از مجموعه کتابهای مثلث نارنجی

مرجع کامل

PLC

کنترل کننده‌های منطقی برنامه‌پذیر

نویسنده: مهندس فربد قابوسی

CD همراه شامل:

- بسته نرم‌افزاری Step 5 for Windows
- پاسخ تمرینات موجود در کتاب
- مثالها و برنامه‌های کاربردی
- نسخه الکترونیکی کتاب

نشر آفرنگ

بهار ۱۳۸۲

POWEREN.IR

تقدیم به

پدر و مادر عزیزم

دیباچه نویسنده

امروزه با پدیدار شدن ریزپردازنده‌ها[1] و پیشرفت فن‌آوری حالت جامد[2] در عرصهٔ علم و تکنولوژی که بی‌شک آن را می‌توان بزرگترین پدیده در علوم الکترونیک دانست چهرهٔ محیط‌های صنعتی به کلی دگرگون شده است.

این مهم که به موازات تحول در عرصهٔ کامپیوتر پا به دنیای الکترونیک گذاشت در نقش پلی بین این دو علم یعنی الکترونیک و کامپیوتر قرار گرفت و برای اولین بار به‌راحتی امکان تلفیق سخت‌افزار توسط مهندسین هر دو فن را ممکن نمود. بدین‌ترتیب در یک کاربرد خاص به جای به‌کارگیری اجزائی متشکل از مدارهای منطقی با سخت‌افزارهای تقریباً ثابت و تلفیق آن با پدیدهٔ مهم نرم‌افزار، امکان طراحی و ساخت هزاران پروژهٔ کاربردی ممکن گردید. یکی از مهمترین خصوصیات این مدارها در صورت لزوم، انجام تغییر و تصحیح در نرم‌افزار و برنامهٔ کاربر مورد نظر است که کمتر لزوم تغییر مدارها و سخت‌افزار را در آن می‌طلبد.

خواص نامبرده و سهولت دستیابی به ریزپردازنده‌ها و اجزای جانبی مربوطه در بازارهای جهانی، این پدیده را به‌عنوان اهرم بسیار مهمی جهت رشد و شکوفایی در صنایع الکترونیک قرار داد. چرا که غیر از ساخت این نوع مدارهای مجتمع[3] که کماکان تکنولوژی پیشرفته‌ای را می‌طلبد، قادر خواهیم بود سایر مراحل انجام یک پروژهٔ کاربردی را خود به انجام برسانیم. این مراحل شامل طراحی و ساخت سخت‌افزار، تدوین نرم‌افزار مربوطه، تلفیق این دو قسمت و تست و رفع عیب می‌باشد که کلاً از عهدهٔ ما برمی‌آید و در واقع فکر و ایده و تخصص را همراه با قوهٔ ابتکار به‌کار می‌گیرد. لذا جهت پرنمودن خلاء به‌وجود آمدهٔ علمی ـ فنی بین دول پیشرفته و کشورهای جهان سوم می‌بایست در جهت آموزش و به‌کارگیری این فن که راه مهمی برای نجات وابستگی تکنولوژیک صرف در مجموعهٔ الکترونیک و کامپیوتر می‌باشد سعی و تلاش پیش گیریم.

"PLC" نیز مولود این پدیده یعنی ظهور ریزپردازنده‌هاست و مخفف عبارت " Programmable Logic Controller" می‌باشد. این سیستم وسیله‌ای است که متناسب با برنامه‌ای که دریافت می‌کند وظیفه‌ای را انجام می‌دهد. به عبارت دیگر PLC نوعی کامپیوتر است که برنامه‌ای خاص را اجرا می‌کند. با ظهور PLC تجهیزات

---

[1] Microprocessors
[2] Solid State
[3] Very Large Scale Integration

و قطعات استفاده شده در کنترل فرآیندهای صنعتی و خطوط تولید تغییر نموده و مدارهای رله کنتاکتوری و سخت‌افزاری حالت جامد کم‌کم جای خود را به کنترل کننده‌های منطقی قابل برنامه‌ریزی یعنی PLCها دادند.

امروزه در طراحی مدارهای کنترل کنندهٔ خطوط تولید و فرآیندهای صنعتی استفاده از مدارهای رله کنتاکتوری منسوخ گردیده و در اکثر کارخانه‌ها و مراکز صنعتی از سیستم‌های PLC استفاده می‌شود.

عدم وجود منبع و مأخذی جامع و کامل که پاسخگوی سؤالات بی‌شمار دانشجویان، تکنسین‌ها و متخصصینی باشد که در مراکز صنعتی با کمبود اطلاعات در زمینهٔ سیستم‌های PLC دست به گریبان هستند، نگارنده را بر آن داشت تا به جمع‌آوری چنین مجموعه‌ای بپردازد.

آنچه که در پیش رو دارید مجموعه‌ای است که حاصل سال‌ها تلاش، تجربه، تحقیق و مطالعه در زمینهٔ انواع سیستم‌های PLC می‌باشد. نحوهٔ ارائهٔ مطالب در این کتاب به‌گونه‌ای است که برای علاقمندان مبتدی و پیشرفته قابل استفاده باشد. جهت استفاده از این کتاب نیازی به اطلاعات پیچیده و جامع نبوده، داشتن اطلاعات مختصری در مورد مدارهای منطقی و سیستم‌های ریزپردازنده کافی است.

در تألیف این کتاب سعی شده مطالب به‌گونه‌ای عنوان شود که حاوی اطلاعات پایه و کاربردی نیز باشد. بنابراین کارفرمایان محترم می‌توانند از آن جهت آموزش پرسنل خود در محیط‌های صنعتی بهره گیرند. این کتاب می‌تواند به‌عنوان منبعی درسی برای مقاطع دانشگاهی (کارشناسی و کارشناسی ارشد) نیز مورد استفاده قرار گیرد.

در پایان لازم است از راهنمایی‌های استاد گرانقدر خود آقای مهندس کریم بهنام نیکو که مشوق نگارنده در به چاپ رساندن این کتاب بوده و تصحیح این مجموعه را نیز بر عهده داشته‌اند سپاسگزاری نمایم.

فربد قابوسی
آبان ۱۳۷۸

پیشگفتار چاپ چهارم

پروردگار مهربان را سپاس فراوان که به نگارندهٔ این کتاب توفیق آن را عنایت فرمود تا به جامعهٔ علمی کشور به‌ویژه دانشجویان، صنعتگران و محققین علوم الکترونیک، کنترل و اتوماسیون خدمتی ناچیز ارائه نماید.

بدون تردید PLC مهمترین و پرکاربردترین وسیلهٔ اتوماسیون در صنایع مدرن امروزی است. در ماشین‌ها و خطوط تولید جدید، کمتر موردی را می‌توان یافت که از کنترل‌کننده‌های منطقی قابل برنامه‌ریزی استفاده نشده باشد. در حقیقت این وسیلهٔ بسیار قابل انعطاف که خود یک کنترل‌کنندهٔ کامل است به‌عنوان قطعه‌ای برنامه‌پذیر در صنایع گوناگون، کاربرد وسیعی یافته است به‌گونه‌ای که با پیشرفت تکنولوژی و حضور اتوماسیون در عرصهٔ صنعت در طراحی کنترل‌کننده‌ها و مدارهای فرمان خطوط تولید و فرآیندهای صنعتی، استفاده از مدارهای فرمان قدیمی منسوخ گردیده و در اکثر مراکز صنعتی از کنترل‌کننده‌های منطقی قابل برنامه‌ریزی استفاده می‌گردد.

پس از برگزاری دوره‌های آموزشی PLC در مراکز صنعتی و دانشگاههای سراسر کشور دریافتم که اکثر داوطلبین این دوره‌ها، مهندسین، متخصصین و کارشناسان شاغل در شرکت‌ها و کارخانه‌های دولتی و خصوصی بوده و اغلب آنها به یک سیستم PLC جهت برنامه‌نویسی، آزمایش و اجرای برنامهٔ نوشتهٔ خود دسترسی دارد. اما تعداد اندکی از داوطلبین شرکت در دوره‌های مذکور که اغلب آنها دانشجویان مستعد و علاقمند به فراگیری این سیستم بودند، فاقد امکانات و منابع برنامه‌نویسی بوده و دستیابی به سیستم PLC برای آنها میسر نبود. زیرا PLC شامل یک سیستم سخت‌افزاری است و اغلب کاربران نیز از عهدهٔ خرید یک دستگاه PLC آموزشی نیز برنمی‌آیند.

پس از عزیمت به خارج از کشور جهت ادامه تحصیل و حضور در دانشگاهها و مراکز صنعتی و دستیابی به چند نرم‌افزار خاص و کاربردی تصمیم گرفتم که این مجموعه را با ویرایشی جدید به چاپ برسانم. به همراه چاپ چهارم این کتاب یک لوح فشرده نیز ارائه می‌گردد. این لوح حاوی نرم‌افزار شبیه‌ساز یک سیستم PLC زیمنس است. در متن و مطالب اصلی این کتاب تغییر خاصی اعمال نشده است. اما پاسخ تمامی تمرینات و مثالهای موجود در این کتاب در لوح فشردهٔ مذکور ذخیره شده است. روش نصب و استفاده از نرم‌افزار شبیه‌ساز

PLC در ضمیمهٔ ٦ آموزش داده شده است. این بستهٔ نرم‌افزاری، سیستم PLC را شبیه‌سازی کرده و به کاربر اجازه می‌دهد تا با نصب این نرم‌افزار از رایانهٔ شخصی خود به جای یک دستگاه PLC استفاده نماید.

شایان ذکر است که این نرم‌افزار تحت نسخهٔ Windows 3.1 بوده، اما در تمامی سیستم‌های عامل XP و Windows 98 نیز قابل استفاده است. نحوهٔ نصب این نرم‌افزار و روش برنامه‌نویسی آن در ضمیمهٔ ٦ به‌طور کامل توضیح داده شده است.

با در اختیار داشتن این نرم‌افزار هریک از کاربران به‌راحتی می‌توانند برنامه‌های خود را به هر یک از سه روش STL، CSF و LAD نوشته و آنها را برروی یک دستگاه رایانهٔ شخصی اجرا کنند. با وجود چنین نرم‌افزاری، برنامه‌نویسی و فراگیری PLC بسیار ساده و لذت‌بخش خواهدبود.

پیشرفت روزافزون سیستم‌های کنترل و اتوماسیون از جمله PLC و SCADA در عرصهٔ صنعت و همچنین کاربرد این کتاب در مراکز دانشگاهی و صنعتی و استقبال کم‌نظیر کاربران این سیستم‌ها موجب ترغیب و تشویق نویسندهٔ این اثر به نگارش مجموعه‌های دیگری از این دست شده است که درآینده‌ای نه چندان دور شاهد چاپ این کتب خواهیم بود.

با توجه به همکاری نزدیک اینجانب با شرکت بین‌المللی National Instruments و دسترسی بسیار آسان به منابع و نرم‌افزارهای مربوط به سیستم‌های کنترل و اتوماسیون، علاقمندان به این‌گونه نرم‌افزارها می‌توانند با ارسال درخواست خود به آدرس الکترونیکی farbod_ghaboosi@yahoo.com مطالب و برنامه‌های دیگری را تقاضا کنند.

از خوانندگان عزیز تقاضامند است نظرات و پیشنهادهای خود را جهت بهبود مطالب این کتاب در چاپ‌های بعدی به آدرس الکترونیکی plc_tips@yahoo.com ارسال فرمایند.

فربد قابوسی
زمستان ۱۳۸۱
ایالت نیوجرسی

فهرست مطالب

عنوان

۱ـ فصل اول ـ مفاهیم منطقی

۲ـ فصل دوم ـ ساختار PLC

۳ـ فصل سوم ـ مقدمه‌ای بر زبان STEP 5

۴ـ فصل چهارم ـ برنامه‌نویسی به زبان STEP 5

۵ـ فصل پنجم ـ شیوه‌های کنترل فرآیند

۶ـ فصل ششم ـ رفع اشکال نرم‌افزاری

۷ـ فصل هفتم ـ برنامه‌نویسی به زبان CSTL

۸ـ فصل هشتم ـ برنامه‌نویسی به روش GRAPH 5

ضمائم

فصل اول

مفاهیم منطقی

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

هدف از ارائهٔ این فصل، معرفی مبناهای عددی گوناگون و یادآوری مفاهیم و عملوندهای منطقی، همچنین ارائهٔ توضیح در مورد برخی اصطلاحات استفاده شده در فصول آینده است که به عنوان یک چهارچوب به منظور مطالعهٔ جزئیات بیشتر فصول بعدی استفاده می‌شوند.

به دلیل گستردگی دامنهٔ استفاده از PLC در صنایع مختلف، این مجموعه جهت استفادهٔ متخصصان و کارشناسان رشته‌های مختلف تدوین شده است. در این فصل سعی شده تا تمام مطالب و مفاهیم اولیهٔ مورد نیاز در مورد سیستم‌های کنترل، عناصر و المان‌های کنترلی، مفاهیم خاص در مورد ریزپردازنده‌ها و ... توضیح داده شود. بنابراین جهت مطالعهٔ این کتاب نیاز به مراجعه به مراجع دیگر وجود ندارد و مطالب مورد نیاز در ضمیمه آمده است.

۱-۱- مبناهای 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 نشان‌دهندهٔ 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 آن هم به نوبهٔ 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 ۲^n عنصر به صورت کد به حداقل n بیت نیاز دارد. زیرا n بیت را می‌توان به ۲^n طریق مجزا در کنار هم قرار داد. به عنوان مثال گروهی مشتمل بر چهار ($۴ = ۲^۲$) مقدار مجزا را می‌توان با استفاده از یک کد دو بیتی نمایش و هر مقدار مجزا را به یکی از ترکیبات بیتی ۰۰، ۰۱، ۱۰ و یا ۱۱ نسبت داد.

