

فصل اول

مفاهیم منطقی

همان گونه که می دانید در سیستم های دیجیتال، اطلاعات ورودی به صورت گسسته (دیجیتال) وارد و به شکل دودویی (binary) نمایش داده می شوند. عملوندهای (Operand) مورد استفاده در محاسبات ممکن است در دستگاه اعداد دودویی و اجزای گسسته دیگر مثل ارقام دهدهی به کدهای دودویی بیان شوند. البته در کامپیوترها مبنای دیگری نظیر مبنای هشت و شانزده نیز مورد استفاده قرار می گیرند.

هدف از ارائه این فصل، معرفی مبنای عددی گوناگون و یادآوری مفاهیم و عملوندهای منطقی، همچنین ارائه توضیح در مورد برخی اصطلاحات استفاده شده در فصول آینده است که به عنوان یک چهارچوب به منظور مطالعه جزئیات بیشتر فصول بعدی استفاده می شوند.

به دلیل گستردگی دامنه استفاده از PLC در صنایع مختلف، این مجموعه جهت استفاده متخصصان و کارشناسان رشته های مختلف تدوین شده است. در این فصل سعی شده تا تمام مطالب و مفاهیم اولیه مورد نیاز در مورد سیستم های کنترل، عناصر و المان های کنترلی، مفاهیم خاص در مورد ریزپردازنده ها و ... توضیح داده شود. بنابراین جهت مطالعه این کتاب نیاز به مراجعه به مراجع دیگر وجود ندارد و مطالب مورد نیاز در ضمیمه آمده است.

۱-۱- مبنای عددی

۱-۱-۱- مبنای دو یا باینری (Binary)

مبنای عددی مورد استفاده در سیستم‌های دیجیتال، مبنای دو است. پیشوند bi به معنی "دو" بوده، این اصطلاح به طور کلی سیستم، مقیاس و یا شرطی را تعریف می‌کند که دو جزء یا حالت دارد. در ریاضیات این اصطلاح نشان‌دهنده سیستم عددی مبنای "دو" می‌باشد که در آن، مقادیر به شکل ترکیب‌هایی از دو رقم صفر و یک بیان می‌شوند.

با استفاده از این دو رقم می‌توان دو حالت (خاموش و روشن یا درست و نادرست) را نشان داد که آن‌ها به نوبه خود با سطح ولتاژ در مدارهای الکترونیکی قابل ارائه است. در حقیقت سیستم عددی دودویی را می‌توان قلب محاسبات رقمی در سیستم‌های دیجیتال دانست.

یک بیت، طبق تعریف یک رقم دودویی است، و هنگامی که به همراه یک کد به کار می‌رود بهتر است که آن را یک کمیت دودویی برابر با "۰" یا "۱" تصور کنیم. نمایش یک گروه از 2^n عنصر به صورت کد به حداقل n بیت نیاز دارد. زیرا n بیت را می‌توان به 2^n طریق مجزا در کنار هم قرار داد. به عنوان مثال گروهی مشتمل بر چهار ($2^2 = 4$) مقدار مجزا را می‌توان با استفاده از یک کد دو بیتی نمایش و هر مقدار مجزا را به یکی از ترکیبات بیتی ۱۰، ۰۱، ۱۱ و ۰۰ نسبت داد. یک گروه با هشت ($2^3 = 8$) جزء، نیازمند یک کد سه بیتی است که هر جزء آن فقط و فقط به یکی از ترکیبات ۰۰۰، ۰۰۱، ۰۱۰، ۰۱۱، ۱۰۰، ۱۰۱، ۱۱۰ و ۱۱۱ نسبت داده می‌شود.

مثالهای فوق نشان می‌دهد که ترکیبات یک کد n بیتی را می‌توان با شمارش دودویی از صفر تا $2^n - 1$ به دست آورد. اعداد دودویی معمولاً به شکل ترکیبات چهار رقمی نوشته می‌شوند و برای اینکه با اعداد دهدهی اشتباه نشوند بعد از آنها حرف b و یا عدد ۲ به صورت زیرنویس ذکر می‌گردد. بنابراین عدد دهدهی ۲ به شکل دودویی 10_b یا $2_{(2)}$ نوشته می‌شود تا با عدد دهدهی ۱۰ اشتباه نگردد.

همان گونه که ذکر شد گاهی برای جلوگیری از بروز اشتباه، مبنای اعداد را در داخل پرانتزی جلوی عدد و به صورت زیرنویس اضافه می‌کنند.

$$2_{(10)} = 10_{(2)} = 10_b \quad (\text{عدد دهدهی } 2)$$

در جدول ۱-۱ معادل اعداد ۰ تا ۹ دهدهی به صورت دودویی نمایش داده شده‌اند.

جدول ۱-۱: معادل دودویی ارقام دهدهی که با چهار بیت نمایش داده شده‌اند.

دهدهی	دودویی
۰	۰۰۰۰
۱	۰۰۰۱
۲	۰۰۱۰
۳	۰۰۱۱
۴	۰۱۰۰
۵	۰۱۰۱
۶	۰۱۱۰
۷	۰۱۱۱
۸	۱۰۰۰
۹	۱۰۰۱

اعداد دودویی نیز به شکل اعداد دهدهی به توان می‌رسند.

$$2^1 = 10_{(2)} \text{ (عدد دهدهی ۲)}$$

$$10^1 = 10_{(10)}$$

$$2^2 = 100_{(2)} \text{ (عدد دهدهی ۴)}$$

$$10^2 = 100_{(10)} \text{ : در مقایسه با}$$

$$2^3 = 1000_{(2)} \text{ (عدد دهدهی ۸)}$$

$$10^3 = 1000_{(10)}$$

در سیستم‌های دیجیتال گاهی لازم است تا مبناهای عددی به یکدیگر تبدیل شوند. این تبدیل

مبنا با مثال زیر روشن می‌گردد.

مثال ۱-۱: عدد $41_{(10)}$ را به مبنا ۲ تبدیل کنید.

روند اجرای عملیات تبدیل به صورت زیر می‌باشد:

ابتدا 41 بر 2 تقسیم شده تا خارج قسمت 20 و باقیمانده 1 به دست آید. خارج قسمت مجدداً بر 2 تقسیم شده تا خارج قسمت و باقیمانده جدیدی حاصل گردد. این روال به همین ترتیب تا زمانی ادامه می‌یابد که خارج قسمت صحیح به دست آمده صفر شود. ضرایب عدد دودویی مطلوب به صورت زیر از باقیمانده‌ها به دست می‌آیند.

خارج قسمت صحیح		باقیمانده	ضریب عدد دودویی
$\frac{41}{2} = 20$	+	۱	$a_0 = 1$
$\frac{20}{2} = 10$	+	۰	$a_1 = 0$
$\frac{10}{2} = 5$	+	۰	$a_2 = 0$
$\frac{5}{2} = 2$	+	۱	$a_3 = 1$
$\frac{2}{2} = 1$	+	۰	$a_4 = 0$
$\frac{1}{2} = 0$	+	۱	$a_5 = 1$

$$\text{جواب: } 41_{(10)} = (a_5 a_4 a_3 a_2 a_1 a_0)_2 = (101001)_2$$

روال ریاضی فوق می تواند به صورت مناسب تری به شکل زیر نمایش داده شود.

عدد صحیح	باقیمانده
41	
20	۱
10	۰
5	۰
2	۱
1	۰
0	۱
جواب = 101001	

تبدیل اعداد صحیح دهدهی به مبنای ۲ شبیه مثال مذکور است بجز اینکه تقسیم می بایست به جای ۲ بر ۲ صورت گیرد.

برای درک چگونگی تبدیل مبنای دودویی به دهدهی مثال ۱-۱ را به صورت عکس بیان می کنیم.

مثال ۱-۲: عدد $(101001)_2$ را به مبنای ۱۰ تبدیل کنید.

$$\begin{aligned} 101001_{(2)} &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 0 + 8 + 0 + 0 + 1 = 41_{(10)} \end{aligned}$$

پس می توان گفت که هر عدد در مبنای ۲ به شکل $(a_n a_{n-1} a_{n-2} \dots a_2 a_1 a_0)_2$ به صورت

حاصلضرب توانهای r در ضرایب مربوطه اش بیان می شود.

$$a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \dots + a_2 \times r^2 + a_1 \times r^1 + a_0$$

اعداد دودویی گرچه برای کامپیوترها استفاده می شوند اما چون رشته های تکراری از صفرها و یک ها هستند تفسیر آنها برای مردم عادی مشکل است. برنامه نویسان و کسانی که با قابلیت های پردازش داخلی کامپیوترها سر و کار دارند جهت سهولت در تفسیر این نوع اعداد از سیستم های عددی اکتال (مبنای ۸) و یا هگزا دسیمال (مبنای ۱۶) استفاده می کنند.

۱-۱-۲- مبنای هشت یا اکتال (Octal)

واژه Octal از کلمه لاتین "Octo" به معنی هشت گرفته شده و به مفهوم سیستم عددی مبنای هشت است که شامل ارقام ۰ تا ۷ می باشد. سیستم مبنای هشت برای ارائه اعداد دودویی به شکل فشرده تر در برنامه نویسی مورد استفاده قرار می گیرد. از آنجایی که این سیستم عددی از هشت رقم تشکیل یافته است و همچنین با استفاده از ۳ بیت می توان هشت ترکیب مختلف را نشان داد، اعداد دودویی برای تبدیل به مبنای ۸ معمولاً به گروه های ۳ بیتی تقسیم می شوند.

به عنوان مثال معادل های دودویی هشت رقم (ارقام ۰ تا ۷) در مبنای هشت در جدول ۱-۲ نشان داده شده است.

جدول ۱-۲: معادل اعداد دودویی در مبنای هشت

مبنای دو	مبنای هشت
۰۰۰	۰
۰۰۱	۱
۰۱۰	۲
۰۱۱	۳
۱۰۰	۴
۱۰۱	۵
۱۱۰	۶
۱۱۱	۷

همان گونه که ذکر شد برای تبدیل اعداد دودویی به مبنای هشت، عدد مورد نظر را به دسته های

۳ بیتی تقسیم می‌کنیم. برای مثال عدد دودویی $(1010011)_2$ را در نظر بگیرید. جهت تبدیل این عدد به مبنای هشت از سمت راست شروع کرده، دسته‌های ۳ بیتی جدا می‌کنیم و در مورد آخرین دسته در صورت کمبود بیت به تعداد تفاضل بیت‌های موجود در دسته و عدد ۳، صفر اضافه می‌نمائیم. با این عمل دسته‌های ۰۱۱، ۰۱۰ و ۰۰۱ به دست می‌آید که با تبدیل این دسته‌ها به مبنای هشت عدد $(123)_8$ حاصل می‌گردد.

$$(1010011)_2 = (001 \ 010 \ 011)_2 \\ = (1 \ 2 \ 3)_8$$

گرچه اعداد در مبنای هشت از نظر ظاهر شبیه اعداد دهدهی هستند اما مفهوم متفاوتی دارند. به عنوان مثال عدد $(123)_8$ معادل با عدد $1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 123$ است در حالی که این عدد در سیستم مبنای هشت به مفهوم $1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$ می‌باشد که معادل عدد دهدهی ۸۳ است. از آنجایی که سیستم مبنای ۸ با مضرب‌هایی از ۳ بیت کار می‌کند و میکرو کامپیوترها معمولاً بر اساس واحدهای ۴، ۸، ۱۶ و ۳۲ و ... هستند این سیستم عددی بیشتر در مینی کامپیوترها و کامپیوترهای بزرگ مورد استفاده قرار می‌گیرد. در مقابل، سیستم عددی مبنای ۱۶ کاربرد بیشتری در میکرو کامپیوترها دارد.

۱-۱-۳- مبنای شانزده یا هگزادسیمال (Hexadecimal)

واژه hexadecimal از ترکیب دو کلمه یونانی hex به معنی ۶ و کلمه لاتین decim به معنی ۱۰ گرفته شده است. سیستم عددی مبنای ۱۶ از ارقام صفر تا ۹ و حروف بزرگ A (معادل اعشاری یا دهدهی عدد ۱۰) تا F (معادل دهدهی عدد ۱۵) تشکیل می‌گردد. هگزا دسیمال که به اختصار "هگز" گفته می‌شود در برنامه‌نویسی برای نشان دادن اعداد دودویی مورد استفاده کامپیوتر در یک قالب فشرده‌تر به کار می‌رود.

اعداد هگزا دسیمال در گروه‌های ۸ بیتی که اساس حافظه و منبع ذخیره اطلاعات در کامپیوتر می‌باشد، جای می‌گیرند. از آنجایی که در چهار بیت می‌توان هر یک از ۱۶ رقم را نشان داد، یک عدد دو رقمی هگز در یک گروه هشت بیتی گنجانده می‌شود. به مثال زیر توجه کنید:

مثال ۱-۳: عدد دهدهی ۸۳ را به مبنای ۱۶ تبدیل کنید.

طبق روش گفته شده در مثال ۱-۱ ابتدا عدد ۸۳ را به ۱۶ تقسیم نموده، خارج قسمت و باقیمانده را به دست می‌آوریم.

خارج قسمت صحیح	باقیمانده	ضریب عدد هگز
$\frac{۸۳}{۱۶} = ۵$	+	$a_0 = ۳$
$\frac{۵}{۱۶} = ۰$	+	$a_1 = ۵$
$۸۳_{(۱۰)} = a_1 a_0_{(۱۶)} = ۵۳_{(۱۶)}$		

برای تبدیل عددی از مبنای ۲ به مبنای ۱۶ باید از سمت چپ بیت‌ها را به دسته‌های ۴ تایی تقسیم نمود و در دسته آخر به تعداد تفاضل بیت‌های موجود و عدد ۴، صفر قرار داد. مثلاً برای تبدیل عدد $(۱۱۱۰۰۱۰۱۰)_۲$ به مبنای ۱۶، دسته‌های ۴ بیتی ۱۰۱۰، ۱۱۰۰ و ۰۰۰۱ را خواهیم داشت و با قرار دادن معادل آنها عدد $(۱CA)_{۱۶}$ حاصل می‌گردد.

$$(۰۰۰۱ \ ۱۱۰۰ \ ۱۰۱۰)_۲$$

$$(\ ۱ \ C \ A)_{۱۶}$$

در جدول ۱-۳ معادل اعداد دودویی و هگز نشان داده شده است.

جدول ۱-۳: معادل دودویی و هگز اعداد در مبنای ۱۰

مبنای ۱۰	مبنای ۱۶	مبنای ۲
۰	۰	۰۰۰۰
۱	۱	۰۰۰۱
۲	۲	۰۰۱۰
۳	۳	۰۰۱۱
۴	۴	۰۱۰۰
۵	۵	۰۱۰۱
۶	۶	۰۱۱۰
۷	۷	۰۱۱۱
۸	۸	۱۰۰۰
۹	۹	۱۰۰۱
۱۰	A	۱۰۱۰
۱۱	B	۱۰۱۱
۱۲	C	۱۱۰۰
۱۳	D	۱۱۰۱
۱۴	E	۱۱۱۰
۱۵	F	۱۱۱۱

بنابر آنچه گفته شد هر گروه ۸ بیتی می تواند هر یک از $2^8 = 256$ عدد مختلف هگزا دسیمال (از 00_H تا FF_H) را در خود جای دهد.

۱-۴-۱- مبنای BCD (Binary Coded Decimal)

اعداد دهدهی کد شده در مبنای دو (دهدهی کد شده به دودویی) با حروف اختصاری BCD نشان داده شده و مبین سیستمی است که در آن، جهت جلوگیری از خطاها و اشتباهات مربوط به گرد کردن ها و تبدیل ها، اعداد دهدهی در مبنای دو کددهی می گردند. در کد BCD، هر رقم در مبنای دهدهی به طور جداگانه به شکل یک عدد دودویی کد می شود. هر یک از ارقام دهدهی صفر تا ۹ در چهار بیت کددهی شده، هر گروه چهار بیتی جهت سهولت در خواندن با یک فاصله از گروه دیگر جدا می گردد. در این روش که ۱-۲-۴-۸ نیز نامیده می شود مطابق جدول ۱-۴ کدهای زیر مورد استفاده قرار می گیرند.

جدول ۱-۴: نمایش اعداد در مبنای دهدهی و معادل BCD آنها

مبنای دهدهی	BCD
۰	۰۰۰۰
۱	۰۰۰۱
۲	۰۰۱۰
۳	۰۰۱۱
۴	۰۱۰۰
۵	۰۱۰۱
۶	۰۱۱۰
۷	۰۱۱۱
۸	۱۰۰۰
۹	۱۰۰۱

به عنوان مثال در کد BCD، عدد $12_{(10)}$ به شکل ۰۰۱۰ ۰۰۰۱ و عدد $96_{(10)}$ به شکل ۱۰۰۱ ۰۱۱۰ نشان داده می شود. در جدول ۱-۵ سیستم های اعداد به صورت یک جا نمایش داده شده اند.

جدول ۱-۵: نمایش اعداد در مبناهای عددی مختلف

دهدهی	باینری	اکتال	هگزادسیمال	BCD
۰	۰۰۰۰	۰	۰	۰۰۰۰
۱	۰۰۰۱	۱	۱	۰۰۰۱
۲	۰۰۱۰	۲	۲	۰۰۱۰
۳	۰۰۱۱	۳	۳	۰۰۱۱
۴	۰۱۰۰	۴	۴	۰۱۰۰
۵	۰۱۰۱	۵	۵	۰۱۰۱
۶	۰۱۱۰	۶	۶	۰۱۱۰
۷	۰۱۱۱	۷	۷	۰۱۱۱
۸	۱۰۰۰	۱۰	۸	۱۰۰۰
۹	۱۰۰۱	۱۱	۹	۱۰۰۱
۱۰	۱۰۱۰	۱۲	A	
۱۱	۱۰۱۱	۱۳	B	
۱۲	۱۱۰۰	۱۴	C	
۱۳	۱۱۰۱	۱۵	D	
۱۴	۱۱۱۰	۱۶	E	
۱۵	۱۱۱۱	۱۷	F	

۱-۲- منطق دودویی

منطق دودویی با متغیرهایی که دو ارزش جدا از هم می‌گیرند و با عملیاتی که مفهوم منطقی دارند سروکار دارد. دو ارزشی که متغیرها می‌گیرند ممکن است با اسامی مختلف نام برده شوند (مثلاً: درست و نادرست یا بلی و خیر و ...) اما برای مقصود ما مناسب است آنها را به صورت بیت در نظر گرفته، مقادیر "۰" و "۱" را به آنها اختصاص دهیم. منطق دودویی به منظور پردازش اطلاعات دودویی به فرم ریاضی به کار می‌رود.

این روش، به خصوص برای تحلیل و طراحی سیستم‌های دیجیتال مناسب است. مدارهای منطقی دیجیتال که محاسبات دودویی را انجام می‌دهند مداراتی هستند که طرز کارشان با توجه به

متغیرهای دودویی و عملیات منطقی به بهترین وجه بیان می‌شود. به این منطق، جبر بول^۱ نیز گفته می‌شود.

۱-۲-۱- تعریف منطق دودویی

منطق دودویی شامل متغیرهای دودویی و عملیات منطقی است. هر متغیر دودویی فقط و فقط می‌تواند دو ارزش "۰" یا "۱" را داشته باشد. در این منطق سه نوع عملیات منطقی اصلی به ترتیب زیر وجود دارد:

۱- AND: این عمل به وسیله یک نقطه و یا عدم وجود عملگر بین دو عملوند (متغیر) نمایش داده می‌شود. مثلاً: $x \cdot y = z$ یا $xy = z$ و بدین صورت تفسیر می‌شود که از نظر منطقی $z = 1$ خواهد بود اگر و تنها اگر $x = 1$ و $y = 1$ باشند، در غیر این صورت $z = 0$ است. (توجه کنید که x و y و z متغیرهای دودویی بوده، تنها می‌توانند مقادیر "۰" یا "۱" را اختیار کنند)

۲- OR: این عمل با علامت "+" نمایش داده می‌شود. برای مثال $x + y = z$ و مفهوم آن این است که $z = 1$ خواهد بود اگر و تنها اگر $x = 1$ یا $y = 1$ باشند و یا اینکه هر دو، مقدار "۱" داشته باشند. در صورتی که $x = 0$ و $y = 0$ باشد آن‌گاه $z = 0$ خواهد بود. به عبارت دیگر $z = 1$ خواهد بود اگر و تنها اگر حداقل یکی از دو متغیر x یا y مقدار "۱" داشته باشند.

۳- NOT: این عمل با علامت پریم (') و یا یک خط در بالای متغیر نشان داده می‌شود. مثلاً $x' = z$ یا $\bar{x} = z$ ، مفهوم آن این است که اگر $x = 1$ باشد آن‌گاه $z = 0$ است و بالعکس اگر $x = 0$ باشد آن‌گاه $z = 1$ خواهد بود.

این تعاریف را می‌توان با استفاده از جداول درستی فهرست نمود. یک جدول درستی، متشکل از تمام ترکیبات ممکن متغیرها و بیانگر ارتباط بین مقادیر آنها و نتایج حاصل از عملیات مربوطه بر روی آنها می‌باشد. جداول درستی AND و OR و NOT در جدول ۱-۶ نشان داده شده‌اند.

جدول ۱-۶: جدول درستی عملیات منطقی AND، OR و NOT

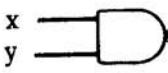
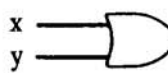
AND			OR			NOT	
x	y	xy	x	y	x + y	x	x'
۰	۰	۰	۰	۰	۰	۰	۱
۰	۱	۰	۰	۱	۱	۱	۰
۱	۰	۰	۱	۰	۱		
۱	۱	۱	۱	۱	۱		



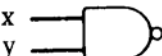
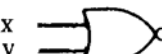
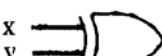
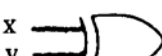
۱-۲-۲- سایر عملگرهای منطقی

هنگامی که عملگرهای منطقی AND و OR بین دو متغیر x و y قرار گیرند به ترتیب توابع بولی $x.y$ و $x + y$ حاصل خواهد شد.

عملگرهای دیگری نیز وجود دارند که توابع جدیدی معرفی می‌نمایند. اگر چه هر تابع را می‌توان بر حسب عملگرهای منطقی AND، OR و NOT بیان کرد اما دلیلی وجود ندارد که عملگر خاصی برای بیان سایر توابع تعیین نگردد. در جدول ۱-۷ کلیه عملگرهای منطقی دیجیتال و توابع متناظر آنها آمده است.

جدول ۱-۷: عملگرهای منطقی و توابع متناظر با آنها

نام	نماد ترسیمی	تابع جبری	جدول درستی	
			x y	F
AND		$F = xy$	۰ ۰	۰
			۰ ۱	۰
			۱ ۰	۰
			۱ ۱	۱
OR		$F = x + y$	۰ ۰	۰
			۰ ۱	۱
			۱ ۰	۱
			۱ ۱	۱

Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = x \oplus y$ $= xy' + x'y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR (XNOR)		$F = x \odot y$ $= xy + x'y'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

۱-۳- مدارهای ترتیبی و فلیپ‌فلاپ‌ها

مدارهای دیجیتالی که با استفاده از توابع ذکر شده مورد توجه قرار می‌گیرند همگی ترکیبی‌اند. خروجی این گونه مدارها در هر لحظه از زمان کاملاً وابسته به ورودی‌های همان زمان است. اگرچه احتمالاً هر سیستم دیجیتال دارای مدارهای ترکیبی است اما در عمل، بیشتر سیستم‌ها شامل عناصر حافظه نیز می‌باشند. این گونه سیستم‌ها باید ضرورتاً بر حسب منطق ترتیبی بررسی شوند. یکی از مهمترین عناصر استفاده شده در مدارهای ترتیبی فلیپ‌فلاپ‌ها هستند.

فلیپ‌فلاپ‌ها سلولهای دودویی هستند که قادر به ذخیره یک بیت اطلاعات می‌باشند. مدار یک فلیپ‌فلاپ دارای دو خروجی است، یکی برای مقدار طبیعی بیت ذخیره شده در آن و دیگری برای متمم آن (Q و Q').

یک فلیپ‌فلاپ قادر است تا وقتی که تحت تأثیر سیگنال ورودی برای تغییر حالت قرار نگرفته، یک حالت دودویی را به طور نامحدود در خود نگهداری کند. (البته تا زمانی که جریان الکتریکی لازم برای فلیپ‌فلاپ تأمین شده باشد).

همان گونه که ذکر شد هر فلیپ‌فلاپ دارای دو ورودی S (ست) و R (ری ست) و دو خروجی Q و Q' می‌باشد. این نوع فلیپ‌فلاپ را گاهی فلیپ‌فلاپ RS و یا لچ RS^1 می‌نامند.

۱-۴- اجزای حافظه و انواع آن

حال که تا حدودی با مفاهیم منطقی آشنا شدیم به ذکر مواردی در زمینه ریز پردازنده‌ها، حافظه‌ها و ... می‌پردازیم.

بیت (bit): این کلمه شکل خلاصه شده $binary\ digit$ بوده، مقدار آن در سیستم‌های عددی دودویی "۰" یا "۱" می‌باشد. در پردازش و ذخیره‌سازی، بیت کوچکترین واحد اطلاعات است که کامپیوتر مورد استفاده قرار می‌دهد و به طور فیزیکی به وسیله پالسی که به یک مدار ارسال می‌گردد و یا به شکل نقطه کوچکی روی دیسک مغناطیسی که قابلیت ذخیره‌سازی "۰" و "۱" را دارد، مشخص می‌گردد.

بیت‌ها کمترین اطلاعات قابل فهم برای انسان را ارائه می‌کنند. (مانند "۰" یا "۱"، درست یا نادرست، ۰ ولت یا ۵ ولت، ۰ ولت یا ۲۴ ولت و ...) بیت‌ها در گروه‌های هشت‌تایی، بایت‌ها را تشکیل می‌دهند که جهت ارائه انواع اطلاعات از جمله حروف الفباء، ارقام صفر تا ۹ و ... مورد استفاده قرار می‌گیرند.

بایت (byte): شکل خلاصه شده **binary term** بوده و یک واحد اطلاعاتی است که از ۸ بیت تشکیل می‌شود. در پردازش و ذخیره‌سازی اطلاعات کامپیوتر، یک بایت معادل یک کاراکتر مثل یک حرف، عدد یا علامت است. چون بایت نشانگر مقدار اطلاعات بسیار کمی می‌باشد مقادیر حافظه و منابع ذخیره اطلاعات کامپیوتر معمولاً به کیلوبایت (۱۰۲۴ بایت) یا مگابایت (۱۰۴۸۵۷۶ بایت) اندازه‌گیری می‌شوند.

از آنجایی که یک بایت از ۸ بیت تشکیل می‌شود و هر بیت می‌تواند یکی از دو مقدار "۰" یا "۱" را داشته باشد پس در یک بایت $2^8 = 256$ حالت مختلف (از ۰۰۰۰۰۰۰۰ تا ۱۱۱۱۱۱۱۱) قابل نمایش است.

کلمه (word): هر کلمه از دو بایت یعنی ۱۶ بیت تشکیل می‌گردد پس در یک کلمه $2^{16} = 65536$ حالت مختلف قابل نمایش است.

۱-۴-۱- گذرگاه یا مسیر عمومی (Bus)

Bus در لغت به معنی اتوبوس یا وسیله حمل و نقل عمومی بوده، در اصطلاح کامپیوتری وسیله‌ای است که حمل و نقل عمومی داده‌ها را بر عهده دارد.

در این گذرگاه، قسمتی که حمل و نقل و جابجایی اطلاعات را بر عهده دارد دیتاباس (**data bus**) می‌نامند و به قسمتی از مسیر عمومی که جابجایی آدرس‌ها را بر عهده دارد آدرس باس (**address bus**) گفته می‌شود.

این گذرگاه مجموعه‌ای از خطوط سخت‌افزاری است که جهت انتقال داده‌ها بین اجزای یک سیستم کامپیوتری مورد استفاده قرار می‌گیرد.

به عبارت دیگر، گذرگاه، یک مسیر مشترک است که بین بخشهای مختلف سیستم از جمله ریزپردازنده، حافظه و درگاههای ورودی خروجی (I/O) و دیگر قسمت‌ها ارتباط برقرار می‌نماید.

در سیستم‌های کامپیوتری، گذرگاه توسط ریزپردازنده کنترل شده، به انتقال انواع مختلفی از اطلاعات اختصاص می‌یابد. به عنوان مثال، گروهی از خطوط، داده‌ها را انتقال داده، گروه دیگر آدرس‌های محل‌های استقرار اطلاعات را منتقل ساخته، یک گروه دیگر سیگنال‌های کنترل را جهت حصول اطمینان از اینکه بخشهای مختلف سیستم از مسیر مشترک خود بدون ایجاد تداخل استفاده می‌کنند، عبور می‌دهند که به این بخش از گذرگاه control bus گویند. گذرگاهها با تعداد بیت‌هایی که در هر لحظه می‌توانند انتقال دهند مشخص می‌شوند به عنوان مثال یک کامپیوتر دارای گذرگاه ۸ بیتی در هر لحظه ۸ بیت از داده‌ها و یک کامپیوتر دارای گذرگاه ۱۶ بیتی در هر لحظه ۱۶ بیت از داده‌ها را انتقال می‌دهند.

۱-۴-۲- حافظه (Memory)

حافظه یکی از قسمت‌های اساسی در سیستم‌های کامپیوتری می‌باشد. از حافظه جهت ذخیره نمودن دستورالعمل‌ها، اطلاعات و نتایج به دست آمده استفاده می‌شود. همان گونه که قبلاً ذکر شد اطلاعات ذخیره شده در حافظه به صورت باینری ("۰" یا "۱") می‌باشد. برای ساختن حافظه معمولاً از نیمه هادی‌ها استفاده می‌شود. در ادامه بحث به ذکر انواع مختلف حافظه در سیستم‌های دیجیتالی و کامپیوتری خواهیم پرداخت.

۱-۴-۳- حافظه‌های با دسترسی تصادفی (RAM)^۱

این حافظه‌ها به گونه‌ای طراحی شده‌اند که در هر لحظه به هر سلول آن می‌توان دست یافت و اطلاعات موجود در آن بخش را خواند.

این نوع حافظه به سه دسته زیر تقسیم می‌گردد:

(الف) حافظه‌های فقط خواندنی (ROM)

(ب) حافظه‌های اغلب خواندنی (RMM)

(ج) حافظه‌های خواندنی نوشتنی (RWM)

در ادامه به شرح این تقسیم‌بندی می‌پردازیم.

الف) حافظه‌های فقط خواندنی (ROM)^۱

حافظه‌های فقط خواندنی امروزه با استفاده از تکنولوژی LSI^۲ ها و VLSI^۳ ها در حد وسیع و انواع مختلف تولید می‌شوند. محتوای این حافظه‌ها غیر قابل تغییر می‌باشد. بدین معنی که وقتی بر روی این حافظه‌ها اطلاعاتی نوشته شود، در حالت عادی نمی‌توان آن را تغییر داد. همچنین با قطع و وصل منبع تغذیه محتوای آنها تغییر نخواهد کرد. به عبارت دیگر این گونه حافظه‌ها پاک نشدنی هستند. به همین جهت این نوع حافظه‌ها برای نگهداری برنامه و یا جداول اطلاعاتی که همیشه ثابت و بدون تغییرند بسیار مناسب می‌باشد.

حافظه‌های فقط خواندنی به دو گروه تقسیم می‌شوند:

۱- ROM (Read Only Memory)

۲- PROM (Programmed Read Only Memory)

در صورتی که محتوای این نوع حافظه در موقع ساخت توسط سازنده برنامه‌ریزی شود به آن ROM گفته می‌شود ولی اگر به گونه‌ای باشد که توسط مصرف کننده و تنها برای یک بار قابل برنامه‌ریزی باشد به آن PROM می‌گویند.

ب) حافظه‌های اغلب خواندنی (RMM)^۳

این نوع حافظه نیز مانند ROM بوده، از آن جهت نگهداری اطلاعات مختلف استفاده می‌شود. اگر در ثبت بیت‌های اطلاعاتی حافظه‌های ROM و PROM^۴ که فقط برای یک بار قابل برنامه‌ریزی هستند اشتباهی رخ دهد راهی جز دور انداختن حافظه وجود ندارد. اما این گروه از حافظه‌ها که می‌توان محتویات آنها را پاک کرد این ضعف را برطرف می‌کند و می‌توان از آنها چندین بار استفاده نمود و برنامه‌های مختلف را در آنها ضبط و پس از اتمام کار آنها را پاک کرد. این نوع حافظه‌ها بر اساس نوع پاک شدن اطلاعات به دو گروه تقسیم‌بندی می‌شوند. (البته خاصیت پاک شدن آنها مربوط به تکنولوژی ساخت آنها است).

۱- EPROM (Erasable Programmed Read Only Memory)

۲- EEPROM (Electrically Erasable Programmed ROM)^۴

1 - Read Only Memory

2 - Large Scale Integration

3 - Read Mostly Memory

۴ - این نوع حافظه را با E² PROM نیز نمایش می‌دهند.

ج) حافظه‌های خواندنی نوشتنی (RWM)^۱

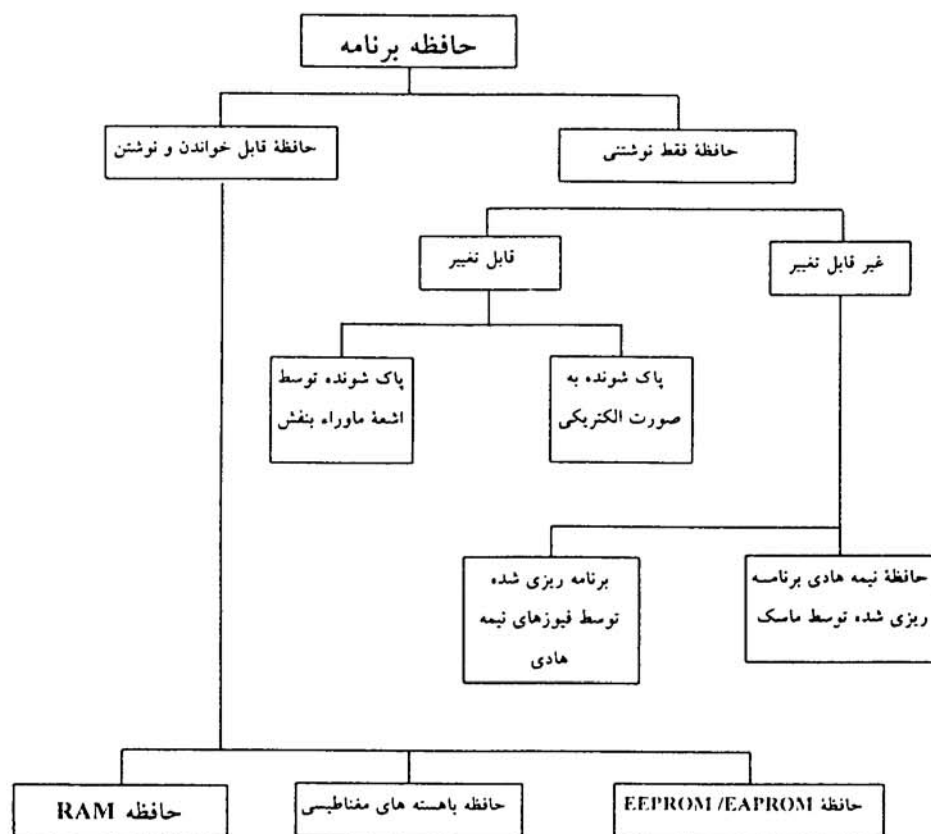
این نوع حافظه اغلب در سیستم‌های میکروپروسسوری برای نگهداری نتایج موقت و میانی در هنگام اجرای یک برنامه به کار می‌روند. ساختمان داخلی هر سلول حافظه RWM اساساً یک فلیپ‌فلاپ می‌باشد که دو حالت پایدار برای ذخیره‌سازی منطق "۱" یا "۰" داشته، پیش‌بینی‌های لازم برای تغییر حالت سریع سلول حافظه در آن به عمل آمده است و سیگنال‌های کنترل تعیین می‌کنند که اطلاعات در محل آدرس داده شده نوشته یا از آن خوانده شود. این گروه به دو بخش استاتیک و دینامیک تقسیم‌بندی می‌شوند.

