهٔ آن را ذکر نمود مثلاً C4. در زیر، برنامه‌نویسی یک شمارنده را به سه روش LAD، STL و CSF ملاحظه می‌کنید.

```
 PB 33

SEGMENT  1            0000
0000      :A    I     0.1
0001      :CU   C     4
0002      :A    I     0.2
0003      :CD   C     4
0004      :A    I     0.0
0005      :L    KC 050
0007      :S    C     4
0008      :A    I     0.3
0009      :R    C     4
000A      :L    C     4
000B      :T    QW    2
000C      :LD   C     4
000D      :T    FW    10
000E      :A    C     4
000F      :=    Q     0.0
0010      :BE
```

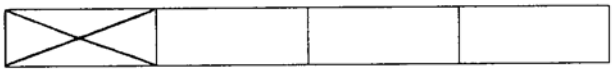
```
 PB 33

SEGMENT  1          0000
           C 4
          +-----+
 I 0.1   --!CU   !
 I 0.2   --!CD   !
 I 0.0   --!S    !
 KC 050  --!CV BI!- QW 2
           !   DE!- FW 10
           !     !   +------+
 I 0.3   --!R   Q!-+-! =    ! Q 0.0
           +-----+   +------+:BE
```

```
 PB 33

SEGMENT  1          0000
!          C 4
!I 0.1     +-----+
+---] [---+-!CU   !
!         ! !     !
!I 0.2    ! !     !
+---] [---+-!CD   !
!         ! !     !
!I 0.0    ! !     !
+---] [---+-!S    !
!KC 050   --!CV BI!- QW 2
!           !   DE!- FW 10
!           !     !
!I 0.3      !     !              Q 0.0
+---] [---+-!R   Q!-+----------+--(    )-!
!           +-----+                    :BE
!
```

در یک شمارنده، ورودی CU جهت شمارش صعودی (Count Up) و ورودی CD جهت شمارش نزولی (Count Down) استفاده می‌شود. لازم به ذکر است که ورودی‌های CU و CD با لبهٔ پالس (لبهٔ بالا رونده یا پائین رونده) فعال می‌شوند و با هر بار فعال شدنِ شمارنده، بسته به نوع شمارشگر، عدد شمارنده افزایش یا کاهش می‌یابد.

ورودی S : (Set) با فعال شدن ورودی S مقدار اولیهٔ شمارنده یعنی CV در شمارنده قرار می‌گیرد.

ورودی CV : (Counter Value) مقدار اولیه‌ای است که در شمارنده قرار گرفته، مبنای شمارش محسوب می‌شود. برای بارگذاری اعداد در شمارنده‌ها باید از فرمت ...KC L استفاده نمود. پس از بارگذاری، مقدار CV در انبارک جای می‌گیرد ولی تنها از ۱۲ بیت انبارک استفاده می‌شود. این اعداد به فرم BCD است و بنابراین حداکثر مقداری که می‌توان به CV نسبت داد عدد 999 می‌باشد.



۱۲ بیت مورد استفاده در بارگذاری اعداد شمارنده‌ها ۴ بیت غیر قابل استفاده

ورودی R : جهت ری‌ست کردن شمارنده استفاده می‌شود.

با توجه به قاعدهٔ بیان شده در مبحث ست و ری‌ست فلیپ‌فلاپ‌ها مبنی بر اینکه هر دستوری که به خط پایانی برنامه (BE) نزدیکتر باشد از نظر اجرایی ارجح است می‌توان گفت که این ورودی نیز بر تمام ورودی‌های دیگر ارجح بوده و هر زمان که اراده کنیم می‌توانیم با فعال نمودن این ورودی، شمارنده را ری‌ست نمائیم. در حالتی که شمارنده ری‌ست می‌شود خروجی شمارنده "۰" خواهد بود.

به محض اینکه شمارنده شروع به شمارش کند (شمارش خواه به صورت مستقیم، خواه به صورت معکوس) ست می‌شود. در ضمن دستورات ...KC L و ...C S به صورت دو دستور پیوسته، لازم و ملزوم یکدیگرند و استفاده از یکی از این دو دستور بدون استفاده از دیگری در برنامه‌نویسی شمارنده کاملاً بی‌مفهوم است. اما می‌توان در یک برنامه از هر دو دستور صرف‌نظر نمود.

شمارش واقعی (Actual Count) : این شمارش به دو صورت باینری در خروجی BI و BCD در خروجی DE قابل نمایش است. PLC هر دو صورت را به شکل کلمه (Word) نشان می‌دهد. مقادیر شمارش شده را می‌توان به خروجی یا فلگ ارسال نمود.

خروجی Q : یک سیگنال خروجی است و تا زمانی که شمارنده در حال شمارش است، مقدار این خروجی (به عنوان مثال Q 0.0) برابر "۱" بوده، در صورت اتمام عملیات شمارش به مقدار "۰"

تغییر وضعیت می‌دهد.

مجدداً یادآور می‌شویم که ورودی‌های CU، CD، S و R تنها با اعمال لبه، فعال شده و با سطح ولتاژ عمل نمی‌کنند. اکنون مجدداً برنامهٔ مربوط به شمارنده به روش STL را در نظر می‌گیریم:

در صورتی که در برنامه، قصد داشته باشیم که از شمارش مستقیم شمارنده استفاده نکنیم به جای این دو سطر از دستور NOP 0 استفاده می‌نمائیم.

```
:A    I     0.1
:CU   C     4
```

اگر در برنامه‌ای نیاز به شمارش معکوس نداشته باشیم از دستور NOP 0 به جای این دو سطر استفاده می‌کنیم.

```
:A    I     0.2
:CD   C     4
```

در صورت نیاز، به کمک این سه دستور می‌توان عدد اولیه را در شمارنده قرار داد و با ورودی مربوطه، به عنوان مثال I 0.0، شمارنده را ست نمود.

```
:A    I     0.0
:L    KC    050
:S    C     4
```

در صورت عدم نیاز به دستور ری‌ست می‌توان از دستور NOP 0 به جای این دو سطر استفاده نمود.

```
:A    I     0.3
:R    C     4
```

این دو سطر، مخصوص نمایش شمارش به فرم باینری است و در صورت عدم تمایل به استفاده از این فرم نمایش از دستور NOP 0 به جای این دو دستور استفاده می‌نمائیم.

```
:L    C     4
:T    QW    2
```

این دو سطر مخصوص نمایش شمارش به صورت BCD است و در صورت عدم تمایل به استفاده از این فرم نمایش می‌توان از دستور NOP 0 به جای این دو دستور استفاده نمود.

```
:LD   C     4
:T    FW    10
```

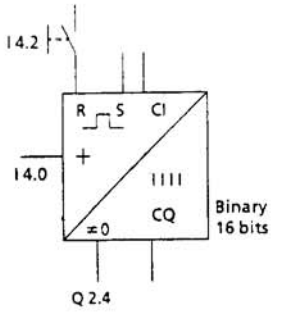
این دو سطر نیز جهت فراخوانی وضعیت شمارنده و ارسال آن به یک بیت خروجی استفاده می‌شوند. در صورت عدم نیاز به این دو سطر از دستور NOP 0 استفاده می‌کنیم.

```
:A    C     4
:=    Q     0.0
:BE
```

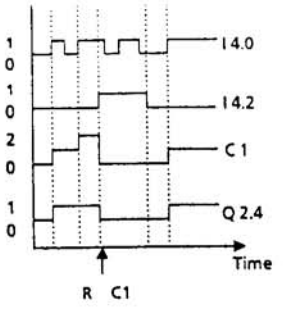
همان‌گونه که قبلاً ذکر شد دستور NOP 0 در شمارنده‌ها نیز به وفور به کار می‌رود. کاربرد این دستور به دلیل عدم نیاز به برخی از قسمتها در برنامه است و با به کارگیری آن به جای دستورات استفاده نشده (دستورات حذف شده) می‌توان برنامهٔ نوشته شده به روش STL را به دیگر روشها تبدیل و ترجمه نمود. در ادامه به ذکر دو نمونه شمارش (شمارش مستقیم و معکوس) می‌پردازیم.