یک گروه با هشت ($۸ = ۲^۳$) 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 یا ۱۱۱ نسبت داده می‌شود.

مثالهای فوق نشان می‌دهد که ترکیبات یک کد n بیتی را می‌توان با شمارش دودویی از صفر تا "$۲^n - ۱$" 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 ۰۰۱۰_b و یا ۱۰_b و یا $۱۰_{(۲)}$ 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.

$۲_{(۱۰)} = ۱۰_{(۲)} = ۱۰_b$ (عدد دهدهی ۲)

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.

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

اعداد دودویی نیز به شکل اعداد دهدهی به توان می‌رسند.

$۱۰^{۱} = ۱۰_{(۱۰)}$ $۱۰_{(۲)}$ = (عدد دهدهی ۲) $۲^{۱}$

$۱۰۰_{(۱۰)}$ $۱۰^{۲} =$: در مقایسه با $۱۰۰_{(۲)}$ = (عدد دهدهی ۴) $۲^{۲}$

$۱۰^{۳} = ۱۰۰۰_{(۱۰)}$ $۱۰۰۰_{(۲)}$ = (عدد دهدهی ۸) $۲^{۳}$

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 $۴۱_{(۱۰)}$ را به مبنای دو تبدیل کنید.

روند اجرای عملیات تبدیل به صورت زیر می‌باشد:

ابتدا ۴۱ بر ۲ تقسیم شده تا خارج قسمت ۲۰ و باقیماندهٔ ۱ به دست آید. خارج قسمت مجدداً بر ۲ تقسیم شده تا خارج قسمت و باقیماندهٔ جدیدی حاصل گردد. این روال به همین ترتیب تا زمانی ادامه می‌یابد که خارج قسمت صحیح به‌دست آمده صفر شود. ضرایب عدد دودویی مطلوب به صورت زیر از باقیمانده‌ها به‌دست می‌آیند.

ضریب عدد دودویی		باقیمانده		خارج قسمت صحیح
$a_0 = ۱$		۱	+	$\frac{۴۱}{۲} = ۲۰$
$a_1 = ۰$		۰	+	$\frac{۲۰}{۲} = ۱۰$
$a_2 = ۰$		۰	+	$\frac{۱۰}{۲} = ۵$
$a_3 = ۱$		۱	+	$\frac{۵}{۲} = ۲$
$a_4 = ۰$		۰	+	$\frac{۲}{۲} = ۱$
$a_5 = ۱$		۱	+	$\frac{۱}{۲} = ۰$

$(۱۰۱۰۰۱)_۲ = (a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_۲ = ۴۱_{(۱۰)}$: جواب

روال ریاضی فوق می‌تواند به صورت مناسب‌تری به شکل زیر نمایش داده شود.

باقیمانده	عدد صحیح	
	۴۱	
۱	۲۰	
۰	۱۰	
۰	۵	
۱	۲	
۰	۱	
۱	۰	۱۰۱۰۰۱ = جواب

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

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

مثال ۱-۲: عدد $(۱۰۱۰۰۱)_۲$ را به مبنای ۱۰ تبدیل کنید.

$$۱۰۱۰۰۱_{(۲)} = ۱\times۲^۵ + ۰\times۲^۴ + ۱\times۲^۳ + ۰\times۲^۲ + ۰\times۲^۱ + ۱\times۲^۰$$

$$= ۳۲ + ۰ + ۸ + ۰ + ۰ + ۱ = ۴۱_{(۱۰)}$$

پس می‌توان گفت که هر عدد در مبنای r به شکل $(a_n\ a_{n-1}\ a_{n-2} \dots a_2\ a_1\ a_0)_r$ به صورت

حاصلضرب توانهای r در ضرایب مربوطه‌اش بیان می‌شود.

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

اعداد دودویی گرچه برای کامپیوترها استفاده می‌شوند اما چون رشته‌های تکراری از صفرها و یک‌ها هستند تفسیر آنها برای مردم عادی مشکل است. برنامه‌نویسان و کسانی که با قابلیت‌های پردازش داخلی کامپیوترها سر و کار دارند جهت سهولت در تفسیر این نوع اعداد از سیستم‌های عددی اکتال (مبنای ۸) و یا هگزا دسیمال (مبنای ۱۶) استفاده می‌کنند.

۱-۱-۲- مبنای هشت یا اُکتال (Octal)

واژهٔ Octal از کلمهٔ لاتین "Octo" به معنی هشت گرفته شده و به مفهوم سیستم عددی مبنای هشت است که شامل ارقام ۰ تا ۷ می‌باشد. سیستم مبنای هشت برای ارائهٔ اعداد دودویی به شکل فشرده‌تر در برنامه‌نویسی مورد استفاده قرار می‌گیرد. از آنجایی که این سیستم عددی از هشت رقم تشکیل یافته است و همچنین با استفاده از ۳ بیت می‌توان هشت ترکیب مختلف را نشان داد، اعداد دودویی برای تبدیل به مبنای ۸ معمولاً به گروههای ۳ بیتی تقسیم می‌شوند.

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

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

۳ بیتی تقسیم می‌کنیم. برای مثال عدد دودویی $(۱۰۱۰۰۱۱)_۲$ 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 به‌دست می‌آید که با تبدیل این دسته‌ها به مبنای هشت عدد $(۱۲۳)_۸$ حاصل می‌گردد.

$$(۱۰۱۰۰۱۱)_۲ = (۰۰۱\ \ ۰۱۰\ \ ۰۱۱)_۲$$
$$= (\ ۱\ \ \ ۲\ \ \ ۳\ )_۸$$

گرچه اعداد در مبنای هشت از نظر ظاهر شبیه اعداد دهدهی هستند اما مفهوم متفاوتی دارند. به عنوان مثال عدد $(۱۲۳)_{۱۰}$ معادل با عدد $۱۲۳=۳\times۱۰^{۰}+۲\times۱۰^{۱}+۱\times۱۰^{۲}$ است در حالی که این عدد درسیستم‌مبنای هشت‌به‌مفهوم $۳\times۸^{۰}+۲\times۸^{۱}+۱\times۸^{۲}$ می‌باشدکه معادل عدد دهدهی ۸۳ است.

از آنجایی که سیستم مبنای ۸ با مضرب‌هایی از ۳ بیت کار می‌کند و میکروکامپیوترها معمولاً بر اساس واحدهای ۴ و 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 میکروکامپیوترها دارد.

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

واژهٔ hexadecimal از ترکیب دو کلمهٔ یونانی hex به معنی ۶ و کلمهٔ 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 ذخیرهٔ 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.

ضریب عدد هگز		باقیمانده		خارج قسمت صحیح
$a_0 = ۳$	+	۳		$\frac{۸۳}{۱۶} = ۵$
$a_1 = ۵$	+	۵		$\frac{۵}{۱۶} = ۰$

$$۸۳_{(۱۰)} = a_1\, a_0{}_{(۱۶)} = ۵۳_{(۱۶)}$$

برای تبدیل عددی از مبنای ۲ به مبنای ۱۶ باید از سمت چپ بیت‌ها را به دسته‌های ۴ تـایی تقسیم نمود و در دستهٔ آخر به تعداد تفاضل بیت‌های موجود و عدد ۴، صفر قرار داد. مثلاً بـرای 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.

$(۰۰۰۱ \quad ۱۱۰۰ \quad ۱۰۱۰)_۲$
$(\quad ۱ \quad C \quad A \quad)_{۱۶}$

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

بنابر آنچه گفته شد هر گروه ۸ بیتی می‌تواند هر یک از ۲۵۶ = 2^8 عدد مختلف هگزا دسیمال (از 00_H تا FF_H) را در خود جای دهد.

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

اعداد دهدهی کد شده در مبنای دو (دهدهی کدشده 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 کردن‌ها و تبدیل‌ها، اعداد دهدهی در مبنای دو کددهی می‌گردند.

درکد BCD، هر رقم در مبنای دهدهی به طور جداگانه به شکل یک عدد دودویی کد می‌شود. 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 می‌گردد. در این روش که ۱-۲-۴-۸ نیز نامیده می‌شود مطابق جدول ۱-۴ کدهای زیر مورد استفاده قرار می‌گیرند.

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 ۰۱۱۰ ۱۰۰۱ نشان داده می‌شود. در جدول ۱-۵ سیستم‌های اعداد به صورت یک جا نمایش داده شده‌اند.

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[1] 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 وسیلهٔ یک نقطه و یا عدم وجود عملگر بین دو عملوند (متغیر) نمایش داده می‌شود. مثلاً: $z = x \cdot y$ یا $z = xy$ و بدین صورت تفسیر می‌شود که از نظر منطقی $z = ۱$ خواهد بود اگر و تنها اگر $x = ۱$ و $y = ۱$ باشند، در غیر این صورت $z = ۰$ 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 $z = x + y$ و مفهوم آن این است که $z = ۱$ خواهد بود اگر و تنها اگر $x = ۱$ یا $y = ۱$ باشند و یا اینکه هر دو، مقدار "۱" داشته باشند. در صورتی که $x = ۰$ و $y = ۰$ باشد آنگاه $z = ۰$ خواهد بود. به عبارت دیگر $z = ۱$ 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 $z = x'$ و یا $z = \bar{x}$، مفهوم آن این است که اگر $x = ۱$ باشد آنگاه $z = ۰$ است و بالعکس اگر $x = ۰$ باشد آنگاه $z = ۱$ 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.

---

1 - Boolean Algebra

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

AND

x	y	xy
۰	۰	۰
۰	۱	۰
۱	۰	۰
۱	۱	۱

OR

x	y	x + y
۰	۰	۰
۰	۱	۱
۱	۰	۱
۱	۱	۱

NOT

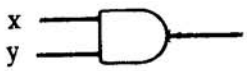
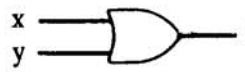
x	x′
۰	۱
۱	۰

۱-۲-۲- سایر عملگرهای منطقی

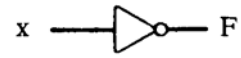
هنگامی که عملگرهای منطقی AND و OR بین دو متغیر x و y قرار گیرند به ترتیب توابع بولی x.y و x + y حاصل خواهد شد.

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 کلیهٔ 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
AND		F = xy	x y | F ۰ ۰ | ۰ ۰ ۱ | ۰ ۱ ۰ | ۰ ۱ ۱ | ۱
OR		F = x + y	x y | F ۰ ۰ | ۰ ۰ ۱ | ۱ ۱ ۰ | ۱ ۱ ۱ | ۱

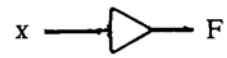
Inverter



$F = x'$

x	F
۰	۱
۱	۰

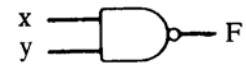
Buffer



$F = x$

x	F
۰	۰
۱	۱

NAND



$F = (xy)'$

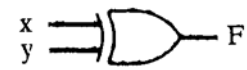
x	y	F
۰	۰	۱
۰	۱	۱
۱	۰	۱
۱	۱	۰

NOR

$F = (x + y)'$

x	y	F
۰	۰	۱
۰	۱	۰
۱	۰	۰
۱	۱	۰

Exclusive-OR (XOR)



$F = x \oplus y$
$= xy' + x'y$

x	y	F
۰	۰	۰
۰	۱	۱
۱	۰	۱
۱	۱	۰

Exclusive-NOR (XNOR)

$F = x \odot y$
$= xy + x'y'$

x	y	F
۰	۰	۱
۰	۱	۰
۱	۰	۰
۱	۱	۱

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

همان‌گونه 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[1] RS می‌نامند.

۱-۴- اجزای حافظه و انواع آن

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

بیت (bit) : این کلمه شکل خلاصه شدهٔ **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 به‌طور فیزیکی به وسیلهٔ پالسی که به یک مدار ارسال می‌گردد و یا به شکل نقطهٔ کوچکی روی دیسک مغناطیسی که قابلیت ذخیره‌سازی "۰" و "۱" را دارد، مشخص می‌گردد.

---

1 - Latch

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، بایت ها را تشکیل می‌دهند که جهت ارائهٔ انواع اطلاعات از جمله حروف الفباء، ارقام صفر تا ۹ و ... مورد استفاده قرار می‌گیرند.

بایت (byte) : شکل خلاصه شدهٔ 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 ذخیرهٔ 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 $۲^۸=۲۵۶$ 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 $۲^{۱۶}=۶۵۵۳۶$ حالت مختلف قابل نمایش است.

۱-۴-۱- گذرگاه یا مسیر عمومی (Bus)

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

دراین‌گذرگاه، 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 به‌دست 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 ادامهٔ بحث به ذکر انواع مختلف حافظه در سیستم‌های دیجیتالی و کامپیوتری خواهیم پرداخت.

۱-۴-۳- حافظه‌های با دسترسی تصادفی (RAM)[1]

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 دستهٔ 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.

1 - Random Access Memory

الف) حافظه‌های فقط خواندنی (ROM)[1]

حافظه‌های فقط خواندنی امروزه با استفاده از تکنولوژی LSIها[2] و VLSIها 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)[3]

این نوع حافظه نیز مانند ROM بوده، از آن جهت نگهداری اطلاعات مختلف استفاده می‌شود. اگر در ثبت بیت‌های اطلاعاتی حافظه‌های ROM و PROM 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)[4]

---

1 - Read Only Memory 2 - Large Scale Integration 3 - Read Mostly Memory

۴ - این نوع حافظه را با E^2PROM نیز نمایش می‌دهند.