در جدول ۸-۱ انواع حافظه‌ها و اطلاعات دیگر در رابطه با نحوه پاک شدن و ... آمده است.

جدول ۸-۱: انواع حافظه‌ها و سایر اطلاعات در مورد آنها

محتوای حافظه پس از قطع تغذیه	برنامه‌ریزی	طریقه پاک شدن	نوع حافظه	ساختمان
ناپایدار	الکتریکی	الکتریکی	حافظه با دسترسی اتفاقی	RAM
پایدار	توسط کارخانه‌سازنده	غیرممکن	حافظه فقط خواندنی	ROM
	الکتریکی		ROM قابل برنامه‌ریزی	PROM
		PROM قابل پاک شدن	EPROM	
		PROM قابل برنامه‌ریزی مجدد	REPROM	
		ROM با قابلیت پاک شدن الکتریکی	EEPROM	
		ROM با قابلیت پاک شدن الکتریکی	EAPROM ^۲	

در شکل ۱-۱ چگونگی تقسیم‌بندی حافظه‌ها به صورتی دیگر آورده شده است.



شکل ۱-۱: نحوه تقسیم بندی حافظه ها

حال که با اجزای تشکیل دهنده حافظه و انواع حافظه ها آشنا شدیم به بحث پیرامون سیستم های کنترل می پردازیم.

۱-۵- سیستم های کنترل

در سالهای اخیر، سیستم های کنترل اهمیت فزاینده ای در توسعه و پیشرفت تکنولوژی جدید

یافته‌اند. هر یک از جنبه‌های فعالیت روزمره ما عملاً تحت تأثیر نوعی سیستم کنترل قرار دارد. مثلاً در محدوده زندگی فردی، کنترل‌کننده‌های خودکار در سیستم‌های تهویه مطبوع، دما و رطوبت هوای خانه‌ها و ساختمان‌ها را در حد مطلوب نگاه می‌دارند.

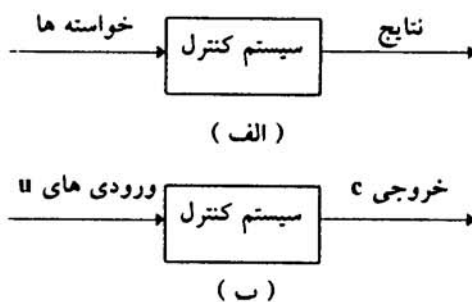
سیستم‌های کنترل در تمام بخشهای صنعت نظیر کنترل کیفیت محصولات، خط مونتاژ خودکار، کنترل ماشین‌ابزار، تکنولوژی فضایی و سیستم‌های نظامی، کنترل کامپیوتری، سیستم‌های حمل و نقل، سیستم‌های قدرت، آدماهای ماشینی و در موارد بسیار دیگری، به فراوانی یافت می‌شوند. صرف‌نظر از اینکه چه نوع سیستم کنترلی در اختیار داریم، سه بخش اساسی را می‌توان در آن مشخص کرد:

۱- خواسته‌های ما از سیستم کنترل

۲- اجزای سیستم کنترل

۳- نتایج

در شکل ۱-۲ (الف) ارتباط اساسی میان این سه بخش به شکل نمودار بلوکی نمایش داده شده است. همان‌طور که در شکل ۱-۲ (ب) دیده می‌شود این سه بخش اساسی به ترتیب با عناوین ورودی‌ها، اجزای سیستم و خروجی‌ها که اصطلاحات علمی تری هستند نیز شناخته می‌شوند.

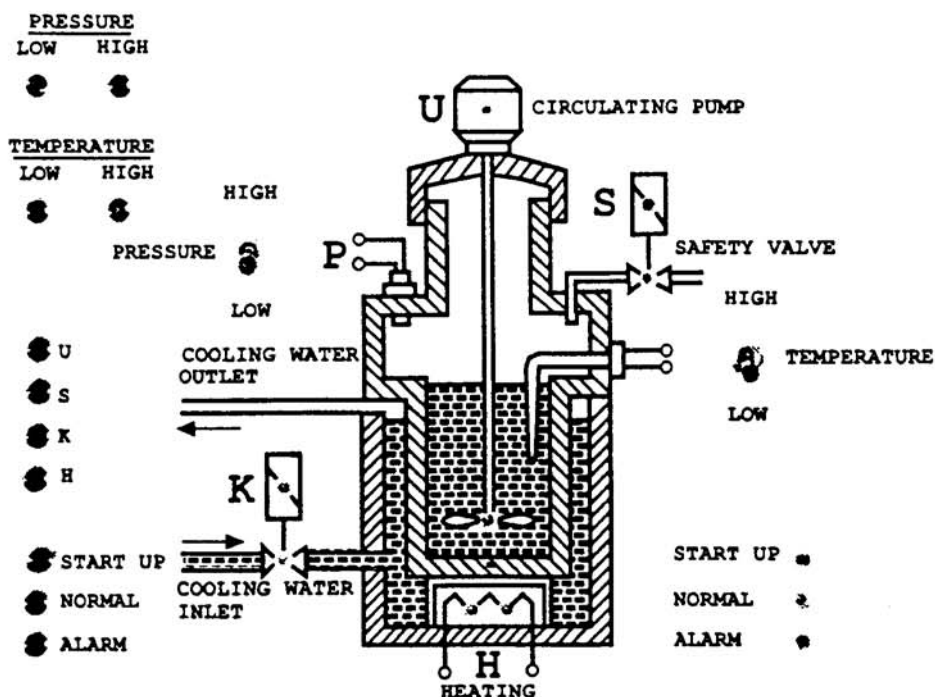


شکل ۱-۲: بخش‌های یک سیستم کنترل

به‌طور کلی، هدف سیستم‌های کنترل این است که خروجی‌های c را به شیوه از پیش تعیین شده‌ای به وسیله ورودی‌های u از طریق اجزای سیستم، کنترل کند. ورودی‌های سیستم کنترل، سیگنال‌های تحریک و خروجی‌های آن، متغیرهای تحت کنترل نیز نامیده می‌شوند.

به عنوان مثال با ثابت نگه داشتن متغیرهای زیر می توان فرآیند شیمیایی موجود در شکل ۳-۱ را کنترل نمود.

- موتور همزن الکتریکی (CIRCULATING PUMP)
- فشار هوای درون مخزن و شیر اطمینان (SAFETY VALVE و PRESSURE)
- درجه حرارت سیال داخل مخزن (TEMPERATURE)
- مقدار آب خنک کننده (COOLING WATER)
- زمان (TIME)



شکل ۳-۱: یک فرآیند شیمیایی تحت کنترل و موارد مهم در کنترل آن

۱-۵-۱- ساختار سیستم‌های کنترل

سیستم‌های کنترل از لحاظ ساختاری به دو بخش زیر تقسیم می‌شوند.

الف) سیستم‌های کنترل حلقه باز (Open Loop)

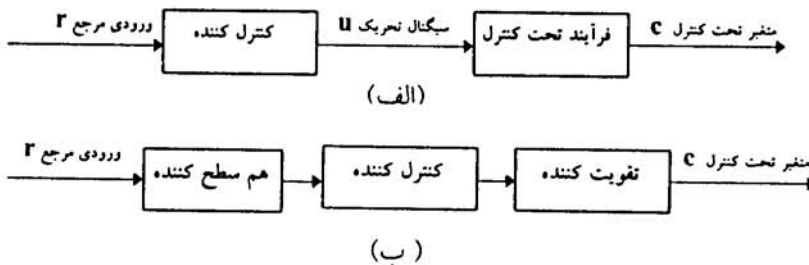
ب) سیستم‌های کنترل حلقه بسته (Closed Loop)

الف) سیستم‌های کنترل مدار باز (Open Loop)

در این گونه سیستم‌ها که در شکل ۴-۱ الف) نشان داده شده است خواسته‌های ما از عملکرد آن به خوبی برآورده نمی‌شود و تنها به دلیل سادگی و اقتصادی بودن سیستم‌های کنترل مدار باز، در بسیاری موارد می‌توان آنها را در حال کار یافت.

ماشین لباسشویی مثال بارزی از یک سیستم کنترل مدار باز است. زیرا عموماً مدت زمان شستشو از طریق قضاوت و تخمین فرد استفاده‌کننده تعیین می‌شود. یک ماشین لباسشویی خودکار باید بتواند دائماً میزان تمیزی لباس‌های در حال شستشو را بررسی کند تا هر زمان که به اندازه کافی تمیز شدند، بطور خودکار خاموش شود.

همان‌طور که در شکل ۴-۱ الف) نشان داده شده است، اجزای یک سیستم کنترل مدار باز را معمولاً می‌توان به دو دسته تقسیم نمود. کنترل‌کننده و فرآیند تحت کنترل. یک فرمان یا سیگنال ورودی r به کنترل‌کننده اعمال می‌شود، خروجی کنترل‌کننده به عنوان سیگنال تحریک u ، فرآیند را کنترل می‌کند به نحوی که متغیر تحت کنترل c بر اساس استانداردهای از پیش تعیین شده‌ای عمل کند.



شکل ۴-۱: قسمت‌های مختلف یک سیستم کنترل مدار باز

در موارد ساده، کنترل‌کننده می‌تواند بر حسب طبیعت سیستم، یک تقویت‌کننده، اتصالات

مکانیکی و یا وسایل کنترلی دیگر باشد. در موارد پیچیده تر کنترل الکترونیکی، کنترل کننده ممکن است یک حسابگر الکترونیکی نظیر یک ریزپردازنده باشد.

همان گونه که در شکل ۱-۴ (ب) نشان داده شده، گاهی ممکن است علاوه بر واحد کنترل کننده قسمتهای دیگری در سیستم کنترل مدار باز وجود داشته باشد. در ادامه بحث به توضیح در مورد اجزای یک سیستم کنترل مدار باز (اجزای نشان داده شده در شکل ۱-۴ (ب)) می پردازیم.

۱- واحد ورودی (ورودی مرجع r): اطلاعات از طریق کلیدها و حس کنندها (سنسورها) به ورودی ها منتقل می گردد. این اطلاعات معمولاً به شکل سیگنال های الکتریکی است که می تواند به صورت دیجیتال یا آنالوگ باشد.

۲- واحد متناسب کننده یا هم سطح کننده (Conditioning Unit): این قسمت در صورتی مورد نیاز است که میزان ولتاژ یا جریان سیگنال های ورودی، برای واحد کنترل کننده مناسب نباشد. بنابراین باید مقدار ولتاژ یا جریان به میزان معینی تنظیم گردد.

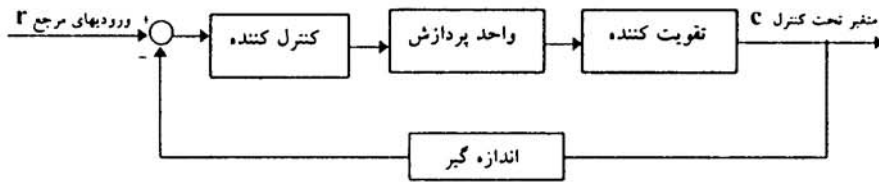
۳- واحد کنترل کننده (واحد پردازش): این واحد، هسته سیستم کنترل را تشکیل داده، اعمال منطقی و محاسباتی در این قسمت صورت می گیرد.

۴- واحد تقویت کننده: در این قسمت سیگنال های ضعیف دریافت شده از واحد پردازش، تقویت و ارسال شده تا وسایل کنترل کننده مانند شیرهای کنترل شونده توسط جریان الکتریکی (Solenoid Valves) یا کنتاکتورها را کنترل، یا لامپ های نشان دهنده را روشن نماید.

۵- واحد خروجی (متغیر تحت کنترل c): از این قسمت فرمانهای سیستم کنترل به اجرا کننده های فرمان یعنی محرک ها (Actuators) و ... ارسال می شود.

ب) سیستم های کنترل مدار بسته (Closed Loop)

آنچه برای کنترل دقیق تر و قابل انعطاف تر لازم بوده و در سیستم کنترل مدار باز وجود ندارد یک اتصال یا فیدبک از خروجی، به ورودی سیستم است. برای دستیابی به کنترل دقیق تر، سیگنال تحت کنترل $c(t)$ باید فیدبک شده، با ورودی مرجع r مقایسه شود و سیگنال تحریکی متناسب با تفاضل ورودی و خروجی به سیستم اعمال و در نتیجه، خطا تصحیح شود. سیستمی با یک یا چند مسیر فیدبک، نظیر آنچه که هم اکنون تشریح شد یک سیستم مدار بسته نامیده می شود. در شکل ۱-۵ طرز کار این گونه سیستم ها نشان داده شده است.



شکل ۱-۵: قسمت‌های مختلف در سیستم‌های کنترل مدار بسته

همان گونه که ذکر شد در این سیستم‌ها، مقدار خروجی توسط عنصری اندازه‌گیر (سنسور) اندازه‌گیری می‌شود و در مقایسه‌ای با مقدار مطلوب، اختلاف بین خروجی و مقدار مطلوب محاسبه شده، به عنوان سیگنال خطا معرفی می‌گردد. این سیگنال خطا به واحد کنترل کننده ارسال و سپس کنترل کننده در حلقه کنترل، به عنوان تصمیم گیرنده عمل می‌نماید. به این ترتیب که با توجه به تنظیم‌های از پیش تعیین شده، فرمان لازم را برای تصحیح خطا صادر می‌نماید.

۱-۶- انواع سیستم‌های کنترل

سیستم‌های کنترل را می‌توان بنا به روش کنترل آنها به دو دسته زیر تقسیم نمود:

۱- سیستم‌های کنترل سخت‌افزاری

۲- سیستم‌های کنترل نرم‌افزاری

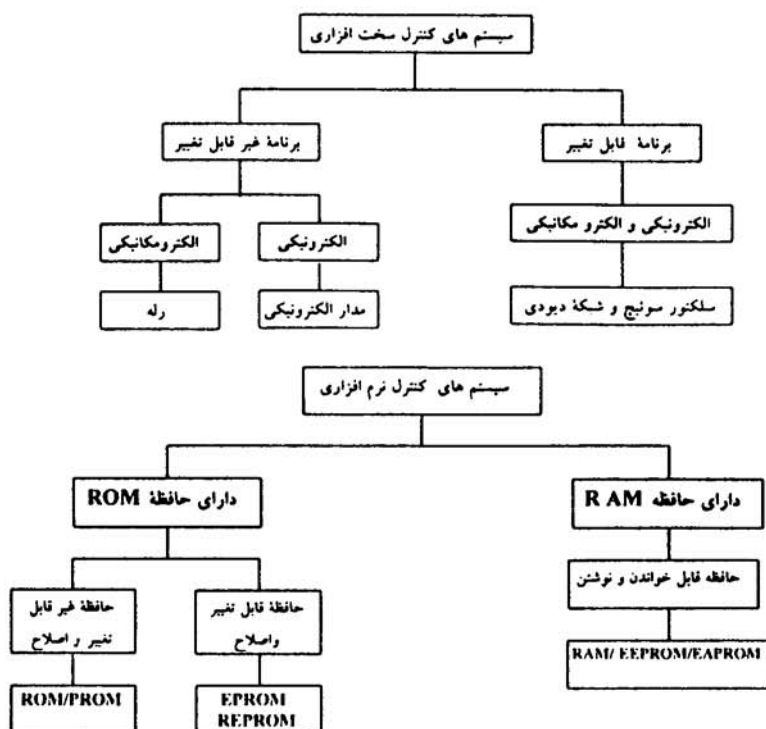
۱-۶-۱- سیستم‌های کنترل سخت‌افزاری

این سیستم‌ها شامل مداراتی هستند که با استفاده از رله‌ها و عناصر الکترونیکی مانند دیودها و ترانزیستورها ساخته می‌شوند. برنامه کنترل در این سیستم‌ها نتیجه روابط بین عناصر مدار الکتریکی است و به راحتی قابل تغییر نمی‌باشد. به عبارت دیگر تغییر در برنامه کنترل به معنی تغییر در سخت‌افزار سیستم است البته در برخی از این کنترل کننده‌ها که با استفاده از کلیدهای انتخاب کننده (Selectors)، یا شبکه دیودی (Diode Matrix) ساخته می‌شوند، برنامه کنترل را می‌توان تا اندازه‌ای تغییر داد. اگرچه اعمال این تغییرات، محدود و در برخی موارد بسیار مشکل است.

۱-۶-۲- سیستم‌های کنترل نرم‌افزاری

این کنترل کننده‌ها دارای حافظه‌ای هستند که برنامه کنترل در آن ذخیره می‌شود. مهمترین مزیت

این سیستم‌ها در آن است که نحوه کنترل را با تغییر برنامه و بدون نیاز به تغییر در سخت‌افزار سیستم می‌توان عوض کرد، زیرا نحوه کنترل سیستم توسط سخت‌افزار سیستم تعیین نمی‌شود بلکه برنامه‌ای که در حافظه ذخیره شده یعنی نرم‌افزار سیستم، نحوه کنترل را مشخص می‌کند لذا این سیستم‌ها بسیار قابل انعطاف بوده، کاربردهای فراوانی دارند. بسته به نوع حافظه این سیستم‌ها، شیوه تغییر در برنامه‌ها متفاوت است. اگر از حافظه RAM استفاده شود، بدون دخالت فیزیکی و تنها با اضافه یا کم نمودن چند سطر برنامه می‌توان برنامه جدید را به اجرا در آورد. در صورتی که از حافظه ROM استفاده شود به اجرا درآوردن برنامه جدید تنها با تعویض حافظه ROM امکان‌پذیر است. در شکل ۱-۶ انواع سیستم‌های کنترل نشان داده شده‌اند.



شکل ۱-۶: سیستم‌های کنترل سخت‌افزاری و نرم‌افزاری

حال که با مفاهیم منطقی و ساختار سیستم‌های کنترل آشنا شدیم در فصل آینده به بحث پیرامون PLC و سخت‌افزار و نرم‌افزار آن خواهیم پرداخت.

فصل دوم

ساختار PLC

۲-۱- PLC

”PLC“ از عبارت Programmable Logic Controller به معنای کنترل کننده منطقی قابل برنامه‌ریزی گرفته شده است. PLC، کنترل کننده‌ای نرم‌افزاری است که در قسمت ورودی، اطلاعاتی را به صورت باینری دریافت و آنها را طبق برنامه‌ای که در حافظه‌اش ذخیره شده پردازش می‌نماید و نتیجه عملیات را نیز از قسمت خروجی به صورت فرمانهایی به گیرنده‌ها و اجراکننده‌های فرمان (Actuators) ارسال می‌کند.

به عبارت دیگر PLC عبارت از یک کنترل کننده منطقی است که می‌توان منطق کنترل را توسط برنامه برای آن تعریف نمود و در صورت نیاز، به راحتی آن را تغییر داد. وظیفه PLC قبلاً بر عهده مدارهای فرمان رله‌ای بود که استفاده از آنها در محیط‌های صنعتی جدید منسوخ گردیده است. اولین اشکالی که در این مدارها ظاهر می‌شود آن است که با افزایش تعداد رله‌ها حجم و وزن مدار فرمان، بسیار بزرگ شده، همچنین موجب افزایش قیمت آن می‌گردد. برای رفع این اشکال، مدارهای فرمان الکترونیکی ساخته شدند ولی با وجود این، هنگامی که تغییری در روند یا عملکرد ماشین صورت می‌گیرد مثلاً در یک دستگاه پرس، ابعاد، وزن، سختی و زمان قرار

گرفتن قطعه زیر بازوی پرس تغییر می‌کند، لازم است تغییرات بسیاری در سخت‌افزار سیستم کنترل داده شود. به عبارت دیگر اتصالات و عناصر مدار فرمان باید تغییر کند.

با استفاده از PLC تغییر در روند تولید یا عملکرد ماشین به آسانی صورت می‌پذیرد، زیرا دیگر لازم نیست سیم‌کشی‌ها (Wiring) و سخت‌افزار سیستم کنترل تغییر کند و تنها کافی است چند سطر برنامه نوشت و به PLC ارسال کرد تا کنترل مورد نظر تحقق یابد.

از طرف دیگر قدرت PLC در انجام عملیات منطقی، محاسباتی، مقایسه‌ای و نگهداری اطلاعات به مراتب بیشتر از تابلوهای فرمان معمولی است. PLC به طراحان سیستم‌های کنترل این امکان را می‌دهد که آنچه را در ذهن دارند در اسرع وقت بیازمایند و به ارتقای محصول خود بیندیشند، کاری که در سیستم‌های قدیمی مستلزم صرف هزینه و به خصوص زمان است و نیاز به زمان، گاهی باعث می‌شود که ایده مورد نظر هیچ‌گاه به مرحله عمل در نیاید.

هر کس که با مدارهای فرمان الکتریکی رله‌ای کار کرده باشد به خوبی می‌داند که پس از طراحی یک تابلوی فرمان، چنانچه نکته‌ای از قلم افتاده باشد، مشکلات مختلفی ظهور نموده، هزینه‌ها و اتلاف وقت بسیاری را به دنبال خواهد داشت. به علاوه گاهی افزایش و کاهش چند قطعه در تابلوی فرمان به دلایل مختلف مانند محدودیت فضا، عملاً غیرممکن و یا مستلزم انجام سیم‌کشی‌های مجدد و پرهزینه می‌باشد.

اکنون برای توجه بیشتر به تفاوت‌ها و مزایای PLC نسبت به مدارات فرمان رله‌ای، مزایای مهم PLC را نسبت به مدارات یاد شده بر می‌شماریم.

- ۱- استفاده از PLC موجب کاهش حجم تابلوی فرمان می‌گردد.
- ۲- استفاده از PLC مخصوصاً در فرآیندهای عظیم موجب صرفه‌جویی قابل توجهی در هزینه، لوازم و قطعات می‌گردد.
- ۳- PLC‌ها استهلاک مکانیکی ندارند، بنابراین علاوه بر عمر بیشتر، نیازی به تعمیرات و سرویس‌های دوره‌ای نخواهند داشت.
- ۴- PLC‌ها انرژی کمتری مصرف می‌کنند.
- ۵- PLC‌ها بر خلاف مدارات رله کنتاکتوری، نویزهای الکتریکی و صوتی ایجاد نمی‌کنند.
- ۶- استفاده از یک PLC منحصربه‌فرد و فرآیند خاصی نیست و با تغییر برنامه می‌توان به آسانی

از آن برای کنترل پروسه‌های دیگر استفاده نمود.

۷- طراحی و اجرای مدارهای کنترل و فرمان با استفاده از PLC ها بسیار سریع و آسان است.

۸- برای عیب‌یابی مدارات فرمان الکترومکانیکی، الگوریتم و منطق خاصی را نمی‌توان پیشنهاد نمود. این امر بیشتر تجربی بوده، بستگی به سابقه‌آشنایی فرد تعمیرکار با سیستم دارد. در صورتی که عیب‌یابی در مدارات فرمان کنترل شده توسط PLC به آسانی و با سرعت بیشتری انجام می‌گیرد.

۹- PLC ها می‌توانند با استفاده از برنامه‌های مخصوص، وجود نقص و اشکال در پروسه تحت کنترل را به سرعت تعیین و اعلام نمایند.

در جدول ۱-۲ مزایای PLC نسبت به مدارات فرمان رله‌ای و همچنین مدارهای منطقی الکترونیکی و کامپیوتر بر شمرده شده است.

جدول ۱-۲: مزایای PLC نسبت به کنترل‌کننده‌های دیگر

PLC	مدارهای رله‌ای	مدارهای منطقی الکترونیکی	کامپیوتر	
ارزان	نسبتاً ارزان	ارزان	گران قیمت	قیمت با توجه به عملکرد
خیلی کوچک	بزرگ و حجیم	خیلی کوچک	نسبتاً کوچک	حجم و ابعاد
خیلی سریع	کند	نسبتاً سریع	خیلی سریع	سرعت کنترل
خوب	عالی	خوب	کاملاً خوب	نویز الکتریکی
نصب و برنامه‌نویسی ساده است	طراحی و نصب مشکل است	طراحی مشکل است	برنامه‌نویسی مشکل است	نصب و بهره‌برداری
آری	خیر	خیر	آری	توانایی محاسبات پیچیده را دارد؟
بسیار آسان	خیلی مشکل	مشکل	آسان	تغییر نحوه کنترل و ایجاد تغییرات

۲-۲- تفاوت PLC با کامپیوتر

استفاده از کامپیوتر معمولی مستلزم آموزش‌های نسبتاً طولانی، صرف وقت و هزینه‌های بسیار

است. چنانچه کنترل فرآیندی مورد نظر باشد استفاده از کامپیوتر معمولی به مراتب پیچیده‌تر و در اغلب موارد عملاً ناممکن می‌شود. علاوه بر آن برای انطباق کامپیوتر با فرآیند مورد نظر، طراحی، ساخت و یا لااقل بررسی و خرید تجهیزات خاص برای انطباق، کاری طاقت فرسا است. بسیاری از صنعتگران نیاز به کارگیری سیستم‌های اتوماتیک را عملاً احساس نموده و دریافته‌اند که تولید بدون به کارگیری اتوماسیون، اقتصادی نمی‌باشد. از طرف دیگر، صنعتگران آموزش‌های مبسوط به این شاخه از صنعت را در محدوده وظایف خود نمی‌دانند.

PLC وسیله‌ای است که درست به همین دلایل ساخته شده و اتوماسیون را با کمترین هزینه و به بهترین شکل ممکن در اختیار قرار می‌دهد. استفاده از PLC بسیار ساده بوده، نیاز به آموزش‌های مفصل، طولانی و پرهزینه ندارد.

از آنجایی که این وسیله به منظور پاسخگویی به کاربردهای صنعتی طراحی شده است، تمامی مسائل مربوط به آن حل شده، هیچ مشکلی در راه استفاده از آن وجود ندارد. طراحان خطوط تولید با بهره‌گیری از این وسیله قابل انعطاف به سرعت می‌توانند نیازمندیهای مصرف‌کنندگان خود را تأمین و در اسرع وقت تواناییهای خود را با نیازمندیهای بازار هماهنگ نمایند.

از شرکت‌های سازنده PLC می‌توان ALLEN BRADLEY، AEG، SIEMENS، MITSUBISHI، OMRON و ... را نام برد. گرچه از عرضه PLC توسط سازندگان مختلف چند دهه سالی می‌گذرد و در ماشین آلات و خطوط تولید خریداری شده از خارج کشور نیز به وفور مشاهده می‌شود استفاده از این وسیله بسیار قابل انعطاف توسط طراحان و ماشین سازان داخلی کمتر به چشم می‌خورد. از جمله عواملی که موجب تأخیر در بهره‌برداری از PLC توسط طراحان داخلی گردیده است عبارتند از:

- ۱- ارتباط مشکل با منابع تأمین‌کننده خارجی.
- ۲- عدم دسترسی به موقع به اطلاعات سیستم‌ها.
- ۳- عدم پشتیبانی مؤثر سازندگان از تجهیزات فروخته شده خود.
- ۴- هزینه بالای تجهیزات خارجی.
- ۵- هزینه بالای آموزش در خارج از کشور.

شرکت‌های داخلی نیز با توجه به مشکلات یاد شده و برای پرکردن خلأ موجود اقدام به

طراحی و ساخت چند نوع PLC نموده‌اند. PLC‌های مذکور، کلیه امکانات استاندارد PLC‌های متداول را داشته، از نمونه‌های خارجی با قابلیت‌های مشابه ارزانترند. این PLC‌ها به خوبی آزمایش گردیده، از پشتیبانی کامل آموزش و خدمات پس از فروش برخوردار می‌باشند.

از شرکت‌های داخلی تولید کننده PLC و سیستم‌های اتوماسیون می‌توان شرکت کنترونیک را نام برد. این شرکت با به کارگیری دانش متخصصین داخلی اقدام به تولید چندین سیستم PLC با قابلیت‌های متفاوت جهت استفاده در صنایع مختلف و کاربردهای متنوع نموده است.

این شرکت همچنین مبتکر زبان برنامه‌نویسی خاصی جهت سیستم‌های PLC تولید شده می‌باشد که بسیار شبیه به زبان برنامه‌نویسی ابداع شده توسط شرکت SIEMENS یعنی STEP 5 است. PLC یاد شده با نمونه‌های خارجی مشابه خود به خوبی رقابت می‌کند.

در فصول آینده با شبیه‌سازهای ابداع شده توسط این شرکت آشنا شده، روش برنامه‌نویسی را توسط این شبیه‌سازها مرور خواهیم نمود. گرچه در این کتاب سعی بر این است که سیستم PLC به طور کلی معرفی شود اما در مورد برنامه‌نویسی به ناچار باید از یک زبان برنامه‌نویسی خاص استفاده نماییم. امروزه کاربرد PLC‌های ساخت شرکت زیمنس در سرتاسر دنیا گسترش یافته، این نوع PLC بیش از هر PLC دیگری در صنایع مختلف به چشم می‌خورد. بنابراین مؤلف ترجیحاً از زبان برنامه‌نویسی STEP 5 (S5) که زبان برنامه‌نویسی سیستم‌های PLC زیمنس می‌باشد استفاده نموده است. همان‌گونه که گفته شد این زبان بسیار شبیه به زبان ابداع شده توسط شرکت کنترونیک یعنی CSTL بوده، و تفاوت این دو زبان برنامه‌نویسی تنها در چند مورد جزئی است. جهت آشنایی بیشتر خوانندگان با این زبان برنامه‌نویسی (CSTL)، در برخی موارد سعی شده تا برنامه مورد نظر برای انجام یک پروسه به هر دو زبان S5 و CSTL نوشته شود تا خوانندگان شباهت‌های این دو زبان را بیشتر درک کنند.

لازم به ذکر است که اصول کلی زبانهای برنامه‌نویسی مختلف تقریباً یکسان بوده، خواننده می‌تواند با یادگیری یکی از زبانهای مذکور، سایر زبانها را به آسانی درک و از آنها استفاده نماید.

سازندگان سیستم‌های PLC برای برنامه‌نویسی سیستم‌های خود، هر یک از زبان منحصر به فردی استفاده می‌نمایند که از نظر اصولی همگی تابع یک سری قوانین منطقی و کلی بوده، تنها تفاوت آنها در ساختار برنامه‌نویسی و نمادهای استفاده شده است.

از زبانهای ابداع شده توسط سازندگان PLC می توان S5 ، FST ، OMRON ، CSTL ، ALLEN BRADLEY و ... را نام برد.

۲-۳- کاربرد PLC در صنایع مختلف

امروزه کاربرد PLC در صنایع و پروسه های مختلف صنعتی به وفور به چشم می خورد. در زیر تعدادی از این کاربردها آورده شده است.

- صنایع اتومبیل سازی - شامل: عملیات سوراخ کاری اتوماتیک، اتصال قطعات و همچنین تست قطعات و تجهیزات اتومبیل، سیستم های رنگ پاش، شکل دادن بدنه به وسیله پرس های اتوماتیک و ...

- صنایع پلاستیک سازی - شامل: ماشین های ذوب و قالب گیری تزریقی، دمش هوا و سیستم های تولید و آنالیز پلاستیک و ...

- صنایع سنگین - شامل: کوره های صنعتی، سیستم های کنترل دمای اتوماتیک، وسایل و تجهیزاتی که در ذوب فلزات استفاده می شوند و ...

- صنایع شیمیایی - شامل: سیستم های مخلوط کننده، دستگاه های ترکیب کننده مواد با نسبت های متفاوت و ...

- صنایع غذایی - شامل: سیستم های سائریفوز، سیستم های عصاره گیری و بسته بندی و ...

- صنایع ماشین - شامل: صنایع بسته بندی، صنایع چوب، سیستم های سوراخ کاری، سیستم های اعلام خطر و هشدار دهنده، سیستم های استفاده شده در جوش فلزات و ...

- خدمات ساختمانی - شامل: تکنولوژی بالابری (آسانسور)، کنترل هوا و تهویه مطبوع، سیستم های روشنایی خودکار و ...

- سیستم های حمل و نقل - شامل: جرثقیل ها، سیستم های نوار نقاله، تجهیزات حمل و نقل و ...

- صنایع تبدیل انرژی (برق، گاز و آب) شامل: ایستگاه های تقویت فشار گاز، ایستگاه های تولید نیرو، کنترل پمپ های آب، سیستم های تصفیه آب و هوای صنعتی، سیستم های تصفیه و بازیافت گاز و ...

۲-۴- سخت افزار PLC

از لحاظ سخت افزاری می توان قسمت های تشکیل دهنده یک سیستم PLC را به صورت زیر تقسیم نمود:

- ۱- واحد منبع تغذیه PS (Power Supply)
- ۲- واحد پردازش مرکزی CPU (Central Processing Unit)
- ۳- حافظه (Memory)
- ۴- ترمینال های ورودی (Input Module)
- ۵- ترمینال های خروجی (Output Module)
- ۶- مدول ارتباط پروسسوری CP (Communication Processor)
- ۷- مدول رابط IM (Interface Module)

۲-۴-۱- مدول منبع تغذیه (PS)

منبع تغذیه ولتاژهای مورد نیاز PLC را تأمین می کند. این منبع معمولاً از ولتاژهای ۲۴ ولت DC و ۱۱۰ یا ۲۲۰ ولت AC، ولتاژ ۵ ولت DC را ایجاد می کند. ماکزیمم جریان قابل دسترسی منطبق با تعداد مدول های خروجی مصرفی است. لازم به ذکر است که ولتاژ منبع تغذیه باید کاملاً تنظیم شده (رگوله)^۱ باشد. جهت دستیابی به راندمان بالا معمولاً از منابع تغذیه سوئیچینگ استفاده می شود. ولتاژی که در اکثر PLC ها استفاده می گردد ولتاژ ۵ یا ۵/۲ ولت DC است. (در برخی موارد، منبع تغذیه و واحد کنترل شونده در فاصله زیادی نسبت به یکدیگر قرار دارند بنابراین ولتاژ منبع، ۵/۲ ولت انتخاب می شود تا افت ولتاژ حاصل از بُعد مسافت بین دو واحد مذکور جبران گردد).

برای تغذیه رله ها و محرک ها (Actuator) معمولاً از ولتاژ ۲۴ ولت DC به صورت مستقیم (بدون استفاده از هیچ کارت ارتباطی) استفاده می شود. در برخی موارد نیز از ولتاژهای ۱۱۰ یا ۲۲۰ ولت AC با استفاده از یک کارت رابط به نام Relay Board استفاده می گردد. (در مورد تغذیه رله ها

احتیاج به رگولاسیون دقیق نیست).

در برخی شرایط کنترلی لازم است تا در صورت قطع جریان منبع تغذیه، اطلاعات موجود در حافظه و همچنین محتویات شمارنده‌ها، تایمرها و فلگ‌های^۱ پایدار بدون تغییر باقی بمانند. در این موارد از یک باتری جنس "Lithium" جهت حفظ برنامه در حافظه استفاده می‌گردد. به این باتری "Battery Back up" می‌گویند. ولتاژ این نوع باتری‌ها معمولاً ۲/۸ ولت تا ۳/۶ ولت می‌باشد. از آنجایی که این باتری نقش مهمی در حفظ اطلاعات موجود در حافظه دارد در اکثر PLC ها یک چراغ نشان دهنده تعبیه شده و در صورتی که ولتاژ باتری به سطحی پائین‌تر از مقدار مجاز ۲/۸ ولت برسد این نشان دهنده روشن می‌گردد. این نشان دهنده به Battery Low LED معروف است. در صورت مشاهده روشن شدن این نشان دهنده لازم است که باتری مذکور تعویض گردد. برای تعویض باتری ابتدا باید به وسیله یک منبع تغذیه، ولتاژ مدول مورد نظر را تأمین و سپس اقدام به تعویض باتری نمود.