مثال ۳-۲۵ : در این مثال، شمارش مستقیم بررسی می‌گردد. همان‌گونه که در برنامه آمده است هنگامی که I 4.0 فعال شود عدد موجود در شمارندهٔ ۱ به اندازهٔ یک واحد افزایش می‌یابد و به محض اینکه ورودی I 4.2 فعال می‌شود شمارنده ری‌ست خواهد شد. دستور A C 1 مقدار "۱" را در هنگام شمارش توسط شمارنده به بیت خروجی Q 2.4 می‌فرستد و به محض اتمام عملیات شمارش، این خروجی "۰" خواهد شد. در ادامه، نماد مداری، نمودار زمانی و برنامه‌های نوشته شده به هر سه روش آمده است.

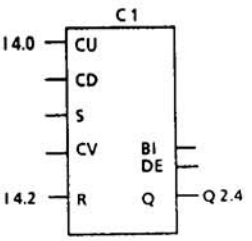
نماد مداری

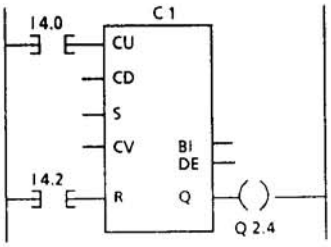


نمودار زمانی



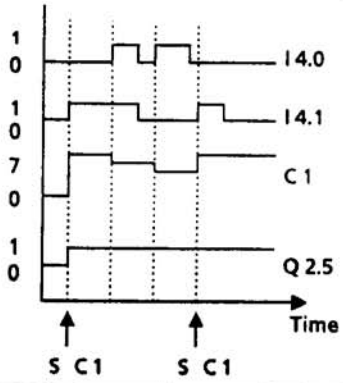
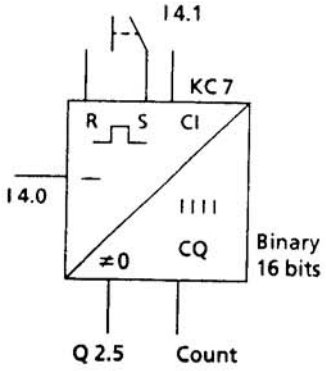
STL			CSF	LAD
A	I	4.0		
CU	C	1		
NOP	0			
NOP	0			
NOP	0			
A	I	4.2		
R	C	1		
NOP	0			
NOP	0			
A	C	1		
=	Q	2.4		

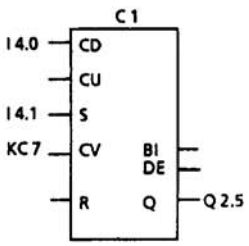
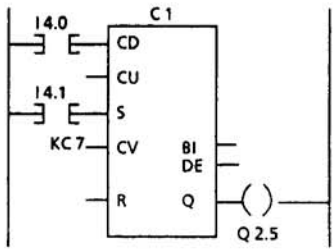




همان‌گونه که ملاحظه می‌کنید در این برنامه (به روش STL) به دلیل عدم نیاز برنامه‌نویس به برخی از ورودی‌ها و یا خروجی‌ها، چندین بار از دستور NOP 0 استفاده شده است.

مثال ۳-۲۶: در این مثال شمارش معکوس مورد بررسی قرار می‌گیرد. به محض اینکه ورودی I 4.1 فعال گردد، عدد اولیه در شمارنده قرار می‌گیرد و خروجی Q 2.5 برابر "۱" می‌شود. هر زمان که ورودی I 4.0 فعال شود عدد موجود در شمارندهٔ ۱ به اندازهٔ ۱ واحد کاهش می‌یابد. هنگامی که شمارش معکوس به عدد صفر برسد بیت خروجی Q 2.5 نیز مقدار "۰" را خواهد داشت.

نمودار زمانی	نماد مداری
	

STL	CSF	LAD
		

```
A    I    4.0
CD   C    1
NOP  0
A    I    4.1
L    KC   7
S    C    1
NOP  0
NOP  0
NOP  0
A    C    1
=    Q    2.5
```

مثال ۳-۲۷: قصد داریم برنامه‌ای بنویسیم که در آن، شمارش از عدد صفر شروع شده، تا عدد ۳۰ ادامه یابد و پس از رسیدن به عدد ۳۰ مجدداً شمارش از صفر شروع شود و تا عدد ۳۰ ادامه پیدا کند و به همین ترتیب شمارش ادامه یابد تا اینکه شمارنده توسط ورودی R در زمان دلخواه ری‌ست شود.

قبل از آغاز برنامه‌نویسی کمی در مورد ماهیت برنامهٔ موردنظر بحث خواهیم نمود. فرض کنید که در یک کارخانه می‌بایست ۳۰ بسته یا ۳۰ واحد از مواد تولیدی در جعبه‌ای قرار گیرد بنابراین در این برنامه شمارنده‌ای را برنامه‌نویسی می‌کنیم که هر بار که به عدد ۳۰ برسد مجدداً شروع به شمارش نماید.

این برنامه در دو بخش (Segment) نوشته می‌شود. در بخش اول برنامه‌ای می‌نویسیم که شمارنده از عدد صفر شروع به شمارش مستقیم (CU) نموده، شمارش تا عدد ۳۰ ادامه یابد. در بخش دوم با کمک یک مقایسه‌کننده، عدد ۳۰ را با خروجی BI مربوط به شمارنده مقایسه می‌نمائیم و در صورت برابری این دو مقدار، بیت خروجی مقایسه‌کننده را ست می‌کنیم. این بیت خروجی را به ورودی R شمارنده اعمال نموده تا به هنگام فعال شدن این بیت یا به عبارت دیگر رسیدن شمارنده به عدد ۳۰، شمارنده ری‌ست شود. در مورد برنامهٔ مذکور تنها روش STL استفاده شده است:

```
 PB 36

SEGMENT  1            0000
0000      :A    I     4.1
0001      :CU   C     6
0002      :NOP  0
0003      :A    I     1.5
0004      :L    KC 030
0006      :S    C     6
0007      :O    I     1.6
0008      :O    F     6.0
0009      :R    C     6
000A      :L    C     6
000B      :T    QW    4
000C      :NOP  0
000D      :A    C     6
000E      :=    Q     5.7
000F      :***

SEGMENT  2            0010
0010      :L    KF +30
0012      :L    QW    4
0013      :!=F
0014      :=    F     6.0
0015      :BE
```

همان‌گونه که ملاحظه می‌کنید در برنامهٔ نوشته شده به روش STL دو بار از دستور NOP 0 در قسمت اول استفاده شده است. از آنجایی که در این شمارنده نیاز به شمارش معکوس نداریم بنابراین از تعریف ورودی نیز جهت شمارش CD خودداری، و به جای دو سطر مربوطه از دستور NOP 0 استفاده کرده‌ایم. دومین دستور NOP 0 به دلیل عدم استفاده از خروجی DE یعنی عدد شمارنده در مبنای BCD است.