ج) حافظه‌های خواندنی نوشتنی (RWM)[1]

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 حافظهٔ 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 سیگنال های 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 ۱-۸ انواع حافظه‌ها و اطلاعات دیگر در رابطه با نحوهٔ پاک شدن و ... آمده است.

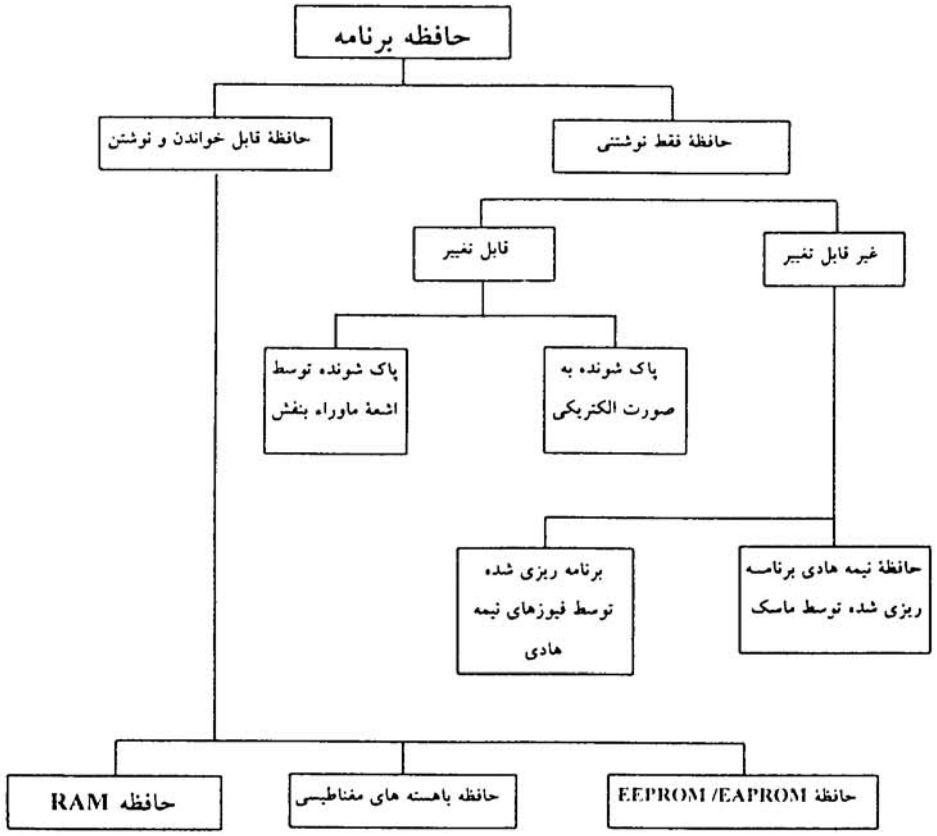
جدول ۱-۸ : انواع حافظه‌ها و سایر اطلاعات در مورد آنها

محتوای حافظه پس از قطع تغذیه	برنامه‌ریزی	طریقهٔ پاک شدن	نوع حافظه	ساختمان
ناپایدار	الکتریکی	الکتریکی	حافظهٔ با دسترسی اتفاقی	RAM
پایدار	توسط کارخانه‌سازنده	غیرممکن	حافظهٔ فقط خواندنی	ROM
	الکتریکی		ROM قابل برنامه‌ریزی	PROM
		توسط اشعهٔ ماوراء بنفش	PROM قابل پاک شدن	EPROM
			PROM قابل برنامه‌ریزی مجدد	REPROM
		الکتریکی	ROM با قابلیت پاک شدن الکتریکی	EEPROM
			ROM با قابلیت پاک شدن الکتریکی	EAPROM[2]

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

---

1 - Read Write Memory 2 - Electrical Alterable PROM



شکل ۱-۱ : نحوهٔ تقسیم‌بندی حافظه‌ها

حال که با اجزای تشکیل دهندهٔ حافظه و انواع حافظه‌ها آشنا شدیم به بحث پیرامون سیستم‌های کنترل می‌پردازیم.

۱-۵- سیستم‌های کنترل

در سالهای اخیر، سیستم‌های کنترل اهمیت فزاینده‌ای در توسعه و پیشرفت تکنولوژی جدید

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

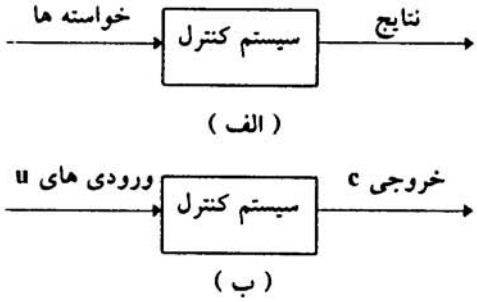
سیستم‌های کنترل در تمام بخشهای صنعت نظیر کنترل کیفیت محصولات، خط مونتاژ خودکار، کنترل ماشین‌ابزار، تکنولوژی فضایی و سیستم‌های نظامی، کنترل کامپیوتری، سیستم‌های حمل و نقل، سیستم‌های قدرت، آدمهای ماشینی و در موارد بسیار دیگری، به فراوانی یافت می‌شوند. صرف‌نظر از اینکه چه نوع سیستم کنترلی در اختیار داریم، 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. همان طور که در شکل ۱-۲ (ب) دیده می‌شود این سه بخش اساسی به ترتیب با عناوین ورودی‌ها، اجزای سیستم و خروجی‌ها که اصطلاحات علمی‌تری هستند نیز شناخته می‌شوند.

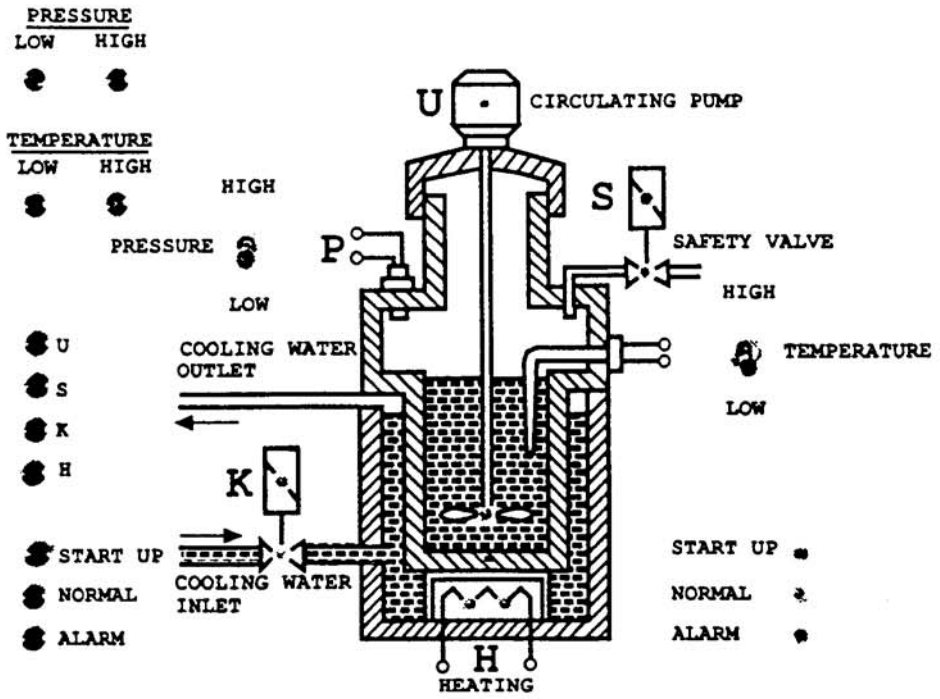


شکل ۱-۲ : بخش‌های یک سیستم کنترل

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

به عنوان مثال با ثابت نگه داشتن متغیرهای زیر می‌توان فرآیند شیمیایی موجود در شکل ۱-۳ راکنترل نمود.

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



شکل ۱-۳ : 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)

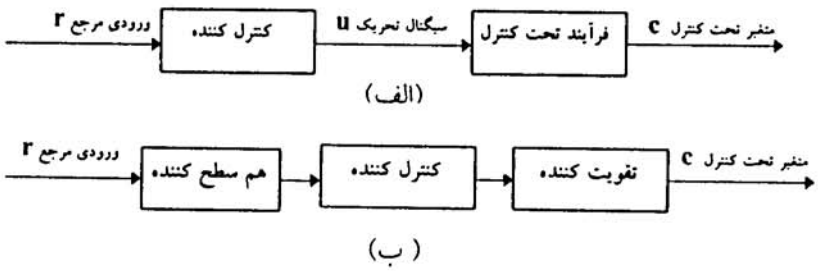
ب) سیستم‌های کنترل حلقه بسته (Closed Loop)

الف) سیستم‌های کنترل مدار باز (Open Loop)

در این گونه سیستم‌ها که در شکل ۱-۴ (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 اندازهٔ کافی تمیز شدند، بطور خودکار خاموش شود.

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



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

در موارد ساده، کنترل کننده می‌تواند بر حسب طبیعت سیستم، یک تقویت کننده، اتصالات

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

همان‌گونه 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 ادامهٔ بحث به توضیح در مورد اجزای یک سیستم کنترل مدار باز (اجزای نشان داده شده در شکل ۱-۴ (ب)) می‌پردازیم.

۱- واحد ورودی (ورودی مرجع r) : اطلاعات از طریق کلیدها و حس‌کنندها (سنسورها) به ورودی‌ها منتقل می‌گردد. این اطلاعات معمولاً به شکل سیگنال‌های الکتریکی است که می‌تواند به صورت دیجیتال یا آنالوگ باشد.

۲- واحد متناسب کننده یا هم‌سطح 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.

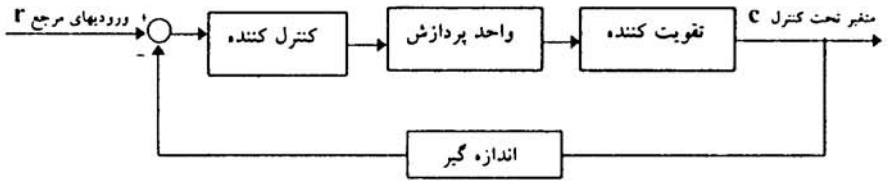
۳- واحد کنترل کننده (واحد پردازش) : این واحد، هستهٔ سیستم کنترل را تشکیل داده، اعمال منطقی و محاسباتی در این قسمت صورت می‌گیرد.

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، یالامپ‌های نشان‌دهنده‌را روشن نماید.

۵- واحد خروجی (متغیر تحت کنترل c) : از این قسمت فرمانهای سیستم کنترل به اجرا کننده‌های فرمان یعنی محرک‌ها (Actuators) و ... ارسال می‌شود.

ب) سیستم‌های کنترل مدار بسته (Closed Loop)

آنچه برای کنترل دقیق‌تر و قابل انعطاف‌تر لازم بوده و در سیستم کنترل مدار باز وجود ندارد یک اتصال یا فیدبک از خروجی، به ورودی سیستم است. برای دستیابی به کنترل دقیق‌تر، 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 می‌شود. در شکل ۱-۵ طرز کار این گونه سیستم‌ها 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 حلقهٔ 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 دستهٔ زیر تقسیم نمود:

۱- سیستم‌های کنترل سخت‌افزاری

۲- سیستم‌های کنترل نرم‌افزاری

۱-۶-۱- سیستم‌های کنترل سخت‌افزاری

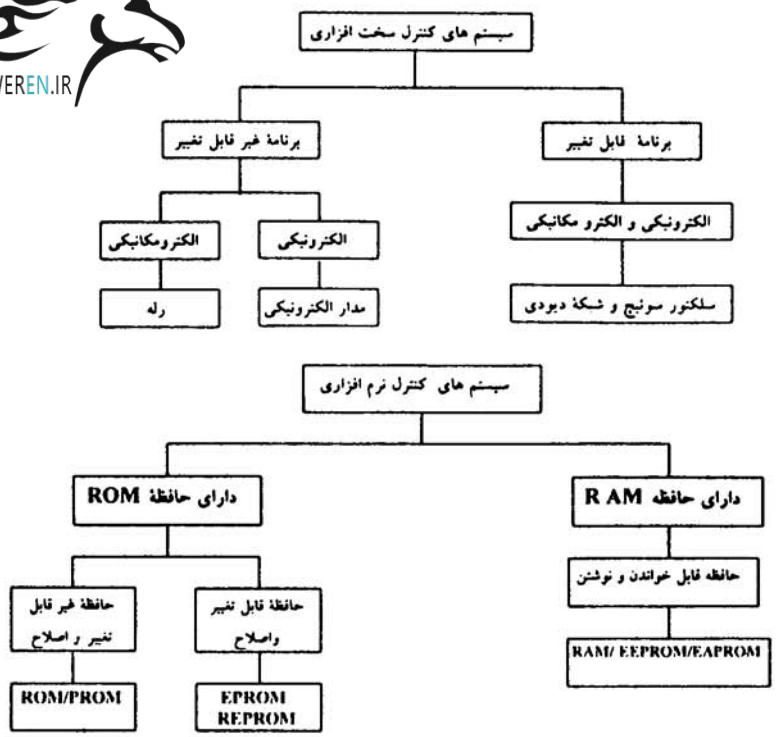
این سیستم‌ها شامل مداراتی هستند که با استفاده از رله‌ها و عناصر الکترونیکی مانند دیودها و ترانزیستورها ساخته می‌شوند. برنامهٔ کنترل در این سیستم‌ها نتیجهٔ روابط بین عناصر مدار الکتریکی است و به راحتی قابل تغییر نمی‌باشد. به عبارت دیگر تغییر در برنامهٔ کنترل به معنی تغییر در سخت‌افزار سیستم است البته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

۱-۶-۲- سیستم‌های کنترل نرم‌افزاری

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

این سیستم‌ها در آن است که نحوهٔ کنترل را با تغییر برنامه و بدون نیاز به تغییر در سخت‌افزار سیستم می‌توان عوض کرد، زیرا نحوهٔ کنترل سیستم توسط سخت‌افزار سیستم تعیین نمی‌شود بلکه برنامه‌ای که در حافظه ذخیره شده یعنی نرم‌افزار سیستم، نحوهٔ کنترل را مشخص می‌کند لذا این سیستم‌ها بسیار قابل انعطاف بوده، کاربردهای فراوانی دارند. بسته به نوع حافظهٔ این سیستم‌ها، شیوهٔ تغییر در برنامه‌ها متفاوت است. اگر از حافظهٔ RAM استفاده شود، بدون دخالت فیزیکی و تنها با اضافه یا کم نمودن چند سطر برنامه می‌توان برنامهٔ جدید را به اجرا در آورد.