۲-۴-۲- واحد پردازش مرکزی (CPU)

CPU یا واحد پردازش مرکزی در حقیقت قلب PLC است. وظیفه این واحد، دریافت اطلاعات از ورودی‌ها، پردازش این اطلاعات مطابق دستورات برنامه و صدور فرمانهایی است که به صورت فعال یا غیرفعال نمودن خروجی‌ها ظاهر می‌شود. واضح است که هر چه سرعت پردازش CPU بالاتر باشد زمان اجرای یک برنامه کمتر خواهد بود.

۲-۴-۳- حافظه (Memory)

همان گونه که در فصل اول اشاره شد، حافظه محلی است که اطلاعات و برنامه کنترل در آن ذخیره می‌شوند. علاوه بر این، سیستم عامل^۲ که عهده‌دار مدیریت کلی بر PLC است در حافظه قرار دارد. تمایز در عملکرد PLC ها، عمدتاً به دلیل برنامه سیستم عامل و طراحی خاص CPU

۱ - در مورد تایمر، شمارنده و فلگ در همین فصل به تفصیل سخن خواهیم گفت.

آنهاست. در حالت کلی در PLC ها دو نوع حافظه وجود دارد:

- ۱- حافظه موقت (RAM) که محل نگهداری فلگ ها، تایمرها، شمارنده ها و برنامه های کاربر^۱ است.
- ۲- حافظه دائم (EPROM, EEPROM) که جهت نگهداری و ذخیره همیشه برنامه کاربر استفاده می گردد.

۲-۴-۴- ترینال ورودی (Input Module)

این واحد، محل دریافت اطلاعات از فرآیند یا پروسه تحت کنترل می باشد. تعداد ورودی ها در PLC های مختلف، متفاوت است. ورودی هایی که در سیستم های PLC مورد استفاده قرار می گیرند در حالت کلی به صورت زیر می باشند:

الف) ورودی های دیجیتال (Digital Input)

ب) ورودی های آنالوگ (Analog Input)

الف) ورودی های دیجیتال یا گسته

این ورودی ها که معمولاً به صورت سیگنال های صفر یا ۲۴ ولت DC می باشند، گاهی برای پردازش توسط CPU به تغییر سطح ولتاژ نیاز دارند. معمولاً برای انجام این عمل مدول هایی خاص در PLC در نظر گرفته می شود. جهت حفاظت مدارات داخلی PLC از خطرات ناشی از اشکالات بوجود آمده در مدار یا برای جلوگیری از ورود نویزهای موجود در محیط های صنعتی ارتباط ورودی ها با مدارات داخلی PLC توسط کوپل کننده های نوری^۲ (Optical Coupler) انجام می گیرد. به دلیل ایزوله شدن ورودی ها از بقیه اجزای مدار داخلی PLC، هر گونه اتصال کوتاه و یا اضافه ولتاژ نمی تواند آسیبی به واحدهای داخلی PLC وارد آورد.

ب) ورودی های آنالوگ یا پیوسته

این گونه ورودی ها در حالت استاندارد $\pm 10 \text{ V DC}$ ، $0 - 20 \text{ mA}$ و $4 - 20 \text{ mA}$ است.

1 - User Programs

۲ - طرز کار این عنصر الکترونیکی در ضمیمه ۱ توضیح داده شده است.

بوده، مستقیماً به مدول‌های آنالوگ متصل می‌شوند. مدول‌های ورودی آنالوگ، سیگنال‌های دریافتی پیوسته (آنالوگ) را به مقادیر دیجیتال تبدیل نموده، سپس مقادیر دیجیتال حاصل توسط CPU پردازش می‌شوند.

۲-۴-۵- ترینال خروجی (Output Module)

این واحد، محل صدور فرمانهای PLC به پروسه تحت کنترل می‌باشد. تعداد این خروجی‌ها در PLC های مختلف متفاوت است. خروجی‌های استفاده شده در PLC ها به دو صورت زیر وجود دارند:

الف) خروجی‌های دیجیتال (Digital Output)

ب) خروجی‌های آنالوگ (Analog Output)

الف) خروجی‌های دیجیتال یا گسسته

این فرمانهای خروجی به صورت سیگنال‌های ۰ یا ۲۴ ولت DC بوده که در خروجی ظاهر می‌شوند، بنابراین هر خروجی از لحاظ منطقی می‌تواند مقادیر "۰" (غیرفعال) یا "۱" (فعال) را داشته باشد. این سیگنال‌ها به تقویت کننده‌های قدرت یا مبدل‌های الکتریکی ارسال می‌شوند تا مثلاً ماشینی را به حرکت در آورده (فعال نمایند) یا آن را از حرکت باز دارند (غیرفعال نمایند). در برخی موارد استفاده از مدول‌های خروجی دیجیتال جهت رسانیدن سطوح سیگنال‌های داخلی PLC به سطح ۰ یا ۲۴ ولت DC الزامی است.

ب) خروجی‌های آنالوگ یا پیوسته

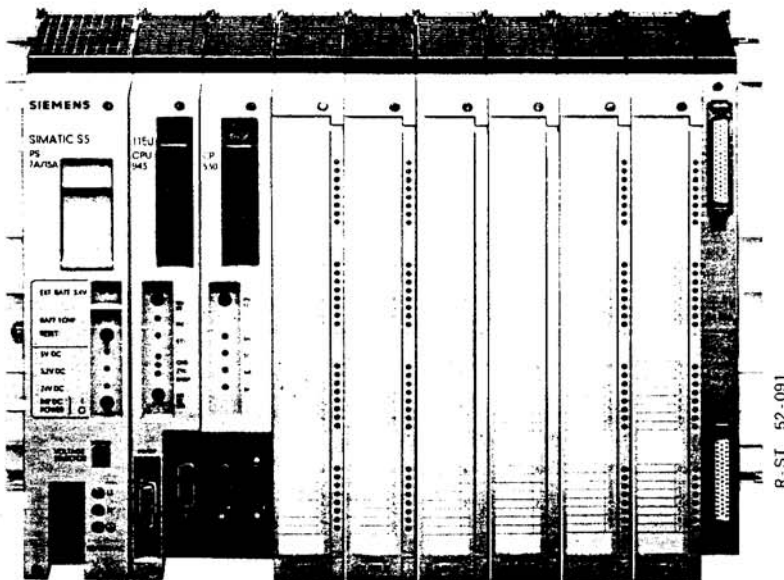
سطوح ولتاژ و جریان استاندارد خروجی می‌تواند یکی از مقادیر $V_{DC} \pm 10$ - ۰، mA ۲۰-۴۰ یا mA ۲۰-۰ باشد. معمولاً مدول‌های خروجی آنالوگ، مقادیر دیجیتال پردازش شده توسط CPU را به سیگنال‌های پیوسته (آنالوگ) مورد نیاز جهت پروسه تحت کنترل تبدیل می‌نمایند. این خروجی‌ها به وسیله واحدی به نام Isolator از سایر قسمت‌های داخلی PLC ایزوله می‌شوند. بدین ترتیب مدارات حساس داخلی PLC از خطرات ناشی از امکان بروز اتصالات ناخواسته خارجی محافظت می‌گردند.

۲-۴-۶- مدول ارتباط پرسوری (CP)

این مدول، ارتباط بین CPU مرکزی را با CPU های جانبی برقرار می‌سازد.

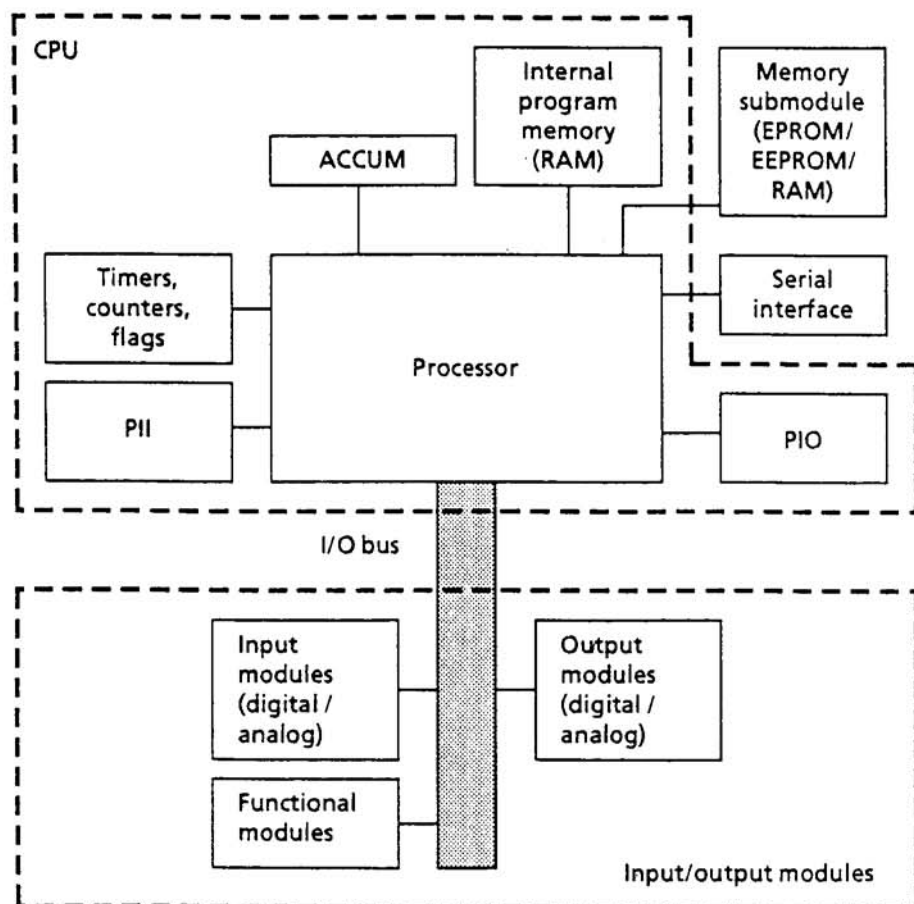
۲-۴-۷- مدول رابط (IM)

در صورت نیاز به اضافه نمودن واحدهای دیگر ورودی و خروجی به PLC یا جهت اتصال پانل اپراتوری و پروگرامر به PLC از این مدول ارتباطی استفاده می‌شود. در صورتی که چندین PLC به صورت شبکه به یکدیگر متصل شوند از واحد IM جهت ارتباط آنها استفاده می‌گردد. در شکل ۱-۲ شمای کلی یک PLC نشان داده شده است.



شکل ۱-۲: شمای کلی یک PLC و قسمت‌های مختلف آن

در شکل ۲-۲ نحوه ارتباط CPU با سایر قسمت‌های PLC نشان داده شده است.



شکل ۲-۲: نحوه ارتباط CPU با سایر قسمت‌های PLC

در ادامه بحث به توضیح در مورد برخی از مفاهیم موجود در شکل ۲-۲ خواهیم پرداخت.

۲-۵- تصویر ورودی‌ها (PII)^۱

قبل از اجرای برنامه، CPU وضعیت تمام ورودی‌ها را بررسی و در قسمتی از حافظه به

نام PII نگهداری می‌نماید. جز در موارد استثنایی و تنها در بعضی از انواع PLC، غالباً در حین اجرای برنامه، CPU به ورودی‌ها مراجعه نمی‌کند بلکه برای اطلاع از وضعیت هر ورودی به سلول مورد نظر در PII رجوع می‌کند. در برخی موارد این قسمت از حافظه، IIT (Input Image Table) نیز خوانده می‌شود.

۲-۶- تصویر خروجی‌ها (PIO)^۱

هرگاه در حین اجرای برنامه یک مقدار خروجی بدست آید، در این قسمت از حافظه نگهداری می‌شود. جز در موارد استثنایی و تنها در برخی از انواع PLC، غالباً در حین اجرای برنامه، CPU به خروجی‌ها مراجعه نمی‌کند بلکه برای ثبت آخرین وضعیت هر خروجی به سلول مورد نظر در PIO رجوع می‌کند و در پایان اجرای برنامه، آخرین وضعیت خروجی‌ها از PIO به خروجی‌های فیزیکی منتقل می‌گردند. در برخی موارد این قسمت از حافظه را OIT (Output Image Table) نیز می‌گویند.

۲-۷- فلگ‌ها، تایمرها و شمارنده‌ها

هر CPU جهت اجرای برنامه‌های کنترلی از تعدادی تایمر، فلگ و شمارنده استفاده می‌کند. فلگ‌ها محل‌هایی از حافظه‌اند که جهت نگهداری وضعیت برخی نتایج و یا خروجی‌ها استفاده می‌شوند. جهت شمارش از شمارنده و برای زمان‌سنجی از تایمر استفاده می‌گردد. فلگ‌ها، تایمرها و شمارنده‌ها را از لحاظ پایداری و حفظ اطلاعات ذخیره شده می‌توان به دو دسته کلی تقسیم نمود. ۱- پایدار (Retentive) به آن دسته از فلگ‌ها، تایمرها و شمارنده‌هایی اطلاق می‌گردد که در صورت قطع جریان الکتریکی (منبع تغذیه) اطلاعات خود را از دست ندهند.

۲- ناپایدار (Non-Retentive) این دسته برخلاف عناصر پایدار، در صورت قطع جریان الکتریکی تغذیه، اطلاعات خود را از دست می‌دهند.

تعداد فلگ‌ها، تایمرها و شمارنده‌ها در PLC‌های مختلف متفاوت می‌باشد اما تقریباً در تمامی

موارد قاعده‌ای کلی جهت تشخیص عناصر پایدار و ناپایدار وجود دارد.

فرض کنید که در یک نوع PLC خاص تعداد فلگ‌ها، تایمرها و شمارنده‌ها به ترتیب m و n و p باشد. تعداد عناصر پایدار و ناپایدار با یکدیگر برابر است. بنابراین تعداد این عناصر به ترتیب $\frac{m}{3}$ و $\frac{n}{3}$ و $\frac{p}{3}$ می‌باشد. المان‌های که شماره آنها از مقادیر نصف یعنی $\frac{m}{3}$ و $\frac{n}{3}$ و $\frac{p}{3}$ کوچکتر باشد پایدار و بقیه، عناصر ناپایدار هستند. به‌طور کلی می‌توان گفت که نیمه اول این عناصر، پایدار و نیمه دوم ناپایدار می‌باشد.

فرض کنید که در یک نوع PLC، ۱۶ شمارنده ($C0 - C15$) تعریف شده باشد بنا بر قاعده مذکور شمارنده‌های $C0 - C7$ همگی پایدار و شمارنده‌های $C8 - C15$ ناپایدار می‌باشند.

۲-۸- انبارک یا آکومولاتور (ACCUM)

انبارک یا آکومولاتور یک ثبات منطقی است که جهت بارگذاری یا به عبارت دیگر لود نمودن^۲ اطلاعات استفاده می‌گردد. از این ثبات جهت بارگذاری اعداد ثابت در تایمرها، شمارنده‌ها، مقایسه‌گرها و ... استفاده می‌شود.

۲-۹- گذرگاه عمومی ورودی / خروجی (I/O bus)

همان‌گونه که قبلاً ذکر شد وظیفه پردازش اطلاعات در PLC برعهده CPU است. بنابراین برای اجرای برنامه بایستی CPU با ورودی‌ها، خروجی‌ها و سایر قسمت‌های PLC در ارتباط بوده، با آنها تبادل اطلاعات داشته باشد. سیستمی که مرتبط کننده CPU با قسمت‌های دیگر است bus نامیده می‌شود. این سیستم توسط CPU اداره می‌شود و در حقیقت علت کاهش چشمگیر اتصالات در PLC به دلیل وجود همین سیستم می‌باشد. سیستم bus از سه بخش زیر تشکیل شده است.

۱- باس داده (Data bus)

۲- باس آدرس (Address bus)

۳- باس کنترل (Control bus)

مشخصات سیستم باس بستگی به نوع CPU مورد استفاده و حجم کلی حافظه دارد. مثلاً برای پردازشگر Z80 باس داده دارای ۸ خط ارتباطی است که ارسال و دریافت هشت بیت یا یک بایت اطلاعات را امکان‌پذیر می‌سازد. بنابراین ورودی‌ها، خروجی‌ها و حافظه‌ها بایستی در دسته‌های هشت بیتی یا یک بایتی سازماندهی شوند.

هر بایت اطلاعات بایستی آدرس منحصر به فردی داشته باشد، هرگاه CPU بخواهد اطلاعاتی را با بایت بخصوصی رد و بدل نماید با استفاده از آدرس منحصر به فرد آن بایت این تبادل اطلاعات امکان‌پذیر می‌گردد. وقتی تمام امکانات CPU با بایت مورد نظر از لحاظ آدرس و خط ارتباطی فراهم شد CPU توسط باس کنترل، جهت حرکت و زمان رد و بدل اطلاعات را سازماندهی می‌کند.

۲-۱۰- روشهای مختلف آدرس دهی

جهت آدرس دهی معمولاً از سه روش زیر استفاده می‌شود.

۱- Fixed Address: در این روش تمام ورودی‌ها و خروجی‌ها دارای آدرس ثابتی می‌باشند.

نظیر این نوع آدرس دهی را در مینی PLC های کنترنیک می‌توان یافت.

۲- Slot Address: در این روش، آدرس دهی قابل تغییر می‌باشد و این تغییر آدرس توسط

شیارهای مورد نظر و فیش‌های زائده‌دار انجام می‌گیرد.

۳- Flexible Address: در این روش آدرس دهی که قابل تغییر نیز می‌باشد سوئیچ‌هایی (دیپ

سوئیچ) در نظر گرفته شده که با استفاده از آنها می‌توان آدرس دهی را تغییر داد.

حال که با سخت افزار سیستم‌های PLC آشنا شدیم به بررسی نرم‌افزار آن می‌پردازیم.

۲-۱۱- نرم‌افزار PLC

در PLC ها سه نوع نرم‌افزار قابل تعریف است:

۱- نرم‌افزاری که کارخانه سازنده با توجه به توان سخت‌افزاری سیستم تعریف می‌کند که به آن

Operating System یا به اختصار OS گویند. مثلاً در PLC زیمنس مدل U 100 تعداد ۱۶ تایمر

(T0 - T15) تعریف شده است و اگر در برنامه نویسی از تایمر شماره ۱۸ یعنی T18 استفاده شود

سیستم عامل دستور مذکور را به عنوان یک دستور اشتباه قلمداد کرده، برنامه اجراء نخواهد شد.

لازم به ذکر است که این نرم افزار ثابت بوده، قابل تغییر نمی باشد بنابراین از نوع فقط خواندنی است و معمولاً در EPROM یا E²PROM ذخیره می شود.

۲- نرم افزاری که برنامه نوشته شده توسط استفاده کننده (User) را به زبان قابل فهم ماشین تبدیل می نماید. این برنامه منحصر به کارخانه سازنده بوده، نام خاصی نیز دارد. معروف ترین و پر کاربردترین این نرم افزارها، نرم افزار S5 می باشد که توسط شرکت زیمنس ابداع گردیده است. این نرم افزار هم مانند OS قابل تغییر نیست و بایستی در ROM ذخیره و برای اجرا به RAM پروگرامر ارسال گردد.

۳- نرم افزار یا برنامه ای که توسط استفاده کننده نوشته می شود و به آن User Program گویند. این نرم افزار در هر لحظه قابل تغییر بوده، خواندنی / نوشتنی است. این نرم افزار در RAM و یا در EPROM و یا در E²PROM ذخیره و در صورت ایجاد هر گونه اشکال در RAM از مدول ذکر شده مجدداً در RAM کپی شده، اجرا می گردد.

همان گونه که ذکر شد هر PLC شامل سخت افزار و نرم افزار می باشد. در صفحات گذشته به طور اجمال به توضیح در مورد سیستم های سخت افزاری و همچنین نرم افزار PLC پرداختیم. واضح است که برای وارد کردن برنامه کنترلی یا نرم افزار کنترلی به سخت افزار، نیاز به یک واحد برنامه نویسی یا پروگرامر می باشد. در ادامه بحث به تشریح واحد برنامه نویسی (Programming Unit) می پردازیم.

۲-۱۲- واحد برنامه نویسی (PG)

در استفاده و به کارگیری PLC علاوه بر آشنایی با نحوه کار، آشنایی با واحد برنامه نویسی آن نیز ضروری است زیرا توسط این واحد قادر خواهیم بود با PLC ارتباط برقرار نماییم. به این ترتیب که برنامه کنترل دستگاه را نوشته، آن را در حافظه PLC قرار داده، اجرای آن را از PLC می خواهیم. این واحد بسیار شبیه به کامپیوترهای معمولی است، یعنی دارای یک صفحه نشان دهنده (مونیتور) و صفحه کلید می باشد. تفاوت این واحد با کامپیوتر معمولی، تک منظوره بودن آن می باشد بدین معنی که از PG تنها می توان جهت ارتباط برقرار نمودن با PLC مربوطه استفاده نمود. با استفاده از PG می توان از وضعیت و چگونگی اجرای برنامه مطلع شد. صفحه نمایش واحد

برنامه‌نویسی به ما نشان می‌دهد که کدام ورودی روشن یا خاموش است، PLC توسط خروجی‌ها دستور فعال شدن یا توقف کار کدام ماشین‌ها را می‌دهد و در حقیقت نحوه اجرای برنامه در صفحه نمایش ظاهر می‌شود. بنابراین در صورتی که اشکالی در برنامه وجود داشته باشد یا ایرادی در اجرای برنامه پیدا شود، از این طریق می‌توان به آن پی برد. پس می‌توان گفت که واحد برنامه‌نویسی در عیب‌یابی برنامه کنترل دستگاهها و سیستم‌های تحت کنترل و بررسی علت توقف آنها نقش به سزایی دارد. به وسیله PG می‌توان تغییرات عملوندها یعنی ورودی‌ها، خروجی‌ها و همچنین تایمرها و شمارنده‌های برنامه در حال اجرا را به صورت Real Time ملاحظه نمود. در اکثر PLCها و به کمک PG می‌توان با دستور خاصی نظیر STATUS وضعیت عملوندها را در حین اجرای برنامه مشاهده نمود.

فصل سوم

مقدمه‌ای بر زبان STEP 5

یک برنامه‌کنترلی مجموعه‌دستورالعمل‌هایی 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 خانواده 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:^۱

- | | |
|-------------|--------------------------------|
| 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 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 مانوس 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 نقشه الکتریکی مدار فرمان همان سیستم است. برخی از نمادهای مورد استفاده در این روش برنامه‌نویسی در زیر آمده است.



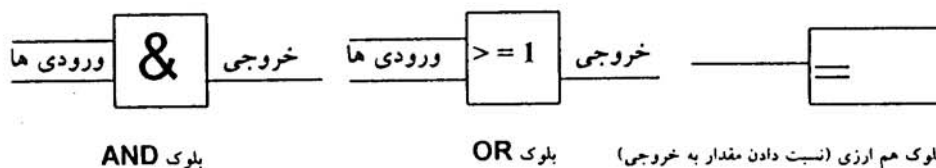
نماد کنتاکت در حالت عادی باز^۱

نماد سیم‌پیچ رله

شکل ۳-۱: برخی نمادهای مورد استفاده 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 هم‌ارزی (نسبت دادن مقدار به خروجی)

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

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 برنامه 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 در عبارت صورت می‌گیرد عملکرد گفته می‌شود. عملکردهای مهم عبارتند از: OR، AND، و ... که در جدول ۱-۳ نشان داده شده‌اند.

جدول ۱-۳: 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 ۲-۳ بخشهای Statement نشان داده شده است.

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

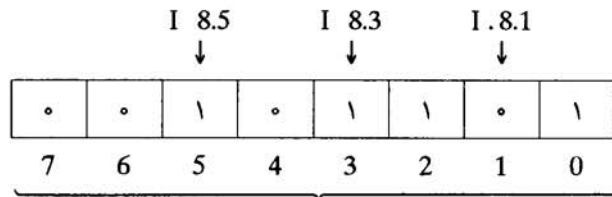
عبارت		Statement
عملوند		عملکرد Operation
آدرس عملوند Parameter	نوع عملوند Operand Identifier	
1.0	I	A
7.1	I	A
2.5	Q	=
0.6	I	O
8.4	I	O
1.3	Q	=

همان گونه که در جدول ۲-۳ 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.

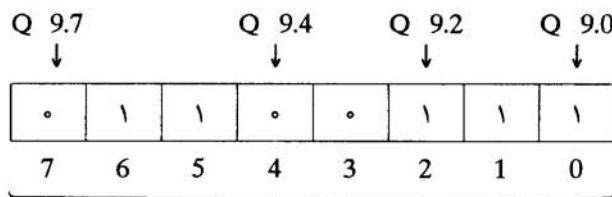
آدرس ورودی‌ها، خروجی‌ها و فلگ‌ها

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 و سپس موقعیت و آدرس بیت مربوطه در آن بایت تعیین گردد.

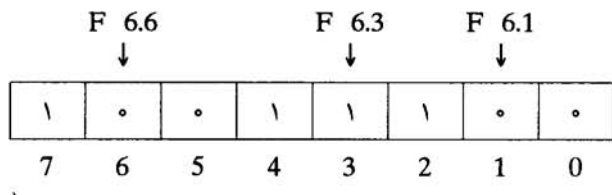
در شکل ۳-۳ نحوه آدرس دهی ورودی، خروجی و فلگ نشان داده شده است.



1IB 8 (بایت ورودی ۸)



2QB 9 (بایت خروجی ۹)



3FY 6 (بایت فلگ ۶)

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

طریقه 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 برنامه کنترل با استفاده از حروف و اعداد لاتین به صورت جملات منطقی و پشت سر هم نوشته می شود و هر حرف، معرف یک واژه انگلیسی است. مثلاً حرف A معرف AND، O معرف OR، I معرف Input و Q معرف Output می باشد.

در روش STL، برنامه به صورت مجموعه ای از دستورات است که به هر دستور یک رشته (خط برنامه) یا Statement گفته می شود و هر دستور یا خط برنامه معمولاً یکی از ترکیب های منطقی ریاضی یعنی ترکیب های AND، OR، NOT، هم ارزی و ... را دربر دارد. علاوه بر ترکیب های فوق وسایل نرم افزاری فلگ، فلیپ فلاپ، شمارنده، تایمر و ... در اکثر PLC ها وجود دارند که در حقیقت بخشی از زبان برنامه نویسی می باشند.

در برنامه نوشته شده به روش STL به چندین سطر که عمل خاصی انجام می دهند یک Segment می گویند. یک برنامه (Program) از چندین Segment تشکیل می گردد. لازم به ذکر است که یک برنامه می تواند تنها شامل یک Segment باشد.

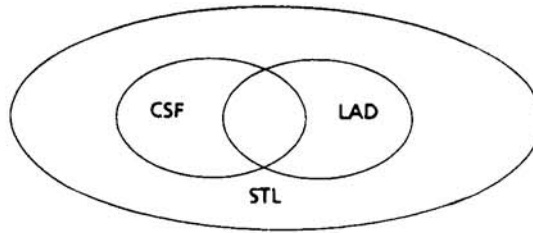
در زیر یک نمونه کاملاً ساده از برنامه نوشته شده به روش STL را می بینیم.

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 می گیرد. سطر انتهایی این برنامه نشان دهنده پایان یافتن برنامه (Block End) است.

هر یک از روشهای برنامه نویسی دارای مشخصات خاصی می باشد. اکثر سازندگان PLC، در طراحی نرم افزار برنامه نویسی به گونه ای عمل کرده اند که می توان برنامه نوشته شده در یک روش را به روشهای دیگر تبدیل نمود. در زبان STEP 5 لزوماً برنامه نوشته شده به روش STL را نمی توان به LAD یا CSF تبدیل نمود اما برنامه نوشته شده به روش LAD و CSF همواره قابل تبدیل به STL می باشد. شکل ۳-۴ گویای این مطلب است.



شکل ۳-۴: نمایش امکان تبدیل و ترجمه برنامه 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 دهنده 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 وظیفه خاصی را انجام می‌دهند و عملکرد آنها بر یکدیگر تأثیرگذار است تشکیل می‌گردند. بنابراین در نوشتن برنامه آنها باید برای هر بخش، برنامه جداگانه‌ای نوشته شود و پس از صحت عملکرد، آنها را که برنامه‌های فرعی نامیده می‌شوند در یک برنامه اصلی فراخوانی نمود. به عبارت دیگر باید این برنامه‌های فرعی را در برنامه اصلی چنان سازمان داد تا بتوان فرآیندهای پیچیده را کنترل نمود. همان گونه که ذکر شد در برنامه‌نویسی سازمان‌یافته، 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 سازمان‌یافته 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 وظیفه 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 نوشته‌اند 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 برنامه استفاده‌کننده (User Program) نوشته می‌شود در بلوک‌های زیر دسته‌بندی شده است.

۳-۳-۱- بلوک‌های برنامه (PB)^۱

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

به عبارت دیگر، PB ها بلوک‌های تشکیل دهنده برنامه کنترل یک فرآیند می‌باشند. به دلایل مختلفی از جمله، جلوگیری از پیچیدگی و طولانی شدن، برنامه کنترل فرآیند به قسمت‌های کوچتری که همان PB ها هستند تفکیک می‌گردد. در این نوع PLC می‌توان از تعداد ۲۵۶ بلوک برنامه (از شماره 0 PB الی 255 PB) 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 مشخص کننده پایان برنامه هر بلوک می‌باشد.

۳-۳-۲- بلوک‌های ترتیبی (SB)^۲

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 تک مرحله‌ای (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 وسیله یک کلید راه‌اندازی می‌شود. بنابراین این گونه بلوک‌ها را بلوک‌های ترتیبی یا Sequence Block می‌گویند.

۳-۳-۳- بلوک‌های تابع ساز (FB)^۳

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

گیرند. به عنوان مثال در برخی پروسه‌ها ضرب دو عدد باینری مکرراً مورد استفاده قرار می‌گیرد. در چنین مواردی لازم نیست که برنامه ضرب هر بار توسط استفاده کننده و در هر جایی 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 تابع ساز 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- بدنه 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 دسته زیر تقسیم نمود:

الف) بلوک‌های تابع ساز 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 دفترچه راهنما که حاوی اطلاعاتی در مورد چگونگی وارد نمودن پارامترها و دیگر اطلاعات می‌باشد در اختیار کاربر قرار می‌گیرد.

ب) بلوک‌های تابع ساز 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 مرحله ۱ از عملوند I 0.0 و در مرحله ۲ از عملوند I 0.6 به عنوان ورودی استفاده نمود.

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

۳-۴- بلوک‌های اطلاعاتی (DB)^۱

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

در این نوع PLC دقیقاً تعداد ۲۵۶ بلوک (DB 0 - DB 255) قابل تعریف است. همان گونه که می‌دانید در برخی از پروسه‌ها لازم است مواردی همچون پیغام‌ها، آلارم‌ها و ... بر روی صفحه 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 کلمه 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 DB بدون 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^۱. DB‌ها می‌توانند در تمامی بلوک‌ها فراخوانده شوند.

۳-۳-۵- بلوک‌های سازماندهی (OB)

OB ها ساختار برنامه استفاده کننده را مشخص می‌کنند و هر OB با یک شماره خاص مشخص می‌شود. در تعریف OB می‌توان گفت که OB ها بلوک‌هایی هستند که هر یک عمل خاصی را انجام می‌دهند و در واقع ارتباط بین سیستم عامل و برنامه 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 برنامه استفاده کننده و آخرین سطر آن، پایان برنامه استفاده کننده است. بنابراین ساختار اجرایی برنامه استفاده کننده در OB 1 تعیین می‌گردد.

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

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

۱ - در مورد اعتبار DB ها در فصل آینده توضیح خواهیم داد.

می شود برنامه این بلوک اجرا می گردد. بنابراین می توان 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 باتری 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 عکس العمل سیستم را در برابر اشکالات باتری می توان به نحو دلخواهی در OB 34 برنامه ریزی نمود.

در مقایسه با روش برنامه نویسی سازمان یافته که جهت برنامه نویسی پروسه های عظیم و پیچیده استفاده می شود روش برنامه نویسی دیگری نیز با نام Linear Programming وجود دارد. این روش جهت نوشتن برنامه های ساده و برنامه هایی که تعداد سطرهای آن از ۵۰۰ سطر کمتر باشد مورد استفاده قرار می گیرد. این برنامه را می توان در OB 1 یا PB 1 نوشت.

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

در زبان برنامه نویسی S5 عملوندهای زیر مورد استفاده قرار می گیرند.

I	(Inputs)	ورودی ها، از پروسه تحت کنترل به PLC.
Q	(Outputs)	خروجی ها، از PLC به پروسه تحت کنترل.
F	(Flags)	فلگ ها، حافظه ای جهت نگهداری مقادیر میانی حاصل از عملیات باینری.
D	(Data)	دیتا، حافظه ای جهت نگهداری مقادیر میانی حاصل از عملیات دیجیتالی.
T	(Timers)	تایمرها، حافظه ای جهت تخصیص زمان سنج ها.
C	(Counters)	شمارنده ها، حافظه ای جهت تخصیص شمارشگرها.
K	(Constants)	ثابت ها.