در بخش دوم این برنامه با استفاده از یک مقایسه‌کننده، عدد ۳۰ را با عدد شمارش شده توسط شمارنده مقایسه و به محض برقراری تساوی بین این دو، بیت F 6.0 برابر "۱" می‌گردد.

توجه داشته باشید که فعال شدن F 6.0 در مدت بسیار کوتاهی می‌باشد و پس از تغییر محتویات QW 4 در شمارش، به دلیل عدم تساوی دو ورودی مقایسه‌کننده، مقدار این بیت "۰" می‌شود.

همان‌طور که ملاحظه می‌کنید در بخش اول در ورودی R شمارنده، حاصل ترکیب فصلی I 1.6 و F 6.0 قرار داده شده است. این بدان معنی است که به مجرد اینکه بیت F 6.0 در بخش دوم برنامه فعال گردد شمارندهٔ ۶ ری‌ست شده، شمارش مجدداً از صفر آغاز می‌شود. در ورودی R علاوه بر F 6.0 ، I 1.6 نیز جهت ری‌ست نمودن شمارنده در نظر گرفته شده است.

توجه داشته باشید که در هنگام اجرای برنامهٔ شمارنده می‌توان روند شمارش را در PG مشاهده نمود. در هنگام اجرای این برنامه ملاحظه می‌شود که مقدار شمارش شده توسط شمارنده یعنی محتویات QW 4 از صفر شروع شده، به ترتیب افزایش می‌یابد تا اینکه به عدد ۲۹ برسد. به محض افزایش ۱ واحد به عدد ۲۹، شمارنده ری‌ست شده، شمارش مجدداً از صفر آغاز می‌گردد. از آنجایی که در اجرای این برنامه عدد ۳۰ حد بالایی شمارش است می‌توان گفت که:

در صورتی که شمارش به صورت مستقیم (CU) انجام شود حد بالایی شمارش و اگر شمارش به صورت معکوس (CD) انجام گیرد حد پائینی شمارش بر روی PG قابل رؤیت نیست.

همواره در اجرای برنامه‌ای که در آن از شمارنده استفاده شده است اطلاعاتی به شرح زیر در پائین صفحهٔ نمایش پروگرامر (PG) دیده می‌شود[۱].

---

۱ - نحوهٔ نمایش این‌گونه اطلاعات ممکن است در پروگرامرهای مختلف، متفاوت باشد.

C ... ACT : ... PAR : ...

شمارهٔ شمارندهٔ مورداستفاده در برنامه

ACT (Actual) : این عدد نشان دهندهٔ مقدار فعلی موجود در شمارنده است. در حقیقت این عدد، همان مقداری است که در حین شمارش کاملاً متغیر بوده و به یکی از دو حالت باینری (BI) و یا BCD (DE) به خروجی‌ها فرستاده می‌شود.

PAR (Parameter) : این مقدار، عدد موجود در ورودی CV را نشان می‌دهد. مقدار این عدد ثابت می‌باشد. به عنوان مثال در برنامهٔ نوشته شده در پائین صفحهٔ نمایش PG مقادیر زیر را خواهیم داشت: C 6 ACT : .... PAR : 30

این مقدار از **0** تا **29** متغیر است.

۳-۲۳- تایمرها (Timers)

تایمرها نقش بسیار مهمی در کنترل اکثر فرآیندها بر عهده دارند. از کنترل چراغهای راهنمایی سر چهارراه‌ها تا کنترل فرآیندهای پیچیدهٔ صنعتی، همگی نیاز به زمان‌سنجی دارند. بنا به کاربرد تایمر می‌توان از انواع مختلف آن استفاده نمود. بنابراین در مورد استفاده از تایمر باید مشخص گردد که از چه نوع تایمری استفاده می‌شود. به طور کلی ۵ نوع تایمر به شرح زیر وجود دارد.

۱- تایمر پله‌ای SP (Pulse Timer)

۲- تایمر پله‌ای گسترده SE (Extended Pulse Timer)

۳- تایمر با تأخیر روشن SD (On - Delay Timer)

۴- تایمر با تأخیر خاموش SF (Off - Delay Timer)

۵- تایمر تأخیر ماندگاری SS (Stored On - Delay Timer)

قبل از ارائه توضیح در مورد هر یک از تایمرها برنامهٔ نوشته شده برای یک نوع تایمر، به عنوان مثال SE را مورد بررسی قرار داده، ورودی‌ها و خروجی‌های آن را بررسی می‌نمائیم. در ادامه، برنامهٔ زمان‌سنجی مربوط به تایمر SE به هر سه روش برنامه‌نویسی آمده است.

```
 PB 37

SEGMENT  1           0000
0000       :A    I     1.0
0001       :L    KT  005.2
0003       :SP   T     7
0004       :A    I     1.1
0005       :R    T     7
0006       :L    T     7
0007       :T    QW   10
0008       :LD   T     7
0009       :T    FW    4
000A       :A    T     7
000B       :=    Q     0.1
000C       :BE
```

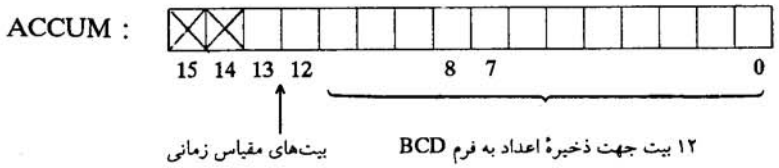
```
 PB 37

SEGMENT  1           0000
            T 7
             +------+
 I 1.0     --!1_-_  !
 KT 005.2  --!TV  BI!- QW 10
             !    DE!- FW 4
             !      !   +-------+
 I 1.1     --!R    Q!-+-! =     ! Q 0.1
             +------+   +-------+:BE
```

```
 PB 37

SEGMENT  1           0000
!            T 7
!I 1.0        +------+
+---] [---+--!1_-_  !
!KT 005.2 --!TV  BI!- QW 10
!           !    DE!- FW 4
!           !      !
!I 1.1      !      !              Q 0.1
+---] [---+--!R    Q!-+----------+--(    )-!
!           +------+              :BE
!
```

حال به توضیح در مورد ورودی‌ها و خروجی‌های تایمر می‌پردازیم:

ورودی S (Set) : با هر بار فعال شدن این ورودی، تایمر فعال می‌گردد.