در صورتی که از حافظهٔ ROM استفاده شود به اجرا درآوردن برنامهٔ جدید تنها با تعویض حافظهٔ 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

PLC – ۱ – ۲

"PLC" از عبارت Programmable Logic Controller به معنای کنترل کنندهٔ 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 نتیجهٔ عملیات را نیز از قسمت خروجی به صورت فرمانهایی به گیرنده‌ها و اجرا کننده‌های فرمان (Actuators) ارسال می‌کند.

به عبارت دیگر PLC عبارت از یک کنترل کنندهٔ منطقی است که می‌توان منطق کنترل را توسط برنامه برای آن تعریف نمود و در صورت نیاز، به راحتی آن را تغییر داد.

وظیفهٔ PLC قبلاً بر عهدهٔ 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 باوجوداین، 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 ایدهٔ مورد نظر هیچ‌گاه به مرحلهٔ 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 منحصر به پروسه و فرآیند خاصی نیست و با تغییر برنامه می‌توان به‌آسانی

از آن برای کنترل پروسه‌های دیگر استفاده نمود.

۷- طراحی و اجرای مدارهای کنترل و فرمان با استفاده از PLCها 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 سابقهٔ آشنایی 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- PLCها می‌توانند با استفاده از برنامه‌های مخصوص، وجود نقص و اشکال در پروسهٔ تحت کنترل را به سرعت تعیین و اعلام نمایند.

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

جدول ۲-۱ : مزایای PLC نسبت به کنترل کننده‌های دیگر

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

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 موردنظر، 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 محدودهٔ 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 وسیلهٔ قابل انعطاف به سرعت می‌توانند نیازمندیهای مصرف‌کنندگان خود را تأمین و در اسرع وقت تواناییهای خود را با نیازمندیهای بازار هماهنگ نمایند.

از شرکت‌های سازندهٔ PLC می‌توان SIEMENS ، AEG ، ALLEN BRADLEY ، OMRON ، MITSUBISHI و ... را نام برد. گر چه از عرضهٔ PLC توسط سازندگان مختلف چند ده سالی می‌گذرد و در ماشین آلات و خطوط تولید خریداری شده از خارج کشور نیز به وفور مشاهده می‌شود استفاده از این وسیلهٔ 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 تأمین کنندهٔ خارجی.

۲- عدم دسترسی به موقع به اطلاعات سیستم‌ها.

۳- عدم پشتیبانی مؤثر سازندگان از تجهیزات فروخته شدهٔ خود.

۴- هزینهٔ بالای تجهیزات خارجی.

۵- هزینهٔ بالای آموزش در خارج از کشور.

شرکت‌های داخلی نیز با توجه به مشکلات یاد شده و برای پر کردن خلاء موجود اقدام به

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

از شرکت‌های داخلی تولید کنندهٔ 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 شبیه سازها 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 برنامهٔ 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 می‌توان S5 ، FST ، OMRON ، CSTL ، ALLEN BRADLEY و ... را نام برد.

۲-۳- کاربرد PLC در صنایع مختلف

امروزه کاربرد PLC در صنایع و پروسه‌های مختلف صنعتی به وفور به چشم می‌خورد. در زیر تعدادی از این کاربردها آورده شده است.

- صنایع اتومبیل‌سازی ـ شامل: عملیات سوراخکاری اتوماتیک، اتصال قطعات و همچنین تست قطعات و تجهیزات اتومبیل، سیستم‌های رنگ پاش، شکل دادن بدنه به وسیلهٔ پرس‌های اتوماتیک و ...

- صنایع پلاستیک سازی ـ شامل: ماشین‌های ذوب و قالب‌گیری تزریقی، دمش هوا و سیستم‌های تولید و آنالیز پلاستیک و ...

- صنایع سنگین ـ شامل: کوره‌های صنعتی، سیستم‌های کنترل دمای اتوماتیک، وسایل و تجهیزاتی که در ذوب فلزات استفاده می‌شوند و ...

- صنایع شیمیایی ـ شامل: سیستم‌های مخلوط کننده، دستگاههای ترکیب کنندهٔ مواد با نسبتهای متفاوت و ...

- صنایع غذایی ـ شامل: سیستم‌های سانتریفوژ، سیستم‌های عصاره‌گیری و بسته‌بندی و ...

- صنایع ماشینی ـ شامل: صنایع بسته‌بندی، صنایع چوب، سیستم‌های سوراخکاری، سیستم‌های اعلام خطر و هشدار دهنده، سیستم‌های استفاده شده در جوش فلزات و ...

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

- سیستم‌های حمل و نقل ـ شامل: جرثقیل‌ها، سیستم‌های نوار نقاله، تجهیزات حمل و نقل و ...

- صنایع تبدیل انرژی (برق، گاز و آب) شامل: ایستگاههای تقویت فشار گاز، ایستگاههای تولید نیرو، کنترل پمپ‌های آب، سیستم‌های تصفیهٔ آب و هوای صنعتی، سیستم‌های تصفیه و بازیافت گاز و ...

۲-۴- سخت‌افزار PLC

از لحاظ سخت‌افزاری می‌توان قسمت‌های تشکیل دهندهٔ 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- ترمینال‌های ورودی (Input Module)

۵- ترمینال‌های 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 می‌کند. این منبع معمولاً از ولتاژهای ۲۴ ولت DC و ۱۱۰ یا ۲۲۰ ولت AC ، ولتاژ ۵ ولت DC را ایجاد می‌کند. ماکزیمم جریان قابل دسترسی منطبق با تعداد مدول‌های خروجی مصرفی است. لازم به ذکر است که ولتاژ منبع تغذیه باید کاملاً تنظیم شده (رگوله)[1] باشد. جهت دستیابی به راندمان بالا معمولاً از منابع تغذیهٔ سوئیچینگ استفاده می‌شود. ولتاژی که در اکثر PLCها استفاده می‌گردد ولتاژ ۵ یا ۵/۲ ولت DC است.(در برخی موارد، منبع تغذیه و واحد کنترل شونده در فاصلهٔ زیادی نسبت به یکدیگر قرار دارند بنابراین ولتاژ منبع، ۵/۲ ولت انتخاب می‌شود تا افت ولتاژ حاصل از بُعد مسافت بین دو واحد مذکور جبران گردد.)

برای تغذیهٔ رله‌ها و محرک‌ها (Actuator) معمولاً از ولتاژ ۲۴ ولت DC به صورت مستقیم (بدون استفاده از هیچ کارت ارتباطی) استفاده می‌شود. در برخی موارد نیز از ولتاژهای ۱۱۰ یا ۲۲۰ ولت AC با استفاده از یک کارت رابط به نام Relay Board استفاده می‌گردد. (در مورد تغذیهٔ رله‌ها

1 - Regulated Voltage

احتیاج به رگولاسیون دقیق نیست.)

در برخی شرایطِ کنترلی لازم است تا در صورت قطع جریان منبع تغذیه، اطلاعات موجود در حافظه و همچنین محتویات شمارنده‌ها، تایمرها و فلگ‌های[1] پایدار بدون تغییر باقی بمانند. در این موارد از یک باطری جنس "Lithium" جهت حفظ برنامه در حافظه استفاده می‌گردد. به این باطری "Battery Back up" می‌گویند. ولتاژ این نوع باطری‌ها معمولاً ۲/۸ ولت تا ۳/۶ ولت می‌باشد. از آنجایی که این باطری نقش مهمی در حفظ اطلاعات موجود در حافظه دارد در اکثر PLC ها یک چراغ نشان دهنده تعبیه شده و در صورتی که ولتاژ باطری به سطحی پائین‌تر از مقدار مجاز ۲/۸ ولت برسد این نشان دهنده روشن می‌گردد. این نشان‌دهنده به Battery Low LED معروف است. در صورت مشاهدهٔ روشن شدن این نشان دهنده لازم است که باطری مذکور تعویض گردد. برای تعویض باطری ابتدا باید به وسیله یک منبع تغذیه، ولتاژ مدول مورد نظر را تأمین و سپس اقدام به تعویض باطری نمود.

۲-۴-۲- واحد پردازش مرکزی (CPU)

CPU یا واحد پردازش مرکزی در حقیقت قلب PLC است. وظیفهٔ 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 برنامهٔ کنترل در آن ذخیره می‌شوند. علاوه بر این، سیستم عامل[2] که عهده‌دار مدیریت کلی بر PLC است در حافظه قرار دارد. تمایز در عملکرد PLCها، عمدتاً به دلیل برنامهٔ سیستم عامل و طراحی خاص CPU

---

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

2 - Operating System

آنهاست. در حالت کلی در PLCها دو نوع حافظه وجود دارد:

۱- حافظهٔ موقت (RAM) که محل نگهداری فلگ‌ها، تایمرها، شمارنده‌ها و برنامه‌های کاربر[1] است.

۲- حافظهٔ دائم (EPROM, EEPROM) که جهت نگهداری و ذخیرهٔ همیشگی برنامهٔ کاربر استفاده می‌گردد.

۲-۴-۴- ترمینال ورودی (Input Module)

این واحد، محل دریافت اطلاعات از فرآیند یا پروسهٔ تحت کنترل می‌باشد. تعداد ورودی‌ها در PLC های مختلف، متفاوت است. ورودی‌هایی که در سیستم‌های PLC مورد استفاده قرار می‌گیرند در حالت کلی به صورت زیر می‌باشند:

الف) ورودی‌های دیجیتال (Digital Input)

ب) ورودی‌های آنالوگ (Analog Input)

الف) ورودی‌های دیجیتال یا گسسته

این ورودی‌ها که معمولاً به صورت سیگنال های صفر یا ۲۴ ولت DC می‌باشند، گاهی برای پردازش توسط CPU به تغییر سطح ولتاژ نیاز دارند. معمولاً برای انجام این عمل مدول‌هایی خاص در PLC در نظر گرفته می‌شود. جهت حفاظت مدارات داخلی PLC از خطرات ناشی از اشکالات بوجود آمده در مدار یا برای جلوگیری از ورود نویزهای موجود در محیط‌های صنعتی ارتباط ورودی‌ها با مدارات داخلی PLC توسط کوپل کننده‌های نوری[2] (Optical Coupler) انجام می‌گیرد. به دلیل ایزوله شدن ورودی‌ها از بقیهٔ اجزای مدار داخلی PLC، هر گونه اتصال کوتاه و یا اضافه ولتاژ نمی‌تواند آسیبی به واحدهای داخلی PLC وارد آورد.

ب) ورودی‌های آنالوگ یا پیوسته

این گونه ورودی‌ها در حالت استاندارد ۱۰ V DC ± - ۰، ۲۰ mA - ۴ و یا ۲۰ mA - ۰

---

1 - User Programs

۲ - طرز کار این عنصر الکترونیکی در ضمیمهٔ 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- ترمینال خروجی (Output Module)

این واحد، محل صدور فرمانهای PLC به پروسهٔ 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 ۰ - ± ۱۰ V DC، ۴-۲۰ mA و یا ۰-۲۰ mA باشد. معمولاً مدول‌های خروجی آنالوگ، مقادیر دیجیتال پردازش شده توسط CPU را به سیگنال های پیوستهٔ (آنالوگ) مورد نیاز جهت پروسهٔ تحت کنترل تبدیل می‌نمایند. این خروجی‌ها به وسیلهٔ واحدی به نام Isolator از سایر قسمتهای داخلی PLC ایزوله می‌شوند. بدین ترتیب مدارات حساس داخلی PLC از خطرات ناشی از امکان بروز اتصالات ناخواستهٔ خارجی محافظت می‌گردند.