۳-۵ - دستورالعمل های زبان S5^۱

دستورات زبان S5 را در حالت کلی می توان به سه دسته زیر تقسیم نمود:

۱ - دستورات زبان S5 در ضمیمه 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 سیستم (System)	دستورالعمل‌های تکمیلی (Supplementary)	دستورالعمل‌های اصلی (Basic)	
تنها در بلوک‌های FB	تنها در بلوک‌های FB	در بلوک‌های FB ، SB ، PB ، OB	کاربرد
STL	STL	STL ، LAD ، CSF	روش‌نمایش
تنها افراد با تجربه می‌توانند از این دستورات استفاده نمایند.	-	-	قابلیت‌های خاص

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 20_{16} یا 10100_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 00010100 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 $AN \dots (AND NOT)$ در روش STL و از نماد \neg در روش LAD و از $d-$ 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)^۱

این اتصال (کنتاکت) هنگامی فعال می‌شود که مثلاً دکمه پوش باتن (Push Button) فشرده یا کلیدی روشن شود. در این حالت در ورودی PLC مقدار "۱" ظاهر می‌گردد و در حالتی که دکمه 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)^۲

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

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

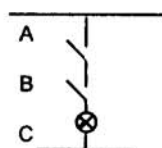
جدول ۳-۴: نحوه نمایش کنتاکت‌های NO و NC در روشهای مختلف برنامه‌نویسی

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 لازمه روشن شدن لامپ 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 3 PB برنامه 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

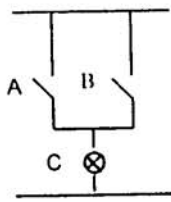
```

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

```

حال فرض کنید که I 1.0 سنسور ورودی نشان دهنده فعال بودن موتور الکتریکی ۱ و I 7.4 سنسور نشان دهنده فعال بودن موتور الکتریکی ۲ باشد. روشن شدن خروجی Q 2.5 به منزله 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

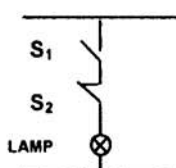
```

SEGMENT 1          0000
!
! I 2.3
+---] [-----+-----+---( )-!
!
! 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 آپراتور را از بروز خطرات احتمالی آگاه می‌سازد.

خوانندگان توجه دارند که نسبت دادن ورودی‌ها و خروجی‌ها به شرایط ورودی و فرمانهای صادره 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 اولیه 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، چراکه 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.

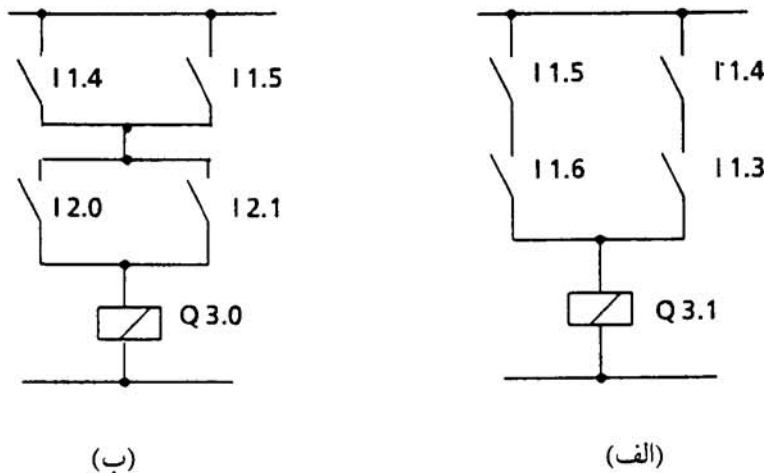
توضیح		نماد مدار
خروجی Q 7.5 مقدار "۱" خواهد داشت در صورتی که ورودی‌های I 2.3 و I 0.4 "۰" بوده و I 3.7 مقدار "۱" داشته باشد. این خروجی برابر "۰" خواهد بود در صورتی که حداقل یکی از شرایط مذکور برقرار نباشد.		
STL	CSF	LAD
<pre> AN I 2.3 A I 3.7 AN I 0.4 = Q 7.5 </pre>		

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

توضیح		نماد مداری
<p>خروجی Q 4.1 برابر "۱" خواهد بود در صورتی که حداقل یکی از شرایط زیر برقرار باشد:</p> <p>I 3.6 = "۱" یا I 4.7 = "۰" یا I 0.0 = "۱"</p> <p>و مقدار این خروجی برابر "۰" خواهد بود اگر I 0.0 = "۰" و I 4.7 = "۱" و I 3.6 = "۰" باشد.</p>		
STL	CSF	LAD
<pre> O I 3.6 ON I 4.7 O I 0.0 = Q 4.1 </pre>		

۳-۹- کاربرد پرانتزها در برنامه‌نویسی به روش STL

نمایش مدارهای زیر را در نظر بگیرید.



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:

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

\uparrow AND \uparrow OR
 (AND قبل از OR)

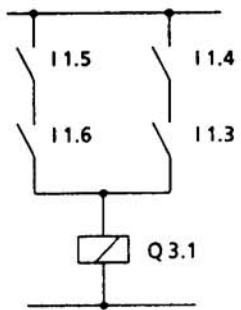
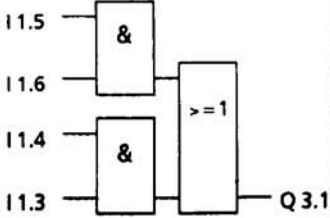
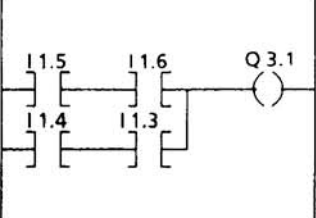
$$\text{ب ۳-۶: } (I\ 1.4 + I\ 1.5) \cdot (I\ 2.0 + I\ 2.1) = Q\ 3.0$$

\uparrow OR \uparrow AND
 (AND قبل از OR)

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، برنامه 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.

توضیح		نماد مداری
<p>خروجی 3.1 Q برابر "۱" خواهد بود هرگاه حاصل حداقل یکی از دو تابع AND برابر "۱" شود. این خروجی برابر "۰" است در صورتی که حاصل هیچ یک از دو تابع AND "۱" نباشد.</p>		
STL	CSF	LAD
<pre> A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q 3.1 </pre>		

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.

توضیح		نماد مداری
<p>خروجی Q 3.0 برابر "۱" خواهد بود اگر حاصل هر دو تابع OR برابر "۱" شود.</p> <p>این خروجی برابر "۰" خواهد بود در صورتی که حاصل حداقل یکی از دو تابع OR "۰" شود.</p>		
STL	CSF	LAD
<pre> A(O I 1.4 O I 1.5) A(O I 2.0 O I 2.1) = Q 3.0 </pre>		

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.

توضیح		نماد مداری
<p>خروجی ۲.۱ Q برابر "۱" خواهد بود هرگاه حداقل یکی از شرایط زیر برقرار باشد:</p> <p>* ورودی ۶.۰ I برابر "۱" باشد.</p> <p>* ورودی ۶.۱ I و حداقل یکی از دو ورودی ۶.۲ I و ۶.۳ I برابر "۱" باشند.</p> <p>مقدار این خروجی برابر "۰" است اگر حاصل هیچ یک از توابع AND برابر "۱" نباشد.</p>		
STL	CSF	LAD
<pre> O I 6.0 O I 6.1 A(O I 6.2 O I 6.3) = Q 2.1 </pre>		

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 حافظه PLC است که آن را می‌توان معادل رله 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 دفترچه یادداشت 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 عمده 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 رله اول، رله دوم را فعال نموده، سپس رله دوم، رله سوم را و به همین ترتیب تا جایی که رله 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 نتیجه 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 پراختها را حذف نمود. البته با این عمل ممکن است در برخی موارد برنامه مورد نظر طولانی‌تر گردد.

هنگامی که در برنامه‌ای حاصل یک دسته از اعمال منطقی باید با حاصل دسته 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 : برنامه 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	}	حاصل دسته اول از اعمال منطقی در F 6.0 قرار می‌گیرد.
: O	I	1.5		
: =	F	6.0		
: O	I	2.0	}	حاصل دسته دوم از اعمال منطقی در F 6.1 قرار می‌گیرد.
: O	I	2.1		
: =	F	6.1		
: A	F	6.0	}	حاصل ترکیب عطفی دو فلگ F 6.0 و F 6.1 در خروجی Q 3.0 قرار داده می‌شود.
: A	F	6.1		
: =	Q	3.0		

۳-۱۱- بیت RLO^۱

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.

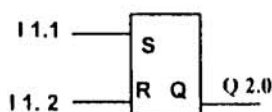
۳-۱۲- ست و ری‌ست در فلگ‌ها و خروجی‌ها

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:

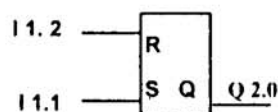
۱- فلیپ فلاپ SR

۲- فلیپ فلاپ RS

در شکل ۳-۶ بلوک این دو نوع فلیپ فلاپ در روش برنامه نویسی CSF را می بینیم.



فلیپ فلاپ SR



فلیپ فلاپ RS

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

تفاوت این دو نوع فلیپ فلاپ تنها در ارجحیت ورودی های ست و ری ست می باشد که در این مورد به طور مفصل در همین بخش سخن خواهیم گفت. صورتهای مختلف نمایش فلیپ فلاپ ها به روش STL در ادامه نشان داده شده است.

فلیپ فلاپ 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

```

فلیپ فلاپ 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 کننده 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. برنامه 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		

```

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

```

```

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

```

- تفاوت بین این دو برنامه در چیست؟

(الف)

(۲)

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

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

در برنامه (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 پروژه تحت کنترل گردد. بسیاری از مشکلات موجود در خطوط تولید و فرآیندهای صنعتی به دلیل کم توجهی یا بی‌توجهی به برخی نکات جزئی مانند تفاوت مذکور می‌باشد.

۳-۱۳ - دستور NOP 0

یک فلیپ‌فلاپ SR را در نظر گرفته، برنامه‌ای جهت ست و ری ست 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

```

حال در صورتی که بخواهیم از خروجی، هیچ استفاده‌ای نکنیم یا به عبارت دیگر سطرهای پنجم و ششم از برنامه 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 برنامه 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

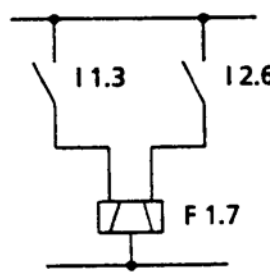
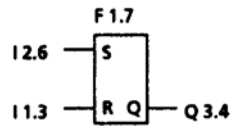
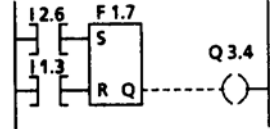
```

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

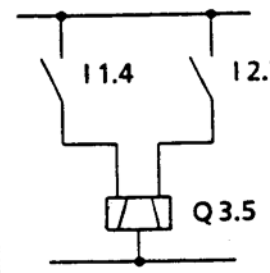
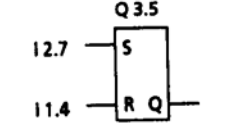
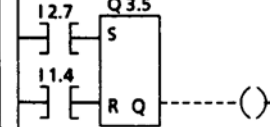
```

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 برنامه 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 برنامه نوشته شده برای یک فلیپ فلاپ می‌پردازیم.

نماد مداری		توضیح
		وجود یک لبه بالا رونده در ورودی I 2.6، I 1.7 F راست می‌کند. در این حالت در صورتی که سیگنال ورودی I 2.6 به "۰" تغییر وضعیت دهد مقدار I 1.7 F بدون تغییر باقی خواهد ماند. در صورت وجود لبه بالا رونده در ورودی I 1.3، I 1.7 F ریست خواهد شد و در صورت تغییر وضعیت I 1.3 به "۰" مقدار فلگ ثابت خواهد ماند. در هر دو حالت فوق مقدار موجود در فلگ I 1.7 F به خروجی Q 3.4 نسبت داده می‌شود.
STL	CSF	LAD
<pre> A I 2.6 S F 1.7 A I 1.3 R F 1.7 A F 1.7 = Q 3.4 </pre>		

مثال ۳-۱۰: در این مثال کاربرد دستور NOP 0 در برنامه نویسی یک فلیپ فلاپ بررسی می‌گردد.

نماد مداری		توضیح
		در صورت وجود لبه بالا رونده در ورودی I 2.7 فلیپ فلاپ Q 3.5 ست خواهد شد و در صورت تغییر وضعیت این ورودی به "۰" خروجی Q 3.5 ثابت می‌ماند. لبه بالا رونده در ورودی I 1.4 فلیپ فلاپ را ریست می‌نماید و تغییر وضعیت این ورودی به "۰" در مقدار فلیپ فلاپ تغییری ایجاد نمی‌کند. در اینجا از خروجی هیچ‌گونه استفاده‌ای نمی‌شود.
STL	CSF	LAD
<pre> A I 2.7 S Q 3.5 A I 1.4 R Q 3.5 NOP 0 </pre>		

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 برنامه 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 نمایید.

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 ۰) 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      !      !
+---] [---+! R      Q! +-----+ Q 9.5
!          +-----+      ( ) -!
!
!
SEGMENT 2          0007
!          Q 9.7
! I 16.2      +-----+
+---] [---+! S      !
!          !      !
! I 16.3      !      !
+---] [---+! R      Q! +-----+ Q 9.6
!          +-----+      ( ) -!
!
!

```



```

SEGMENT 3 000E
!
! I 16.6 Q 10.0
+---] [---+!R
!
! I 16.7 F 10.2
+---]/[---+!S Q!+---+---( )-!
!
!
SEGMENT 4 0015
!
! I 17.0 Q 10.1
+---] [---+!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

[illegible]

[illegible]

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.

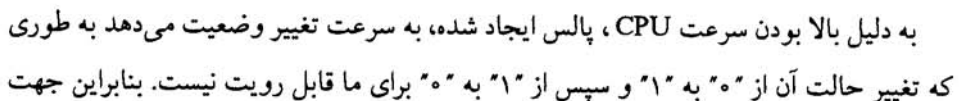
مثال ۳-۱۲: در برخی از فرایندهای کنترلی گاه لازم است دستگاهها و سیستم‌های مشابه به صورت همزمان با برقراری شرایطی خاص مانند شرایط ایمنی و ... فعال شوند. در این گونه برنامه‌ها برای جلوگیری از تکرار خطوط برنامه می‌توان این شرایط را در یک کانکتور قرار داد و سپس هر سیستمی که برای فعال شدن به شرایط مذکور نیاز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

```
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 خلاصه نمودن برنامه بسیار مؤثر است.



PB 28

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

```

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 برنامه تشخیص لبه پالس را با استفاده از فلیپ فلاپ SR بازنویسی می‌کنیم.

توضیح			نماد مداری
<p>در صورت وجود لبه بالا رونده در I 1.7 شرایط لازم برای "۱" شدن ترکیب عطفی I 1.7 و نقیض F 4.0 مهیا می‌شود و F 2.0 و F 4.0 ست خواهند شد.</p> <p>در اجرای سیکل بعدی حاصل ترکیب عطفی I 1.7 و نقیض F 4.0 برابر "۰" می‌باشد زیرا در سیکل قبلی F 4.0 ست شده بود.</p> <p>در این حالت F 2.0 ری ست می‌شود.</p> <p>بنابراین F 2.0 تنها در خلال اجرای یک مرتبه برنامه "۱" خواهد شد.</p> <p>در صورتی که "۰" = I 1.7 شود، F 4.0 ری ست می‌شود.</p>			
STL	CSF	LAD	
<pre> 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 </pre>			

روش دیگری جهت برنامه‌نویسی تشخیص دهنده لبه پالس وجود دارد. در روش مذکور از فلیپ فلاپ استفاده نمی‌شود. این برنامه کاملاً کاربردی بوده، در بسیاری از پروژه‌ها مورد استفاده

قرار می‌گیرد. برنامه 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 لبه تیز خواهیم داشت.

مثال ۳-۱۳: فرض کنید در کنترل یک پروسه صنعتی یک پوش باتن (Push Button) وجود دارد.

می‌خواهیم برنامه‌ای جهت کنترل خروجی مرتبط با این پوش باتن بنویسیم به گونه‌ای که:

اگر پوش باتن 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 پوش باتن 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 پوش باتن به ذکر یک مثال در زندگی روزمره 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 ری ست شدن فلیپ‌فلاپ‌ها الزامی است.
 برنامه نوشته شده به صورت زیر می‌باشد: (I 0.0 در تمامی موارد نشان دهنده 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      !         !
+---] [---+---] [---+---! R      Q 2.0
!                   +-----+   +---( )-!
!
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 در نظر 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 "0" = F 8.1 می‌باشد بنابراین حاصل ترکیب عطفی نقیض آن و ورودی I 0.0 که در ترمینال S فلیپ‌فلاپ قرار گرفته‌اند F 8.0 را ست می‌کند.)

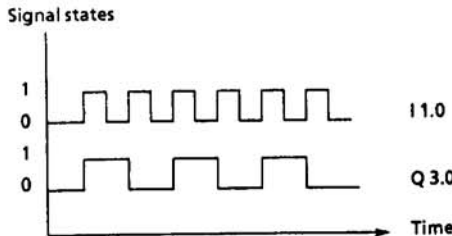
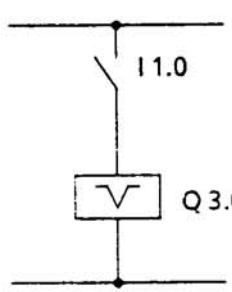
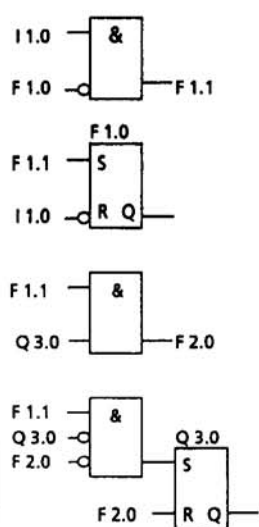
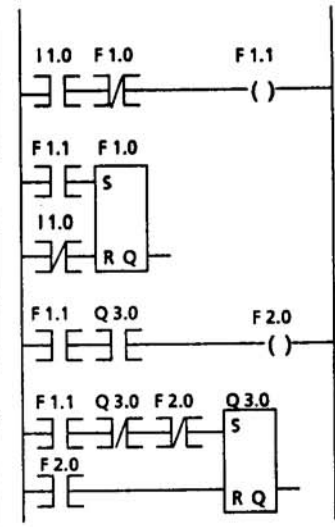
با رها کردن پوش باتن ورودی، I 0.0 "0" خواهد شد و ترمینال S فلیپ‌فلاپ موجود در SEGMENT 2 فعال می‌شود. (زیرا ترمینال S این فلیپ‌فلاپ حاصل ترکیب عطفی F 8.0 و نقیض I 0.0 می‌باشد و از آنجایی که F 8.0 در مرحله قبل فعال شده بود حاصل ترکیب عطفی نقیض I 0.0 و F 8.1 برابر "1" خواهد بود.) در این مرحله این فلیپ‌فلاپ 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 ادامه 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 استفاده برنامه‌نویس از خروجی فلیپ‌فلاپ موجود در SEGMENT 2 می‌باشد.

مثال ۳-۱۴: در این مثال به برنامه‌نویسی "Binary Scaler" یا برنامه نصف‌کننده 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، دوره تناوب ورودی $I. 1.0$ در خروجی $Q. 3.0$ دو برابر شده است. بررسی عملکرد این برنامه را به عهده خواننده واگذار می‌کنیم.

نمودار زمانی	نماد مداری	
<p>Signal states</p>  <p>Time</p>		
STL	CSF	LAD
<pre>A I 1.0 AN F 1.0 = F 1.1 *** A F 1.1 S F 1.0 AN I 1.0 R F 1.0 NOP 0 *** A F 1.1 A Q 3.0 = F 2.0 *** A F 1.1 AN Q 3.0 AN F 2.0 S Q 3.0 A F 2.0 R Q 3.0 NOP 0</pre>		

۳-۱۶- دستور پرش غیر شرطی (JU)^۱

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

بنابراین، ارزش بیت RLO به نتیجه عملیات منطقی سطر بستگی دارد. انجام برخی از دستورات به RLO وابسته (RLO Dependent) و برخی دیگر غیر وابسته اند (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 غیر شرطی 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)^۲

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 18 PB اجرا و در صورت غیرفعال نمودن همان کلید بلوک دیگری مثلاً 19 PB اجرا شود.

با اندکی دقت در متن مثال در می‌یابیم که این برنامه باید در بلوک 1 OB نوشته شود زیرا همان‌گونه که اشاره شد ساختار کلی سیستم در این بلوک تعیین می‌گردد. بنا به تعریف دو دستور JU و JC برای پرش باید از دستور JC به همراه شرط فعال بودن یا غیرفعال بودن کلید مربوطه استفاده نماییم.

حال فرض کنید کلید مورد نظر در مثال، ورودی 0.0 I باشد برای مشاهده عملکرد این برنامه در دو بلوک 18 PB و 19 PB دو برنامه متفاوت می‌نویسیم. برنامه‌های نوشته شده در بلوک‌های 1 OB، 18 PB و 19 PB 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 0.0 I فعال یعنی "۱" باشد بیت RLO نیز برابر "۱" است و در نتیجه، دستور پرش شرطی به 18 PB انجام می‌شود. پس از اینکه پردازنده به دستور BE در 18 PB رسید به سطر بعدی در 1 OB یعنی سطر بعدی خطی از برنامه که از آنجا عمل پرش صورت گرفته مراجعه می‌کند. چون 0.0 I فعال است پس نقیض آن و بیت RLO برابر "۰" می‌باشند بنابراین دستور پرش شرطی

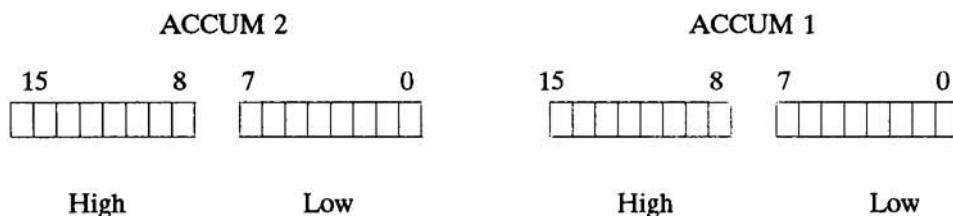
به 19 PB انجام نخواهد شد. سپس پردازنده به دستور BE در بلوک 1 OB رسیده، برنامه پایان یافته تلقی می‌شود. در اینجا مجدداً پردازنده به سطر اول بلوک 1 OB مراجعه نموده و در صورت فعال بودن 0.0 I همین روند مجدداً تکرار می‌شود. اما در صورتی که 0.0 I غیرفعال یعنی برابر "۰" باشد بیت RLO نیز "۰" است و دستور پرش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

با اندکی دقت در این برنامه ملاحظه می‌کنیم که این کلید یعنی ورودی 0.0 I می‌تواند نقش کلید Auto/Manual را در فرآیندهای صنعتی ایفا نماید. به عنوان مثال 18 PB شامل دستوراتی باشد که سیستم را در حالت Auto کنترل و 19 PB 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 مبادله اعداد احتیاج به یک حافظه واسطه می‌باشد. بنابراین قسمتی از حافظه که به آن آکومولاتور یا انبارک (انباره) 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^۱

واژه 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 1 و ACCUM 2) استفاده شده است.

حال دستور L IW 4 را در نظر بگیرید. این دستور بدان معنی است که ۱۶ بیت موجود در کلمه ورودی شماره 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

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^۱

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.

$$T \quad QW \quad 8 \quad , \quad T \quad FW \quad 52$$

در اجرای دستور T QW 8، محتویات ACCUM 1 به کلمه 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^۲، 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 نحوه 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.

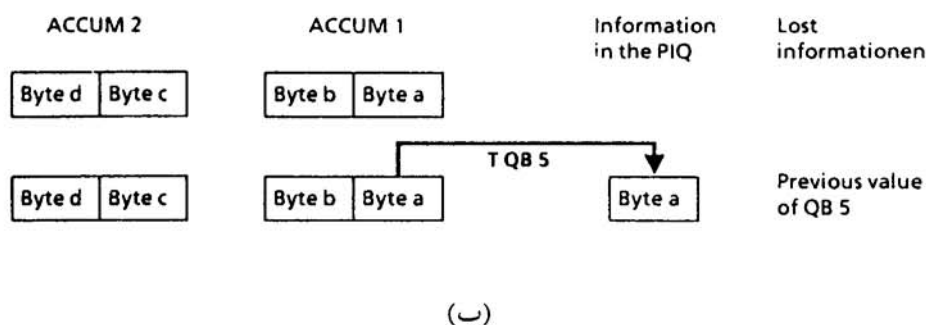
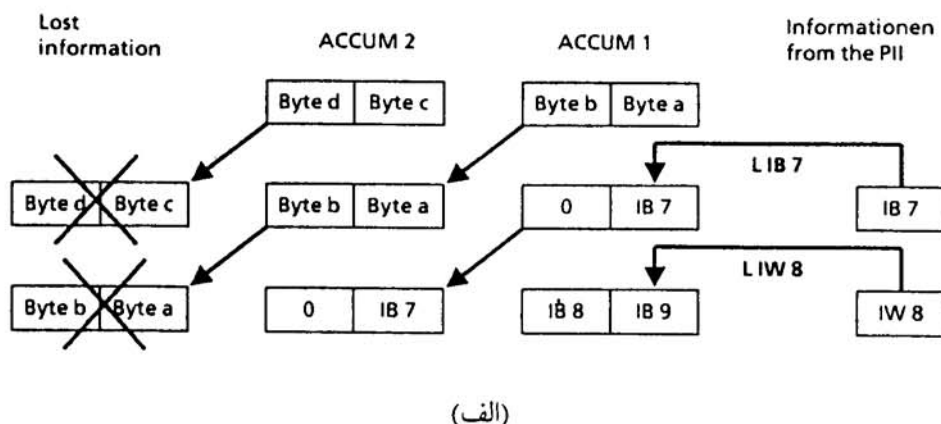
1 - Transfer

2 - FW: Flag Word

3 - QW: Output Word

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

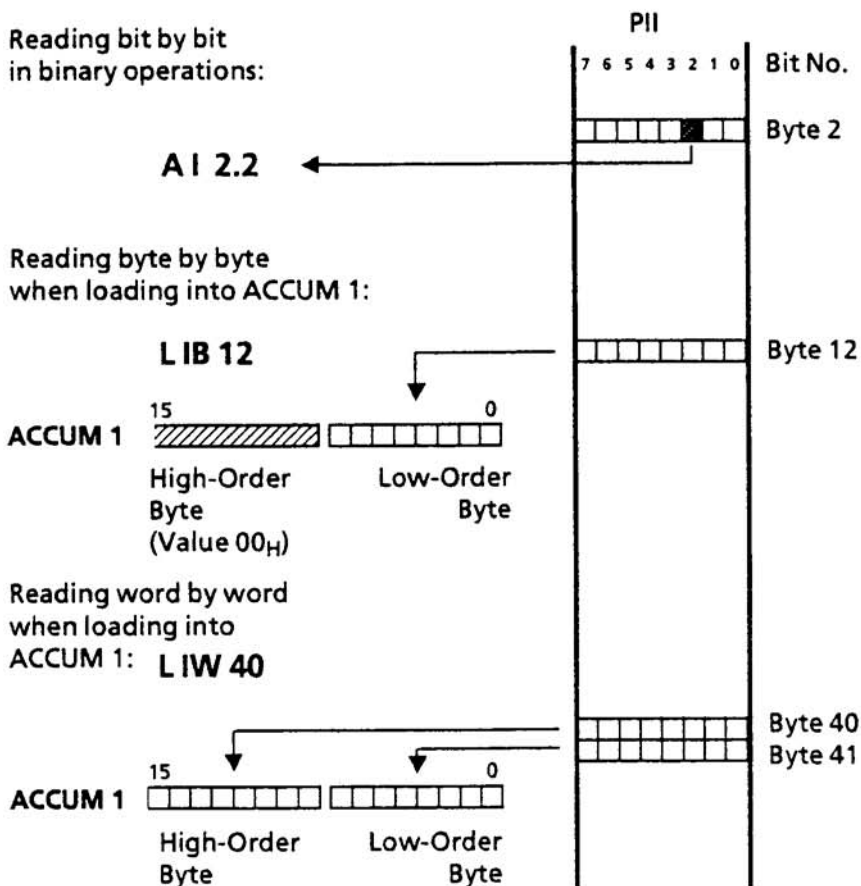
همان‌گونه که در شکل ۸-۳ (الف) ملاحظه می‌کنید دو دستور L IB 7 و L IB 8 اجرا شده است، و طی اجرای این دو دستور محتویات ابتدایی ACCUM 2 دور ریخته می‌شوند. ضمناً در دستور بارگذاری اطلاعات از PII به انبارک، اطلاعات موجود در PII ثابت باقی مانده، تنها رونوشتی از آنها به انبارک بارگذاری می‌شود.



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

در شکل ۸-۳ (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 نحوه بارگذاری از PII و انتقال به PIO به صورت بیتی، بایتی و کلمه‌ای نشان داده شده است.



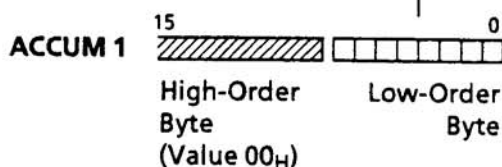
شکل ۳-۹: نحوه بارگذاری اطلاعات به صورت بیتی، بایتی و کلمه‌ای از PII

Writing bit by bit
in binary operations:

=Q 4.6

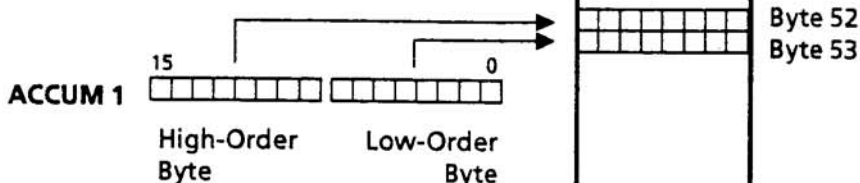
Writing byte by byte
to transfer from ACCUM 1:

T QB 36



Writing word by word
to transfer from ACCUM 1:

T QW 52



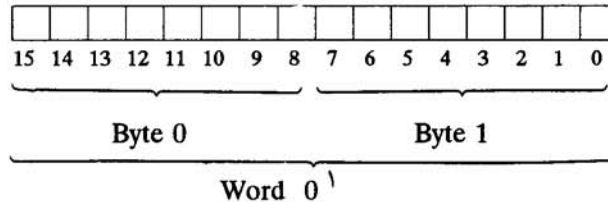
شکل ۳-۱۰: نحوه انتقال اطلاعات به صورت بیتی، بایتی و کلمه‌ای به PIO

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:

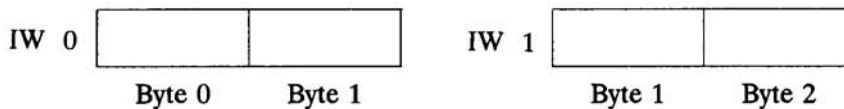
مثال ۳-۱۶: دستور 0 IW L را در نظر بگیرید. چگونگی بارگذاری بایت‌های این کلمه یعنی کلمه ورودی صفر را بیان کنید.

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 نحوه بارگذاری اطلاعات به داخل انبارک ۱ نشان داده شده است.



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 شماره فرد همیشه در هنگام اجرای این گونه دستورات هم‌پوشانی^۲ 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 PLC ها شماره گذاری بایت ها برعکس روش مذکور در PLC های SIEMENS انجام می‌شود.

می‌بریم تا از بروز چنین مواردی جلوگیری شود. با به کارگیری این قاعده در استفاده از کلمات با شماره زوج یا فرد همواره شماره کلمه مثلاً صفر در 0 IW با شماره 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 ۳۱ و ۳۲ بارگذاری می‌شوند.
		L IW 57	بایت‌های ۵۷ و ۵۸ بارگذاری می‌شوند.

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

۳-۲۰- موارد استفاده 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 برنامه زیر که به روش STL نوشته شده است حاصل جمع دو کلمه ورودی ۰ و ۲ به کلمه 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 برنامه 20 PB 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 کلمه ورودی قرار داد.

$$\begin{aligned}
 1456_{(10)} &= 00000101 \ 10110000_{(2)} + \rightarrow IW \ 0 \\
 7659_{(10)} &= 00011101 \ 11101011_{(2)} \rightarrow IW \ 2 \\
 \hline
 &= 00100011 \ 10011011_{(2)} \rightarrow QW \ 4 \\
 &= 0 \times 2^{15} + 0 \times 2^{14} + 1 \times 2^{13} + 0 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 \\
 &\quad + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 9115_{(10)}
 \end{aligned}$$

بنابراین در خروجی (QW 4) عدد دودویی $(0010001110011011)_2$ را خواهیم داشت که پس از تبدیل مبنا عدد $9115_{(10)}$ 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 برنامه زیر این مطلب نشان داده شده است.

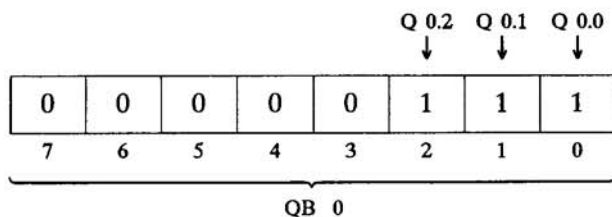
PB 10

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

در صورتی که دو عدد $2_{(10)}$ و $5_{(10)}$ را در بایتهای ورودی ۴ و ۱۲ داشته باشیم حاصل جمع این دو عدد که همان $7_{(10)}$ می‌باشد به بایت خروجی صفر انتقال می‌یابد.

$$\begin{aligned}
 2_{(10)} &= 00000010 + \rightarrow IB \ 4 \\
 5_{(10)} &= 00000101 \rightarrow IB \ 12 \\
 \hline
 &= 00000111 \rightarrow QB \ 0 \\
 &= 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7_{(10)}
 \end{aligned}$$

عدد $7_{(10)}$ در QB 0 به صورت زیر انتقال می‌یابد.



بنابراین داریم:

$$Q\ 0.0 = "۱" \quad \text{و} \quad Q\ 0.1 = "۱" \quad \text{و} \quad Q\ 0.2 = "۱"$$

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 نماییم.

نتیجه‌ای که از این مثال به دست می‌آید این است که:

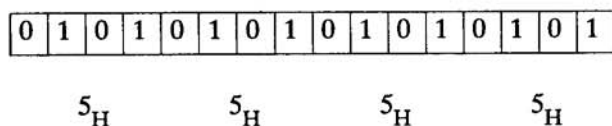
با انجام عملیات محاسباتی بر روی ورودی‌ها می‌توان به خروجی‌ها فرمان داد.

همان‌گونه که در فصل اول بیان شد به دلیل مشکلات موجود در تبدیل اعداد باینری به دهدهی و بالعکس، از مبنای اول بالاتر از مبنای دو یعنی مبنای ۸ و ۱۶ استفاده می‌کنیم. در صورتی که از مبنای ۱۶ استفاده کنیم کلمه ۱۶ بیتی را به ۴ گروه ۴ بیتی تقسیم می‌نماییم. 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 ... KD استفاده می‌کنیم در حالی که جهت بارگذاری انبارک با عدد ثابتی در مبنای ۱۶ از دستور ... KH استفاده می‌نماییم. (به جای نقطه چین 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.



1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 A_H A_H A_H A_H

بنابراین می‌توان یکی از دو عدد 5555_H و یا $AAAA_H$ را در 2 QW 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. نمونه 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 2 IB از محتویات 1 IB 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 به دست آورید. ابتدا دو عدد داده شده را به مبنای ۲ تبدیل می‌کنیم.

$$\begin{array}{rcl}
 ۵۴_{(۱۰)} = ۰۰۱۱۰۱۱۰_{(۲)} & \rightarrow & \text{IB 1} \\
 ۳۶_{(۱۰)} = ۰۰۱۰۰۱۰۰_{(۲)} & \rightarrow & \text{IB 2} \\
 \hline
 & \rightarrow & \text{QB 3} \\
 & = & ۰۰۰۱۰۰۱۰_{(۲)} \\
 & = & ۱ \times ۲^۳ + ۱ \times ۲^۱ \\
 & = & ۱۶ + ۲ = ۱۸
 \end{array}$$

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 در نظر می‌گیریم:

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

$$\begin{array}{rcl}
 \text{IB 1 : } ۰۰۰۰۰۰۰۰ & - & ۰۰۰۰۰۰۰۰ \\
 \text{IB 2 : } ۰۰۰۰۰۰۰۱ & \xrightarrow{\text{جمع با مکمل ۲}} & \begin{array}{r} ۱۱۱۱۱۱۰ \\ ۱۱۱۱۱۱۰ \\ ۱ \\ \hline ۱۱۱۱۱۱۱ \end{array} \\
 & ? & \leftarrow \text{مکمل ۱} \\
 & & \rightarrow \text{QB 3}
 \end{array}$$

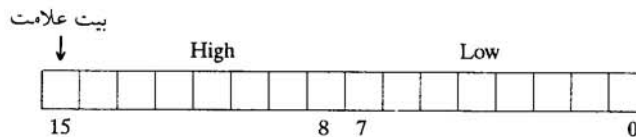
بنابراین عدد $۱۱۱۱۱۱۱_{(۲)}$ به بایت خروجی ۳ منتقل می‌گردد. توجه داشته باشید که با انجام این عمل توانسته‌ایم تمام بیت‌های بایت خروجی ۳ را فعال نمائیم. پس تفریق دو عدد نیز همانند جمع آنها می‌تواند به عنوان روشی جهت فعال یا غیرفعال نمودن بیت‌های خروجی مورد استفاده قرار گیرد.

مثال عملی متناظر با تفریق عدد ۱ از ۰ را می‌توان در شمارنده‌های (کانترهای) مکانیکی ضبط صوت یافت. (فرض کنید که این شمارنده ۴ رقمی است یعنی از عدد ۰۰۰۰ شروع شده، تا

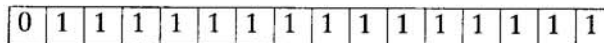
9999 ادامه می‌یابد و پس از 9999 مجدداً به 0000 باز می‌گردد و ... اگر شمارنده مکانیکی ضبط صوت بر روی 0000 قرار گرفته باشد و شما توسط کم ارزش‌ترین رقم (یعنی رقم سمت راست)، شمارنده را به اندازه یک واحد در خلاف جهت افزایش شماره‌ها بچرخانید، با این عمل در حقیقت از عدد 0000، یک واحد کم کرده‌اید. عددی که بر 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 ۱۱۱۱۱۱۱ به دست 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 $7FFF_H$ و $FFFF_H$ به ترتیب کوچکترین عدد منفی و بزرگترین عدد مثبت در مبنای ۱۶ می‌باشند.

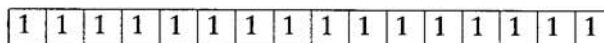


7 (بزرگترین عدد مثبت مبنای ۱۶)

F

F

F



F (کوچکترین عدد منفی مبنای ۱۶)

F

F

F

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 به دست 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 استفاده مبنای ۱۶، از فرم ...KF استفاده می‌کنیم. در این حالت می‌توان از اعداد مثبت، منفی و صفر استفاده نمود. محدوده تغییرات اعداد دهمی در فاصله $[-32768 \text{ و } +32768]$ می‌باشد.

جهت بارگذاری انبارک می‌توان از اعداد BCD نیز استفاده نمود. برای بارگذاری اعداد به فرم 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 محدوده این اعداد در فاصله $[7999, 9999-]$ می‌باشد.

با استفاده از دستورات L و T و بارگذاری اعداد مختلف در مبناهای متفاوت می‌توان صحت ارتباط بین ورودی‌ها و خروجی‌ها^۱ را در یک سیستم بررسی نمود. بسیاری از خوانندگانی که محیط‌های صنعتی را تجربه نموده‌اند به خوبی آگاهند که قبل از راه‌اندازی هر فرآیند، بایستی 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 تلفن 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 برنامه 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.

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

فرض کنید یک دسته 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 دسته ۱۶ تایی، در مورد دسته دوم نیز همین عمل به همراه همین برنامه اجرا می‌شود تا اینکه صحت ارتباط تمامی قسمت‌های مدار بررسی گردد.

۳-۲۱- مقایسه‌کننده‌ها (Comparators)

یکی دیگر از موارد استفاده انبارک در مقایسه‌کننده است. یک مقایسه‌کننده مقدار دو ورودی را دریافت نموده، با توجه به نوع و نتیجه مقایسه، خروجی مقایسه‌کننده را فعال و یا غیرفعال می‌کند. اعدادی که می‌توان در مقایسه‌کننده استفاده نمود به صورت بایتی بوده، شامل IBها، QBها و FYها می‌باشند. علاوه بر موارد مذکور اعداد ثابت نیز می‌توانند با فرمت $^{1}KF \dots$ برای مقایسه در برنامه وارد شوند.

در جدول ۳-۵ انواع مقایسه، نمادهای ریاضی و نمادهای برنامه‌نویسی^۲ 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 مقایسه 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 برنامه مقایسه 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 100 KF L در سطر دوم محتویات ACCUM 1 یعنی IB 5 به ACCUM 2 انتقال یافته، عدد دهدهی ۱۰۰ در ACCUM 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 1.4 Q نیز فعال می‌شود. در غیر این صورت خروجی مذکور "۰" خواهد شد.

مثال ۳-۲۲: در این مثال حالت دیگری از مقایسه 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 فعال می‌شود.
 مثال ۳-۲۳: برای درک بیشتر حالات مختلف مقایسه، برنامه کاملی شامل تمام حالات مقایسه نوشته شده است. با تکمیل جدول ۳-۶ در هر حالت چگونگی عملکرد را توضیح دهید. بلوک برنامه 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!
!      !! = !!
! IB 17  --!V2  Q!-----+---( )---!
!      +-----+
!
!
    
```