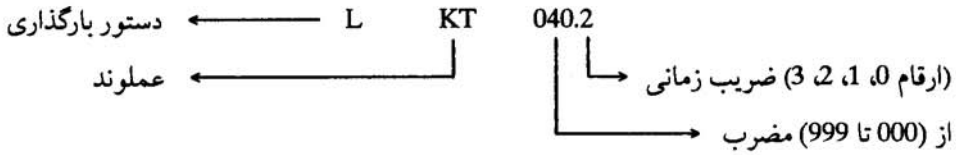
ورودی TV (Timer Value) : عددی است که پریود زمانی تایمر را معین می‌کند. معمولاً این عدد با فرمت ... KT L در تایمر بارگذاری می‌شود. پس از بارگذاری، مقدار KT در انبارک قرار می‌گیرد. KT در انباره‌ها شامل ۱۴ بیت بوده، به فرم BCD می‌باشد. دو بیت آخر در انبارک بدون استفاده می‌باشند. دو بیت باارزش بالاتر KT (موجود در انبارک) بیت‌های ضریب یا مقیاس زمانی (Time Base) نامیده می‌شوند. از ۱۲ بیت باقیمانده نیز برای ذخیرهٔ اعداد BCD از 000 تا 999 استفاده می‌شود.



عدد ثابت تایمر یا KT از یک مضرب و یک مقیاس زمانی تشکیل می‌گردد. مضرب می‌تواند عددی در فاصلهٔ 000 تا 999 باشد و مقادیری که مقیاس یا ضریب زمانی اختیار می‌کند ۰، ۱، ۲ یا ۳ است. این ارقام، تولرانس تایمر را نیز معرفی می‌نمایند. ارقام مقیاس زمانی توسط PLC به صورت زیر تفسیر می‌شوند.

	بیت ۱۲	بیت ۱۳
۰/۰۱ثانیه : ۰ =	۰	۰
۰/۱ثانیه : ۱ =	۱	۰
۱ثانیه : ۲ =	۰	۱
۱۰ثانیه : ۳ =	۱	۱

بنابراین برای وارد نمودن یا بارگذاری عدد ثابت تایمر به صورت زیر عمل می‌نمائیم:



کمترین و بیشترین مقادیری که می‌توان به TV نسبت داد به صورت زیر می‌باشد:

کمترین زمان ممکن : KT 001.0 ⟶ ۰/۰۱ ثانیه

بیشترین‌زمان‌ممکن : KT 999.3 ⟶ ۹۹۹۰ ثانیه

به عنوان مثال اگر مضرب KT عدد 5 باشد با هر یک از مقیاسهای زمانی مذکور زمانهای زیر به‌دست می‌آیند.

دستور بارگذاری	زمان پریود	مقیاس زمانی	مضرب
L KT 5.0	۰/۰۵ ثانیه	۰ (۰/۰۱ ثانیه)	۵
L KT 5.1	۰/۵ ثانیه	۱ (۰/۱ ثانیه)	۵
L KT 5.2	۵ ثانیه	۲ (۱ ثانیه)	۵
L KT 5.3	۵۰ ثانیه	۳ (۱۰ ثانیه)	۵

همانگونه که ملاحظه می‌شود زمان پریود از ضرب نمودن مقیاس زمانی در مضرب به‌دست می‌آید.

حال فرض کنید که قصد داریم با استفاده از یک تایمر، تأخیری به مدت ۴۰ ثانیه[1] ایجاد نمائیم. برای این کار از دستور .... L KT استفاده می‌کنیم. این دستور می‌تواند به یکی از حالات زیر وارد شود.

جدول ۳-۷ : حالات مختلف برای بارگذاری زمان ۴۰ ثانیه در تایمر

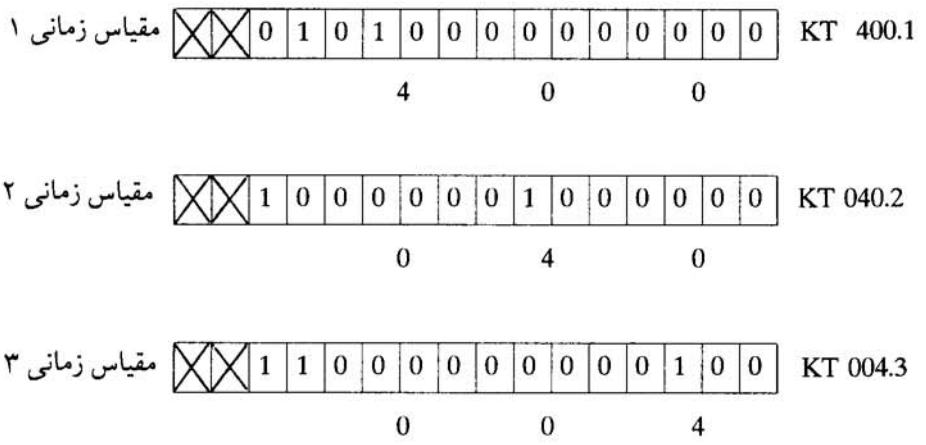
تولرانس	فاصلهٔ زمانی	دستور	
۰/۱ ثانیه	ثانیه ۰/۱ ± ۴۰ = ۱/۰ × ۴۰۰	L KT 400.1	زمانهای ممکن استفاده شده‌برای‌بارگذاری‌زمان ۴۰ ثانیه در تایمر
۱ ثانیه	ثانیه ۱ ± ۴۰ = ۱ × ۴۰	L KT 40.2	
۱۰ ثانیه	ثانیه ۱۰ ± ۴۰ = ۱۰ × ۴	L KT 4.3	

در جدول فوق ملاحظه می‌کنید که در تمامی موارد، زمان ۴۰ ثانیه در تایمر بارگذاری می‌شود

---

۱ - برای ایجاد زمان تأخیر به مدت ۴۰ ثانیه نمی‌توان از ضریب زمانی ۰ استفاده نمود زیرا در این حالت باید از دستور L KT 4000.0 استفاده کنیم و همانگونه که ذکر شد انبارک در این حالت گنجایش چنین عددی را ندارد.

ولی مقدار تولرانس زمانی هر یک متفاوت با دیگری است. بنابراین در مواردی که دقت عمل بالا در محاسبهٔ زمانهای حساس نیاز باشد از ضرائب زمانی کوچکتر (۰ یا ۱) و در موارد دیگر از ضرائب بزرگتر (۲ یا ۳) استفاده می‌کنیم. در زیر نحوهٔ بار شدن عدد ۴۰ در انباره آمده است.



ورودی R (Reset) : با فعال شدن این ورودی، سنجش زمان متوقف می‌گردد. از آنجایی که این ورودی نسبت به سایر ورودی‌ها به دستور پایانی برنامه نزدیکتر است از نظر اجرایی نسبت به ورودی‌های دیگر ارجحیت دارد و هرگاه که در این ورودی لبه پالسی داشته باشیم خروجی تایمر ری‌ست می‌شود.

خروجی BI : زمان باقیماندهٔ تایمر نسبت به TV به صورت عددی در مبنای دو در یک کلمهٔ خروجی یا کلمهٔ فلگ می‌تواند ظاهر شود. در صورتی که نیازی به این خروجی نباشد می‌توان از وارد نمودن آن در برنامه خودداری و به جای آن از دستور NOP 0 استفاده کنیم.