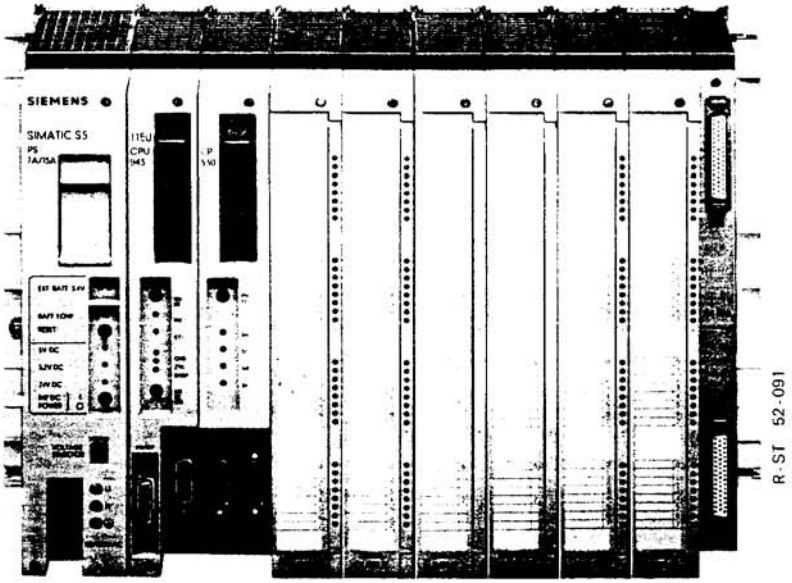
۲-۴-۶- مدول ارتباط پروسسوری (CP)

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

۲-۴-۷- مدول رابط (IM)

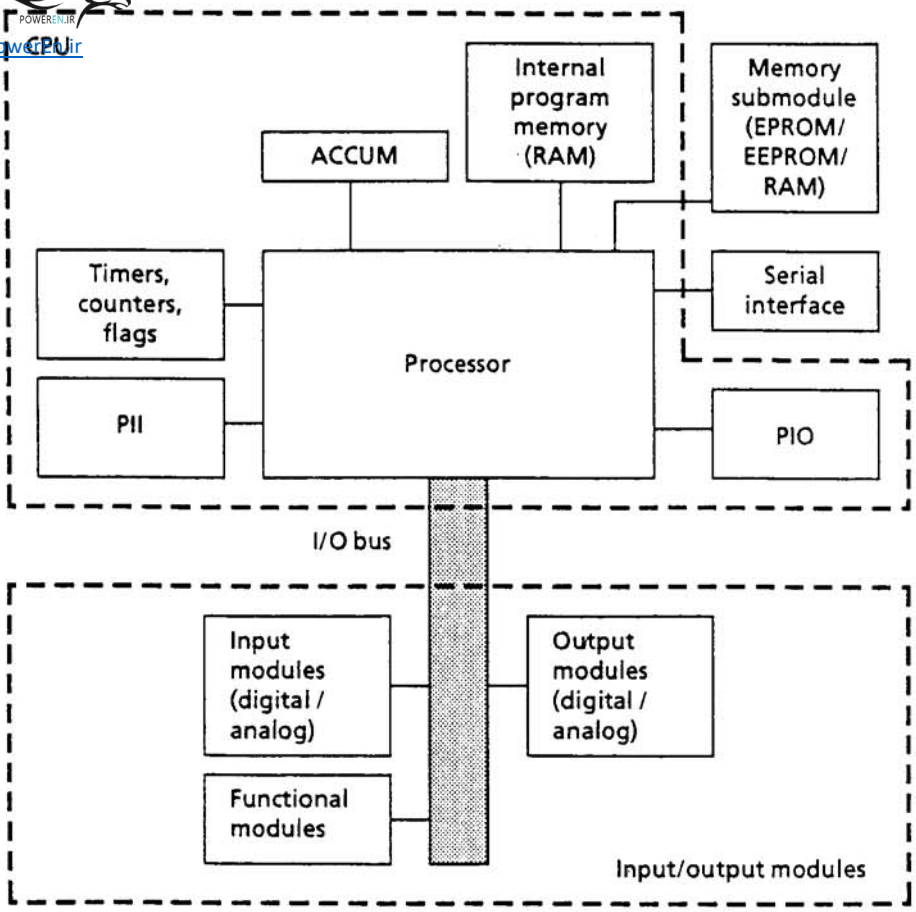
در صورت نیاز به اضافه نمودن واحدهای دیگر ورودی و خروجی به PLC یا جهت اتصال پانل اپراتوری و پروگرامر به PLC از این مدول ارتباطی استفاده می‌شود. در صورتی که چندین PLC به صورت شبکه به یکدیگر متصل شوند از واحد IM جهت ارتباط آنها استفاده می‌گردد.

در شکل ۲-۱ شمای کلی یک PLC نشان داده شده است.



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

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



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

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

۲-۵- تصویر ورودی‌ها (PII)[1]

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

1 - Process Image Input

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)[1]

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 دستهٔ 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 PLC های مختلف متفاوت می‌باشد اما تقریباً در تمامی

---

1 - Process Image Output

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 $\frac{m}{2}$ و $\frac{n}{2}$ و $\frac{p}{2}$ می‌باشد. المان‌های که شمارهٔ آنها از مقادیر نصف یعنی $\frac{m}{2}$ و $\frac{n}{2}$ و $\frac{p}{2}$ کوچکتر باشد پایدار و بقیه، عناصر ناپایدار هستند. به‌طور کلی می‌توان گفت که نیمهٔ اول این عناصر، پایدار و نیمهٔ دوم ناپایدار می‌باشد.

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

۲-۸- انبارک یا آکومولاتور (ACCUM)[1]

انبارک یا آکومولاتور یک ثبات منطقی است که جهت بارگذاری یا به عبارت دیگر لود نمودن[2] 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)

همان گونه که قبلاً ذکر شد وظیفهٔ پردازش اطلاعات در PLC برعهدهٔ 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 کنندهٔ 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)

---

1 - Accumulator 2 - Load

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 آدرس‌دهی

جهت آدرس‌دهی معمولاً از سه روش زیر استفاده می‌شود.

۱- Fixed Address : در این روش تمام ورودی‌ها و خروجی‌ها دارای آدرس ثابتی می‌باشند. نظیر این نوع آدرس‌دهی را در مینی PLCهای کنترونیک می‌توان یافت.

۲- Slot Address : در این روش، آدرس‌دهی قابل تغییر می‌باشد و این تغییر آدرس توسط شیارهای مورد نظر و فیش‌های زائده‌دار انجام می‌گیرد.

۳- Flexible Address : در این روش آدرس‌دهی که قابل تغییر نیز می‌باشد سوئیچ‌هایی (دیپ سوئیچ) در نظر گرفته شده که با استفاده از آنها می‌توان آدرس‌دهی را تغییر داد.

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

۲-۱۱- نرم‌افزار PLC

در PLCها سه نوع نرم‌افزار قابل تعریف است:

۱- نرم‌افزاری که کارخانهٔ سازنده با توجه به توان سخت‌افزاری سیستم تعریف می‌کند که به آن Operating System یا به اختصار OS گویند. مثلاً در PLC زیمنس مدل 100 U تعداد ۱۶ تایمر (T0 - T15) تعریف شده است و اگر در برنامه‌نویسی از تایمر شمارهٔ 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 نرم‌افزار ثابت بوده، قابل تغییر نمی‌باشد بنابراین از نوع فقط خواندنی است و معمولاً در EPROM یا E^2PROM ذخیره می‌شود.

۲- نرم‌افزاری که برنامهٔ نوشته شده توسط استفاده کننده (User) را به زبان قابل فهم ماشین تبدیل می‌نماید. این برنامه منحصر به کارخانهٔ سازنده بوده، نام خاصی نیز دارد. معروف‌ترین و پر کاربردترین این نرم‌افزارها، نرم‌افزار S5 می‌باشد که توسط شرکت زیمنس ابداع گردیده است. این نرم‌افزار هم مانند OS قابل تغییر نیست و بایستی در ROM ذخیره و برای اجرا به RAM پروگرامر ارسال گردد.

۳- نرم‌افزار یا برنامه‌ای که توسط استفاده کننده نوشته می‌شود و به آن User Program گویند. این نرم‌افزار در هر لحظه قابل تغییر بوده، خواندنی / نوشتنی است. این نرم‌افزار در RAM و یا در EPROM و یا در E^2PROM ذخیره و در صورت ایجاد هرگونه اشکال در RAM از مدول ذکر شده مجدداً در RAM کپی شده، اجرا می‌گردد.

همان گونه که ذکر شد هر PLC شامل سخت‌افزار و نرم‌افزار می‌باشد. در صفحات گذشته به‌طور اجمال به توضیح در مورد سیستم‌های سخت‌افزاری و همچنین نرم‌افزار PLC پرداختیم. واضح است که برای وارد کردن برنامهٔ کنترلی یا نرم‌افزار کنترلی به سخت‌افزار، نیاز به یک واحد برنامه‌نویسی یا پروگرامر می‌باشد. در ادامهٔ بحث به تشریح واحد برنامه‌نویسی (Programming Unit) می‌پردازیم.

۲-۱۲- واحد برنامه‌نویسی (PG)

در استفاده و به کارگیری PLC علاوه بر آشنایی با نحوهٔ کار، آشنایی با واحد برنامه‌نویسی آن نیز ضروری است زیرا توسط این واحد قادر خواهیم بود با PLC ارتباط برقرار نماییم. به این ترتیب که برنامهٔ کنترل دستگاه را نوشته، آن را در حافظهٔ PLC قرار داده، اجرای آن را از PLC می‌خواهیم. این واحد بسیار شبیه به کامپیوترهای معمولی است، یعنی دارای یک صفحهٔ نشان دهنده (مونیتور) و صفحه کلید می‌باشد. تفاوت این واحد با کامپیوتر معمولی، تک منظوره بودن آن می‌باشد بدین معنی که از PG تنها می‌توان جهت ارتباط برقرار نمودن با PLC مربوطه استفاده نمود.

با استفاده از PG می‌توان از وضعیت و چگونگی اجرای برنامه مطلع شد. صفحهٔ 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 نحوهٔ اجرای برنامه در صفحهٔ 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 برنامهٔ کنترل دستگاهها و سیستم‌های تحت کنترل و بررسی علت توقف آنها نقش به سزایی دارد. به وسیلهٔ PG می‌توان تغییرات عملوندها یعنی ورودی‌ها، خروجی‌ها و همچنین تایمرها و شمارنده‌های برنامهٔ 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 برنامهٔ کنترلی مجموعهٔ دستورالعمل‌هایی است که به سیستم PLC فرمانهایی جهت کنترل پروسه صادر می‌کند. بنابراین باید این برنامه به زبانی خاص و مطابق با قوانین و دستورات قابل درک برای PLC نوشته شود. زبان برنامه‌نویسی که خانوادهٔ SIEMENS از آن استفاده می‌کند STEP 5 (S5) نامیده می‌شود.

۳-۱- اشکال مختلف نمایش برنامه‌ها

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

۱- نردبانی (Ladder) LAD

۲- فلوچارتی (Control System Flowchart) CSF

۳- عبارتی (Statement List) STL

---

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

۳-۱-۱- روش نمایش نردبانی (LAD)

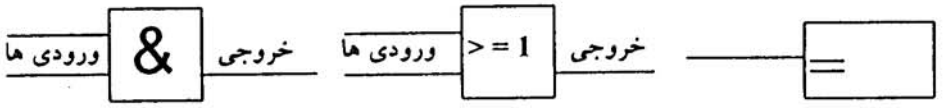
در نمایش نردبانی، هر دستور یا خط برنامه به صورت نماد اتصال و سیم‌پیچ مدارهای فرمان رله‌ای نشان داده می‌شود. در نتیجه ساختار برنامه در این روش تقریباً شبیه به شکل مدارهای فرمان رله‌ای می‌باشد. این طرز نمایش از قدیم در سیستم‌های رله‌ای متداول بوده، نقشه‌های مدار فرمان اکثراً به این روش ترسیم می‌شوند. به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

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

شکل ۳-۱ : برخی نمادهای مورداستفاده در نمایش LAD

۳-۱-۲- روش نمایش فلوچارتی (CSF)

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



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

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

1 - NO (Normally Open)

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

۳-۱-۳- روش نمایش عبارتی (STL)

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

عبارت یا Statement

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

الف) عملکرد (Operation)

ب) عملوند (Operand)

الف) عملکرد (Operation)

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

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

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

ب) عملوند (Operand)

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

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

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

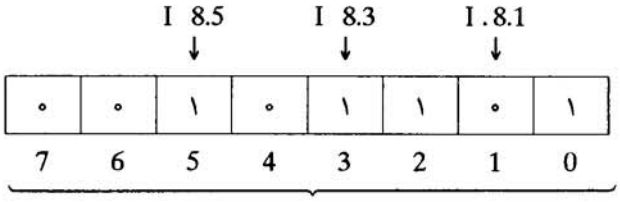
Statement	عبارت	
Operation عملکرد	*Operand*	عملوند
	Operand Identifier نوع عملوند	*Parameter* آدرس عملوند
A	I	1.0
A	I	7.1
=	Q	2.5
O	I	0.6
O	I	8.4
=	Q	1.3

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

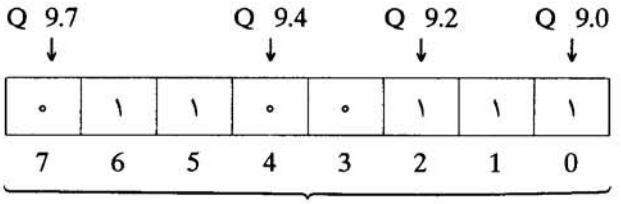
آدرس ورودی‌ها، خروجی‌ها و فلگ‌ها

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

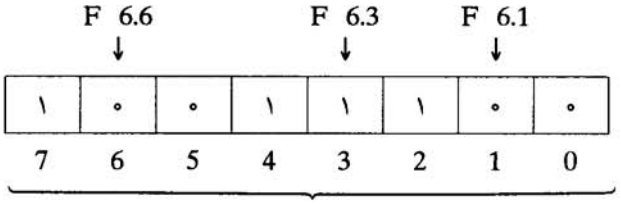
در شکل ۳-۳ نحوهٔ آدرس دهی ورودی، خروجی و فلگ نشان داده شده است.