ادامہ یلوک 1 PB:

SEGMENT 2 0005

```

      +-----+
IB 16 --!V1 F!
      !>< !
      +-----+
IB 17 --!V2 Q!--+-----+Q 8.1( )--!
      +-----+

```

SEGMENT 3 000A

```

!          +-----+
!IB 16    --!V1   F!
!          !>=   !
!          +-----+
!IB 17    --!V2   Q!--+-----+---(      )--!
!          +-----+

```

SEGMENT 4 000F

```

!          +-----+
!IB 16    --!V1   F!
!          !>     !
!          +-----+
!IB 17    --!V2   Q!--+-----+-----Q 8.3
!          +-----+

```

SEGMENT 5 0014

```

|      +-----+
| IB 16 --!V1  F!
|          !<=
| IB 17 --!V2  Q!--+-----+---(    )--!
|          +-----+

```

SEGMENT 6 0019

```

|      +-----+
| IB 16 --!V1  F!  
|      !<    !  
| IB 17 --!V2  Q!--+-----+---Q 8.5  
|          +-----+---( )--!
|      +-----+

```

: 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 8.0 ! = F
10	10						
9	10						
10	9						

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

موارد استفاده مقایسه کننده ها در کنترل پروسه های صنعتی بسیار زیاد است. مثلاً می توان به یکی از ورودی ها 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 غیر فعال نمود. در مثال زیر از این مطلب استفاده شده است. مثال ۳-۲۴: قصد داریم در یک فرآیند صنعتی درجه حرارت روغن خنک کننده سیستم را کنترل نمائیم. برنامه ای بنویسید که اگر درجه حرارت روغن به کمتر از 20°C برسد گرم کننده روغن (Heater) موجود در مخزن روغن روشن و در صورتی که درجه حرارت روغن به بیش از 30°C برسد Heater خاموش شود.

برای درک بهتر این مثال، ابتدا برنامه را به روش CSF می نویسیم. همان گونه که حدس زده اید در این برنامه باید از دو مقایسه کننده همراه با دو حالت مقایسه جداگانه و در دو Segment استفاده کنیم.

در برنامه PB 29 فرض بر این است که درجه حرارت روغن توسط بیت ورودی ۵ (IB 5) به مقایسه کننده وارد می شود. در قسمت اول، این عدد با عدد ۲۰ مقایسه شده و در صورتی که درجه حرارت روغن از 20°C کمتر باشد خروجی مقایسه کننده اول فعال می شود و $Q 0.7$ را ست می کند بنابراین خواهیم داشت: $Q 0.7 = 1$. این خروجی می تواند فرمان روشن شدن گرم کننده

الکتریکی (Heater) باشد. در قسمت دوم، درجه حرارت روغن با عدد ۳۰ مقایسه می‌شود و در صورتی که درجه حرارت روغن از 30°C تجاوز نماید خروجی این مقایسه‌کننده فعال می‌شود. یعنی $Q = 0.7$ راریست 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 PLC‌های مختلف تعداد شمارنده‌هایی که می‌توان استفاده نمود متفاوت است. برای استفاده از هر شمارنده باید شماره 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
!
! I 0.1      C 4
! +---] [---+-!CU      |
! +---] [---+-!CD      |
! I 0.2
! +---] [---+-!S      |
! I 0.0
! +---] [---+-!CV BI!- QW 2
! KC 050  --!  DE!- FW 10
!
! I 0.3
! +---] [---+-!R  Q!-+-----+---( )-!
!                               Q 0.0
!                               :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 لبه پالس (لبه بالا رونده یا پائین رونده) فعال می‌شوند و با هر بار فعال شدن شمارنده، بسته به نوع شمارشگر، عدد شمارنده افزایش یا کاهش می‌یابد.

ورودی S : (Set) با فعال شدن ورودی S مقدار اولیه 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 قاعده 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 $S C... L$ 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 0.0 Q) برابر "۱" بوده، در صورت اتمام عملیات شمارش به مقدار "۰" خواهد بود.

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 برنامه‌ی 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 0.0 I، شمارنده را ست نمود.

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

در صورت عدم نیاز به دستوری ست 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 برنامه 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 شمارنده ۱ به اندازه یک واحد افزایش می‌یابد و به محض اینکه ورودی I 4.2 فعال می‌شود شمارنده ریست خواهد شد. دستور 1 C A 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.

نمودار زمانی		نماد مداری
STL	CSF	LAD
<pre> 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 </pre>		

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 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 شمارنده ۱ به اندازه 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
<pre> 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 </pre>		

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 برنامه مورد نظر 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 برنامه 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 برنامه نوشته شده به روش STL دو بار از دستور NOP 0 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 6.0 F برابر "۱" می‌گردد. توجه داشته باشید که فعال شدن 6.0 F در مدت بسیار کوتاهی می‌باشد و پس از تغییر محتویات 4 QW 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 1.6 I و 6.0 F قرار داده شده است. این بدان معنی است که به مجرد اینکه بیت 6.0 F در بخش دوم برنامه فعال گردد شمارنده ۶ ریست شده، شمارش مجدداً از صفر آغاز می‌شود. در ورودی R علاوه بر 6.0 F، 1.6 I نیز جهت ریست نمودن شمارنده در نظر گرفته شده است.

توجه داشته باشید که در هنگام اجرای برنامه شمارنده می‌توان روند شمارش را در PG مشاهده نمود. در هنگام اجرای این برنامه ملاحظه می‌شود که مقدار شمارش شده توسط شمارنده یعنی محتویات 4 QW 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 صفحه نمایش پروگرامر (PG) دیده می‌شود^۱.

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

C ...	ACT : ...	PAR : ...
شماره شمارنده مورد استفاده در برنامه		
<p>ACT (Actual) : این عدد نشان دهنده 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 خروجی ها فرستاده می شود.</p> <p>PAR (Parameter) : این مقدار، عدد موجود در ورودی CV را نشان می دهد. مقدار این عدد ثابت می باشد. به عنوان مثال در برنامه نوشته شده در پائین صفحه نمایش PG مقادیر زیر را خواهیم داشت:</p>		
C 6	ACT :	PAR : 30
این مقدار از 0 تا 29 متغیر است.		

۳-۲۳- تایمرها (Timers)

تایمرها نقش بسیار مهمی در کنترل اکثر فرایندها بر عهده دارند. از کنترل چراغهای راهنمایی سر چهارراه ها تا کنترل فرایندهای پیچیده صنعتی، همگی نیاز به زمان سنجی دارند. بنا به کاربرد تایمر می توان از انواع مختلف آن استفاده نمود. بنابراین در مورد استفاده از تایمر باید مشخص گردد که از چه نوع تایمری استفاده می شود. به طور کلی ۵ نوع تایمر به شرح زیر وجود دارد.

۱- تایمر پله ای	SP	(Pulse Timer)
۲- تایمر پله ای 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 برنامه نوشته شده برای یک نوع تایمر، به عنوان مثال SE را مورد بررسی قرار داده، ورودی ها و خروجی های آن را بررسی می نمائیم. در ادامه، برنامه زمان سنجی مربوط به تایمر SE به هر سه روش برنامه نویسی 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

```

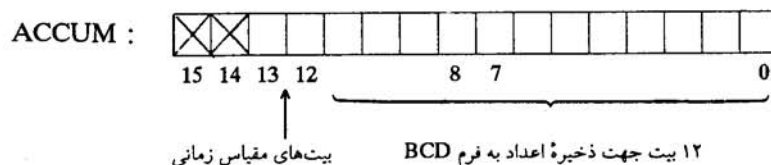
SEGMENT 1          0000
!      T 7
! I 1.0      +-----+
+---] [---+!1 - !
! KT 005.2 --!TV BI!- QW 10
!      ! DE!- FW 4
!      ! !
! I 1.1      !      !      Q 0.1
+---] [---+!R Q!-+-----+---( )-!
!      +-----+      :BE
!

```

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 ذخیره اعداد BCD از ۰۰۰ تا ۹۹۹ استفاده می‌شود.



عدد ثابت تایمر یا KT از یک مضرب و یک مقیاس زمانی تشکیل می‌گردد. مضرب می‌تواند عددی در فاصله ۰۰۰ تا ۹۹۹ 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

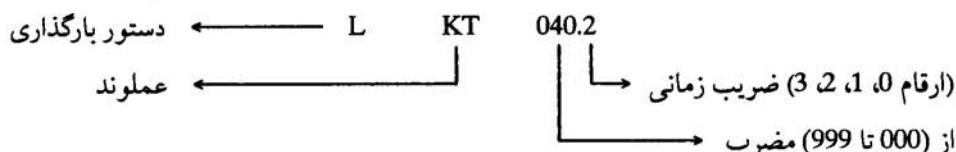
۰ ۰ = ۰ : ثانیه ۰/۰۱

۰ ۱ = ۱ : ثانیه ۰/۱

۱ ۰ = ۲ : ثانیه ۱

۱ ۱ = ۳ : ثانیه ۱۰

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:

ثانیه ۰/۰۱ \rightarrow 001.0 KT : کمترین زمان ممکن

ثانیه ۹۹۹۰ \rightarrow 999.3 KT : بیشترین زمان ممکن

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

حال فرض کنید که قصد داریم با استفاده از یک تایمر، تأخیری به مدت ۴۰ ثانیه^۱ ایجاد نماییم. 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	فاصله زمانی	دستور	زمانهای ممکن استفاده شده برای بارگذاری زمان ۴۰ ثانیه در تایمر
۰/۱ ثانیه	ثانیه $40 \pm 0.1 = 40.1 \times 0.1$	L KT 400.1	
۱ ثانیه	ثانیه $40 \pm 1 = 40.2 \times 1$	L KT 40.2	
۱۰ ثانیه	ثانیه $40 \pm 10 = 4.3 \times 10$	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 محاسبه 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 نحوه بار شدن عدد ۴۰ در انباره آمده است.

مقیاس زمانی ۱

×	×	×	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 KT 400.1
4 0 0

مقیاس زمانی ۲

×	×	×	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 KT 040.2
0 4 0

مقیاس زمانی ۳

×	×	×	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 KT 004.3
0 0 4

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 باقیمانده تایمر نسبت به TV به صورت عددی در مبنای دو در یک کلمه خروجی یا کلمه 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 باقیمانده تایمر نسبت به TV به صورت عددی در مبنای BCD به یک کلمه خروجی یا کلمه 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 لبه 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)^۱

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

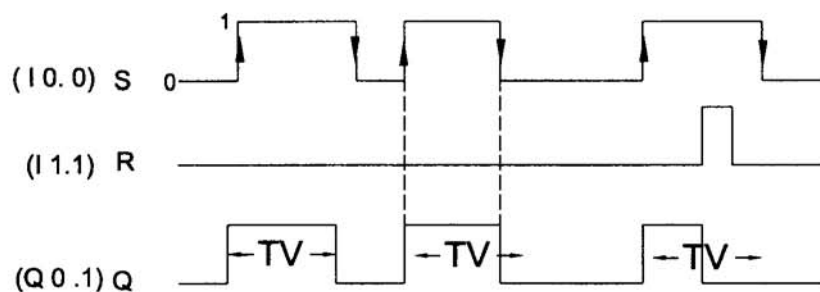
```



```

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

```



با نگاهی گذرا در برنامه 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)^۱

خروجی این تایمر تنها به لبه بالارونده ورودی S حساس است. با لبه 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 لبه پایین‌رونده داشته باشیم، این لبه، بر خروجی بی‌تأثیر بوده و پس از گذشت مدت زمان TV، خروجی غیرفعال می‌شود. در ادامه، برنامه نوشته شده جهت این نوع تایمر به همراه شکل موجهای S، R، 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

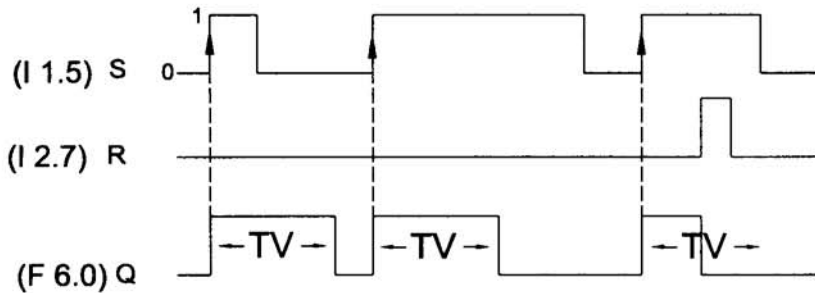
```

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
!

```



همان‌گونه که دیده می‌شود لبه پائین‌رونده ورودی S در عملکرد تایمر بی‌تأثیر است. در برنامه نوشته شده به روش STL به دلیل استفاده از خروجی BI (ارسال زمان باقیمانده 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)^۱

خروجی این تایمر هم به لبه بالا رونده و هم به لبه پائین‌رونده ورودی S حساس است. در طول مدت زمان تایمر، ورودی S باید فعال باقی بماند. با لبه بالا رونده S، خروجی تایمر پس از مدت زمان TV ثانیه فعال و با لبه پائین‌رونده S، غیرفعال می‌شود. با اندکی تأمل در می‌یابیم که عملکرد این تایمر درست برعکس تایمر SP است. در ادامه برنامه نوشته شده به همراه شکل موجهای S، R، Q و چگونگی تأثیر ورودی‌ها بر خروجی ارائه شده است.

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

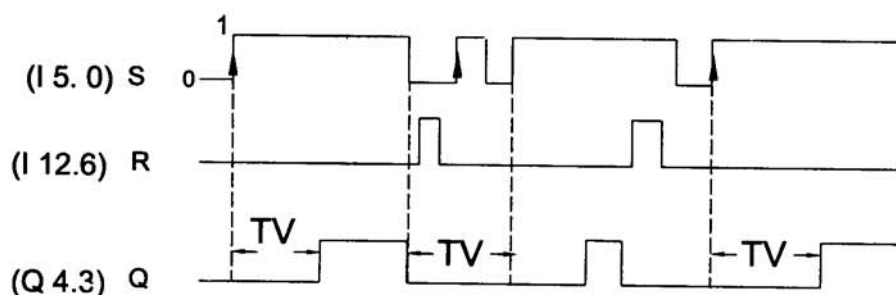
```

PB 13

```

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

```



۳-۲۳-۴- تایمر با تأخیر خاموش (SF)^۱

خروجی این تایمر با لبه پیش‌رونده ورودی S فعال و با لبه پس‌رونده 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

```

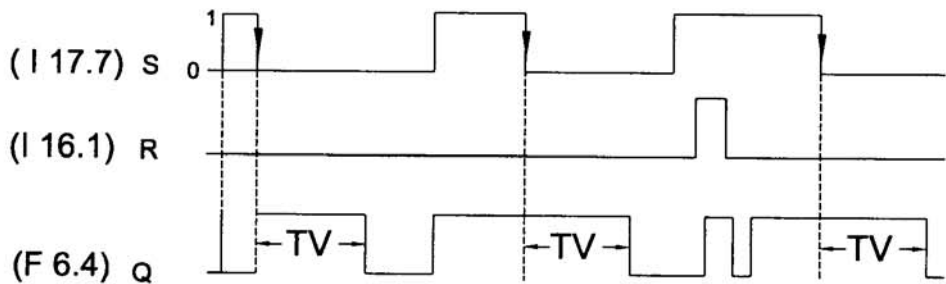
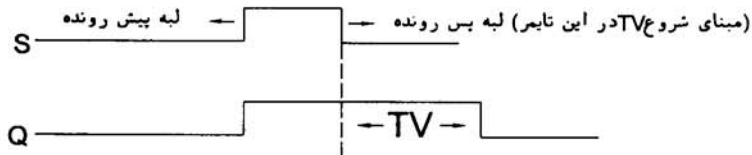
PB 14

```

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

```

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)^۱

خروجی این تایمر فقط به لبهٔ بالاروندهٔ ورودی حساس است. این تایمر با لبهٔ بالاروندهٔ 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.

PB 15

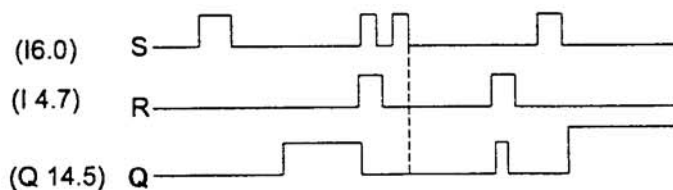
```

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      !          !
+----] [----+ !R      Q!-+-----+---( )-!
!          +-----+
!
!          :BE
    
```



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

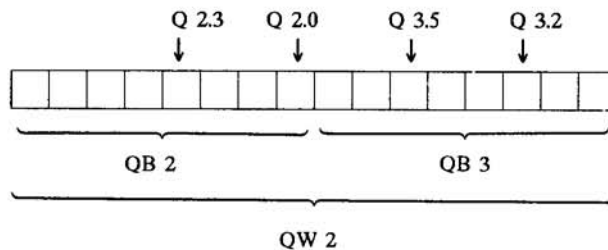
PB 16

```

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 کلمه‌ی خروجی ۲ (QW 2) را به یک سری نشان‌دهنده‌ی LED اعمال نمائیم مشاهده می‌کنیم که بیت‌های این کلمه‌ی خروجی با فرکانس‌های متفاوت (سرعت‌های متفاوت) فعال و غیرفعال می‌شوند یا به عبارت دیگر چشمک می‌زنند. در ادامه، این کلمه‌ی 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 $0/1$ ثانیه می‌باشد بنا به مطالب عنوان شده کم ارزش ترین بیت یعنی $Q\ 3.0$ با فرکانس $0/1$ 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 ارزش ترین 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 کلمه خروجی، این خواسته را برآورده ساخت. از این گونه برنامه‌ها در مواردی که لازم است عملی به صورت پرودیک انجام گیرد (به عنوان مثال: چشمک زدن LEDها و ...) استفاده می‌شود.

تنها اشکالی که در این برنامه وجود دارد (برنامه 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 پرودیک 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 برنامه 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 "°" = 6.0 F شده و چون در ورودی S تایمر ترکیب عطفی نقیض 6.0 F به همراه 0.0 I 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 برنامه 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 کلمه خروجی مثلاً 20 QW انتقال یابد. در بخش دوم بیت‌های این کلمه خروجی را به دو چراغ اعمال می‌کنیم. واضح است که برای داشتن

فرکانس ۱ ثانیه در چشمک زدن چراغ ۱ باید بیت 21.0 Q را به این چراغ اعمال نمود. در مورد چراغ دوم نیاز به فرکانسی چند برابر چراغ اول داریم بنابراین به اختیار یکی از بیت‌های 21.3 Q یا 21.4 Q 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 1 SEGMENT 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      5.0
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^۱

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 می‌بیند.

جدول ۳-۸: دستورات اعلام پایان برنامه

عملکرد	عملوند	توضیحات
BE	—	پایان برنامه - برنامه صرفنظر از اینکه بیت RLO چه مقداری داشته باشد پایان می‌یابد.
^۱ BEU	—	پایان برنامه بدون شرط - برنامه صرفنظر از اینکه بیت RLO چه مقداری داشته باشد پایان می‌یابد.
^۲ BEC	—	پایان برنامه با شرط - 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^۳ 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 زمینس 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 برنامه فعلی که با LABEL مشخص گردیده پرش و ادامه برنامه را از آن سطر دنبال نماید. LABEL یا برچسب، در هنگام اجرای برنامه علامت مشخصه‌ای جهت پرش پردازنده به سطر

1 - Block End Unconditional

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 حلقه 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 ادامه اجرای برنامه را از آن سطر دنبال می‌کند.

مثال ۳-۲۹: در یک بلوک تابع ساز مثلاً 4 FB 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 55 KH L اجرا و عدد 55H 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

M001 = JC 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 "۱" $I\ 0.2 = 1$ 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 تابع ساز، 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 "۱" $I\ 0.2 = 1$) قسمت‌های دیگر برنامه را به مرحله 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 "۱" $I\ 0.2 = 1$ باشد قسمت LABEL اجرا، ولی اگر "۰" $I\ 0.2 = 0$ 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 ارائه مثال‌های کاربردی و نمونه‌هایی از پروژه‌های صنعتی، روش برنامه‌نویسی را بررسی نمائیم.

فصل چهارم

۴-۱ - روش برنامه‌نویسی

آنچه که تاکنون در مورد آن به بحث پرداختیم، دستورات برنامه‌نویسی و ذکر مثالهایی ساده بود. در این فصل قصد داریم تا روش برنامه‌نویسی و روند کلاسیک برخورد یک برنامه‌نویس با پروژه یا فرآیندی را که قرار است توسط PLC کنترل شود بررسی کنیم. پروسه و روند برنامه‌نویسی به شرح زیر است:

۱- تعریف پروژه و فرآیند تحت کنترل

اولین گام برای نوشتن یک برنامه، تعریف برنامه و پروسه است. به عبارت دیگر برنامه‌نویس باید بداند که چه امکانات و ورودیهایی در دست دارد تا برای کنترل فرآیند مورد نظر بتواند از آنها استفاده نماید. از آنجایی که ممکن است یک برنامه‌نویس به تمام موارد و جزئیات سیستم و پروژه تسلط و اشراف نداشته باشد معمولاً در این مرحله از برنامه‌نویسی کارشناسانی که در مورد پروسه تحت کنترل اطلاعات کافی دارند، برنامه‌نویس را در جریان کلیه جزئیات سیستم قرار می‌دهند.

۲- رسم فلوچارت^۱ برنامه

معمولاً در مورد کنترل پروژه‌های بزرگ و حتی پروژه‌های کوچک پس از تعریف پروژه، مرحله رسم فلوچارت برنامه است. زیرا رسم فلوچارت ساده‌ترین روش جهت معرفی و بررسی عملکرد یک پروژه است. در برنامه‌نویسی PLC نیز این عمل با نوشتن برنامه‌ای مشابه با روش CSF انجام می‌گیرد.

۳- تهیه لیستی از ابزار مورد نیاز

در این مرحله، برنامه‌نویس به تعریف ورودی‌ها، خروجی‌ها، مشخص نمودن تایمرها، شمارنده‌ها و ... می‌پردازد.

۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی STL، LAD و CSF

در این مرحله از برنامه‌نویسی، برنامه‌نویس با استفاده از موارد ذکر شده قبلی یعنی رسم فلوچارت و ابزار مورد نیاز، برنامه اصلی کنترل پروسه را به یکی از سه روش مذکور می‌نویسد.

۵- در نظر گرفتن شرایط ایمنی و اقدامات حفاظتی

این مرحله، یکی از مهمترین مراحل برنامه‌نویسی است، چراکه اگر شرایط ایمنی در پروژه‌ها در نظر گرفته نشود ممکن است خسارات جبران‌ناپذیری به سیستم وارد آید. برای درک بیشتر این مطلب به ذکر یک مثال می‌پردازیم:

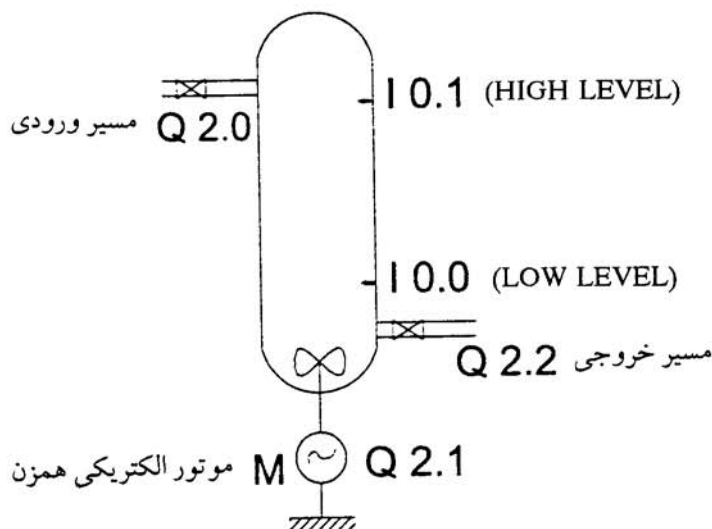
مثال ۴-۱: مخزنی را مطابق شکل ۴-۱ در نظر بگیرید. در این مخزن دو سنسور جهت تشخیص سطح مواد داخل مخزن در نظر گرفته شده است. 0.0 معرف کمترین سطح ممکن (Low Level) و 0.1 نشان دهنده بالاترین سطح ممکن (High Level) می‌باشد. دو شیر (Valve) یکی برای

۱ - منظور از فلوچارت در اینجا روش برنامه‌نویسی CSF نمی‌باشد بلکه فلوچارت، روند برنامه‌نویسی را به روشی مشابه با CSF مشخص می‌نماید. در مورد رسم فلوچارت می‌توان از نمادهای غیراستاندارد نیز استفاده نمود.

ورود مواد و دیگری جهت خروج آنها تعبیه شده است. این مخزن دارای یک همزن الکتریکی نیز می‌باشد. روند اجرای این پروسه به شرح زیر است:

در صورتی که سطح مواد داخل مخزن به Low Level برسد یعنی مقدار بیت ورودی I 0.0 برابر "۱" شود بایستی شیر ورودی باز شده ($Q\ 2.0 = "۱"$)، مواد از طریق این شیر به داخل مخزن راه یابند. این عمل تا زمانی انجام می‌گیرد که سطح مواد داخل مخزن به High Level نرسیده باشد در صورتی که سطح مواد به بالاترین مقدار ممکن برسد و یا به عبارت دیگر بیت ورودی I 0.1 برابر "۱" شود باید شیر ورودی بسته شده ($Q\ 2.0 = "۰"$)، موتور الکتریکی همزن شروع به چرخش نماید ($Q\ 2.1 = "۱"$) پس از مدت زمانی که برای هم زدن این مواد تعریف می‌شود، موتور الکتریکی همزن خاموش و شیر خروجی باز می‌شود ($Q\ 2.2 = "۱"$) تا اینکه مخزن از مواد موجود تخلیه گردد. پس از تخلیه مواد، مجدداً بیت ورودی I 0.0 برابر "۱" شده، سیکل مذکور مجدداً انجام می‌شود.

برای بررسی شرایط ایمنی تنها شرایطی خاص و مربوط به یک قسمت از برنامه را دنبال می‌نماییم و در مثالهای بعدی به طور مفصل در مورد این پروسه به بحث خواهیم پرداخت.



شکل ۴-۱: شمای پروسه تحت کنترل جهت بررسی شرایط ایمنی

حال فرض کنید که بیت 0.0 I برابر "۱" باشد در این حالت باید فرمان برای باز شدن شیر ورودی صادر شود.

خوانندگان توجه دارند که در این حالت تنها باز شدن شیر ورودی دال بر صحت عملکرد سیستم نیست، چرا که ممکن است شیر خروجی نیز در همین حالت باز بوده، مواد بدون اینکه در مرحله میانی با یکدیگر مخلوط شوند از مخزن خارج شوند. مورد دیگری که باید مدنظر داشت آن است که در زمان باز بودن شیر ورودی بایستی موتور الکتریکی همزن خاموش باشد. همچنین باید توجه داشت که اگر 0.0 I برابر "۱" باشد ورودی 0.1 I که معرف High Level است نمی تواند مقدار "۱" داشته باشد، چرا که در این صورت باید به صحت عملکرد یکی از دو سنسور شک نمود زیرا که در این وضعیت هر دو سنسور مقدار "۱" دارند و این مطلب بدان معنی است که سطح مواد داخل مخزن هم در وضعیت Low Level و هم در وضعیت High Level قرار دارد که چنین چیزی غیرممکن است.

نکته دیگر این که تا قبل از مرحله صدور فرمان باز شدن شیر ورودی یعنی 2.0 Q، این شیر باید کاملاً بسته باشد. حال همین موارد را در برنامه‌ای که به روش STL نوشته شده است به وضوح می‌بینیم:

در صورتی که سطح مواد داخل مخزن به Low Level برسد ("۱" = 0.0 I اگر) $A \rightarrow I \ 0.0$
و موتور همزن الکتریکی خاموش باشد ("۰" = 2.1 Q) $AN \rightarrow Q \ 2.1$
و شیر خروجی بسته باشد ("۰" = 2.2 Q) $AN \rightarrow Q \ 2.2$
و سنسور مربوط به High Level فعال نشده باشد ("۰" = 0.1 I) $AN \rightarrow I \ 0.1$
و تا قبل از این فرمان، شیر ورودی کاملاً بسته باشد ("۰" = 2.0 Q) $AN \rightarrow Q \ 2.0$
اکنون در صورت برقراری تمامی شرایط فوق دستور باز شدن ("۱" = 2.0 Q) $S \rightarrow Q \ 2.0$
شیر ورودی صادر می‌شود

همان‌گونه که ملاحظه شد بررسی شرایط ایمنی و گنجاندن آنها در برنامه، نیاز به اندکی تجربه در برخورد با پروژه‌های کوچک و بزرگ صنعتی دارد. مواردی که ذکر گردید تنها برای حالتی است که سطح مواد داخل مخزن به Low Level رسیده باشد و در این حالت نیاز به صدور فرمان باز شدن شیر ورودی وجود دارد، در سایر موارد یعنی روشن شدن موتور الکتریکی همزن و همچنین باز

شدن شیر خروجی باید شرایط ایمنی دیگری را تقریباً مشابه شرایط مذکور در نظر بگیریم. این قسمت از برنامه یعنی بررسی شرایط ایمنی در مورد روشن شدن همزن الکتریکی و باز شدن شیر خروجی به خواننده واگذار می‌گردد.

حال که روش برخورد با یک برنامه و پروسه را آموختیم به ذکر مثالهایی در این زمینه می‌پردازیم. این مثالها کاملاً کاربردی و عملی بوده، در پروسه‌های کوچک و بزرگ با آنها برخورد خواهیم داشت.

نحوه ارائه مطالب در این مثالها به گونه‌ای است که با توضیحات مفصل، تمامی خوانندگان (حتی خوانندگانی که تجربه‌ای در مورد پروژه‌های صنعتی ندارند) بتوانند در موارد بعدی، مراحل برنامه‌نویسی را خود دنبال نمایند. اکثر این مثالها با استفاده از شکل‌های مناسب و یا شبیه‌سازهای^۱ صنعتی و غیرصنعتی مدل‌سازی شده‌اند و اجرای برنامه را می‌توان به طور کامل و مرحله به مرحله بر روی این شبیه‌سازها دنبال نمود.

مثال ۴-۲: برنامه چراغ راهنمایی:

در گروهی از برنامه‌ها با استفاده از مقایسه‌کننده‌های اعداد، فلگ‌های خاص، دستورات T، L و ... می‌توان برنامه را به طور چشمگیری ساده و خلاصه نمود. نمونه‌ای از این برنامه‌ها، چراغ راهنمایی است که در شکل ۴-۲ شمای شبیه‌ساز برنامه آن را ملاحظه می‌کنید.

در این مثال قصد داریم با نوشتن یک برنامه، پروسه چراغ راهنمایی را به صورت زیر کنترل نماییم:

۱- هنگامی که کلید S1 فعال شود سیستم کنترل شروع به کار کند.

۲- زمانی که چراغ سبز برای اتومبیل‌ها روشن است چراغ قرمز مخصوص عابر پیاده نیز در سمت دیگر چهار راه روشن باشد.

۳- زمانی که چراغ زرد برای اتومبیل‌ها روشن است چراغ زرد مخصوص عابر پیاده نیز در سمت دیگر چهارراه روشن باشد.