خروجی DE : عملکرد این خروجی نیز همانند خروجی BI است با این تفاوت که در این خروجی، زمان باقیماندهٔ تایمر نسبت به TV به صورت عددی در مبنای BCD به یک کلمهٔ خروجی یا کلمهٔ فلگ ارسال می‌شود.

خروجی Q : این بیت از زمان شروع به کار تایمر به مدت TV ثانیه فعال می‌ماند. البته این مطلب در صورتی صادق است که در حین سپری شدن زمان تایمر، ورودی R فعال نشده باشد. این بیت را می‌توان به یک بیت فلگ یا بیت خروجی نسبت داد. باید توجه داشت که تایمر برای ست و

ری‌ست شدن تنها نیاز به لبهٔ پالس در ورودی‌های S و R دارد.

حال که با مفاهیم و اصطلاحات استفاده شده در تایمرها آشنا شدیم به ارائه توضیح در مورد انواع تایمرها می‌پردازیم.

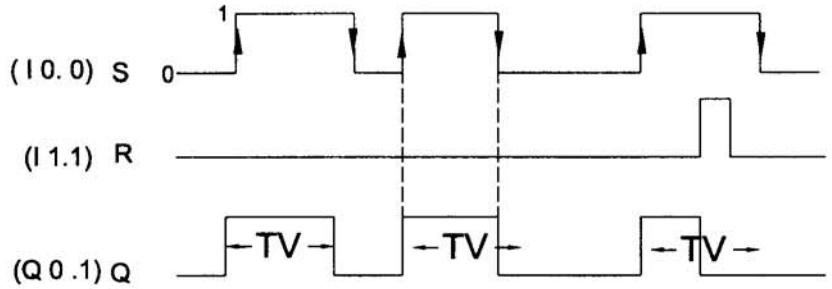
۳-۲۳-۱- تایمر پله‌ای (SP)[1]

در این تایمر، خروجی، هم به لبهٔ بالارونده و هم به لبهٔ پائین‌رونده حساس است. خروجی تایمر با لبهٔ بالارونده S به مدت TV ثانیه فعال و سپس غیرفعال می‌گردد. با لبهٔ پائین‌روندهٔ S، خروجی نیز "۰" خواهد شد. به عبارت دیگر، خروجی تایمر بستگی به ورودی S خواهد داشت. در ادامه، برنامهٔ نوشته شده جهت این نوع تایمر به دو روش STL و LAD به همراه شکل موجهای R، S، Q و چگونگی تأثیر ورودی‌های S و R بر خروجی ارائه شده است.

```
 PB 11

SEGMENT  1            0000
0000      :A    I     0.0
0001      :L    KT  005.2
0003      :SP   T     3
0004      :A    I     1.1
0005      :R    T     3
0006      :NOP  0
0007      :NOP  0
0008      :A    T     3
0009      :=    Q     0.1
000A      :BE
```

1 - Pulse Timer

```
 PB 11

SEGMENT  1          0000
!           T 3
!I 0.0      +------+
+---] [---+-!1_ _  !
!KT 005.2 --!TV BI!-
!           !   DE!-
!           !     !
!I 1.1      !     !            Q 0.1
+---] [---+-!R   Q!-+----------+--(    )-!
!           +------+
!
!                                :BE
!
```



همان‌گونه که ملاحظه می‌شود خروجی، هم به لبهٔ بالارونده و هم به لبهٔ پائین‌روندهٔ ورودی S حساس است. با دقت در شکل موجها می‌بینیم که در صورتی که هنوز زمان TV ثانیه از زمان فعال شدن خروجی سپری نشده باشد و در ورودی S لبهٔ پائین‌رونده داشته باشیم خروجی، قبل از به پایان رسیدن زمان TV، خاموش یا غیرفعال می‌شود. در این شکل موجها، عملکرد ورودی R نیز به وضوح دیده می‌شود.

با نگاهی گذرا در برنامهٔ نوشته شده به روش STL ملاحظه می‌کنید که به دلیل عدم استفاده از خروجی‌های BI و DE، از دستور NOP 0 دو بار استفاده شده است.

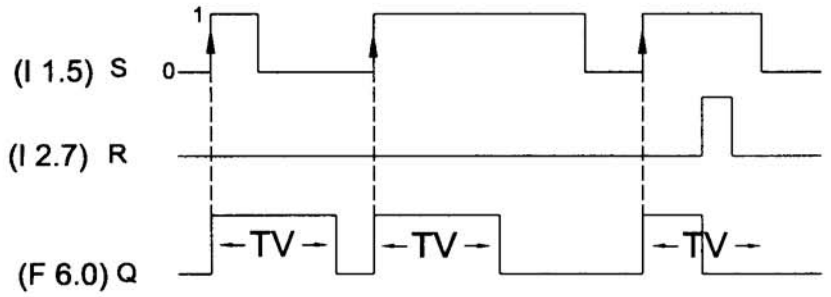
۲-۲۳-۲- تایمر پله‌ای گسترده (SE)[1]

خروجی این تایمر تنها به لبهٔ بالارونده ورودی S حساس است. با لبهٔ بالارونده ورودی S، خروجی شمارنده به مدت TV ثانیه فعال و سپس خاموش می‌شود. در صورتی که در مدت زمانی کمتر از TV ثانیه در ورودی S یک لبهٔ پایین‌رونده داشته باشیم، این لبه، بر خروجی بی‌تأثیر بوده و پس از گذشت مدت زمان TV، خروجی غیرفعال می‌شود. در ادامه، برنامهٔ نوشته شده جهت این نوع تایمر به همراه شکل موجهای R، S، Q و چگونگی ورودی‌ها بر خروجی ارائه شده است.

```
 PB 12

SEGMENT  1            0000
0000       :A    I     1.5
0001       :L    KT  004.1
0003       :SE   T     1
0004       :A    I     2.7
0005       :R    T     1
0006       :L    T     1
0007       :T    FW    50
0008       :NOP  0
0009       :A    T     1
000A       :=    F     6.0
000B       :BE
```

```
 PB 12

SEGMENT  1            0000
!            T 1
!I 1.5       +------+
+---] [---+-!1   V!
!KT 004.1 --!TV  BI!- FW 50
!            !   DE!-
!            !     !
!I 2.7       !     !              F 6.0
+---] [---+-!R    Q!-+----------+--(    )-!
!            +------+                    :BE
!
```

1 - Extended Pulse Timer



همان‌گونه که دیده می‌شود لبهٔ پائین‌روندهٔ ورودی S در عملکرد تایمر بی‌تأثیر است.

در برنامهٔ نوشته شده به روش STL به دلیل استفاده از خروجی BI (ارسال زمان باقیماندهٔ تایمر نسبت به TV به صورت باینری در FW 50) از دستور NOP 0 استفاده نشده است. ولی به دلیل عدم استفاده از خروجی DE، فقط یک بار از دستور NOP 0 استفاده گردیده است.