8 IB[1] (بایت ورودی ۸)



9 QB[2] (بایت خروجی ۹)



6 FY[3] (بایت فلگ ۶)

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

طریقهٔ آدرس دهی را با چند مثال بررسی می‌کنیم:

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

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

توضیح اینکه FB علامت اختصاری Function Block می‌باشد.

روش نمایش STL

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

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

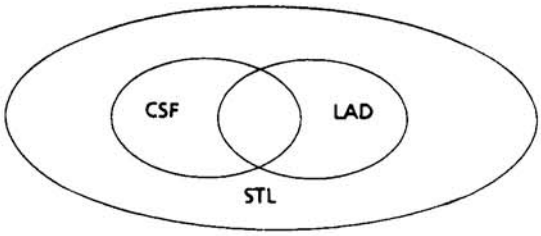
در برنامهٔ نوشته شده به روش STL به چندین سطر که عمل خاصی انجام می‌دهند یک Segment می‌گویند. یک برنامه (Program) از چندین Segment تشکیل می‌گردد. لازم به ذکر است که یک برنامه می‌تواند تنها شامل یک Segment باشد.

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

```
 PB 1

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

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

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



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

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

برنامه نوشته شده به روش STL را در نظر بگیرید.

```
 PB 2

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

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

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

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

۱- به کارگیری پردازنده‌هایی با سرعت و قابلیت بالاتر

۲- استفاده از Structured Programming در برنامه‌نویسی

۳-۳- برنامه‌نویسی سازمان‌یافته (Structured Programming)

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

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

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

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

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

۳-۳-۱- بلوک‌های برنامه (PB)[1]

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

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

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

۳-۳-۲- بلوک‌های ترتیبی (SB)[2]

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

۱- کنترل دستی (Hand یا Manual)

۲- کنترل اتوماتیک (Auto)

۳- کنترل تک‌مرحله‌ای (Single Step یا Inching)

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

۳-۳-۳- بلوک‌های تابع‌ساز (FB)[3]

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

---

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

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

هر FB از دو قسمت تشکیل شده است:

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

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

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

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

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

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

۳-۳-۴- بلوک‌های اطلاعاتی (DB)[1]

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

---

1 - Data Block

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

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

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

۳-۳-۵- بلوک‌های سازماندهی (OB)[2]

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

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

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

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

---

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

2 - Organization Block

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

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

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

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

در زبان برنامه‌نویسی S5 عملوندهای زیر مورد استفاده قرار می‌گیرند.

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

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

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

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

T (Timers) تایمرها، حافظه‌ای جهت تخصیص زمان‌سنج‌ها.

C (Counters) شمارنده‌ها، حافظه‌ای جهت تخصیص شمارشگرها.

K (Constants) ثابت‌ها.

۳-۵- دستورالعمل‌های زبان S5[1]

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

---

۱ - دستورات زبان S5 در ضمیمهٔ ۲ آمده است.

۱- دستورالعمل‌های اصلی (Basic)

۲- دستورالعمل‌های تکمیلی (Supplementary)

۳- دستورالعمل‌های سیستم (System)

۳-۵-۱- دستورالعمل‌های اصلی (Basic)

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

۳-۵-۲- دستورالعمل‌های تکمیلی (Supplementary)

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

۳-۵-۳- دستورالعمل‌های سیستم (System)

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

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

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

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

۳-۶- خواندن صفر (Scanning for Zero)

برای درک این مطلب به ذکر یک مثال می‌پردازیم.

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

0	0	0	1	0	1	0	0

با توجه به بیت‌های قرار داده شده در این بایت این مطلب استنباط می‌گردد که پردازنده تمام بیت‌ها را اعم از "۰"ها و "۱"ها خوانده و عدد ۰۰۰۱۰۱۰۰ را به عنوان یک بایت ورودی دریافت می‌کند. حال اگر برنامه‌نویس بخواهد با بیت "۰" حافظه، عملی انجام دهد (به عنوان مثال آن را از حافظه بخواند) باید از دستور ... AN (AND NOT) در روش STL و از نماد -|/|- در روش LAD و از -|0|- در روش CSF استفاده نماید. به این عمل در اصطلاح خواندن صفر یا Scanning for Zero می‌گویند.

۳-۷- کنتاکت در حالت عادی باز (NO)[1]

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

۳-۸- کنتاکت در حالت عادی بسته (NC)[2]

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

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

---

1 - Normally Open Contact 2 - Normally Close Contact

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

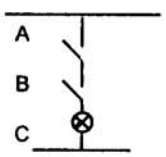
جدول ۳-۴ : نحوهٔ نمایش کنتاکت‌های NO و NC در روشهای مختلف برنامه‌نویسی

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

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

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

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



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

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

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

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

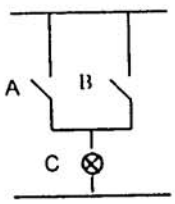
```
 PB 3

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

```
 PB 3

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

```
 PB 3

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

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

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



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

```
 PB 4

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

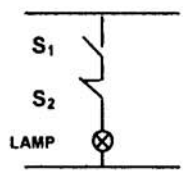
```
 PB 4

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

```
 PB 4

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

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

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

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



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

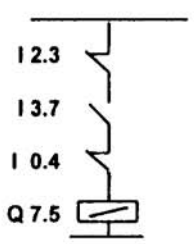
```
 PB 5

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

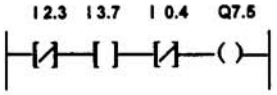
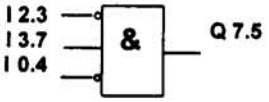
```
 PB 5

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

```
 PB 5

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

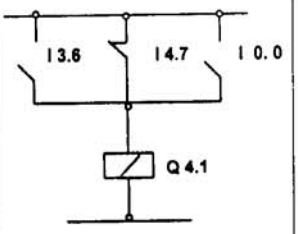
در مثالهای بعدی به پیچیدگی برنامه اضافه نموده و از ذکر جزئیات خودداری می‌کنیم.

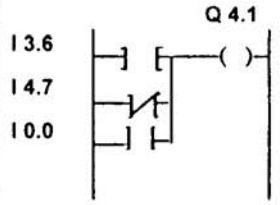
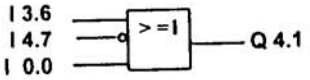
مثال ۳-۴: در این مثال دوباره قصد داریم تا چگونگی استفاده از کنتاکت‌های NC را در برنامه‌نویسی بررسی نماییم.

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

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

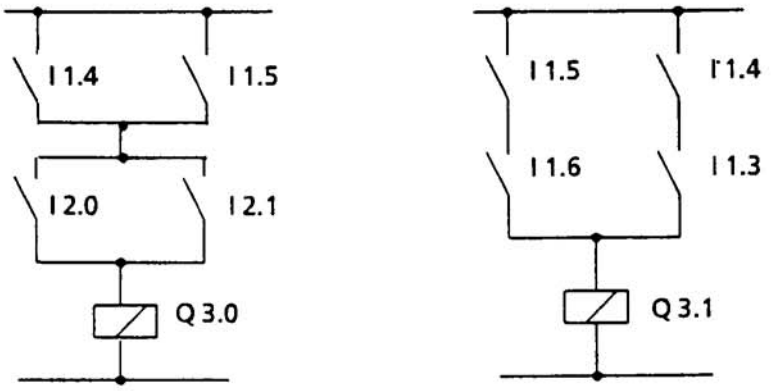
مثال ۳-۵ : در این مثال نیز ترکیب فصلی ورودی‌ها را به همراه کنتاکت‌های NC مورد بررسی قرار می‌دهیم.

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

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

۳-۹- کاربرد پرانتزها در برنامه‌نویسی به روش STL

نمایش مدارهای زیر را در نظر بگیرید.



(ب) (الف)

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

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

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

(AND قبل از OR) AND OR

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

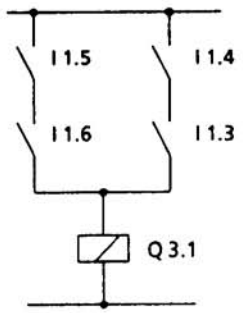
(OR قبل از AND) OR AND

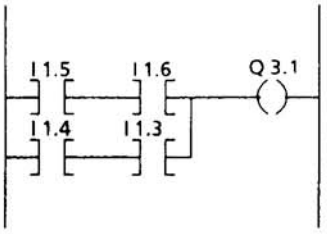
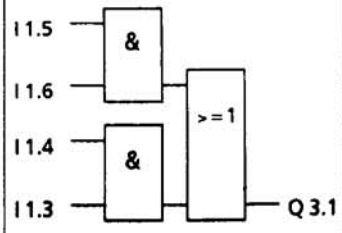
با اندکی دقت در این دو تابع در می‌یابیم که در تابع اول دستور AND قبل از دستور OR استفاده شده است در حالی که در تابع دوم دستور OR قبل از دستور AND به کار برده شده است.

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

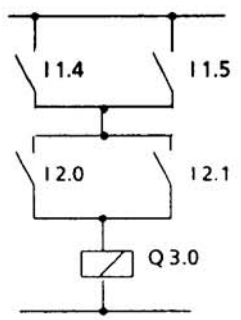
برای روشن شدن مطالب عنوان شده، در دو مثال بعدی، برنامهٔ عملکرد این دو مدار فرمان را بررسی می‌نمائیم.

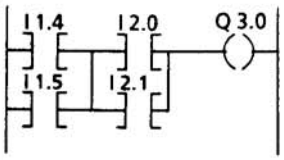
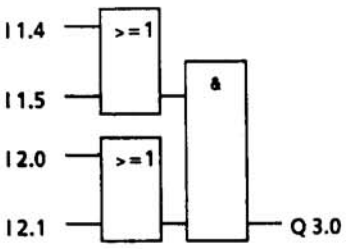
مثال ۳-۶ : در این مثال عدم استفاده از پرانتز در حالتی که دستور AND قبل از دستور OR قرار گرفته باشد مورد بررسی قرار می‌گیرد.

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

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

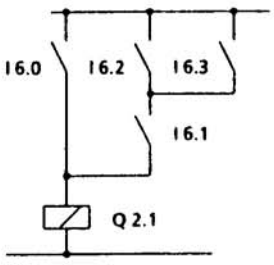
مثال ۳-۷: در این مثال چگونگی استفاده از پرانتز در حالتی که دستور OR قبل از دستور AND قرار گرفته باشد نشان داده می‌شود.

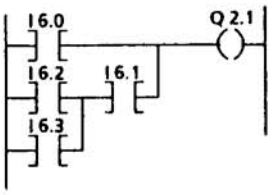
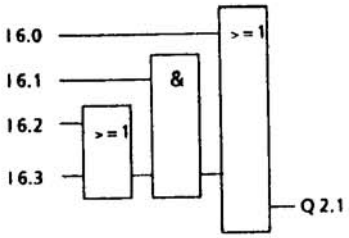
نماد مداری	توضیح
	خروجی Q 3.0 برابر "۱" خواهد بود اگر حاصل هر دو تابع OR برابر "۱" شود. این خروجی برابر "۰" خواهد بود در صورتی که حاصل حداقل یکی از دو تابع OR "۰" شود.

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

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

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

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

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

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

۳-۱۰- فلگ یا پرچم (Flag)

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

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

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

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

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

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

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

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

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

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

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

```
 PB 11

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

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

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

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

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

۳-۱۱- بیت RLO[1]

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

۳-۱۲- ست و ری‌ست در فلگ‌ها و خروجی‌ها

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

---

1 - Result of Logic Operation

۱- فلیپ‌فلاپ SR

۲- فلیپ‌فلاپ RS

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

فلیپ‌فلاپ SR فلیپ‌فلاپ RS

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

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

فلیپ‌فلاپ SR

```
PB 16

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

فلیپ‌فلاپ RS

```
PB 17

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

طرز کار فلیپ‌فلاپ SR به صورت زیر است:

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

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

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

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

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

```
 PB 18

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

```
 PB 18

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

```
 PB 18

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

مبحث فلیپ‌فلاپ‌ها را با پاسخ به یک سؤال ادامه می‌دهیم.

- تفاوت بین این دو برنامه در چیست؟

```
 PB 19                              PB 20

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

(الف) (ب)

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

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

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

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

۳-۱۳- دستور NOP 0[1]

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

```
 PB 21

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

 PB 21

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

1 - No Operation Zero

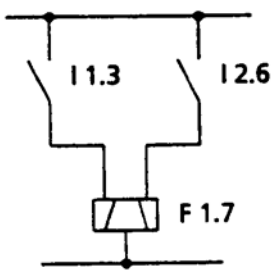
حال در صورتی که بخواهیم از خروجی، هیچ استفاده‌ای نکنیم یا به عبارت دیگر سطرهای پنجم و ششم از برنامهٔ نوشته شده به روش STL را حذف نماییم به گونه‌ای که مقادیر ورودی در ترمینال‌های R و S به خروجی منتقل نشوند از دستور NOP 0 استفاده می‌نماییم و با استفاده از این دستور برنامهٔ قبلی را بازنویسی می‌کنیم.

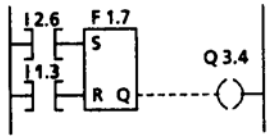
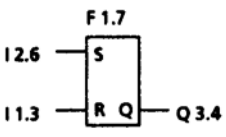
```
 PB 22