۴- زمانی که چراغ قرمز برای اتومبیل‌ها روشن است چراغ سبز مخصوص عابر پیاده نیز در

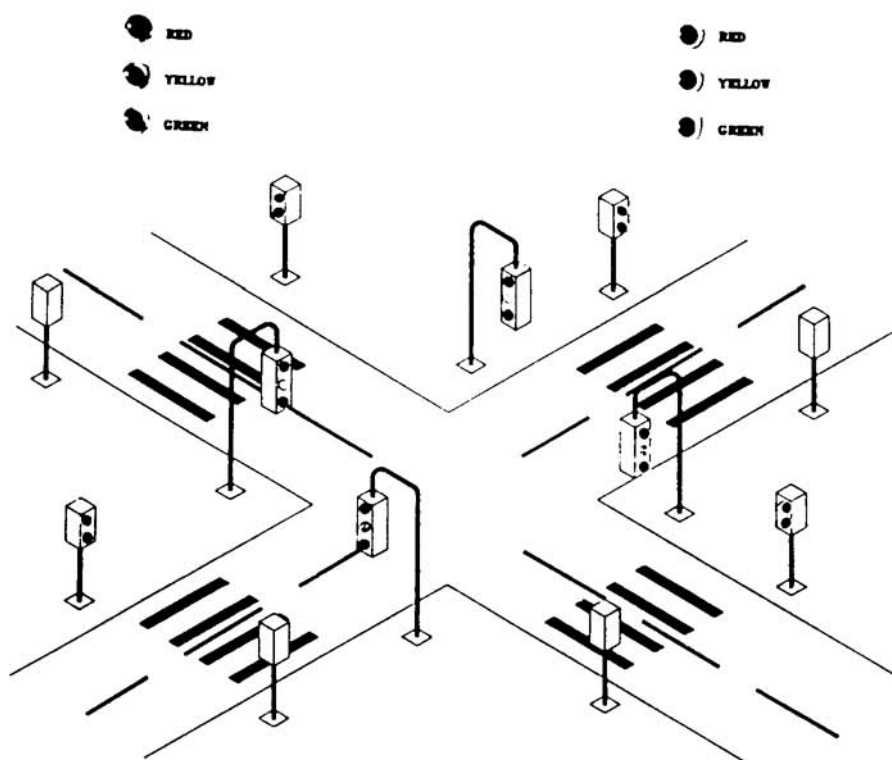
۱ - این شبیه‌سازها (Simulators) در شرکت کنترونیک طراحی شده‌اند.

سمت دیگر چهارراه روشن باشد.

۵- همواره یکی از چراغهای مخصوص اتومبیل‌ها به همراه چراغ متناظر آن، برای عابر پیاده روشن باشد. (مثلاً چراغ قرمز برای اتومبیل به همراه چراغ سبز برای عابر پیاده).

۶- مدت زمان روشن بودن چراغهای سبز و قرمز ۶۰ ثانیه و مدت زمان روشن بودن چراغ زرد ۵ ثانیه باشد.

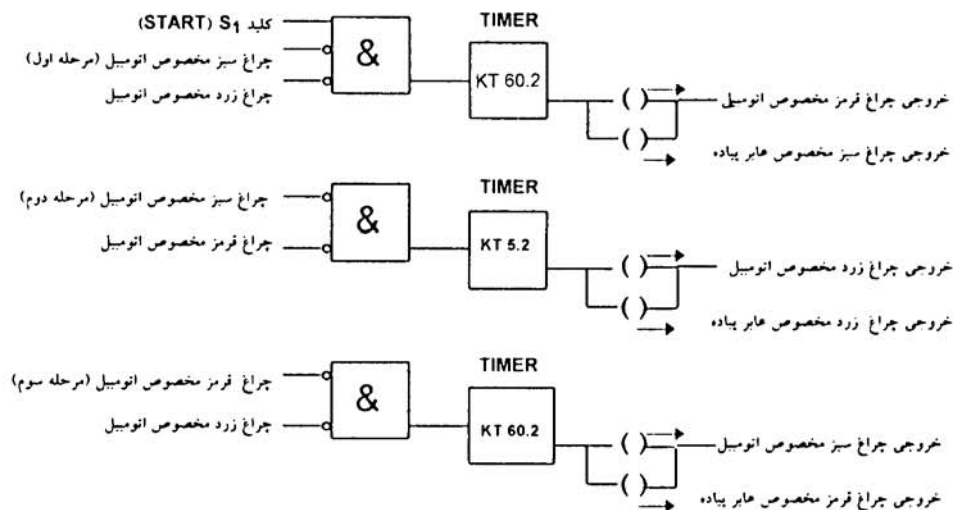
CONTRONIC CO. TRAFFIC LIGHTS



شکل ۴-۲: شبیه‌ساز برنامه چراغ راهنمایی

اکنون برنامه‌نویسی را به همان روشی که در ابتدای فصل ذکر شد دنبال می‌کنیم:

- ۱- تعریف پروژه: پروژه کنترل چراغ راهنمایی در ۶ مورد فوق تعریف و توصیف گردید.
 - ۲- رسم فلوچارت: از آنجایی که این پروژه شامل برنامه‌نویسی برای ۳ حالت می‌باشد برای رسم فلوچارت آن از سه مرحله استفاده می‌کنیم. این مراحل عبارتند از:
 - مرحله ۱: چراغ قرمز مخصوص اتومبیل به همراه چراغ سبز مخصوص عابر پیاده.
 - مرحله ۲: چراغ زرد مخصوص اتومبیل به همراه چراغ زرد مخصوص عابر پیاده.
 - مرحله ۳: چراغ سبز مخصوص اتومبیل به همراه چراغ قرمز مخصوص عابر پیاده.
- در ادامه، فلوچارت هر سه مرحله آورده شده است.

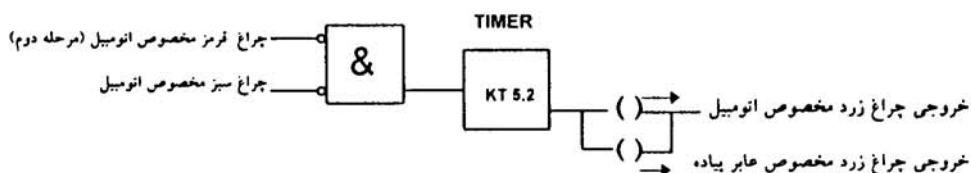


در رسم این فلوچارت، شرایط ایمنی نیز در نظر گرفته شده است. به عنوان مثال در مرحله اول برای به کار افتادن تایمر باید کلید استارت فعال باشد. این مورد تنها شرط لازم جهت ست نمودن تایمر نیست. همان‌گونه که در فلوچارت مرحله اول ملاحظه می‌کنید از نقیض خروجی مراحل دیگر یعنی نقیض خروجی چراغ سبز و چراغ زرد مخصوص اتومبیل به صورت ترکیب عطفی با کلید استارت جهت ست نمودن تایمر برای فعال نمودن خروجی چراغ قرمز مخصوص اتومبیل استفاده شده است. در مراحل دیگر نیز نقیض خروجی‌های دو مرحله دیگر جهت ست نمودن تایمر به کار

برده شده‌اند.

توجه داشته باشید که خروجی هر تایمر جهت روشن نمودن دو چراغ متناظر استفاده شده است. به عنوان مثال در مرحله اول، خروجی تایمر جهت روشن نمودن چراغ قرمز مخصوص اتومبیل و چراغ سبز مخصوص عابر پیاده به کار برده شده است زیرا در مدت زمانی که چراغ قرمز مخصوص اتومبیل روشن است چراغ سبز مخصوص عابر پیاده هم روشن می‌باشد و به همین ترتیب در مراحل دیگر مشابه این مرحله را خواهیم داشت.

با توجه به این نکته می‌توان شرایط ایمنی معادل را در مورد هر مرحله در نظر گرفت. به عنوان مثال در مرحله دوم به جای استفاده از فلوچارت رسم شده قبلی می‌توان فلوچارت دیگری مشابه با همان فلوچارت اولیه رسم نمود که شرایط ایمنی در نظر گرفته شده در آن کاملاً متناظر با شرایط در نظر گرفته شده در فلوچارت اولیه باشد.



۳- تهیه لیستی از ابزار مورد نیاز: ابزار مورد نیاز در این پروژه، تعدادی ورودی، خروجی و همچنین تایمر و نوع تایمر استفاده شده می‌باشد. لیست ورودی‌ها، خروجی‌ها و تایمرهای در نظر گرفته شده در این پروژه به ترتیب زیر است.

کلید استارت S1	I 16.0	خروجی چراغ سبز مخصوص اتومبیل	Q 8.4
خروجی چراغ قرمز مخصوص اتومبیل	Q 8.0	خروجی چراغ قرمز مخصوص عابر پیاده	Q 8.5
خروجی چراغ سبز مخصوص عابر پیاده	Q 8.1	تایمر مرحله اول	T1
خروجی چراغ زرد مخصوص اتومبیل	Q 8.2	تایمر مرحله دوم	T2
خروجی چراغ زرد مخصوص عابر پیاده	Q 8.3	تایمر مرحله سوم	T3

از آنجایی که این برنامه باید به گونه‌ای نوشته شود که خروجی تایمر وابسته به ورودی باشد از

تایمر SP استفاده می‌کنیم زیرا همان‌طور که گفته شد یکی از شرایط استفاده شده در تعریف پروژه آن است که با فعال شدن کلید S1 (ورودی استارت) سیستم کنترل نیز فعال شده، با غیرفعال شدن کلید مذکور سیستم متوقف گردد. چنانچه از تایمر SE استفاده نماییم در صورت وجود یک لبه در ورودی یا تک‌پالس (پالس ورودی با زمان خیلی کوتاه مثل استارت سریع و قطع مجدد کلید استارت) سیستم شروع به کار کرده، سیکل زمانی تایمر پروسه به کار خود ادامه می‌دهد.

در این برنامه چون به ورودی R در تایمر و همچنین خروجی‌های BI و DE نیازی نیست از تعریف آنها نیز در فلوچارت خودداری و در برنامه‌نویسی به روش STL (در مرحله بعدی برنامه‌نویسی) به جای این ورودی و خروجی‌ها از دستور NOP استفاده می‌کنیم.

۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: در اینجا برای تمرین بیشتر از دو روش STL و CSF جهت برنامه‌نویسی این پروژه استفاده شده است. برنامه مذکور در PB 10 و در سه بخش (Segment) نوشته شده است.

PB 10

```

SEGMENT 1          0000
0000      :A      I      16.0
0001      :AN     Q      8.2
0002      :AN     Q      8.4
0003      :L      KT     005.2
0005      :SP     T      1
0006      :NOP    0
0007      :NOP    0
0008      :NOP    0
0009      :A      T      1
000A      :      Q      8.0
000B      :      Q      8.1
000C      :***

```

```

SEGMENT 2          000D
000D      :AN     Q      8.4
000E      :AN     Q      8.0
000F      :L      KT     003.2
0011      :SP     T      2
0012      :NOP    0
0013      :NOP    0
0014      :NOP    0
0015      :A      T      2
0016      :      Q      8.2
0017      :      Q      8.3
0018      :***

```

```

SEGMENT 3          0019
0019      :AN  Q    8.0
001A      :AN  Q    8.2
001B      :L   KT 005.2
001D      :SP  T    3
001E      :NOP 0
001F      :NOP 0
0020      :NOP 0
0021      :A   T    3
0022      : =   Q    8.4
0023      : =   Q    8.5
0024      :BE
    
```

PB 10

```

SEGMENT 1          0000
      +---+
I 16.0  ---! & !   T 1
Q 8.2    --O!      +-----+
Q 8.4    --O!      !-----! 1_ _ !
      +---+      !
      KT 005.2  --!TV BI!-
                  !  DE!-
                  !
                  --!R   Q!-+-! =   ! Q 8.0
                  +-----+
                  !
                  +-----+
                  +-! =   ! Q 8.1
                  +-----+
    
```

```

SEGMENT 2          000D
      +---+      T 2
Q 8.4    --O! & !   +-----+
Q 8.0    --O!      !-----! 1_ _ !
      +---+      !
      KT 003.2  --!TV BI!-
                  !  DE!-
                  !
                  --!R   Q!-+-! =   ! Q 8.2
                  +-----+
                  !
                  +-----+
                  +-! =   ! Q 8.3
                  +-----+
    
```



```

SEGMENT 3          0019
                    +---+ T 3
Q 8.0      --O! & !      +-----+
Q 8.2      --O!      !-----! 1 - !
                    +---+      !
                    KT 005.2 --! TV BI! -
                               ! DE! -
                               !      !
                               !      ! +-----+
                               --! R  Q! -+--! =      ! Q 8.4
                               +-----+ ! +-----+
                               ! +-----+
                               +-! =      ! Q 8.5
                               +-----+ : BE

```

در اینجا به ذکر چند نکته در مورد این برنامه می‌پردازیم:

- در مورد تعریف TV: از آنجایی که در این برنامه تولرانس زمانی (حساسیت یا خطای زمانی) از اهمیت چندانی برخوردار نیست از ضرب زمان ۲ استفاده شده است. (به عنوان مثال در دستور (L KT 60.2

- در مورد تعریف تایمر: همان‌گونه که ذکر شد در این برنامه به دلیل تعریف پروژه مبنی بر وابستگی فعالیت سیستم به ورودی S1 و نقیض دو خروجی دیگر استفاده از تایمر SP الزامی است. در هنگام استفاده از تایمر حتماً باید نوع و شماره آن ذکر شود مثلاً در مرحله سوم دستور SP T 3 ، تایمر شماره ۳ از نوع SP معرفی می‌شود.

در فصل قبل در مورد استفاده از تایمرها توضیحاتی ارائه شد. همان‌گونه که عنوان شد تعداد تایمرها در PLCهای مختلف محدود و متفاوت است. اکنون با ذکر یک مثال در رابطه با رفع اشکال مورد بحث در آن، راه حل مناسبی پیشنهاد می‌کنیم.

مثال ۳-۴: فرض کنید که در یک پروژه کنترلی و برای تعریف پروژه و برنامه‌نویسی به ۲۰ تایمر نیاز داریم اما PLC موجود در دسترس تنها دارای ۱۶ تایمر (T0 - T15) است. برای رفع این شکل چه می‌کنید؟

برای حل این مشکل برنامه PB 34 پیشنهاد می‌گردد:

PB 34

```

SEGMENT 1          0000
0000      :AN  F    6.0
0001      :L   KT  050.1
0003      :SE  T    1
0004      :A   T    1
0005      : =   F    6.0
0006      : ***

```

```

SEGMENT 2          0007
0007      :AN  F    6.0
0008      :CU  C    5
0009      :L   C    5
000A      :L   KF  +10
000C      : !=F
000D      :S   Q    2.0
000E      :S   Q    2.1
000F      :L   C    5
0010      :L   KF  +20
0012      : !=F
0013      :R   Q    2.0
0014      :L   C    5
0015      :L   KF  +30
0017      : !=F
0018      :R   Q    2.1

```

در فصل قبل عنوان شد که می‌توان بدون استفاده از تعریف تایمر، یک تایمر را برنامه‌نویسی نمود. برنامه PB 34 همان برنامه مذکور است. در اینجا می‌توان با استفاده از تولید پالس توسط یک تایمر و یک شمارنده و ست و ری ست نمودن پی‌درپی، تایمرهای زیادی ایجاد نمود.

در برنامه PB 34، فلگ F 6.0 مولد پالس بوده، به فواصل زمانی هر ۵ ثانیه برای یک سیکل زمانی، منفی و سپس مثبت می‌شود و هر بار یک واحد به شمارنده ۵ اضافه می‌کند. حال اگر عدد موجود در شمارنده را با ۱۰ مقایسه نمائیم یعنی ۵۰ ثانیه ($10 \times 5 = 50$) بعد از این پریود زمانی خروجی‌های Q 2.0 و Q 2.1 فعال می‌شود و با در نظر گرفتن مقایسه‌های انجام شده، خروجی

2.0 Q برای مدت ۵۰ ثانیه و خروجی 2.1 Q برای مدت زمان ۱۰۰ ثانیه فعال خواهند بود.
 حال به بررسی یک مشکل دیگر می‌پردازیم. در فصل قبل ذکر شد که حداکثر عددی که می‌توان در ورودی یک شمارنده اعمال نمود ۹۹۹ است. در صورتی که نیاز به شمارش عددی بیش از این مقدار داشته باشیم راه حل مناسبی پیشنهاد کنید.

برای رفع این مشکل می‌توان چند سطر به انتهای برنامه 34 PB اضافه نمود.

0019	: L	C	5	در این برنامه شمارنده ۵ بعد از هر ۹۹۹ بار
001A	: L	KF	+999	شمارش یک بار صفر می‌شود. اگر از لبه
001C	: !=F			پائین‌رونده شمارنده ۵ به شمارنده ۶ ارسال
001D	: R	C	5	نماییم شمارنده ۶ یک واحد افزایش خواهد
001E	: AN	C	5	یافت، بنابراین در صورتی که به عنوان مثال
001F	: CU	C	6	شمارنده ۶ حاوی عدد ۱۰۰ باشد.
0020	: BE			

بدین معنی است که به تعداد (۱۰۰×۹۹۹) بار
 شمارش انجام شده است. پس با استفاده از این
 دو شمارنده می‌توان به تعداد (۹۹۹×۹۹۹) بار
 شمارش انجام داد.

از این پس در نوشتن برنامه‌ها از DBها (Data Block) و FBها (Function Block) استفاده می‌کنیم. در این قسمت قصد داریم به تفصیل در مورد این بلوک‌ها که در اکثر برنامه‌ها به کار برده می‌شوند توضیحاتی ارائه دهیم.

۴-۲- بلوک‌های اطلاعاتی (DB)

همان‌گونه که در فصل گذشته عنوان شد این بلوک‌ها شامل اطلاعاتی نظیر پارامترها و پیامها می‌باشند.

افرادی که با محیط‌های صنعتی آشنایی دارند به خوبی می‌دانند که هنگام ایجاد اشکال در سیستم‌های کنترلی مدرن (سیستم‌های مونیتورینگ) پیغامهای خطایی نظیر HIGH TEMPERATURE ، TANK LEVEL LOW و ... بر روی صفحه نمایش ظاهر

می شود. این گونه پیغامها در DB وجود دارد و PLC به هنگام ایجاد مشکل در سیستم، پیغامهای مناسب را از DB خوانده، به صفحهٔ مونیتور می فرستد. در PLC مورد بحث ما می توان ۲۵۶ بلوک اطلاعاتی (DB 0 - DB 255) تعریف نمود. هر DB می تواند شامل ۲۵۶ سطر باشد که در هر سطر آن ۱۶ بیت (۱ کلمه) وجود دارد. به همین دلیل، هر یک از اطلاعات موجود در سطرهای DB را یک DW (Data Word) می گویند.

اطلاعاتی که می توان در DB قرار داد به یکی از صورتهای زیر می باشد:

۱- Data (مقادیر و پارامترها)

۲- Text (متن پیامها و ...)

۳- Bit Pattern (این اطلاعات شامل تعدادی بیت های "۰" و "۱" است که به صورت بایتی یا کلمه ای به خروجی ارسال شده، عمل سیگنالیینگ یا فعال و غیر فعال نمودن خروجی ها را برعهده دارند) در فصل گذشته دیدیم که در مثال ۳-۱۵ با استفاده از دستورات JU و JC که باعث پرش از یک بلوک به بلوک دیگر می شوند می توان بلوک های PB را اجرا نمود. اما در مورد DB ها فراخوانی اطلاعات با دستور دیگری انجام می گیرد. برای خواندن اطلاعات هر DB ابتدا باید آن را در طول اجرای برنامه صدا^۱ زد.

در مورد PB ها ملاحظه شد که اجرای برنامه بدین صورت است که پردازنده از سطر اول، دستورات را به ترتیب اجرا می نماید تا اینکه به سطر انتهایی برنامه برسد اما در مورد DB ها می توان به سطر دلخواه دست یافت. به عنوان مثال برای خواندن اطلاعات سطر صدم از DB 50 به ترتیب زیر عمل می کنیم.

ابتدا DB شماره ۵۰ را با استفاده از دستور C صدا زده، اطلاعات (DW) موجود در سطر صدم آن را با استفاده از دستور L در اتبازک بارگذاری می کنیم:

```

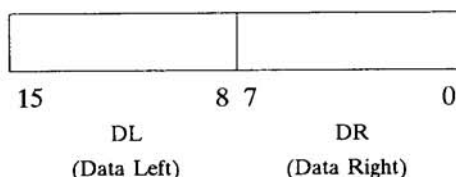
:      :
:  C   DB  50
:      :
:  L   DW  100
:      :

```

اطلاعاتی که در DBها قرار می‌گیرند به یکی از فرمت‌های زیر می‌باشد:

- ۱) KH : 16 BITS $(0000_{(H)} \rightarrow FFFF_{(H)})$ برای اعداد در مبنای ۱۶
 - ۲) KF : 16 BITS $(-32768 \rightarrow +32768)$ برای اعداد در مبنای ۱۰
 - ۳) KT : 14 BITS $(001.0 \rightarrow 999.3)$ برای اعداد ثابت TV
 - ۴) KC : 12 BITS $(000 \rightarrow 999)$ برای اعداد ثابت شمارنده‌ها
- یادآوری: اعداد با فرمت KT و KC در مبنای BCD می‌باشند.
- ۵) KY : 16 BITS (2 BYTES) در این حالت ۱۶ بیت به دو بایت چپ و راست تقسیم می‌شوند.

این دو بایت کاملاً مجزا بوده، هر یک از آنها می‌تواند مقادیر ۰۰۰ الی ۲۵۵ را داشته باشد.



- ۶) KM : 16 BITS $((000 \dots 00) \rightarrow (111 \dots 11))$
۱۶ بیت
۱۶ بیت

- ۷) KG : 32 BITS (DOUBLE WORD یا DWORD)

جهت نمایش اعداد با ممیز اعشاری (Floating Point) و نیز اعداد بسیار بزرگ یا بسیار کوچک از این شکل نمایش استفاده می‌شود. مثلاً عدد ۱۵۲۴۶۷۰ ± ۰.۹ را در نظر بگیرید، در PLC زیمنس این عدد ابتدا به $۱۰^۶$ تقسیم می‌شود. در مرحله بعد در صورت مثبت بودن علامت توان، PLC عدد حاصل را در $۱۰^۹$ و در صورت منفی بودن علامت توان، آن را در $۱۰^{-۹}$ ضرب می‌کند. محدودیتی که در این روش وجود دارد آن است که دو رقم مشخص کننده توان حداکثر ۳۸ می‌باشد.

- ۸) KS : TANK LEVEL LOW.

جهت نمایش پیامها و متون به کار برده شده در سیستم‌های کنترلی مونیتورینگ از این شکل نمایش استفاده می‌شود. در این حالت دقیقاً شبیه به کد ASCII در ازای هر کاراکتر یک بایت از

حافظه اختصاص می‌یابد، یعنی برای ذخیره پیغام فوق ۱۵ بایت از حافظه اشغال می‌گردد. در این حالت هر حرف، علامت و حتی جای خالی و نقطه انتهایی به عنوان یک کاراکتر محسوب می‌شود. بلوک اطلاعاتی زیر را در نظر بگیرید:

DB 25:

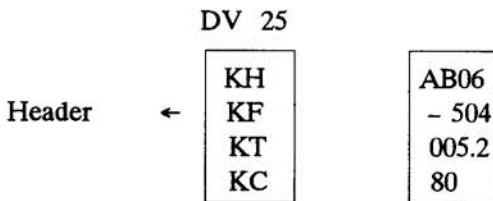
KH = AB06

KF = - 504

KT = 005.2

KC = 080

پس از وارد کردن اطلاعات این DB به سیستم، PLC دو بلوک به شرح زیر ایجاد می‌کند. یکی از بلوک‌ها حاوی شکل، فرمت اعداد، پارامترها و پیامها بوده، بلوک دیگری با نام DV و هم شماره با DB اولیه است و اصطلاحاً به آن Block Header گفته می‌شود حاوی اطلاعات و پارامترها می‌باشد. در زمان انتقال DB از واحد برنامه‌نویسی PG به PLC تنها اطلاعات محض منتقل شده، Header بر روی فلاپی (هارد دیسک یا فلاپی دیسک) باقی می‌ماند. این عمل جهت جلوگیری از اشغال حجم حافظه انجام می‌گیرد.



DBهای ایجاد شده را از لحاظ محل ذخیره‌سازی می‌توان به دو دسته کلی زیر تقسیم نمود:

۱- DBهایی که حاوی پارامترهای ثابت فرآیند و یا خط تولید بوده، اطلاعات آنها در EPROM یا EEPROM ذخیره می‌گردد.

۲- DBهایی که حاوی اطلاعات موقتی بوده و برای مصارف کوتاه مدت و موقت استفاده می‌شوند. این‌گونه DBها در RAM ذخیره می‌گردند.

از آنجایی که در طول یک برنامه ممکن است چندین DB فراخوانده شده و یا با پرش به برنامه‌های دیگر از DBهای موجود در بلوک جدید استفاده شود در این قسمت به بحث در مورد

اعتبار DBها به هنگام پرش (JUMP) و یا توقف برنامه (STOP) می‌پردازیم. برای روشن شدن مفهوم اعتبار DBها دو بلوک برنامه زیر را در نظر بگیرید.

PB 20				PB 22			
SEGMENT	1		0000	SEGMENT	1		0000
0000	:C	DB	16	0000	:C	DB	25
0001	:L	KF	+150	0001	:L	FW	20
0003	:T	DW	2	0002	:T	DW	0
0004	:L	DW	6	0003	:L	DW	2
0005	:!=F			0004	:L	DW	4
0006	:JC	PB	22	0005	:+F		
0007	:L	DW	3	0006	:T	DW	16
0008	:L	KF	+10	0007	:L	DW	8
000A	: -F			0008	:!=F		
000B	:T	QW	2	0009	:S	Q	4.5
000C	:BE			000A	:BE		

در سطر اول PB 20 دستور DB 16 باعث معتبر یا فعال شدن DB 16 می‌گردد. بنابراین DWهای موجود در سطرهای سوم و چهارم مربوط به DB 16 می‌باشند. در سطر ششم در صورتی که بیت RLO برابر "۱" باشد و یا به عبارت دیگر در صورتی که اطلاعات موجود در DW 6 موجود در DB 16 با عدد KF 150 مساوی باشد پرش به PB 22 صورت می‌گیرد. در این بلوک باز هم DB 16 فعال است تا اینکه در سطر سوم با صدا زدن DB 25، این DB معتبر شده و از این پس عملیات L و T بر روی DWهای این DB انجام می‌گیرد تا اینکه اجرای PB 22 به پایان برسد. پس از پایان PB 22، پردازنده، اجرای برنامه را در سطر بعدی خطی از برنامه که در آن، دستور پرش وجود دارد دنبال می‌کند. پس از بازگشت به PB 22 مجدداً DB 16 (همان بلوک داده‌ای که در هنگام پرش معتبر بود) فعال می‌شود و عملیات L و T بر روی DWهای این DB انجام می‌گیرد تا اینکه اجرای این بلوک نیز به پایان رسد.

در هنگام پرش (رفت و برگشت) بین دو بلوک، اطلاعاتی به شرح زیر در حافظه PLC ذخیره می‌شود تا هنگام بازگشت، اجرای برنامه روند قبلی خود را طی نماید:

۱- شماره DB فعال یا معتبر در زمان پرش و یا توقف برنامه.

۲- شماره سطری از برنامه اول که در آن، دستور پرش یا توقف وجود دارد.

۳- شماره و نوع بلوکی که در آن، دستور پرش یا توقف وجود دارد.

این اطلاعات در محلی از حافظه به نام B - STACK (BLOCK STACK) قرار می‌گیرد و از آنجایی که تعداد بیت‌های موجود در این قسمت از حافظه محدود است و گنجایش ذخیره اطلاعات فراوانی ندارد، تعداد پرش‌های افقی و عمودی نیز محدود بوده و این تعداد در PLC‌های مختلف، متفاوت است. منظور از پرش افقی، پرش از یک بلوک به بلوک دیگر است در حالی که پرش عمودی، پرش به سطرهای همان برنامه (سطرهای بالاتر یا پائین‌تر) می‌باشد.

مثال ۴-۴: فرض کنید در سطری از یک DB، اطلاعات 0000_H وجود دارد. برنامه‌ای بنویسید که در اجرای هر سیکل برنامه ۱ واحد به مقدار موجود در این کلمه اضافه کند.

برنامه خواسته شده را در یک PB نوشته و در آن بلوک یک DB را که خود ایجاد نموده‌ایم صدا می‌کنیم. در ادامه، این برنامه به روش STL آمده است.

PB 30

SEGMENT	1		0000
0000	:C	DB	27
0001	:L	DW	0
0002	:L	DW	1
0003	:+F		
0004	:T	DW	0
0005	:T	QW	2
0006	:BE		

DB27

0:	KF = +00000;
1:	KF = +00001;
2:	

روند اجرای برنامه به صورت زیر است:

در سطر اول PB 30، بلوک داده شماره ۲۷ معتبر شده، در سطرهای دوم و سوم این PB اطلاعات موجود در سطرهای ۰ و ۱ یعنی DW 0 و DW 1 در انبارک‌ها بارگذاری می‌شود. در سطر چهارم به کمک دستور + F محتویات این دو کلمه با هم جمع و در سطر پنجم توسط دستور

0 DW T حاصل جمع این دو عدد به 0 DW در 27 DB فرستاده می‌شود. پس در هر بار اجرای سیکل این برنامه یک واحد به 0 DW اضافه می‌شود.

در انتهای برنامه، حاصل جمع دو عدد موجود در 0 DW و 1 DW به کلمه خروجی شماره ۲ یعنی 2 QW فرستاده می‌شود. کلمه خروجی ۲ می‌تواند به یک سری نشان دهنده مثلاً LED متصل شده باشد. در این صورت چگونگی افزایش محتویات 0 DW در هر سیکل اجرای برنامه با خاموش و روشن شدن LEDها قابل رویت می‌گردد. بیشترین عددی که در 0 DW به عنوان حاصل جمع قرار می‌گیرد عدد $FFFF_H$ می‌باشد. در صورت اضافه شدن این عدد به عدد ثابت ۱ مجدداً عدد 0000_H در 0 DW قرار می‌گیرد و این سیکل مرتباً تکرار می‌شود.

با اندکی دقت در این برنامه در می‌یابیم که می‌توان به کمک سرعت روشن و خاموش شدن LEDهای مذکور سرعت پردازنده PLC را در اجرای یک سیکل برنامه اندازه‌گیری نمود. اما سرعت پردازش و انتقال اطلاعات توسط پردازنده به اندازه‌ای بالا است که نمی‌توان سرعت اجرای یک سیکل برنامه را به دست آورد. سرعت اجرای یک سیکل برنامه معادل با سرعت خاموش و روشن شدن LEDهای متصل به 2 QW می‌باشد. بنابراین سرعت اجرای تعداد سیکل برنامه مثلاً ۲۰۰۰ مرتبه را اندازه‌گیری و سپس زمان به دست آمده را بر عدد ۲۰۰۰ تقسیم می‌کنیم.

مثال ۴-۵: برنامه‌ای بنویسید که سرعت اجرای یک سیکل زمانی اجرای برنامه توسط ریزپردازنده PLC را اندازه‌گیری کند.

برای نوشتن این برنامه از برنامه نوشته شده در مثال ۴-۴ استفاده می‌کنیم. با این تفاوت که در

27 DB، سطر دیگری را که همان $KF = 2000$ است وارد نموده، با ایجاد تغییراتی در 30 PB

برنامه موجب می‌شویم که تنها ۲۰۰۰ سیکل از برنامه اجرا شود. PB 31

SEGMENT	1		0000
0000	:C	DB	28
0001	:L	DW	0
0002	:L	DW	1
0003	:+F		
0004	:T	DW	0
0005	:L	DW	2
0006	:!=F		
0007	:S	Q	2.0
0008	:BE		

DB28

```

0:      KF = +00000;
1:      KF = +00001;
2:      KF = +02000;
3:

```

روند اجرای این برنامه مانند برنامه 30 PB است با این تفاوت که در اجرای این برنامه و در هر سیکل، اطلاعات موجود در 2 DW یعنی عدد ۲۰۰۰ با حاصل جمع موجود در 0 DW یا تعداد سیکل‌های انجام شده مقایسه می‌گردد. در صورت برابر بودن این دو مقدار بیت خروجی Q 2.0 ست می‌گردد. Q 2.0 می‌تواند فرمان ارسال شده جهت روشن نمودن یک LED باشد. به محض روشن شدن LED مذکور، زمان مربوطه را ثبت می‌نمائیم. این زمان، مدت زمان اجرای ۲۰۰۰ سیکل برنامه است. بنابراین برای به دست آوردن سرعت اجرای یک سیکل برنامه، این عدد را بر ۲۰۰۰ تقسیم می‌کنیم. عدد به دست آمده در PLC مورد بحث حدود ۳ میلی ثانیه برای اجرای یک سیکل این برنامه می‌باشد. البته به دلیل تفاوت سرعت پردازنده‌ها در PLC‌های مختلف عدد به دست آمده در مورد هر PLC با PLC‌های دیگر متفاوت خواهد بود.

در اینجا به ذکر یک سؤال می‌پردازیم:

- دلیل استفاده از دستور S Q 2.0 در سطر هشتم برنامه 31 PB چیست؟ و آیا می‌توان به جای آن از دستور Q 2.0 = استفاده نمود یا خیر؟

پاسخ به این سؤال بسیار ساده است. از آنجایی که در دستور هم‌ارزی (=) وضعیت خروجی کاملاً به وضعیت ورودی وابسته است در صورتی که در این برنامه از دستور Q 2.0 = استفاده می‌شد بیت خروجی Q 2.0 تنها برای یک لحظه بسیار کوتاه، معادل با زمان لازم جهت مساوی شدن حاصل جمع عدد موجود در 0 DW و عدد ۲۰۰۰ روشن و سپس خاموش می‌شد. مسلماً در این حالت و با این تغییر وضعیت، با سرعت بالا نمی‌توان مدت زمان اجرای یک سیکل را به دست آورد زیرا این سرعت به قدری بالا است که چشم انسان قدرت تشخیص وضعیت روشن و خاموش بودن LED مذکور را ندارد. اما به دلیل استفاده از دستور S Q 2.0 جهت فعال نمودن خروجی، برای روشن شدن و روشن باقی ماندن LED تنها به لبه پالس نیاز است و در لحظه‌ای که محتویات موجود در 0 DW با عدد ۲۰۰۰ مساوی می‌شود بیت خروجی Q 2.0 فعال شده، LED مربوطه

در وضعیت روشن باقی می‌ماند.

- در شمارش تعداد سیکل‌های اجرای برنامه (تعداد ۲۰۰۰ سیکل برنامه) لازم است که شمارش حتماً از عدد صفر آغاز گردد بنابراین لازم است که با فشار دادن یک کلید، شمارش تعداد سیکل‌های برنامه را از صفر آغاز نموده، تا ۲۰۰۰ ادامه یابد. برای این مسأله چه برنامه‌ای پیشنهاد می‌کنید؟ همان‌گونه که در فصل سوم توضیح داده شد بلوک 1 OB ساختار برنامه استفاده کننده را مشخص می‌کند زیرا سیستم عامل در شروع هر سیکل برنامه به بلوک 1 OB رجوع می‌کند. بنابراین کلید معرف آغاز شمارش را در این بلوک تعریف می‌کنیم. برنامه خواسته شده در ادامه آمده است.