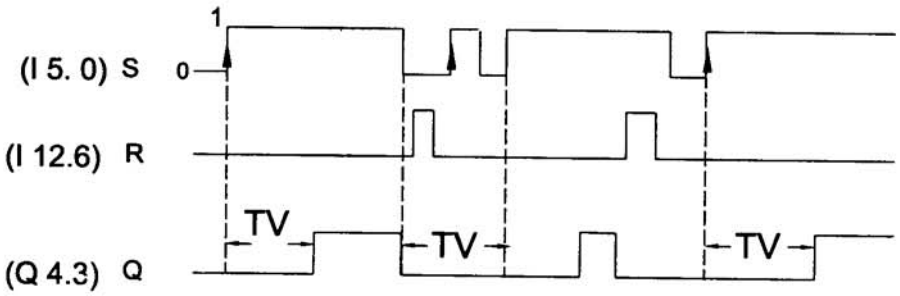
۳-۲۳-۳- تایمر با تأخیر روشن (SD)[1]

خروجی این تایمر هم به لبهٔ بالارونده و هم به لبهٔ پائین‌روندهٔ ورودی S حساس است. در طول مدت زمان تایمر، ورودی S باید فعال باقی بماند. با لبهٔ بالارونده S، خروجی تایمر پس از مدت زمان TV ثانیه فعال و با لبهٔ پائین‌روندهٔ S، غیرفعال می‌شود. با اندکی تأمل در می‌یابیم که عملکرد این تایمردرست برعکس تایمر SP است. درادامه برنامهٔ نوشته شده به همراه شکل موجهای S، R، Q و چگونگی تأثیر ورودی‌ها بر خروجی ارائه شده است.

1 - On - Delay Timer

```
PB 13

SEGMENT  1                0000
0000        :A    I      5.0
0001        :L    KT   100.0
0003        :SD   T      5
0004        :A    I     12.6
0005        :R    T      5
0006        :NOP  0
0007        :LD   T      5
0008        :T    QW     5
0009        :A    T      5
000A        :=    Q      4.3
000B        :BE
```

```
 PB 13

SEGMENT  1                0000
!              T 5
!I 5.0        +-----+
+---] [---+-!T!-!0!
!KT 100.0 --!TV BI!-
!             !   DE!- QW 5
!             !     !
!I 12.6       !     !              Q 4.3
+---] [---+-!R    Q!-+----------+--(    )-!
!             +-----+
!
!                                  :BE
```



۳-۲۳-۴- تایمر با تأخیر خاموش (SF)[1]

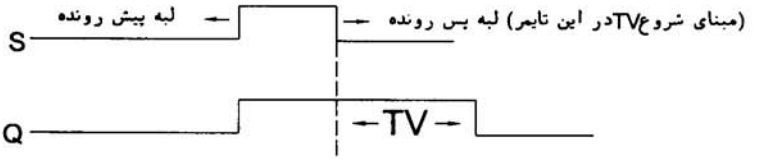
خروجی این تایمر با لبهٔ پیش‌روندهٔ ورودی S فعال و با لبهٔ پس‌روندهٔ ورودی S پس از TV ثانیه غیرفعال می‌گردد. برنامه‌های نوشته شده به روشهای STL و LAD به همراه شکل موجهای خروجی و ورودی و تأثیر ورودی‌های R و S بر خروجی در ادامه ارائه شده است.

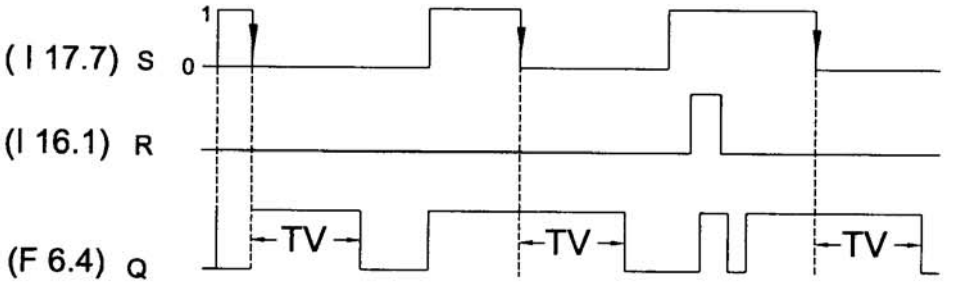
```
 PB 14

SEGMENT  1              0000
0000       :A    I     17.7
0001       :L    KT  015.3
0003       :SF   T     15
0004       :A    I     16.1
0005       :R    T     15
0006       :L    T     15
0007       :T    QW     1
0008       :LD   T     15
0009       :T    FW    12
000A       :A    T     15
000B       :=    F      6.4
000C       :BE
```

```
 PB 14

SEGMENT  1              0000
!            T 15
!I 17.7      +------+
+---] [---+-!0!-!T!
!KT 015.3 --!TV BI!- QW 1
!           !   DE!- FW 12
!           !     !
!I 16.1     !     !              F 6.4
+---] [---+-!R   Q!-+----------+--(    )-!
!           +------+
!
!                                  :BE
```

1 - Off - Delay Timer

در این نوع تایمر، با لبهٔ پیش‌رونده (Leading Edge) خروجی فعال می‌شود ولی مبنای سنجش زمان TV ، لبهٔ پس‌رونده (Lagging Edge) خواهد بود. در شکل زیر لبهٔ پیش‌رونده و پس‌روندهٔ پالس نشان داده شده است.





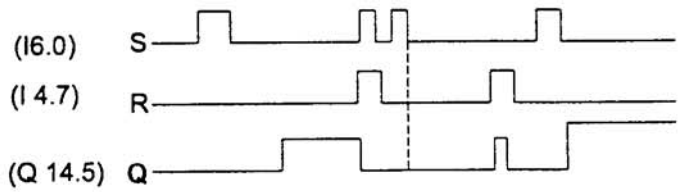
۳-۲۳-۵- تایمر تأخیر ماندگاری (SS)[1]

خروجی این تایمر فقط به لبهٔ بالارونده ورودی حساس است. این تایمر با لبهٔ بالارونده ورودی S پس از TV ثانیه فعال شده، در همین وضعیت باقی می‌ماند و تنها با فعال شدن ورودی R غیرفعال می‌شود. عملکرد این تایمر بر عکس تایمر SE است. در ادامه، برنامهٔ نوشته شده جهت این نوع تایمر به همراه شکل موج ورودی‌های S ، R و Q آورده شده است.