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

```
 PB 22

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

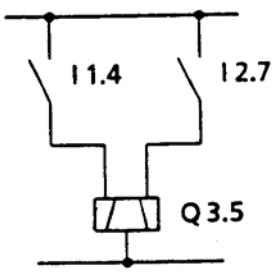
دستور NOP 0 دستوری مختص PLCهای زیمنس می‌باشد. با استفاده از این دستور در برنامه‌نویسی به روش STL (تنها در صورت عدم استفاده از بخشی از برنامه می‌توان دستور NOP 0 را جایگزین آن بخش از برنامه نمود) می‌توان برنامهٔ نوشته شده به روش STL را به LAD و CSF تبدیل نمود. در آینده از این دستور در تعریف تایمرها، شمارنده‌ها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

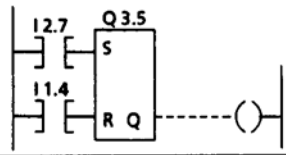
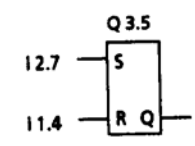
مثال ۳-۹ : در این مثال به بررسی برنامهٔ نوشته شده برای یک فلیپ فلاپ می‌پردازیم.

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

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

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

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

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

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

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

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

```
 PB 25

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

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

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

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


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


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

PB 25

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

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

جدول مربوط به مثال ۳-۱۱

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

ادامهٔ جدول مربوط به مثال ۳-۱۱

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

۳-۱۴- کانکتور (Connector)

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

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

```
 PB 26

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

که ورودی‌ها به صورت زیر تعریف شده‌اند:

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

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

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

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

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

```
 PB 27

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

طرز عملکرد این برنامه به صورت زیر است:

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

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

لبهٔ تیز بوجود آمده در F 100.0

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

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

```
 PB 28

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

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

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

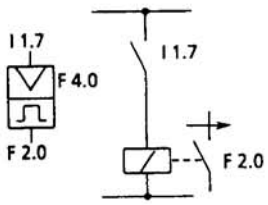
افرادی که در محیط‌های صنعتی با فرآیندهای صنعتی کوچک و بزرگ سروکار دارند به خوبی می‌دانند که در برخی از سیستم‌ها جهت ری‌ست نمودن تمامی موارد (مواردی نظیر آلارم‌ها و ...) از کلیدی با نام General Reset و یا Common Reset استفاده می‌گردد.

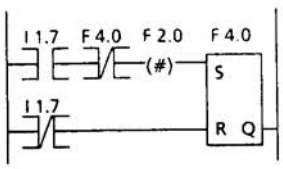
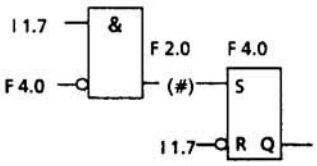
در ادامه، این دو برنامه به روش STL آورده شده‌اند.

```
 PB 27

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

```
PB 28

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

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

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

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

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

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

```
 PB 30

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

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

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

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

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

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

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

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

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

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

```
 PB 31

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

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

```
 PB 31

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

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

```
 PB 31

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

حال به تشریح عملکرد این برنامه می‌پردازیم:

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

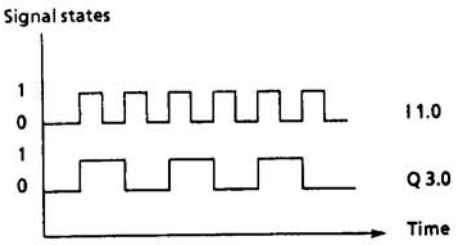
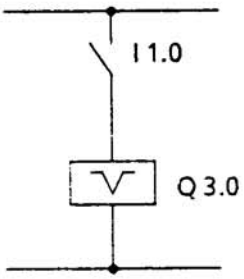
با رها کردن پوش باتن ورودی، I 0.0 "۰" خواهد شد و ترمینال S فلیپ‌فلاپ موجود در SEGMENT 2 فعال می‌شود. (زیرا ترمینال S این فلیپ‌فلاپ حاصل ترکیب عطفی F 8.0 و نقیض I 0.0 می‌باشد و از آنجایی که F 8.0 در مرحلهٔ قبل فعال شده بود حاصل ترکیب عطفی نقیض I 0.0 و F 8.1 برابر "۱" خواهد بود.) در این مرحله این فلیپ‌فلاپ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

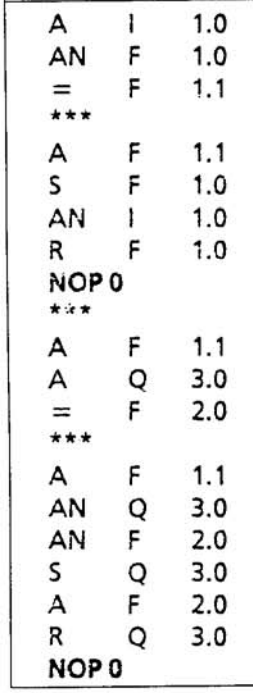
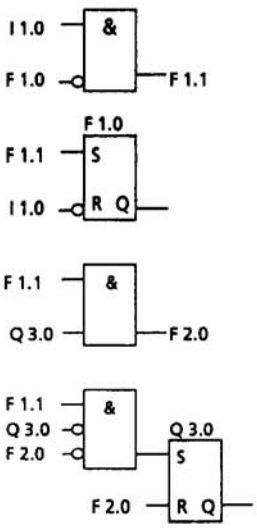
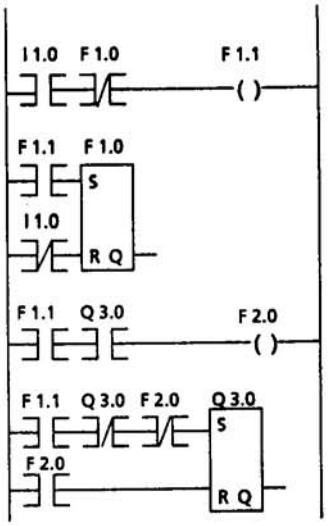
داشت: "۱" = F 8.1. در این حالت در صورتی که پوش باتن را مجدداً فشار دهیم، I 0.0 فعال شده و ترمینال R فلیپ‌فلاپ موجود در SEGMENT 1 نیز فعال می‌گردد (این ترمینال حاصل ترکیب عطفی F 8.1 و I 0.0 است که در این مرحله مقدار هر دو "۱" بوده، در نتیجه ترمینال R نیز "۱" خواهد شد) که موجب ری‌ست شدن این فلیپ‌فلاپ می‌گردد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

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

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

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

نمودار زمانی	نماد مداری
	

STL	CSF	LAD
		

۳-۱۶- دستور پرش غیرشرطی (JU)[1]

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

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

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

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

۳-۱۷- دستور پرش شرطی (JC)[2]

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

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

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

---

1 - Jump Unconditional 2 - Jump Conditional

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

```
OB 1

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


```
PB 18

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

PB 19

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

حال به توصیف عملکرد برنامه می‌پردازیم.

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

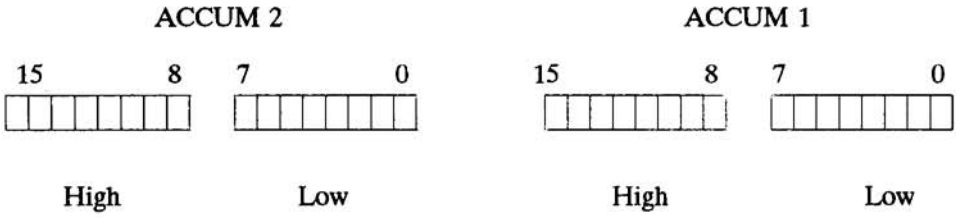
به PB 19 انجام نخواهد شد. سپس پردازنده به دستور BE در بلوک OB 1 رسیده، برنامه پایان یافته تلقی می‌شود. در اینجا مجدداً پردازنده به سطر اول بلوک OB 1 مراجعه نموده و در صورت فعال بودن I 0.0 همین روند مجدداً تکرار می‌شود. اما در صورتی که I 0.0 غیرفعال یعنی برابر "۰" باشد بیت RLO نیز "۰" است و دستور پرش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

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

۳-۱۸- دستورهای بارگذاری و انتقال

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

پائین (Low) می‌باشد.



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

۳-۱۸-۱- دستور L[1]

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

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

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

---

1 - Load

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

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

و اطلاعات ACCUM 2 دور ریخته می‌شود.

۳-۱۸-۲- دستور T[1]

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

T QW[3] 8 ، T FW[2] 52

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

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

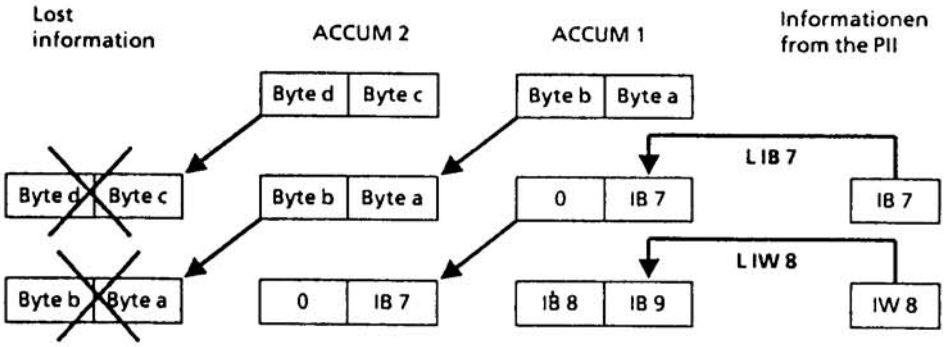
در شکل ۳-۸ چگونگی اجرای چندین دستور بارگذاری و انتقال و نحوهٔ جابجایی اطلاعات ذخیره شده در انبارک‌ها و همچنین اطلاعاتی که ممکن است در صورت وجود چندین دستور بارگذاری دور ریخته شوند نشان داده شده است.

---

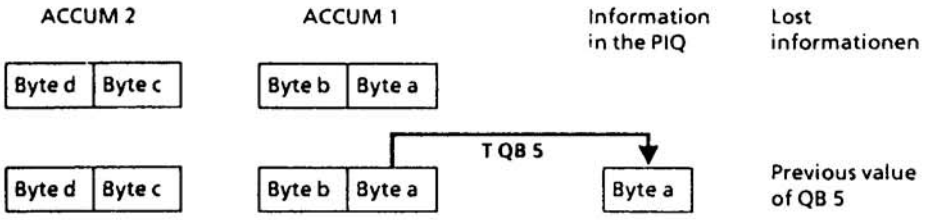
1 - Transfer 2 - FW: Flag Word 3 - QW: Output Word

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

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



(الف)

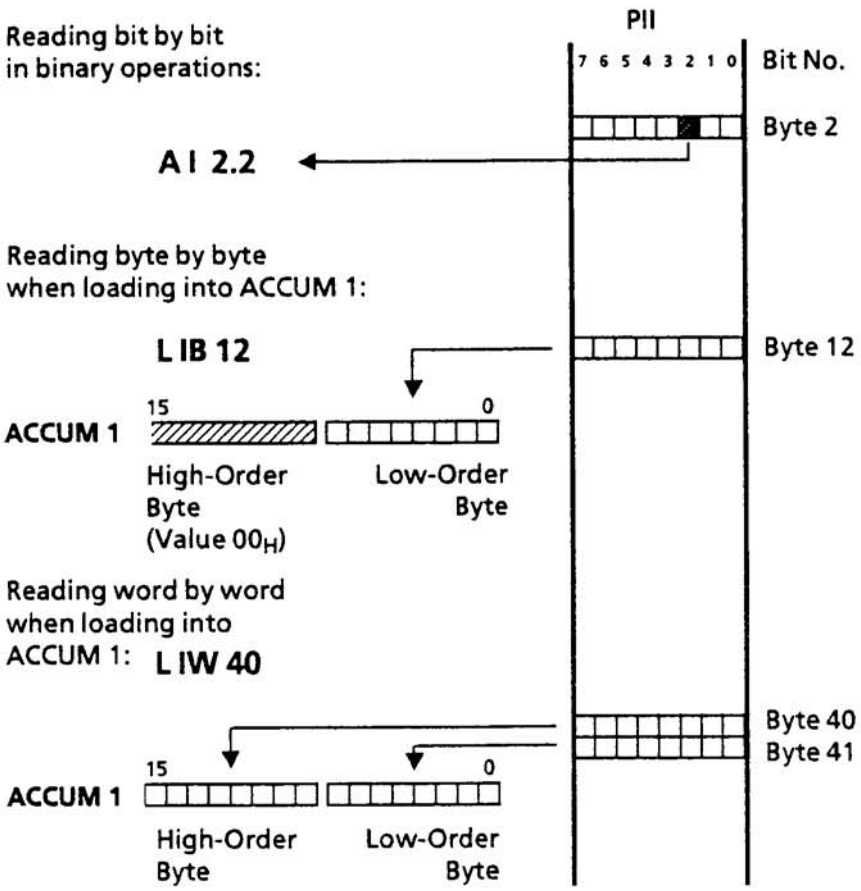


(ب)

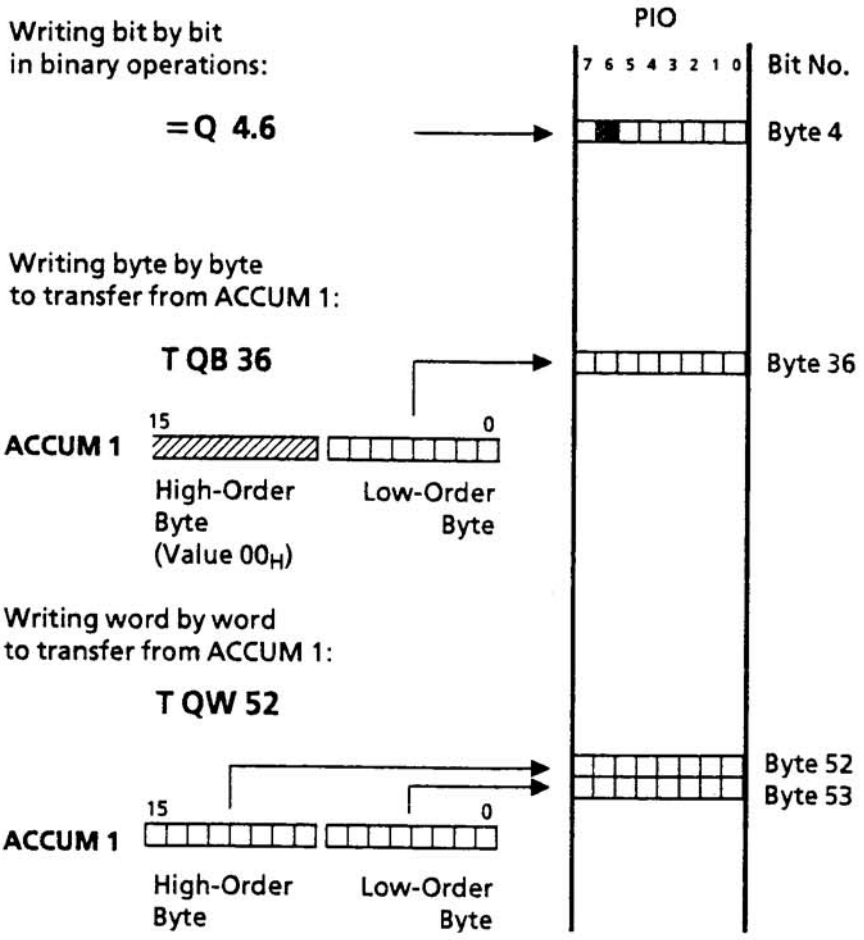
شکل ۳-۸ : چگونگی اجرای دستورات بارگذاری از PLC و انتقال به PIO

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

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



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



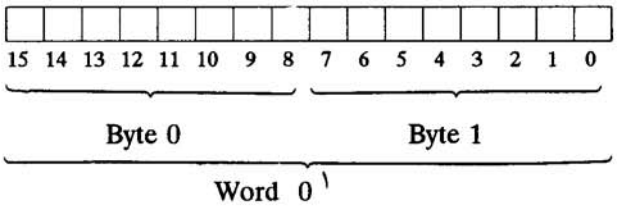
شکل ۳-۱۰ : نحوهٔ انتقال اطلاعات به صورت بیتی، بایتی و کلمه‌ای به PIO

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

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

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

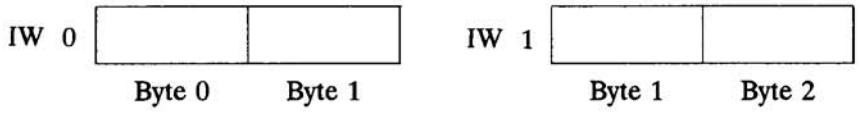
در صورت اجرای این دستور، بایت‌های ۰ و ۱ به داخل انبارک ۱ بارگذاری می‌شوند. بدین ترتیب که بایت ۰ در بایت باارزش بالا و بایت ۱ در بایت باارزش پائین جای می‌گیرد. در شکل زیر نحوهٔ بارگذاری اطلاعات به داخل انبارک ۱ نشان داده شده است.



[1]

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

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



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

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

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

---

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

2 - Overlap

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

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

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

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

۳-۲۰- موارد استفادهٔ انبارک‌ها

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

۳-۲۰-۱- دستور جمع دو عدد (+ F)

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