OB 1

```

SEGMENT 1          0000
0000      :C      DB  28
0001      :A      I    0.0
0002      :JC     PB  31
0003      :AN     I    0.0
0004      :JC     PB  40
0005      :BE

```

PB 31

```

SEGMENT 1          0000
0000      :L      DW   0
0001      :L      DW   1
0002      :+F
0003      :T      DW   0
0004      :L      DW   2
0005      :!=F
0006      :S      Q    2.0
0007      :BE

```

PB 40

```

SEGMENT 1          0000
0000      :L      KF +0
0002      :T      DW   0
0003      :R      Q    2.0
0004      :BE

```

DB28

```

0:      KF = +00000;
1:      KF = +00001;
2:      KF = +02000;
3:

```

روند اجرای برنامه به صورت زیر است:

در سطر اول بلوک 1 OB، بلوک اطلاعاتی 28 DB فعال می‌گردد تا تمامی اطلاعات از این بلوک خوانده شود. در صورتی که کلید استارت شمارش یا 0.0 I برابر "۱" شود PB 31 که همان برنامه قبلی است اجرا می‌شود. در این برنامه، شمارش از صفر شروع شده، تا 2000 ادامه می‌یابد. ولی اگر کلید استارت شمارش غیرفعال باشد PB 40 اجرا خواهد شد. در این برنامه ابتدا عدد صفر در 0 DW فرستاده می‌شود، در سطر سوم با ریست نمودن بیت خروجی 2.0 Q امکان انجام این کار برای چندین بار عملی شده است.

بنابراین ملاحظه می‌کنید که در هر دو صورت، شمارش از صفر آغاز شده، تا 2000 ادامه می‌یابد. بنابراین زمان به دست آمده تا روشن شدن LED مربوط به 2.0 Q زمان پردازش 2000 سیکل برنامه می‌باشد.

بنا به دلایل مختلفی از جمله خطای چشم در مشاهده زمان روشن شدن بیت 2.0 Q و عوامل دیگر ممکن است در محاسبه سرعت اجرای یک سیکل برنامه دچار اشتباه شده و نتوان به مقدار واقعی سرعت پردازنده در اجرای یک سیکل برنامه دست یافت. بنابراین در مثال بعد با استفاده از یک تایمر سرعت مذکور را به دست می‌آوریم.

مثال ۴-۶: با استفاده از یک تایمر، سرعت اجرای یک سیکل برنامه را به دست آورید.

این برنامه به صورت زیر نوشته می‌شود.

OB 1

```

SEGMENT 1      0000
0000      :C    DB 20
0001      :JU   PB 32
0002      :BE

```

PB 32

```

SEGMENT 1          0000
0000      :L      DW      0
0001      :L      DW      1
0002      :+F
0003      :T      DW      0
0004      :T      FW      2
0005      :A      I      0.0
0006      :L      KT      002.2
0008      :SP      T      1
0009      :A      F      3.0
000A      :A      T      1
000B      :CU      C      5
000C      :AN      I      0.0
000D      :R      C      5
000E      :BE
    
```

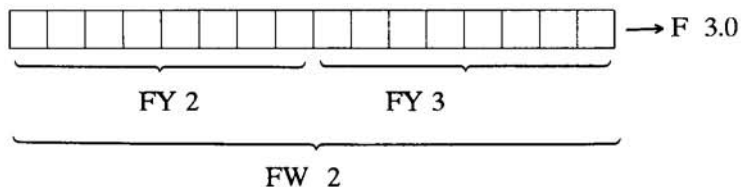
DB20

```

0:      KF = +00000;
1:      KF = +00001;
2:
    
```

در زیر اجرای برنامه تشریح شده است.

در 1 OB پس از معتبر شدن 20 DB پرش به 32 PB صورت می‌گیرد. در این بلوک محتویات 1 DW به 0 DW اضافه شده، حاصل مجدداً در 0 DW قرار می‌گیرد و همین حاصل جمع به 2 FW فرستاده می‌شود. به شکل زیر دقت کنید.



شکل ۴-۲: نمایش کلمه فلگ ۲ و بیت‌های مربوط به آن

به دلیل افزوده شدن به محتویات 0 DW و یا 2 FW در هر لحظه، سرعت تغییر وضعیت 3.0 F یعنی کم ارزش ترین بیت همیشه سرعت اجرای یک چرخه از برنامه و یا یک سیکل زمانی را مشخص می‌کند. حال در صورتی که کلید آغاز شمارش یعنی 0.0 I فعال باشد تایمر T1 که از فوئ SP نیز می‌باشد با مقدار 2.2 KT بارگذاری می‌شود. در سطر نهم، دهم و یازدهم برنامه جهت افزایش شمارنده C5 از سرعت تغییرات 3.0 F استفاده شده است. در سطر دوازدهم، دستور 0.0 I AN جهت اطمینان از شمارش از صفر به کار برده شده است.

با اجرای این برنامه پس از ۲ ثانیه (2.2 KT) تعداد سیکل‌های انجام شده به دست می‌آید. این تعداد به دست آمده همان محتویات شمارنده C5 می‌باشد. بنابراین می‌توان با تقسیم عدد مذکور بر ۲، تعداد سیکل‌های انجام شده در یک ثانیه را به دست آورد.

در مورد PLC مورد بحث، عدد به دست آمده در شمارنده، ۳۰۲ می‌باشد که با تقسیم آن بر ۲، عدد ۱۵۱ حاصل می‌شود. این حاصل بدان معنی است که در طول ۱ ثانیه ۱۵۱ سیکل از برنامه 32 PB انجام شده است. بنابراین مدت زمان اجرای یک سیکل از این برنامه $\frac{1}{151}$ ثانیه یا تقریباً ۶ میلی ثانیه است. همان‌گونه که قبلاً ذکر شد در مواردی که با فاصله‌های زمانی کوچک سروکار داریم و یا نیاز به دقت در محاسبه زمان باشد از تولرانس‌های ۰ و ۱ استفاده می‌کنیم. بنابراین به جای استفاده از 2.2 KT L در این برنامه می‌توان از دستور 20.1 KT L و یا 200.0 KT L نیز استفاده نمود. یکی از مواردی که در فرآیندهای صنعتی و خطوط تولید با آن روبرو می‌شویم محاسبه زمان دقیق در مورد انجام هر قسمت از خط یا فرآیند است. می‌توان همین برنامه را در انتهای برنامه کنترل هر قسمت از خط نوشته، برای هر قسمت سرعت یک سیکل زمانی را اندازه‌گیری نماییم. در این برنامه روش اندازه‌گیری زمان یک سیکل اجرای برنامه تغییر می‌کند و دیگر نیازی به طی ۲۰۰۰ سیکل و سپس تقسیم زمان به دست آمده بر ۲۰۰۰ نیست. همان‌گونه که در این برنامه ملاحظه کردید تعداد سیکل‌های پیموده شده در طی X ثانیه محاسبه می‌شود و سپس زمان اجرای یک سیکل برنامه از رابطه
$$\frac{X}{\text{تعداد سیکل‌های انجام شده در مدت X ثانیه}}$$
 به دست می‌آید.

همان‌گونه که می‌دانید خروجی یک شمارنده، عددی متغیر است. پس با استفاده از تعریف یک بلوک اطلاعاتی و مثالهای قبلی می‌توان برنامه کنترل چراغ راهنما را نوشت.

مثال ۴-۷: برنامه کنترل چراغ راهنمایی را با استفاده از تعریف یک DB بازنویسی کنید.

برنامه نوشته شده شامل چندین بلوک می‌باشد که در ادامه آمده است.

OB 1

```

SEGMENT 1      0000
0000      :C    DB    20
0001      :JU    PB    25
0002      :BE

```

PB 25

```

SEGMENT 1      0000
0000      :L    DW    0
0001      :L    DW    1
0002      :+F
0003      :T    DW    0
0004      :L    DW    1
0005      :!=F
0006      :S    Q     8.0
0007      :S    Q     8.1
0008      :R    Q     8.2
0009      :R    Q     8.3
000A      :R    Q     8.4
000B      :R    Q     8.5
000C      :L    DW    0
000D      :L    DW    2
000E      :!=F
000F      :S    Q     8.2
0010      :S    Q     8.3
0011      :R    Q     8.0
0012      :R    Q     8.1
0013      :R    Q     8.4
0014      :R    Q     8.5
0015      :L    DW    0
0016      :L    DW    3
0017      :!=F
0018      :S    Q     8.4
0019      :S    Q     8.5
001A      :R    Q     8.0
001B      :R    Q     8.1
001C      :R    Q     8.2
001D      :R    Q     8.3

```

ادامهٔ بلوک 25 PB:

```

001E      :L    DW    0
001F      :L    DW    4
0020      :!=F
0021      :O    I      0.0
0022      :JC   PB    26
0023      :BE

```

PB 26

```

SEGMENT 1      0000
0000      :L    KF    +0
0002      :T    DW    0
0003      :BE

```

DB20

```

0:      KF = +00000;
1:      KF = +00001;
2:      KF = +01500;
3:      KF = +02500;
4:      KF = +04000;
5:

```

روند اجرای برنامه به صورت زیر است:

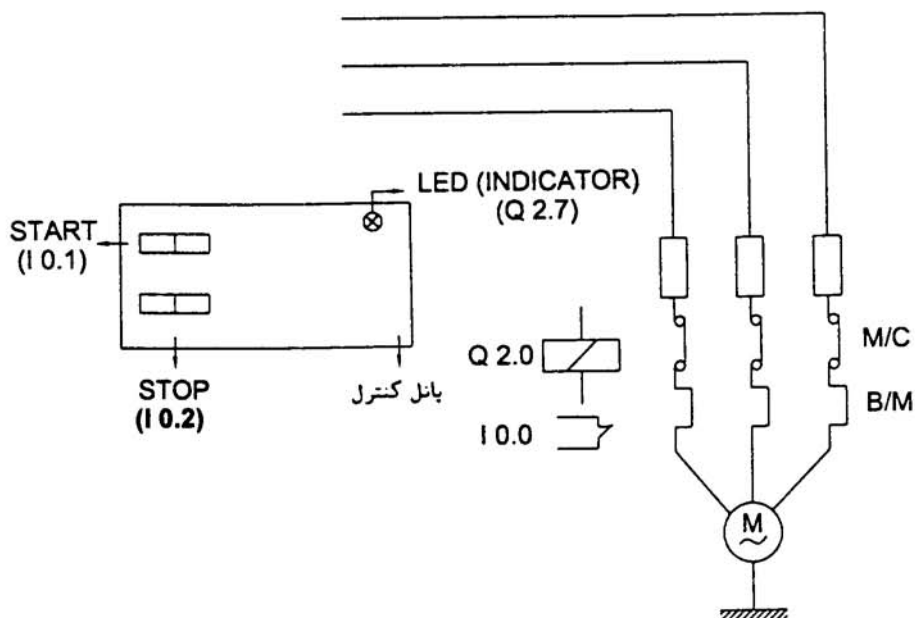
در بلوک 1 OB، ابتدا DB 20 فعال و سپس دستور پرش به 25 PB (بلوک برنامه‌ای حاوی برنامهٔ کنترلی) صادر گردیده است. در 25 PB ابتدا دو مقدار DW 0 و DW 1 یعنی KF 0000 و KF 0001 بارگذاری شده، حاصل جمع آنها در DW 0 قرار می‌گیرد. سپس این مقدار با مقدار موجود در DW 1 مقایسه می‌گردد و در صورت تساوی، دستور ست شدن چراغ قرمز مخصوص اتومبیل و چراغ سبز مخصوص عابر پیاده صادر می‌گردد. چهار دستور بعدی با توجه به شرایط ایمنی در برنامه گنجانده شده‌اند. در پایان این مرحله محتویات DW 0 و DW 1 برابر و DW 0 حاوی KF 1500 می‌باشد. در مرحلهٔ بعد محتویات همین DW 0 با DW 2 یعنی KF 2500 مقایسه می‌شود. در صورت مساوی بودن این دو مقدار، خروجی چراغهای زرد

مخصوص اتومبیل و عابر پیاده فعال و سایر چراغها غیرفعال می‌گردند. روند اجرای مرحله سوم یعنی ارسال فرمان به خروجی‌های چراغهای سبز مخصوص اتومبیل و قرمز مخصوص عابر پیاده نیز مانند مراحل قبل است. در حقیقت زمان روشن بودن چراغهای سه مرحله به ترتیب 1500 و $1000 = (2500-1500)$ و $1500 = (4000-2500)$ برابر سیکل زمانی می‌باشد.

در سطرهای انتهایی برنامه، دستور $O \quad I \quad 0.0$ دیده می‌شود دلیل استفاده از این دستور به شرح زیر است:

هنگامی که عدد $DW \quad 0$ به هر دلیلی بیشتر از ۴۰۰۰ باشد پردازنده باید محتوای $DW \quad 0$ را با عدد ۶۵۵۳۵ ($FFFF_H$) پر نموده، مجدداً صفر شود ولی با استفاده از $I \quad 0.0$ در هر لحظه امکان پرش به PB 26 میسر است و این کلید کنترل دستی در واقع برای شروع از عدد صفر می‌باشد. در این برنامه ملاحظه نمودید که به سادگی و بدون استفاده از تایمر، توانستیم ۳ تایمر تعریف کنیم. همان‌گونه که در فصل قبل نیز ذکر شد با اجرای چنین برنامه‌هایی می‌توان بدون استفاده از تایمر در برنامه، n تایمر را تعریف نمود.

مثال ۴-۸: شکل زیر را در نظر بگیرید. این شکل، نمایش دهنده یک مدار حفاظتی است که با استفاده از کنتاکتور بی‌متال (B/M) صحت عملکرد و وجود هر سه فاز در سیستم‌های ۳ فاز بررسی می‌گردد. در پانل کنترل، دو کلید START و STOP جهت روشن و خاموش کردن موتور تعبیه شده است. در این پانل یک نشان دهنده (Indicator) نیز جهت مشخص شدن وضعیت موتور وجود دارد. در این مثال قصد داریم برنامه‌ای بنویسیم که هرگاه کنتاکتور بی‌متال سالم و بی‌عیب و نقص و یا به عبارت دیگر موتور الکتریکی موجود با ۳ فاز در حال کار باشد LED مربوطه روشن و ثابت باشد ولی در صورت قطع این کنتاکتور، LED مذکور چشمک بزند. (توجه: کلید STOP در حالت عادی بسته و به صورت NC است.)



PB 41

برنامه کنترل مورد نظر در ادامه آمده است:

```

SEGMENT 1      0000
0000           :AN  F    6.0
0001           :L   KT  050.1
0003           :SE  T    1
0004           :A   T    1
0005           : =  F    6.0
0006           :L   T    1
0007           :T   FW   20
0008           :A   I    0.1
0009           :S   Q    2.0
000A           :AN  I    0.2
000B           :R   Q    2.0
000C           :A   I    0.0
000D           :A   Q    2.0
000E           : =  F    10.0
000F           :AN  I    0.0
0010           :A   Q    2.0
0011           :A   F    21.2
0012           : =  F    10.1
0013           :O   F    10.0
0014           :O   F    10.1
0015           : =  Q    2.7
0016           :BE

```

در برنامه فوق در صورتی که کلید START فعال شود خروجی $Q\ 2.0$ یعنی فرمان اتصال سه فاز موتور صادر شده، موتور به کار می‌افتد و در صورت فعال شدن کلید STOP موتور خاموش می‌شود. حال اگر کنتاکتور B/M سالم و یا به عبارت دیگر $I\ 0.0 = 1$ باشد، و در این حالت موتور نیز روشن بماند $F\ 10.0$ به حالت 1 ثابت می‌ماند ولی چنانچه کنتاکتور B/M قطع شده، موتور کار کند، فلگ $F\ 21.2$ که با فاصله زمانی خاص فعال و غیرفعال می‌گردد به $F\ 10.1$ نسبت داده می‌شود و حاصل ترکیب فصلی دو فلگ $F\ 10.0$ و $F\ 10.1$ به خروجی $F\ 2.7$ ارسال می‌گردد.

۴-۳- بلوک‌های تابع ساز (FB)

چنانچه به خاطر داشته باشید در فصل سوم در مورد این بلوک‌ها توضیحاتی ارائه شد. در تعریف این بلوک‌ها و کاربرد آنها یادآور می‌شویم که در کنترل پروسه‌ها و خطوط تولید گاه ممکن است توابعی به صورت مداوم و تکراری مورد استفاده قرار گیرند.

برای مثال ضرب دو عدد باینری بعضاً در کنترل فرآیندها مکرراً مورد استفاده قرار می‌گیرد، یا اینکه فرض کنید در یک خط تولید، عملیات پرس و خم‌کاری ورقه‌های فولادی به صورت تکراری انجام می‌شود. برای انجام این عملیات تنها لازم است برنامه این پروسه را یک بار در FB تعریف و سپس در صورت نیاز، بلوک مذکور فراخوانی شده، اطلاعات لازم جهت انجام عملیات موردنظر به این بلوک تابع ساز داده شود.

سازندگان PLC معمولاً برخی از FB‌هایی را که توسط استفاده‌کنندگان و کاربران PLC مورد نیاز می‌باشد به صورت بسته‌های نرم‌افزاری نوشته و به همراه دفترچه‌های راهنمای هر نرم‌افزار در اختیار کاربران قرار می‌دهند. این دفترچه راهنما معمولاً شامل اطلاعاتی در مورد تعداد ورودی‌ها، خروجی‌ها، چگونگی وارد نمودن اطلاعات و ... می‌باشد. هر بلوک FB از دو بخش اصلی زیر تشکیل شده است.

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

۲- بدنه بلوک (Block Body) که شامل توابع و دستوراتی است که می‌بایست در FB قرار گیرند. علاوه بر دستورات معمول، گروهی از دستورها که دستورالعمل‌های تکمیلی نامیده می‌شوند در این بلوک مورد استفاده قرار می‌گیرند. به طور کلی می‌توان FB‌ها را به دو دسته زیر تقسیم نمود:

۱- بلوک تابع ساز استاندارد (Standard FB)

این بلوک‌ها شامل بلوک‌هایی هستند که عملیات منطقی نظیر ضرب، تقسیم و ... در آنها تعریف شده، توسط سازندگان PLC به همراه دفترچه راهنما در اختیار کاربر قرار می‌گیرد.

۲- بلوک تابع ساز انتسابی (Assignable FB)

در اجرای این بلوک‌ها می‌توان عملوندها یعنی ورودی‌ها، خروجی‌ها و ... را در هر پروسه تغییر داد و عملوندهای جدیدی تعریف نمود.

در ادامه بحث در مورد برنامه‌نویسی FBها، مثالهایی از توابع استاندارد و انتسابی ارائه خواهد شد. در PLCهای زیمنس، FBهای استاندارد با شماره‌های خاصی وجود دارند به عنوان مثال، FB 242 یک FB استاندارد است که جهت ضرب دو عدد به صورت کلمه‌ای (۱۶ بیتی) به کار می‌رود. در ابتدای برنامه‌نویسی این بلوک‌ها، PLC از کاربر نام بلوک را می‌پرسد. استفاده کننده می‌تواند به دلخواه نامی برای این بلوک انتخاب نماید، سپس PLC پارامترهای ورودی یعنی دو عددی که قرار است عمل ضرب بر روی آنها انجام شود و پارامترهای خروجی یعنی خروجی‌ای که باید حاصل ضرب به آن فرستاده شود را از کاربر می‌پرسد. در ادامه، یک FB 242 که یک بلوک استاندارد می‌باشد برنامه‌نویسی شده است. جهت اجرای این برنامه نیاز به برنامه‌نویسی یک بلوک PB می‌باشد. در ادامه، برنامه‌نویسی این بلوک را ملاحظه می‌کنید.

PB 40

```

SEGMENT 1          0000
0000      :JU  FB 242
0001 NAME :MUL:16
0002 Z1    :   IW  16
0003 Z2    :   IW  22
0004 Z3=0  :    Q  10.0
0005 Z32   :   QW  14
0006 Z31   :   QW   8
0007      :BE

```

همان‌گونه که گفته شد FBها فقط به روش STL قابل برنامه‌نویسی هستند. بلوک FB 242 را از نظر شماتیکی در ادامه نمایش داده شده است.

```

      FB 242
      +-----+
      !      MUL: 16      !
IW 16  --! Z1              Z3=0! -- Q 10.0
IW 22  --! Z2              Z32 ! -- QW 14
      !                    Z31 ! -- QW 8
      +-----+

```

شکل ۴-۳: نمایش شماتیکی بلوک استاندارد FB 242

اکنون به توضیح در مورد پارامترهای این بلوک می‌پردازیم:

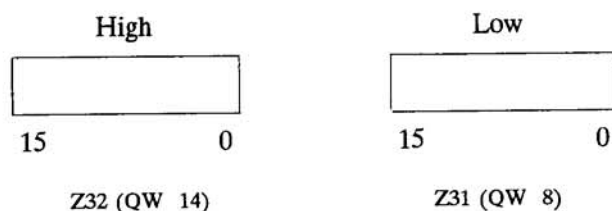
Z1: عدد اول (IW 16). این عدد به صورت کلمه‌ای به PLC داده می‌شود.

Z2: عدد دوم (IW 21). این عدد به صورت کلمه‌ای به PLC داده می‌شود.

Z3: در صورتی که نتیجه حاصلضرب دو عدد صفر شود یا به عبارت دیگر لااقل یکی از دو عدد Z1 یا Z2 صفر باشد بیت RLO "۱" شده، به یک بیت خروجی (Q 10.0) ارسال می‌شود.

Z31: نتیجه حاصلضرب دو عدد Z1 و Z2 در یک کلمه خروجی (QW 8) ظاهر می‌شود.

Z32: در صورتی که نتیجه حاصلضرب از یک کلمه یا ۱۶ بیت بیشتر شده باشد بیت‌های بعدی در یک کلمه خروجی دیگر (QW 14) قرار می‌گیرد. با این تعاریف می‌توان گفت که نتیجه حاصلضرب در دو کلمه خروجی نمایش داده می‌شود.



همان‌گونه که ملاحظه می‌شود حاصلضرب دو عدد Z1 و Z2 در دو کلمه خروجی نشان داده می‌شود که این دو کلمه الزاماً شماره‌های پی‌درپی ندارند. Z31 یا QW 8 شامل بیت‌های با ارزش پائین‌تر و Z32 یا QW 14 حاوی بیت‌هایی با ارزش بالاتر می‌باشد.

مثال ۴-۹: به کمک بلوک FB 242 حاصلضرب دو عدد $14_{(10)}$ و $25_{(10)}$ را به دست آورید.

قبل از هر کار، دو عدد داده شده را در مبنای ۲ تبدیل کرده، آنها را به صورت کلمه ۱۶ بیتی

نمایش می‌دهیم.

$$14_{(10)} = 1110_{(2)} = 0000000000001110 \rightarrow IW \ 16$$

$$25_{(10)} = 11001_{(2)} = 00000000000011001 \rightarrow IW \ 21$$

در صورت وارد نمودن این اعداد به PLC، خروجی‌های FB 242 به صورت زیر خواهند بود:

به دلیل اینکه حاصلضرب، صفر نشده است مقدار بیت خروجی 10.0 برابر "۰" می‌باشد.

$$Z3 = 0 \rightarrow Q \ 10.0$$

$$Z31 = 000000001011110_{(2)} \rightarrow QW \ 8$$

$$= 1 \times 2^6 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 = 350_{(10)}$$

$$Z32 = 000000000000000_{(2)} \rightarrow QW \ 14$$

از آنجایی که حاصلضرب دو عدد مذکور بیش از ۱۶ بیت نشده است مقدار ذخیره شده در Z32، صفر خواهد بود.

حال اگر بخواهیم حاصلضرب دو عدد $5698_{(10)}$ و $3265_{(10)}$ را به دست آوریم همان روند قبلی را طی کرده، خواهیم داشت:

$$5698_{(10)} = 0001011001000010_{(2)} \rightarrow IW \ 16$$

$$3265_{(10)} = 0000110011000001_{(2)} \rightarrow IW \ 21$$

خروجی‌ها به صورت زیر می‌باشند:

$$Z3 = 0$$

$$Z31 = 110111111000010_{(2)} \rightarrow QW \ 8$$

$$Z32 = 0000000100011011_{(2)} \rightarrow QW \ 14$$

$$نتیجه حاصلضرب = Z32 \ Z31 = 0000000100011011 \ 110111111000010$$

$$= 18603970_{(10)}$$

همان‌گونه که ملاحظه می‌کنید نتیجه حاصلضرب بسیار بزرگ بوده، تبدیل این عدد در مبنای ۲ به مبنای ۱۰ وقت‌گیر و مشکل است. برای حل این مشکل دو کلمه موجود در خروجی (۳۲ بیت) را به ۸ دسته ۴ بیتی تقسیم می‌نماییم که هر دسته معرف عددی در مبنای ۱۶ می‌باشد. سپس این عدد را بر مبنای ۱۶ به دست آورده، در این مرحله تبدیل مبنا را انجام می‌دهیم.

16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
0000	0001	0001	1011	1101	1111	1100	0010
0	1	1	B	D	F	C	2

۸ دسته ۴ بیتی

بنابراین عدد حاصل در مبنای ۱۶ به صورت $011BDFC2_H$ می‌باشد که تبدیل آن به مبنای ۱۰ این‌گونه است:

$$011BDFC2 = 0 \times 16^7 + 1 \times 16^6 + 1 \times 16^5 + 11 \times 16^4 + 13 \times 16^3 + 15 \times 16^2 + 12 \times 16^1 + 2 \times 16^0$$

$$= 18603970_{(10)}$$

اکنون به ذکر مثالی در مورد Assignable FB می‌پردازیم.

در این مثال به بررسی 20 FB که در آن تابع منطقی "XOR" تعریف شده است خواهیم پرداخت. حاصل تابع منطقی XOR دارای دو ورودی، در شرایطی "۱" خواهد بود که ورودی‌ها مساوی نباشند یا به عبارت دیگر یکی از ورودی‌ها "۰" و دیگری "۱" باشد.

در صورتی که بخواهیم حاصل تابع منطقی XOR دو ورودی 0.1 I و 0.0 I را در Q 2.0 قرار

دهیم برنامه PB 10 را خواهیم داشت:

PB 10

```

SEGMENT 1          0000
0000      :A      I      0.0
0001      :AN     I      0.1
0002      :O
0003      :AN     I      0.0
0004      :A      I      0.1
0005      :      Q      2.0
0006      :BE

```

PB 10

```

SEGMENT 1          0000
!
! I 0.0      I 0.1      Q 2.0
+---] [---+---] / [---+-----+---( )-!
!
! I 0.0      I 0.1      !
+---] / [---+---] [---+      :BE
!

```

مثال ۴-۱۰: با استفاده از تعریف تابع منطقی XOR در بلوک تابع ساز، برنامه‌ای بنویسید که هرگاه تنها یکی از دو موتور با شماره‌های ۱ و ۲ روشن باشد LED موجود بر روی پانل کنترل روشن شود. برای نوشتن این برنامه به دو بلوک FB و PB نیاز داریم، که باید در بلوک تابع XOR را به همراه پارامترهای موردنظر تعریف کنیم. این برنامه در ادامه آمده است.

PB 3

```

SEGMENT 1          0000
0000      :JU  FB  20
0001 NAME :XOR
0002 MOT1 :      I      0.0
0003 MOT2 :      I      0.1
0004 LED  :      Q      2.0
0005      :BE

```

FB 20

```

SEGMENT 1          0000
NAME :XOR
DECL :MOT1          I/Q/D/B/T/C: I  BI/BY/W/D: BI
DECL :MOT2          I/Q/D/B/T/C: I  BI/BY/W/D: BI
DECL :LED           I/Q/D/B/T/C: Q  BI/BY/W/D: BI

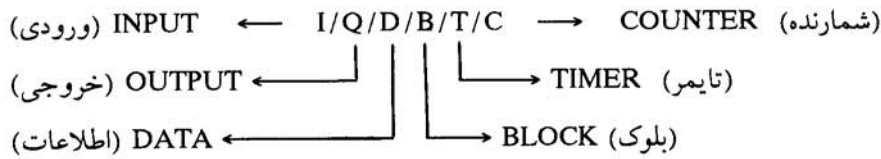
000E      :A      =MOT1
000F      :AN     =MOT2
0010      :O
0011      :AN     =MOT1
0012      :A      =MOT2
0013      :      =LED
0014      :BE

```

روند وارد نمودن اطلاعات به شرح زیر است:

در ابتدای بلوک FB، PLC نام برنامه را از استفاده کننده می‌پرسد. کاربر می‌تواند به دلخواه نام مناسب با عملیات انجام شده مثلاً XOR را وارد کند. در مرحله بعد، نام پارامترها، نوع و ماهیت پارامترها و همچنین نحوه ارسال و یا دریافت پارامترها از کاربر خواسته می‌شود. در مورد اولین ورودی، کاربر پارامتر 1 MOT را معرفی نموده است. سپس در همین سطر، PLC از استفاده کننده

نوع و ماهیت این پارامتر را سؤال می‌کند و استفاده‌کننده با انتخاب یکی از موارد I/Q/D/B/T/C نوع پارامتر را مشخص می‌کند.



بنابراین با توجه به اینکه پارامتر 1 MOT یکی از ورودی‌های سیستم است در مورد نوع پارامتر، "I" انتخاب می‌گردد. سپس در ادامه همین سطر چگونگی ارسال و یا دریافت پارامترها از استفاده‌کننده خواسته می‌شود. کاربر با انتخاب یکی از حالات BI/BY/W/D نحوه ارسال و یا دریافت پارامترها را به PLC وارد می‌کند.



با توجه به اینکه پارامتر 1 MOT ورودی سیستم می‌باشد و وضعیت روشن و خاموش بودن موتور توسط یک بیت قابل ارسال به ورودی PLC است، کاربر با انتخاب BI نحوه ارسال این ورودی را به PLC به صورت بیتی وارد می‌کند.

در سطر بعدی همین عمل در مورد پارامتر دوم انجام می‌گیرد. در این سطر کاربر نام، نوع و نحوه ارسال پارامتر را به ترتیب 2 MOT، I و BI انتخاب می‌نماید.

در سطر بعدی نام پارامتر سوم یعنی LED وارد می‌شود. در اینجا به دلیل اینکه روشن و یا خاموش شدن LED به صورت یک مقدار خروجی ظاهر می‌شود در مرحله انتخاب نوع پارامتر، Q انتخاب شده است. نحوه دریافت این پارامتر از PLC به صورت بیتی با انتخاب BI انجام می‌گیرد. در سطرهای بعدی، برنامه‌نویسی در FB (Block Body) آغاز می‌شود. در FBها می‌توان به جای استفاده از عملوندها از نام پارامترهای در نظر گرفته شده در بلوک استفاده نمود. برنامه‌نویسی در این مرحله تقریباً نظیر برنامه‌نویسی در PBها می‌باشد با این تفاوت که در اینجا از نام پارامترها به عنوان عملوند استفاده شده است. مثلاً به جای دستور $A \ I \ 0.0$ از دستور $A = MOT \ 1$ استفاده می‌شود.

پس از برنامه‌نویسی بلوک FB باید در بلوک PB پارامترهای استفاده شده در FB را به

عملوندهای مورد نظر نسبت دهیم.

PB 3

```

SEGMENT 1          0000
0000      :JU  FB  20
0001 NAME :XOR
0002 MOT1 :    I    0.0
0003 MOT2 :    I    0.1
0004 LED  :    Q    2.0
0005      :BE

```

واضح است که برای اجرای PB نوشته شده نیاز به تعریف 1 OB به صورت زیر می باشد:

OB 1

SEGMENT 1 :

: JU PB 3

: BE

در بلوک 3 PB دستور پرش به 20 FB و در حقیقت دستور اجرای این بلوک صادر می گردد. سپس در همین بلوک یعنی 3 PB نام بلوک FB و همچنین عملوندهای متناظر با پارامترهای وارد شده در 20 FB از کاربر خواسته می شود. مثلاً سطر 0.0 I : MOT 1 معرف آن است که عملوند متناظر با پارامتر 1 MOT، 0.0 I می باشد.

در بلوک 1 OB نیز با دستور 3 PB JU در حقیقت فرمان اجرای 3 PB و یا به عبارت دیگر 20 FB را به PLC صادر می کنیم.

از آنجایی که این بلوک های تابع ساز، انتسابی می باشند می توان در هر بار صدا زدن بلوک، عملوندهای متناظر با پارامترهای تعریف شده در بلوک FB را تغییر داد. حتی می توان در یک بلوک 3 PB چندین بار بلوک FB خاصی را صدا زده، هر بار عملوندهای آن را تغییر داد. برنامه نوشته شده در 6 PB این مطلب را روشن می سازد.

PB 6

```

SEGMENT 1      0000
0000      :JU   FB   20
0001 NAME :XOR
0002 MOT1 :      I    0.0
0003 MOT2 :      I    0.1
0004 LED  :      Q    2.0
0005      :JU   FB   20
0006 NAME :XOR
0007 MOT1 :      I    0.2
0008 MOT2 :      I    0.3
0009 LED  :      Q    3.4
000A      :BE

```

۴-۴- دستورات تکمیلی (Supplementary)

در PLC های زیرمنس دستورالعمل‌هایی به نام دستورات تکمیلی یا Supplementary وجود دارند که استفاده از آنها فقط در FB ها مجاز است. در این قسمت به ذکر برخی از این دستورات مهم می‌پردازیم.

۴-۴-۱- دستور AW^۱

فرض کنید که در برنامه‌ای قصد داریم به گونه‌ای عمل شود که هر بیت خروجی حاصل ترکیب عطفی (AND) دو ورودی متناظر با خود باشد. در صورتی که ورودی‌ها در دو کلمه ورودی (IW) قرار گرفته باشند و خروجی‌های متناظر با حاصل ترکیب عطفی دو ورودی در کلمه خروجی (QW) باشند برای نسبت دادن حاصل ترکیب عطفی بیت‌های متناظر به بیت‌های خروجی، برنامه‌ای با تعداد سطرهای زیاد مورد نیاز است زیرا برای نسبت دادن ترکیب عطفی دو بیت ورودی به یک بیت خروجی به ۳ سطر برنامه احتیاج می‌باشد و از آنجایی که هر کلمه شامل ۱۶ بیت است برای تعریف چنین برنامه‌ای ناچار به برنامه‌نویسی یک بلوک با حدود ۵۰ سطر می‌باشیم.

1 - AND WORD

دستور AW این عمل را خلاصه نموده، حاصل ترکیب عطفی دو کلمه ورودی را در یک کلمه خروجی قرار می‌دهد. در این عمل هر بیت خروجی هم‌ارز با حاصل ترکیب عطفی دو بیت متناظر در دو کلمه ورودی است. فرض کنید کلمات ورودی IW 16 و IW 22 و کلمه خروجی QW 8 باشد.

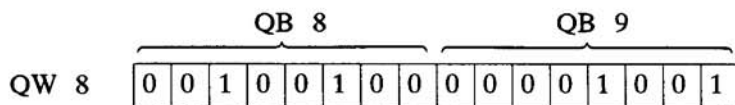
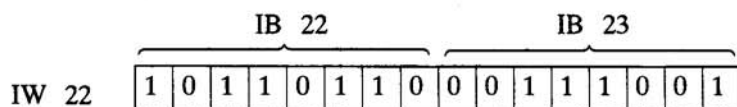
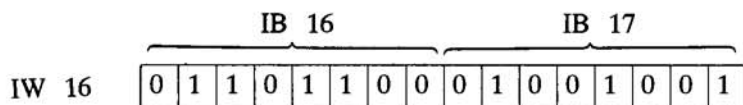
در برنامه زیر دستور AW بر روی ورودی‌های مذکور انجام و حاصل ترکیب این دو کلمه به خروجی QW 8 نسبت داده شده است:

FB 10

SEGMENT 1 0000
NAME :AND

0005 :L IW 16
0006 :L IW 22
0007 :AW
0008 :T QW 8
0009 :BE

به عنوان مثال در صورتی که کلمات ورودی IW 16 و IW 21 شامل بیت‌های نشان داده شده باشند خروجی QW 8 به صورت زیر است:



همان‌گونه که ملاحظه می‌شود هر بیت کلمه خروجی، هم‌ارز با ترکیب عطفی بیت‌های متناظر در کلمات ورودی است. یعنی به عنوان مثال: $23.3 \cdot I \ 17.3 = I \ 9.3$. لازم به ذکر است که این‌گونه دستورات در زبان S5 تنها منحصر به استفاده در FBها بوده، در برخی PLCها و زبانهای برنامه‌نویسی دیگر می‌توان این دستورات را در بلوک‌های OB، PB و FB استفاده نمود.