1 - Stored On - Delay Timer

```
 PB 15

SEGMENT  1            0000
0000       :A    I     6.0
0001       :L    KT 012.2
0003       :SS   T    13
0004       :A    I     4.7
0005       :R    T    13
0006       :NOP  0
0007       :NOP  0
0008       :A    T    13
0009       :=    Q    14.5
000A       :BE
```

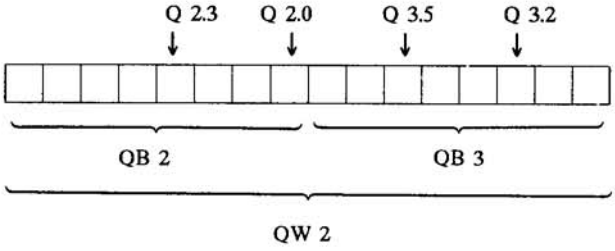
```
 PB 15

SEGMENT  1            0000
!            T 13
!I 6.0       +-----+
+---] [---+-!T!-!S!
!KT 012.2 --!TV BI!-
!            !   DE!-
!            !     !
!I 4.7       !     !              Q 14.5
+---] [---+-!R   Q!-+----------+--(    )-!
!            +-----+
!
!                                 :BE
```



در فصل بعد خواهید دید که چگونه می‌توان یک تایمر را بدون استفاده از تعریف تایمر، برنامه‌نویسی نمود. حال برنامهٔ تایمر زیر راکه از نوع SP می‌باشد در نظر بگیرید.

```
 PB 16

SEGMENT  1            0000
0000       :A    I     0.0
0001       :L    KT  005.2
0003       :SP   T     3
0004       :A    I     0.1
0005       :R    T     3
0006       :L    T     3
0007       :T    QW    2
0008       :NOP  0
0009       :A    T     3
000A       :=    F     6.0
000B       :BE
```

اگر پس از اجرای این برنامه، مقادیر ارسالی به کلمهٔ خروجی ۲ (QW 2) را به یک سری نشان‌دهنده LED اعمال نمائیم مشاهده می‌کنیم که بیت‌های این کلمهٔ خروجی با فرکانس‌های متفاوت (سرعت‌های متفاوت) فعال و غیرفعال می‌شوند یا به عبارت دیگر چشمک می‌زنند. در ادامه، این کلمهٔ خروجی به همراه بیت‌های آن نشان داده شده است.



از آنجایی که عدد ثابت تایمر KT 5.2 می‌باشد در نتیجه به دلیل وجود مقیاس زمانی ۲، تولرانس زمانی برابر ۱ ثانیه است و کم ارزش‌ترین بیت موجود در این کلمه یعنی Q 3.0 با فرکانس ۱ ثانیه روشن و خاموش و یا فعال و غیرفعال می‌شود. بیت‌های بعدی یعنی Q 3.1، Q 3.2 و ... با فرکانس‌هایی متفاوت با فرکانس کم ارزش‌ترین بیت و همچنین متفاوت با فرکانس‌های یکدیگر

روشن و خاموش می‌شوند.

در صورتی که در سطر دوم برنامه، L KT 500.1 را داشته باشیم به دلیل وجود عدد ۱ که مقیاس زمانی بوده و معرف تولرانس ۰/۱ ثانیه می‌باشد بنا به مطالب عنوان شده کم ارزش‌ترین بیت یعنی Q 3.0 با فرکانس ۰/۱ ثانیه روشن و خاموش می‌شود. این سرعت روشن و خاموش شدن به اندازه‌ای است که چشم انسان قدرت تشخیص وضعیت LED مربوط به این بیت خروجی را دارد. بیت‌های بعدی با فرکانس‌های مختلف روشن و خاموش می‌شوند.

بنابراین نتیجه‌ای که حاصل می‌شود آن است که در صورت استفاده از چنین برنامه‌ای و ارسال اعداد شمارش شده در خروجی BI ، همواره کم ارزش‌ترین بیت با سرعت تولرانس تایمر روشن و خاموش شده و بیت‌های دیگر با سرعتی چندین برابر سرعت مذکور فعال و غیرفعال می‌شوند. پس می‌توان از بیت‌های خروجی ارسال شده در موارد مختلف به صورت باینری استفاده نمود.

فرض کنید یک چراغ هشداردهنده (آلارم) بایستی طوری برنامه‌ریزی شود که با سرعتهای متفاوت چشمک بزند. می‌توان با ارسال بیت‌های گوناگون کلمهٔ خروجی، این خواسته را برآورده ساخت. از این گونه برنامه‌ها در مواردی که لازم است عملی به صورت پریودیک انجام گیرد (به عنوان مثال: چشمک زدن LEDها و ...) استفاده می‌شود.

تنها اشکالی که در این برنامه وجود دارد (برنامهٔ PB 16) آن است که تایمر، این سیکل را تنها برای یک بار انجام می‌دهد و پس از گذشت TV ثانیه، تایمر متوقف می‌شود. بنابراین در صورتی که لازم باشد در انجام عملیاتی پریودیک از این برنامه استفاده گردد باید با ایجاد تغییری در برنامه موجب شویم تا در صورت اتمام زمان TV ، تایمر ری‌ست شده و سیکل مجدداً انجام شود. در برنامهٔ PB 17 این تغییرات اعمال شده است.

```
 PB 17

SEGMENT  1              0000
0000      :A    I      0.0
0001      :AN   F      6.0
0002      :L    KT  005.2
0004      :SE   T      3
0005      :A    I      0.1
0006      :R    T      3
0007      :L    T      3
0008      :T    QW     20
0009      :NOP  0
000A      :A    T      3
000B      :=    F      6.0
000C      :BE
```

در این برنامه به دلیل نیاز به لبه برای فعال شدن تایمر از تایمر SE استفاده شده است. عملکرد این برنامه بدین گونه است که:

به محض اینکه تایمر زمان TV را پشت سر گذاشت خروجی تایمر یعنی "۰" = F 6.0 شده و چون در ورودی S تایمر ترکیب عطفی نقیض F 6.0 به همراه I 0.0 وجود دارد بلافاصله تایمر ست شده، مجدداً سیکل شمارش اجرا می‌گردد و به همین ترتیب این عمل ادامه می‌یابد تا زمانی که تایمر ری‌ست شود.

مثال ۳-۲۸ : برنامه‌ای بنویسید که کنترل دو چراغ چشمک‌زن را برعهده داشته باشد به صورتی که چراغ ۱ با فرکانس ۱ ثانیه روشن و خاموش شود و چراغ ۲ با فرکانسی چندین برابر کمتر از فرکانس چراغ ۱ چشمک بزند.

از آنجایی که قصد داریم سرعت چشمک زدن چراغها مضربی از ۱ ثانیه باشد بنابراین برنامهٔ تایمری را می‌نویسیم که دارای تولرانس ۱ ثانیه باشد. به عبارت دیگر مقیاس زمانی موجود در عدد شمارنده، ۲ باشد. این برنامه را در دو بخش می‌نویسیم به طوری که در بخش اول، تایمر فعال شده و عدد شمارش شده توسط تایمر به صورت باینری به یک کلمهٔ خروجی مثلاً QW 20 انتقال یابد. در بخش دوم بیت‌های این کلمهٔ خروجی را به دو چراغ اعمال می‌کنیم. واضح است که برای داشتن

فرکانس ۱ ثانیه در چشمک زدن چراغ ۱ باید بیت Q 21.0 را به این چراغ اعمال نمود. در مورد چراغ دوم نیاز به فرکانسی چند برابر چراغ اول داریم بنابراین به اختیار یکی از بیت‌های Q 21.3 یا Q 21.4 را انتخاب و به چراغ دوم اعمال می‌کنیم. جهت انجام سیکل شمارش و روشن نمودن LED ها در این برنامه از PB 18 استفاده شده که در SEGMENT 1 این برنامه PB 17 به کار برده شده است.

```
 PB 18

SEGMENT  1              0000
0000       :A    I      0.0
0001       :AN   F      6.0
0002       :L    KT  005.2
0004       :SE   T      3
0005       :A    I      0.1
0006       :R    T      3
0007       :L    T      3
0008       :T    QW     20
0009       :NOP  0
000A       :A    T      3
000B       :=    F      6.0
000C       :***

SEGMENT  2              000D
000D       :A    Q      21.0
000E       :=    Q      10.5
000F       :A    Q      21.3
0010       :=    Q      10.7
0011       :BE
```

۳-۲۴- دستورهای اعلام پایان برنامه[1]

در انتهای هر برنامه لازم است به نحوی به PLC اطلاع داده شود که برنامه به پایان رسیده است. این عمل با استفاده از دستورات مربوط به پایان برنامه صورت می‌گیرد. این دستورات را همراه با

---

۱ - این دستورات تنها در روش برنامه‌نویسی STL قابل استفاده‌اند.

توضیح و چگونگی عملکرد آنها در جدول ۳-۸ می‌بینند.

جدول ۳-۸ : دستورات اعلام پایان برنامه

عملکرد	عملوند	توضیحات
BE	_	پایان برنامه ـ برنامه صرفنظر از اینکه بیت RLO چه مقداری داشته باشد پایان می‌یابد.
BEU[1]	_	پایان برنامه بدون شرط ـ برنامه صرفنظر از اینکه بیت RLO چه مقداری داشته باشد پایان می‌یابد.
BEC[2]	_	پایان برنامه با شرط ـ چنانچه مقدار RLO "۱" باشد برنامه پایان یافته، در غیر این صورت اجرای برنامه ادامه می‌یابد.

لازم به ذکر است که دستور BE تنها در انتهای برنامه استفاده می‌شود در صورتی که دستورات BEU و BEC در طول برنامه مورد استفاده قرار می‌گیرند. همانگونه که در جدول ۳-۸ مشاهده می‌کنید این عملکردها فاقد عملوند بوده، در سطری که از این دستورات استفاده می‌گردد هیچ عملوندی دیده نمی‌شود. دو دستور BEU و BEC جزء دستورات تکمیلی[3] هستند.

در صورتی که به خاطر داشته باشید در مبحث پرش شرطی و غیرشرطی (JC و JU) بیان شد که این دستورات جهت پرش به برنامه‌های دیگر مورد استفاده قرار می‌گیرند. اکنون یکی دیگر از کاربردهای این دستورات را بیان کرده سپس با تلفیق دو دستور پرش و دستورات BEU و BEC به ذکر چند مثال جهت روشن شدن مطلب و چگونگی عملکرد این دستورات می‌پردازیم.

در PLC زیمنس و در دستورات تکمیلی، دستوری به صورت JUMP TO LABEL وجود دارد که به برنامه‌نویس این امکان را می‌دهد که در صورت برقراری یا عدم برقراری یک شرط به سطری از برنامهٔ فعلی که با LABEL مشخص گردیده پرش و ادامهٔ برنامه را از آن سطر دنبال نماید. LABEL یا برچسب، در هنگام اجرای برنامه علامت مشخصه‌ای جهت پرش پردازنده به سطر

---

1 - Block End Uncoditional 2 - Block End Conditional

۳ - در مورد تعدادی از دستورات تکمیلی در فصل بعد توضیح خواهیم داد.

حاوی LABEL می‌باشد. برنامه‌نویس با استفاده از LABEL و به کمک دستورات BEU و BEC می‌تواند پردازنده را تا برقراری و یا عدم برقراری برخی شرایط دلخواه در یک حلقهٔ اجرایی برنامه قرار دهد. LABELهای استفاده شده در زبانهای برنامه‌نویسی به شکلهای مختلف می‌باشند. در PLC زیمنس برچسب‌ها از M000 الی M127 هستند. روش استفاده از این برچسب‌ها مثلاً به صورت JC = M012 است. در اجرای این دستور، PLC در صورت برقراری شرط و یا برابر بودن مقدار بیت RLO با "۱" به سطری که با برچسب M012 مشخص شده پرش و ادامهٔ اجرای برنامه را از آن سطر دنبال می‌کند.

مثال ۳-۲۹: در یک بلوک تابع‌ساز مثلاً FB 4 برنامه‌ای به صورت زیر نوشته شده است. می‌خواهیم عملکرد دستورات BEU و BEC و JUMP TO LABEL را بررسی نمائیم.

```
 FB 4

SEGMENT  1            0000
NAME :JUMP

0005        :A   I     0.0
0006        :A   I     0.1
0007        :=   Q     2.0
0008        :=   Q     2.1
0009        :A   I     0.2
000A        :JC  =M001
000B        :BEU
000C M001   :L   KH 0055
000E        :T   QB    3
000F        :BE
```

روند اجرای این برنامه به صورت زیر است:

در سطر پنجم برنامه در صورت برقراری شرط "۱" = I 0.2 (یا به عبارت دیگر "۱" = RLO) اجرای برنامه از سطری که با برچسب M001 مشخص شده ادامه می‌یابد. دستورالعمل L KH 55 اجرا و عدد 55_H به بایت خروجی ۳ یعنی QB 3 فرستاده می‌شود و پس از رسیدن به دستور BE، برنامه پایان می‌یابد.

ولی در صورتی که ورودی I 0.2 برابر "۰" باشد (یا به عبارت دیگر "۰" = RLO) سطر

JC = M001 نادیده فرض می‌شود و سطر بعدی یعنی BEU (اتمام برنامه بدون شرط) اجرا شده، و اجرای برنامه در همین سطر پایان می‌یابد و پردازنده جهت دریافت و پردازش ورودی‌ها و اطلاعات جدید مجدداً به ابتدای برنامه رجوع می‌کند. این سیکل همچنان ادامه می‌یابد تا اینکه شرط I 0.2 = "۱" برقرار شده، PLC به دستور انتهایی برنامه یعنی BE برسد.

با اندکی تأمل در می‌یابیم که از این گونه برنامه‌ها (بلوک‌های FB) می‌توان در مواردی استفاده نمود که انجام عملی منوط به برقراری شرط خاصی باشد. پس می‌توان به کمک این بلوک تابع‌ساز، قسمتی از نرم‌افزار را در یک حلقه (Loop) قرار داد تا شرط مورد نظر برقرار شده، پس از برقراری شرط (به عنوان مثال در اینجا I 0.2 = "۱") قسمتهای دیگر برنامه را به مرحلهٔ اجرا در آورد. در صورتی که در همین برنامه از دستور BEC به جای دستور BEU استفاده کنیم عملکرد برنامه به ترتیب زیر خواهد بود.

در صورتی که I 0.2 = "۱" باشد قسمت LABEL اجرا، ولی اگر I 0.2 = "۰" باشد تمام برنامه تا انتها اجرا می‌شود.

در این فصل و فصل گذشته به ذکر دستورات مهم و کاربردی در برنامه‌نویسی PLC پرداختیم. در فصل آینده سعی خواهیم داشت تا با ارائهٔ مثالهای کاربردی و نمونه‌هایی از پروسه‌های صنعتی، روش برنامه‌نویسی را بررسی نمائیم.