ملاحظه نمودید که دستور AW در خلاصه نمودن برنامه تا چه اندازه به برنامه‌نویس کمک می‌نماید. در صورتی که از این دستور استفاده نمی‌شد باید یک بلوک مطابق بلوک PB 4 برنامه‌نویسی می‌کردیم:

PB 4

SEGMENT	1		0000
0000	:A	I	17.0
0001	:A	I	23.0
0002	:=	Q	9.0
0003	:A	I	17.1
0004	:A	I	23.1
0005	:=	Q	9.1
0006	:	:	
0007	:	:	
0008	:A	I	16.0
0009	:A	I	22.0
000A	:=	Q	8.0
000B	:	:	
000C	:	:	
000D	:A	I	16.7
000E	:A	I	22.7
000F	:=	Q	8.7
0010	:BE		

۴-۴-۲- کاربرد عملی دستور AW

در برخی از فرایندها لازم است که بعضی از بیت‌های ورودی پوشانده و تنها برخی از آنها در

خروجی ظاهر شوند. به این عمل ماسک^۱ کردن می‌گویند. چگونگی انجام این عمل را در مثال زیر می‌بینید.

مثال ۴-۱۱: برنامه‌ای بنویسید که تنها بیت‌های با شماره فرد یک کلمه ورودی را در خروجی ظاهر کند یا به عبارت دیگر بیت‌های با شماره زوج را بپوشاند.

برای نوشتن این برنامه لازم است برنامه‌نویس، یک ورودی تعریف نماید به طوری که بیت‌های فرد آن "۱" و بیت‌های زوج آن "۰" باشد. سپس حاصل ترکیب عطفی این ورودی را با ورودی اصلی، به عنوان مثال کلمه ورودی ۱۴ (IW 14)، به دست آورده، عدد حاصل را به یک کلمه خروجی منتقل نماید. این برنامه در FB 11 نوشته شده است.

FB 11

```
SEGMENT 1          0000
NAME :AND1
```

```
0005      :L   KH AAAA
0007      :L   IW  14
0008      :AW
0009      :T   QW   2
000A      :BE
```

KH AAAA

1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

IW 14

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

QW 2

1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

همان‌طور که دیده می‌شود در تمامی بیت‌های فرد خروجی "۱" و در بیت‌های زوج آن "۰" وجود دارد. هر بیت کلمه خروجی حاصل ترکیب عطفی (AND) بیت‌های متناظر در دو کلمه تعریف شده در ورودی می‌باشد.

۴-۴-۳- دستور OW^۱

عملکرد این دستور نیز مانند دستور AW است با این تفاوت که در اینجا حاصل ترکیب فصلی دو کلمه ورودی، در یک کلمه خروجی ظاهر می‌گردد، به صورتی که هر بیت کلمه خروجی حاصل ترکیب فصلی بیت‌های متناظر در دو کلمه ورودی است. در بلوک زیر برنامه‌ای با استفاده از این دستور نوشته شده است:

FB 12

```

SEGMENT 1          0000
NAME :OR

0005      :L      IW  16
0006      :L      IW  22
0007      :OW
0008      :T      QW   8
0009      :BE

```

با استفاده از دستور OW می‌توانیم ترکیبی از ورودی‌ها را در خروجی ایجاد نمائیم، که به این عمل ترکیب^۲ ورودی‌ها نیز گویند.

۴-۴-۳- دستور XOW^۳

در این دستور حاصل تابع منطقی XOR دو کلمه ورودی در یک کلمه خروجی ظاهر می‌شود. در بلوک FB 13 چگونگی کاربرد این دستور آمده است:

FB 13

```

SEGMENT 1          0000
NAME :XR

0005      :L      IW  16
0006      :L      IW  22
0007      :XOW
0008      :T      QW   8
0009      :BE

```

به کمک این دستور می توان اختلاف بین ورودی ها را در خروجی آشکار ساخت که به این عمل Detect نمودن می گویند. به این ترتیب که هرگاه دو بیت ورودی از لحاظ ارزش مخالف یکدیگر باشند حاصل خروجی آنها "۱" بوده، در غیر این صورت حاصل خروجی متناظر با آنها "۰" می باشد.

۴-۴-۵- دستور CFW^۱

با استفاده از این دستور می توان مکمل ۱ یک کلمه ورودی را در یک کلمه خروجی قرار داد. در این دستور برخلاف دستورات قبلی تنها از یک کلمه در ورودی استفاده می شود. همان گونه که می دانید مکمل "۰"، "۱" و مکمل "۱"، "۰" می باشد، بنابراین برای به دست آوردن مکمل ۱ یک کلمه ورودی کافی است ارزش بیت های آن را عکس نمود. در ادامه، یک برنامه با استفاده از این دستور نوشته شده است.

FB 14

SEGMENT 1 0000
NAME : FIRS

0005 :L IW 16
0006 :CFW
0007 :T QW 8
0008 :BE

IW 16

1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

QW 8

0	1	0	0	1	1	1	0	1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

در این برنامه مکمل ۱ بیت های کلمه ورودی ۱۶ در کلمه خروجی ۸ قرار گرفته اند یا به عبارت دیگر ارزش بیت های ورودی عکس شده است.

۴-۴-۶- دستور CSW^۱

به کمک این دستور می‌توان مکمل ۲ یک کلمه ورودی را در یک کلمه خروجی قرار داد. در این دستور نیز تنها از یک کلمه ورودی استفاده می‌شود. همان‌گونه که می‌دانید برای به دست آوردن مکمل ۲ یک عدد کافی است مکمل ۱ آن را به دست آورده، یک عدد ۱ به آن اضافه نمود. در زیر برنامه‌ای شامل این دستور آورده شده است:

FB 15

```
SEGMENT 1          0000
NAME :TWOS
```

```
0005      :L      IW   16
0006      :CSW
0007      :T      QW    8
0008      :BE
```

در صورتی که در 16 IW عدد $AAAA_{(H)}$ قرار گرفته باشد مکمل ۲ آن به صورت زیر به دست می‌آید:

$$AAAA_{(H)} \xrightarrow{\text{مکمل ۱}} 5555_{(H)} +$$

$$\xrightarrow{1} 5556_{(H)} \longrightarrow \text{این عدد در 8 QW قرار می‌گیرد}$$

IW 16

1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

A A A A

QW 8

0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

5 5 5 6

۴-۴-۷- دستور SLW^۱

به کمک این دستور می‌توان بیت‌های یک کلمه ورودی را به سمت چپ حرکت (شیفت) داد. تعداد حرکت بیت‌ها توسط عددی که بعد از دستور SLW قرار گرفته مشخص می‌شود. این عدد می‌تواند مقادیر بین ۰ تا ۱۵ را اختیار کند. بلوک 16 FB را در نظر بگیرید:

FB 16

```

SEGMENT 1          0000
NAME :SHIFT

0005      :L      IW  22
0006      :SLW      3
0007      :T      QW  14
0008      :BE

```

در این بلوک FB، کلمه ورودی IW 22 در انبارک بارگذاری شده، سپس تک‌تک بیت‌های این کلمه به اندازه ۳ بیت به سمت چپ حرکت داده می‌شوند. به هنگام حرکت بیت‌های با ارزش پائین‌تر به سمت چپ، به تعداد بیت‌های شیفت داده شده بیت "۰" از طرف راست وارد انبارک می‌شود و در این حرکت بیت‌ها، بیت‌های با ارزش بالاتر حذف می‌گردند. بنابراین در برنامه فوق پس از اجرای برنامه، تمام بیت‌های IW 22 به اندازه ۳ بیت به سمت چپ شیفت داده شده، ۳ بیت "۰" به سمت راست انبارک وارد می‌شوند. واضح است که در این حرکت بیت‌ها، تعداد ۳ بیت از بیت‌های با ارزش بالاتر کلمه ورودی ۲۲ حذف می‌شوند. در ادامه، محتویات اولیه IW 22 و همچنین محتویات QW 14 پس از انجام شیفت نشان داده شده است.

IW 22

1	1	0	1	0	1	1	1	0	1	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

این ۳ بیت با ارزش بالاتر حذف می‌شوند

QW 14

1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

۳ بیت "۰" وارد انبارک می‌شوند

توجه داشته باشید که در اجرای دستور 3 SLW، حرکت بیت‌ها به سمت چپ انجام گرفته است و در این حرکت، چرخش بیت‌های با ارزش بالاتر به بیت‌های با ارزش پایین‌تر وجود ندارد.

مثال ۴-۱۲: عدد $(10)_2$ را به عنوان کلمه ورودی در 22 IW در نظر گرفته، دستورات 1 SLW، 2 SLW و 3 SLW را در مورد آن اعمال کنید. پس از مقایسه عدد حاصل در هر قسمت با عدد اولیه چه نتیجه‌ای می‌گیرید؟

ابتدا عدد $2_{(10)}$ را به مبنای دو تبدیل نموده، آن را به صورت کلمه نشان می‌دهیم.

[illegible]

IW 22

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $\lambda_{\sigma(r)} = 2$

SLW 1

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $100_{(Y)} = 4$

SLW 2

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $\lambda \dots_{(Y)} = \lambda$

SLW 3

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 10000₍₇₎ = 16

همان‌گونه که دیده می‌شود با اجرای دستور 1 SLW بر روی 22 IW که حاوی عدد $2_{(1,0)}$ می‌باشد، عدد $4_{(1,0)}$ یعنی ۲ برابر مقدار اولیه به دست می‌آید. با اجرای دستور 2 SLW، حاصل $8_{(1,0)}$ یعنی 2^2 برابر عدد اول یعنی محتویات اولیه 22 IW به دست می‌آید و در دستور 3 SLW نتیجه حاصل شده 2^3 برابر عدد اولیه خواهد بود. پس نتیجه می‌گیریم که با هر بار حرکت بیت‌ها به سمت چپ، عدد به دست آمده ۲ برابر می‌شود. بنابراین دستور 5 SLW عدد اولیه موجود در 22 IW را در $32 (2^5)$ ضرب می‌کند. اما جواب این حاصل ضرب تنها هنگامی صحیح خواهد بود که عدد اول موجود در 22 IW به هنگام شیفِت به سمت چپ، بیت‌های با ارزش بالای خود که حاوی بیت "۱" می‌باشد را از دست ندهد. پس می‌توان گفت که برای ضرب اعداد در توانهایی از ۲ به عنوان مثال ۲، ۴، ۸، ۱۶ و ... از دستورات 1 SLW، 2 SLW، 3 SLW، 4 SLW و ... استفاده می‌شود به شرطی که پس از شیفِت، بیت‌های با ارزش بالا با محتوای "۱" حذف نگردند.

۴-۴-۸- دستور SRW^۱

عملکرد این دستور دقیقاً بر عکس دستور SLW است به این صورت که بیت‌های کلمه ورودی را به سمت راست حرکت می‌دهد. تعداد انتقال بیت‌ها از ۰ تا ۱۵ قابل تغییر است. بلوک ۱۷ FB را در نظر بگیرید:

FB 17

```

SEGMENT 1          0000
NAME : RIGHT

0005      :L      IW   22
0006      :SRW     4
0007      :T      QW   14
0008      :BE

```

در برنامه فوق پس از بارگذاری کلمه ورودی ۲۲ در انبارک، تک تک بیت‌ها به اندازه ۴ بیت به سمت راست حرکت داده می‌شوند. در سمت چپ یا به عبارت دیگر بیت‌های با ارزش بالاتر ۴ بیت "۰" وارد انبارک شده، ۴ بیت از بیت‌های با ارزش پائین‌تر نیز حذف می‌گردند. در این حالت نیز چرخش بیت‌ها وجود ندارد. در ادامه محتویات اولیه ۲۲ IW و همچنین محتویات ۱۴ QW پس از انجام عمل شیفت نشان داده شده است.

این ۴ بیت حذف می‌شوند

IW 22

1	0	0	1	0	1	0	0	0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

QW 14

0	0	0	0	1	0	0	1	0	1	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

۴ بیت "۰" وارد انبارک می‌شوند

مثال ۴-۱۳: عدد $۱۹۲_{(۱۰)}$ را به عنوان کلمه ورودی ۲۲ در نظر گرفته، دستورات ۱ SRW، ۲ SRW و ۳ SRW را در مورد آن اعمال نمایید و پس از مقایسه اعداد به دست آمده با عدد اول نتیجه را بیان کنید.

ابتدا عدد $۱۹۲_{(۱۰)}$ را به مبنای ۲ تبدیل نموده، آن را به صورت کلمه نشان می‌دهیم:

$$۱۹۲_{(۱۰)} = ۱۱۰۰۰۰۰۰_{(۲)} = ۰۰۰۰۰۰۰۰۱۱۰۰۰۰۰۰_{(۲)} \rightarrow IW\ 22$$

$$IW\ 22 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} ۱۱۰۰۰۰۰۰_{(۲)} = ۱۹۲_{(۱۰)}$$

$$SRW\ 1 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline \end{array} ۱۱۰۰۰۰۰۰_{(۲)} = ۹۶_{(۱۰)}$$

$$SRW\ 2 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} ۱۱۰۰۰۰۰_{(۲)} = ۴۸_{(۱۰)}$$

$$SRW\ 3 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} ۱۱۰۰۰۰_{(۲)} = ۲۴_{(۱۰)}$$

با مقایسه اعداد به دست آمده در هر قسمت با عدد اول یعنی $۱۹۲_{(۱۰)}$ ملاحظه می‌شود که با اجرای هر بار دستور SRW نصف عدد قبلی به دست می‌آید. با اجرای دستور SRW 1 عدد اول بر ۲^1 تقسیم شده است. با اجرای دستور SRW 2 عدد به دست آمده برابر $\frac{۱}{۴}$ (عدد اول است، با اجرای دستور SRW 3 به $\frac{۱}{۸}$ (عدد اول دست خواهیم یافت. مثال ۴-۱۴: عدد $۲۱۵_{(۱۰)}$ را در نظر گرفته، دستور SRW 1 را در مورد آن اعمال نمائید. پس از مقایسه عدد به دست آمده با عدد اول چه نتیجه‌ای می‌گیرید؟

ابتدا عدد $۲۱۵_{(۱۰)}$ را به مبنای ۲ برده، آن را به صورت کلمه در می‌آوریم:

$$۲۱۵_{(۱۰)} = ۱۱۰۱۰۱۱۱_{(۲)} = ۰۰۰۰۰۰۰۰۱۱۰۱۰۱۱۱_{(۲)} \rightarrow IW\ 22$$

$$IW\ 22 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} ۱۱۰۱۰۱۱۱_{(۲)} = ۲۱۵_{(۱۰)}$$

$$SRW\ 1 \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array} ۱۱۰۱۰۱۱_{(۲)} = ۱۰۷_{(۱۰)}$$

$$\neq \frac{۲۱۵}{۲}$$

با مقایسه عدد به دست آمده با عدد اول ملاحظه می‌کنید که به دلیل فرد بودن آن، عدد حاصل برابر با نصف عدد اول نمی‌باشد. پس در حالت کلی می‌توان گفت که برای تقسیم اعداد زوج به

توان‌هایی از ۲ به عنوان مثال ۲، ۴، ۸، ۱۶ و ... می‌توان از دستورات 1 SRW، 2 SRW، 3 SRW و 4 SRW ... استفاده نمود به شرطی که پس از شیفت، بیت‌های کم ارزش عدد اول که محتوی "۱" است از دست نرود.

۴-۴-۹- دستور I^۱

به کمک این دستور می‌توان حداکثر ۲۵۵ واحد معادل با یک بایت به کلمه ورودی اضافه نمود.
به بلوک 18 FB توجه کنید.

FB 18

```

SEGMENT 1          0000
NAME :PLUS

0005      :L      IW  22
0006      :I      115
0007      :T      QW  14
0008      :BE

```

در این بلوک به محتویات کلمه ورودی ۲۲، ۱۱۵ واحد افزوده شده است. توجه داشته باشید که مقدار اضافه شونده، عددی در مبنای ۱۰ می‌باشد.

مثال ۴-۱۵: با استفاده از دستور I، ۱۵ واحد به عدد ۴۶۱_(۱۰) اضافه کنید.

ابتدا عدد ۴۶۱_(۱۰) را به مبنای ۲ تبدیل نموده، آن را به صورت یک کلمه ورودی نمایش می‌دهیم.

$$461_{(10)} = 111001101_{(2)} = 00000000111001101_{(2)} \rightarrow IW\ 22$$

حال با استفاده از بلوک 19 FB حاصل را به 14 QW می‌فرستیم.

FB 19

```

SEGMENT 1          0000
NAME :PLUS1

0005      :L      IW  22
0006      :I      15
0007      :T      QW  14
0008      :BE

```

در زیر محتویات 22 IW و 14 QW نشان داده شده است.

IW 22	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	1	$111001101_{(2)} = 461_{(10)}$
0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	1			
QW 14	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	$111011100_{(2)} = 476_{(10)}$
0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0			

۴-۴-۱۰- دستور D

به کمک این دستور می‌توان حداکثر ۲۵۵ واحد از کلمه ورودی تفریق نمود. بلوک 20 FB را در نظر بگیرید.

FB 20

```

SEGMENT 1          0000
NAME :MINUS

0005      :L      IW  22
0006      :D              12
0007      :T      QW  14
0008      :BE
    
```

در اجرای برنامه این بلوک از محتویات کلمه ورودی ۲۲، ۱۲ واحد کسر می‌گردد.

مثال ۴-۱۶: با استفاده از دستور D از عدد $461_{(10)}$ ، ۱۰۰ واحد کم کنید.

ابتدا عدد $461_{(10)}$ را به مبنای ۲ تبدیل نموده، آن را به صورت یک کلمه ورودی در نظر می‌گیریم:

$$461_{(10)} = 111001101_{(2)} = 00000000111001101_{(2)} \rightarrow \text{IW 22}$$

سپس با استفاده از بلوک 21 FB حاصل این تفریق را به کلمه خروجی ۸ می‌فرستیم.

FB 21

```

SEGMENT 1          0000
NAME :MINU1

0005      :L      IW  22
0006      :D              100
0007      :T      QW   8
0008      :BE
    
```

در زیر محتویات 22 IW و 8 QW نشان داده شده است.

IW 22

0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $111001101_2 = 461_{(10)}$

QW 8

0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 $101101001_2 = 361_{(10)}$

۴-۱۱- دستور ADD

در مورد دستورات I و D خاطر نشان کردیم که حداکثر مقداری که می‌توان به کلمه ورودی اضافه و یا از آن کسر نمود ۲۵۵ واحد می‌باشد. حال در صورتی که در اجرای برخی از برنامه‌ها به جمع و یا تفریق نمودن یک مقدار ورودی با مقادیری بیش از مقادیر ذکر شده نیاز داشته باشیم با مشکل مواجه خواهیم شد. برای رفع این مشکل از دستور ADD استفاده می‌کنیم. به کمک این دستور می‌توان اعدادی را که در فاصله $[-32768 + 32768]$ قرار دارند با عدد مورد نظر جمع نمود. در هنگام استفاده از این دستور عدد اضافه شونده می‌تواند مثبت یا منفی باشد و با فرمت KF در برنامه استفاده شود.

در بلوک 54 FB نحوه استفاده از این دستور در برنامه آورده شده است.

FB 54

SEGMENT 1 0000
NAME : ADD

0005 :L IW 16
0006 :ADD KF +3000
0008 :T QW 8
0009 :BE

در این برنامه بر خلاف برنامه‌های نوشته شده با دستورات I و D تنها از یک کلمه ورودی استفاده می‌شود و پس از انجام عملیات، عدد حاصل به یک کلمه خروجی فرستاده می‌شود. واضح است که می‌توان با اضافه نمودن عددی منفی به کلمه ورودی، عملیات تفریق را نیز انجام داد. در بلوک 55 FB این عمل انجام شده است.

FB 55

```

SEGMENT 1          0000
NAME :MINUS

```

```

0005      :L      IW      16
0006      :ADD    KF      -1000
0008      :T      QW       8
0009      :BE

```

۴-۴-۱۲- دستور JZ^۱

این دستور در انجام برخی اعمال ریاضی قابل استفاده و اجرا خواهد بود. در بلوک FB 50 طرز استفاده و کاربرد این دستور نشان داده شده است.

FB 50

```

SEGMENT 1          0000
NAME :JPZERO

```

```

0005      :L      IW      16
0006      :L      IW      22
0007      :-F
0008      :JZ      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

همان‌گونه که ملاحظه می‌کنید برنامه‌نویس به اختیار نام JPZERO را برای این بلوک FB انتخاب و همچنین به دلیل عدم نیاز به پارامتر، از تعریف پارامترهای FB خودداری نموده است. در سطرهای بعدی، بدنه بلوک برنامه‌نویسی شده است.

در این برنامه ابتدا حاصل تفریق دو کلمه ورودی IW 16 و IW 22 را با استفاده از دستور - F

1 - Jump if Zero

به دست می‌آوریم. در صورتی که حاصل تفریق صفر باشد و یا به عبارت دیگر دو عدد ورودی مساوی باشند اجرای برنامه از سطری که با برچسب M001 مشخص گردیده دنبال می‌شود و با استفاده از دستور STP که فرمان توقف اجرای برنامه است، برنامه متوقف می‌گردد. در غیر این صورت سطر $JZ = M001$ نادیده فرض شده، سطر بعدی یعنی BEU اجرا خواهد شد. در این مرحله اجرای برنامه به اتمام می‌رسد.

با اندکی تأمل در کاربرد این دستور در می‌یابیم که داشتن حاصل صفر در تفریق دو عدد را می‌توان معادل با شرط مساوی بودن دو عدد مذکور در نظر گرفت. بنابراین از این دستور می‌توان به عنوان یک دستور جایگزین در دستورات مقایسه‌ای ($F = !$) برای حالت تساوی دو مقدار ورودی استفاده نمود. در بلوک 14 FB این برنامه نوشته شده است. پس در صورت نیاز به تعریف دستور مقایسه‌ای ($F = !$) می‌توان از دستور JZ نیز استفاده نمود.

FB 14

```

SEGMENT 1          0000
NAME :LABEL

0005      :L      IW   16
0006      :L      IW   22
0007      :!=F
0008      :JC      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

۴-۴-۱۳ - دستور JN^۱

بر خلاف دستور JZ، در این دستور پرش هنگامی صورت می‌گیرد که حاصل عملیات ریاضی مخالف صفر باشد. در بلوک 51 FB این دستور استفاده شده است.

FB 51

```

SEGMENT 1          0000
NAME : JPNZERO

0005      :L      IW   16
0006      :L      IW   22
0007      :-F
0008      :JN      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

روند اجرای برنامه به این صورت است که اگر حاصل تفریق کلمه ورودی IW 22 از IW 16 برابر صفر نشود پرش به سطری از برنامه که با برچسب M001 مشخص شده انجام می‌گیرد و با اجرای فرمان STP، PLC متوقف می‌شود. در غیر این صورت سطر JN = M001 نادیده فرض شده، پس از اجرای دستور BEU، اجرای برنامه به اتمام می‌رسد.

با کمی دقت در کاربرد این دستور در می‌یابیم که حاصل تفریق دو عدد هنگامی مخالف صفر است که آن دو عدد نامساوی باشند. بنابراین می‌توان از این دستور به جای دستور مقایسه‌ای نامساوی ($><F$) در مقایسه دو کلمه استفاده نمود. در بلوک FB 45 برنامه مقایسه دو کلمه ورودی با استفاده از دستور مقایسه‌ای ($><F$) آمده است.

FB 45

```

SEGMENT 1          0000
NAME : NOT

0005      :L      IW   16
0006      :L      IW   22
0007      :><F
0008      :JC      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

۴-۴-۱۴ - دستور JP^۱

در این دستور، پرش هنگامی صورت می‌گیرد که حاصل عملیات ریاضی عددی مثبت باشد. در بلوک 52 FB چگونگی استفاده از این دستور آمده است:

FB 52

```

SEGMENT 1          0000
NAME :JPPOSI

0005      :L      IW  16
0006      :L      IW  22
0007      :-F
0008      :JP      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

در این برنامه در صورتی که حاصل تفریق دو عدد موجود در کلمه‌های ورودی IW 16 و IW 22 مثبت بوده، یا به عبارت دیگر عدد موجود در IW 16 از عدد موجود در IW 22 بزرگتر باشد پرش به سطری از برنامه که با برچسب M 001 مشخص شده است انجام خواهد گرفت و با فرمان STP، PLC متوقف خواهد شد. در غیر این صورت سطر JP = M001 نادیده فرض شده، پس از اجرای دستور BEU اجرای برنامه به پایان می‌رسد.

همان‌گونه که ذکر شد در صورتی که عدد اول بزرگتر از عدد دوم باشد این دستور اجرا می‌شود، بنابراین می‌توان از دستور مقایسه‌ای (F >) به جای دستور JP استفاده نمود. در بلوک 46 FB این جایگزینی انجام شده است.

FB 46

```

SEGMENT 1          0000
NAME :JMINU

```

```

0005      :L      IW  16
0006      :L      IW  22
0007      :>F
0008      :JC      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

۴-۴-۱۵- دستور JM

بر خلاف دستور JP، در این حالت، پرش هنگامی صورت می‌گیرد که حاصل عملیات ریاضی عددی منفی باشد. در بلوک FB 53 چگونگی استفاده و عملکرد این دستور آمده است:

FB 53

```

SEGMENT 1          0000
NAME :JMINUS

```

```

0005      :L      IW  16
0006      :L      IW  22
0007      :-F
0008      :JM      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

در این برنامه در صورتی پرش به سطری که با برچسب M001 مشخص گردیده است انجام می‌شود که حاصل تفریق دو عدد موجود در کلمه‌های IW 16 و IW 22 عددی منفی باشد. به بیان

دیگر، پرش هنگامی صورت می‌گیرد که عدد اول یعنی 16 IW از عدد دوم یعنی 22 IW کوچکتر باشد. بنابراین از این دستور می‌توان به جای دستور مقایسه‌ای ($F <$) استفاده نمود. در بلوک 47 FB این جایگزینی اعمال شده است.

FB 47

```

SEGMENT 1          0000
NAME :JMINUS1

0005      :L      IW   16
0006      :L      IW   22
0007      :<F
0008      :JC      =M001
0009      :BEU
000A M001 :STP
000B      :BE

```

اکنون که با تعریف و طرز استفاده از FBها در برنامه‌نویسی آشنا شدیم به ذکر یک مثال می‌پردازیم.

مثال ۴-۱۷: در مثال ۵-۴ سرعت اجرای یک سیکل برنامه را با استفاده از تعریف چندین بلوک (31 PB، 40 PB، 28 DB، 1 OB و ...) اندازه‌گیری نمودیم. در اینجا قصد داریم تا با استفاده از بلوک تابع ساز، برنامه مذکور را بازنویسی کنیم.

FB 10

```

SEGMENT 1          0000
NAME :CYCLET

0005      :A      I      0.0
0006      :JC      =M001
0007      :AN      I      0.0
0008      :JC      =M002
0009      :BEU
000A M001 :L      FW   30
000B      :L      KF  +1
000D      :+F
000E      :T      FW   30
000F      :L      KF +2000

```

ادامهٔ بلوک 10 FB:

```

0011      :!=F
0012      :S   Q      2.0
0013      :BEU
0014 M002 :L   KF  +0
0016      :T   FW   30
0017      :R   Q      2.0
0018      :BE

```

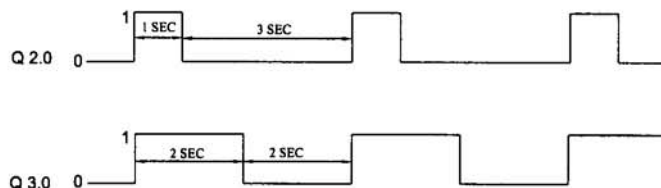
روند اجرای برنامه به ترتیب زیر است:

در سطر اول چنانچه بیت ورودی I 0.0 برابر "۱" باشد پرش به سطری که با برچسب M001 مشخص شده، انجام می‌گیرد. در صورتی که "۰" = I 0.0 باشد پرش به سطری که با برچسب M002 مشخص شده، انجام می‌شود.

حالتی را در نظر بگیرید که "۱" = I 0.0 باشد. در این حالت اجرای برنامه از سطری که با برچسب M001 مشخص شده، دنبال می‌گردد. در این قسمت از برنامه به محتویات FW 30 ازای پیمایش هر سیکل زمانی، یک واحد افزوده خواهد شد و در صورتی که تعداد سیکل‌های پیموده شده به ۲۰۰۰ برسد خروجی Q 2.0 روشن و زمان اندازه‌گیری شده، ثبت می‌گردد. روشن شدن بیت خروجی Q 2.0 به این معنی است که در مدت زمان اندازه‌گیری شده تعداد ۲۰۰۰ سیکل از برنامه طی شده است. بنابراین برای به دست آوردن مدت زمان اجرای یک سیکل، زمان اندازه‌گیری شده را بر ۲۰۰۰ تقسیم می‌کنیم.

در صورتی که "۰" = I 0.0 باشد پرش به سطری که با برچسب M002 مشخص شده، انجام می‌شود. در این قسمت از برنامه جهت اطمینان از شمارش تعداد سیکل‌های پیموده شده از عدد صفر تا ۲۰۰۰، عدد ۰ را در FW 30 قرار می‌دهیم و با ریست نمودن خروجی Q 2.0 باعث می‌شویم که این عمل را بتوان برای چندین بار انجام داد.

مثال ۴-۱۸: برنامه‌ای بنویسید که دو شکل موج زیر را در دو خروجی جداگانه ایجاد نماید. یا به عبارت دیگر نسبت روشن و خاموش شدن دو خروجی را مطابق شکل موجهای زیر تنظیم نماید.



PB 40

برنامه مطلوب در PB 40 نوشته شده است.

SEGMENT	1	0000	
0000	:AN	F	6.0
0001	:L	KT	120.1
0003	:SE	T	1
0004	:A	T	1
0005	: =	F	6.0
0006	:L	T	1
0007	:L	KF	+0
0009	: !=F		
000A	:S	Q	2.0
000B	:S	Q	3.0
000C	:L	T	1
000D	:L	KF	+90
000F	: !=F		
0010	:R	Q	2.0
0011	:L	T	1
0012	:L	KF	+60
0014	: !=F		
0015	:R	Q	3.0
0016	:BE		

در این قسمت از برنامه، تایمری داریم که با عدد 120.1 KT بارگذاری شده و دائماً در حال کار است.

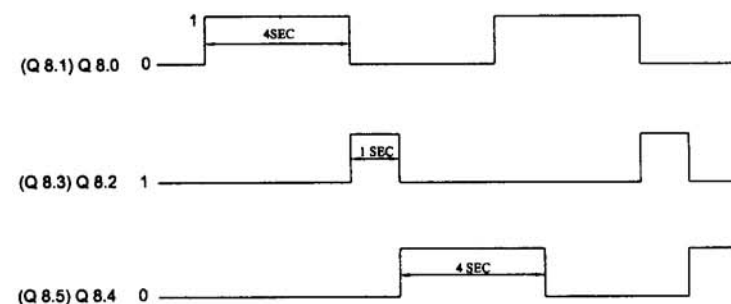
در صورتی که تایمر به صفر برسد هر دو خروجی فعال می‌شوند.

در صورتی که تایمر به 90 برسد خروجی 2.0 Q را ریست می‌کند.

در صورتی که تایمر به 60 برسد خروجی دوم را نیز ریست می‌کند.

اکنون که با برنامه‌نویسی تولید شکل موجهای مختلف با Duty Cycle های متفاوت آشنا شدیم به ذکر یک مثال می‌پردازیم:

مثال ۴-۱۹: با استفاده از روند برنامه‌نویسی جهت تولید شکل موجهای مختلف برنامه کنترل چراغ راهنمایی را بازنویسی کنید.



در کنترل چراغ راهنمایی خروجی‌های مورد نظر دارای شکل موجهای زیر می‌باشند.

پس در این برنامه لازم است تا ۳ شکل موج مطابق شکل موجهای فوق ایجاد شوند. ادامه این برنامه را به خوانندگان واگذار می‌کنیم.

در اکثر پروژه‌ها و فرآیندهای صنعتی گاه لازم است که پیغام هشدار دهنده‌ای مثلاً روشن شدن یک چراغ (LED) چشمک زن به کاربران اعلام شود. سرعت چشمک زدن این چراغ نیز باید به گونه‌ای باشد که کاربران قدرت تشخیص آن را از فواصل دور و در شرایط کاری مختلف داشته باشند. برنامه چراغ چشمک زن در بیشتر موارد با استفاده از تایمر نوشته می‌شود، در این مثال سعی شده تا این برنامه در یک بلوک FB نوشته شود. در این قسمت به ذکر یک مثال در این مورد می‌پردازیم. مثال ۴-۲۰: برنامه‌ای برای یک چراغ چشمک زن^۱ بنویسید که به هنگام صدا زدن FB مربوطه از کاربر دو سؤال بپرسد:

۱- کدام یک از تایمرها در حال حاضر توسط برنامه‌های دیگر اشغال نشده‌اند یا به عبارت دیگر کدام تایمر را می‌توان بدون تداخل در برنامه‌های زمانی بلوک‌های دیگر استفاده نمود. (در پاسخ به این سؤال، کاربر شماره تایمر را وارد می‌کند).

۲- سرعت چشمک زدن چراغ با چه فرکانسی باشد به عنوان مثال با سرعت ۱ ثانیه به ۱ ثانیه و یا ۰/۴ ثانیه به ۰/۴ ثانیه و ...

در مورد این برنامه باید حتماً از تایمر SE استفاده کنیم زیرا خروجی فقط وابسته به لبه بالارونده ورودی است.

از آنجایی که در هر بار اجرای برنامه قصد تغییر پارامترهای برنامه را داریم بنابراین از یک بلوک تابع ساز انتسابی استفاده می‌کنیم. این برنامه در FB 100 تعریف شده است.

```

SEGMENT 1          0000
NAME : BLINK
DECL : TIM          I/Q/D/B/T/C: T
DECL : DATA        I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/
                        KT/KC/KG: KT
000B      : AN      =TIM          نام تایمر
000C      : LW      =DATA          زمان
000D      : SEC      =TIM          نوع تایمر
000E      : L        =TIM          خروجی باینری تایمر
000F      : T        FW 20        و ارسال به FW 20
0010      : BE

```

بلوک 1 OB نیز به صورت زیر برنامه نویسی می گردد.

OB 1

SEGMENT 1 0000

0000 :JU FB 100

0001 NAME :BLINK

0002 TIM : T 8 انتخاب شماره تایمر

0003 DATA : KT 050.1 زمان موردنظر در تایمر به همراه تولرانس آن

0004 :A F 21.2

0005 : = Q 8.0

0006 :A F 21.3

0007 : = Q 9.0

0008 :BE

سرعت خاموش و روشن به فاصله زمانی

هر ۰/۴ ثانیه یک بار ارسال فرمان روشن و

خاموش شدن به خروجی Q 8.0

سرعت خاموش و روشن به فاصله زمانی

هر ۰/۸ ثانیه یک بار ارسال فرمان روشن

و خاموش شدن به خروجی Q 9.0