```
 PB 20

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

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

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

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

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

$= ۰۰۱۰۰۰۱۱\ \ ۱۰۰۱۱۰۱۱_{(۲)}$ ⟶ QW 4

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

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

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

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

```
 PB 10

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

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

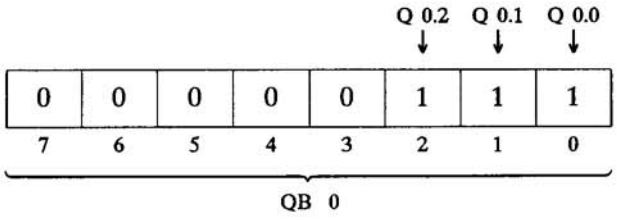
$۲_{(۱۰)} = ۰۰۰۰۰۰۱۰$ + ⟶ IB 4

$۵_{(۱۰)} = ۰۰۰۰۰۱۰۱$ ⟶ IB 12

$= ۰۰۰۰۰۱۱۱$ ⟶ QB 0

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

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



بنابراین داریم:

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

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

نتیجه‌ای که از این مثال به‌دست می‌آید این است که:

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

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

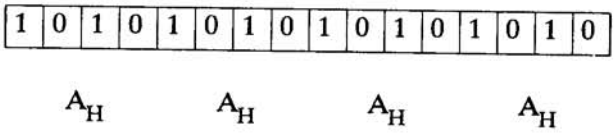
جهت بارگذاری انبارک با اعداد ثابت در مبنای دهدهی از دستور ... L KD استفاده می‌کنیم در حالی که جهت بارگذاری انبارک با عدد ثابتی در مبنای ۱۶ از دستور ... L KH استفاده می‌نمائیم. (به جای نقطه‌چین عدد بارگذاری شده قرار می‌گیرد.)

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

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

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

5_H 5_H 5_H 5_H



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

```
 PB 5

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

```
 PB 8

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

۳-۲۰-۲- دستور تفریق دو عدد (F -)

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

```
 PB 15

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

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

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

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

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

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

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

$$= ۱\times۲^{۴} + ۱\times۲^{۱}$$

$$= ۱۶+۲ = ۱۸$$

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

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

IB 1 : ۰۰۰۰۰۰۰۰ −

IB 2 : ۰۰۰۰۰۰۰۱

?

جمع با مکمل ۲ ←

۰۰۰۰۰۰۰۰ +

مکمل ۱ ← ۱۱۱۱۱۱۱۰

۱۱۱۱۱۱۱۰ +

۱

۱۱۱۱۱۱۱۱ → QB 3

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

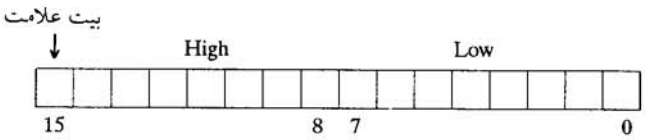
مثال عملی متناظر با تفریق عدد ۱ از ۰ را می‌توان در شمارنده‌های (کانترهای) مکانیکی ضبط‌صوت یافت. (فرض کنید که این شمارنده ۴ رقمی است یعنی از عدد 0000 شروع شده، تا

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

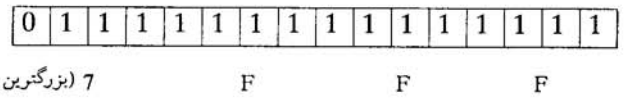
در مورد مثال تفریق عدد ۱ از عدد ۰ نیز حاصل تفریق، تقریباً مشابه این عدد یعنی ۱۱۱۱۱۱۱۱ به‌دست آمد، عدد حاصل منفی است. حال این سؤال مطرح می‌شود که:

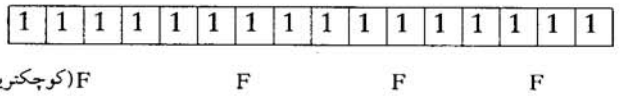
طرز تشخیص علامت اعداد (مثبت یا منفی بودن آنها) چگونه است؟

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



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





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

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

جهت بارگذاری انبارک می‌توان از اعداد BCD نیز استفاده نمود. برای بارگذاری اعداد 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 محدودهٔ این اعداد در فاصلهٔ [7999, 9999-] می‌باشد.

با استفاده از دستورات L و T و بارگذاری اعداد مختلف در مبناهای متفاوت می‌توان صحت ارتباط بین ورودی‌ها و خروجی‌ها[1] را در یک سیستم بررسی نمود. بسیاری از 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 برنامهٔ زیر را در مورد هر دسته جداگانه اجرا کرد، تا اینکه تمام ارتباطات بررسی شوند.

```
 PB 14

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

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

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

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

۳-۲۱- مقایسه کننده‌ها (Comparators)

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

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

---

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

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

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

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

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

مثال ۳-۲۱ :به برنامه PB 25 توجه کنید.

```
 PB 25

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

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

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

```
 PB 36

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

عملکرد این برنامه به شرح زیر است:

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

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

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

```
 PB 1

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


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

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


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


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


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


 PB 1

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

ادامهٔ بلوک PB 1:

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

جدول ۳-۶ : مربوط به مثال ۳-۲۳

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

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

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

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

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

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

```
 PB 29

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

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

در ادامه، همین برنامه به روش STL آمده است.

```
 PB 29

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


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

۳-۲۲- شمارنده‌ها (Counters)

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

```
 PB 33

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

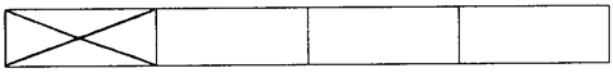
```
 PB 33

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

```
 PB 33

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

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

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

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



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

ورودی R : جهت ری‌ست کردن شمارنده استفاده می‌شود.

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

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

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

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

تغییر وضعیت می‌دهد.

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

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

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

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

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

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

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

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

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

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

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

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

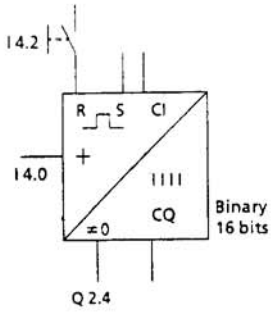
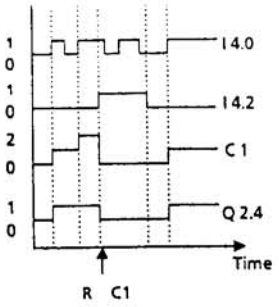
```
:LD   C     4
:T    FW    10
```

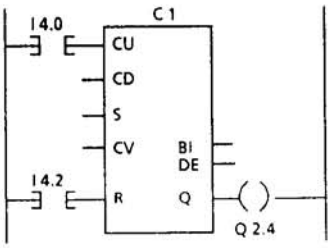
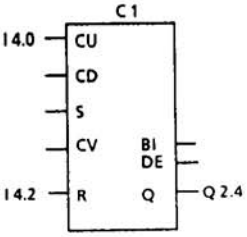
این دو سطر نیز جهت فراخوانی وضعیت شمارنده و ارسال آن به یک بیت خروجی استفاده می‌شوند. در صورت عدم نیاز به این دو سطر از دستور NOP 0 استفاده می‌کنیم.

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

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

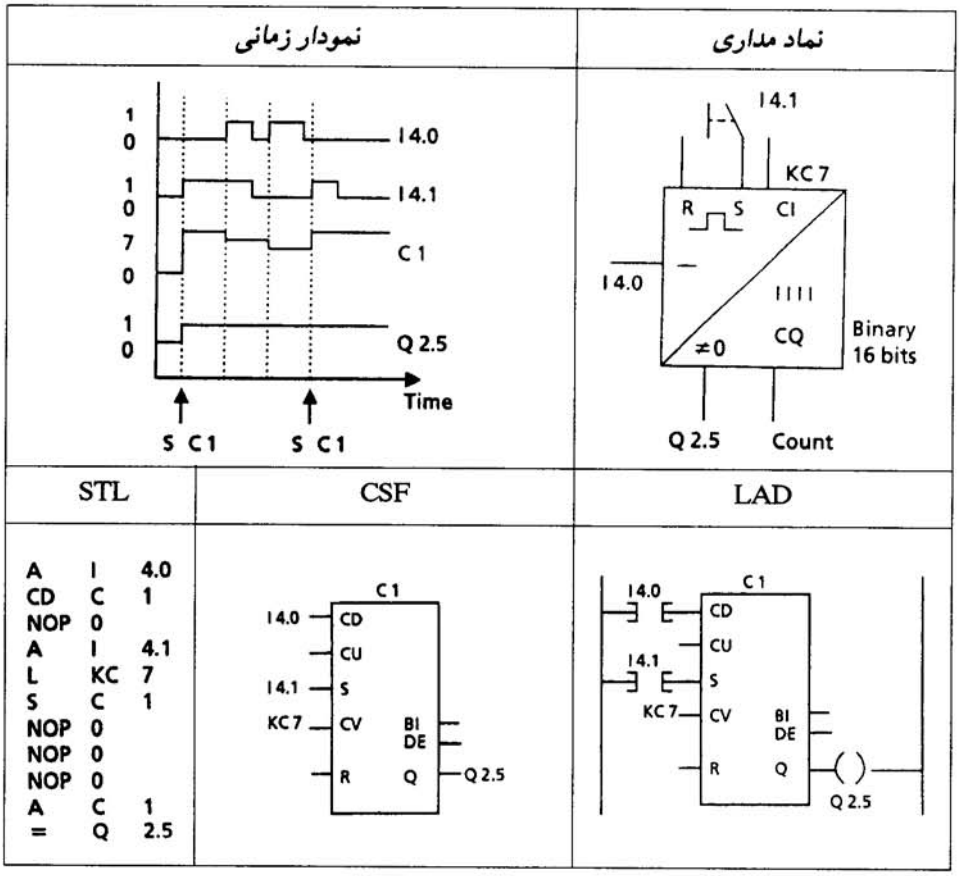
مثال ۳-۲۵ : در این مثال، شمارش مستقیم بررسی می‌گردد. همان‌گونه که در برنامه آمده است هنگامی که I 4.0 فعال شود عدد موجود در شمارندهٔ ۱ به اندازهٔ یک واحد افزایش می‌یابد و به محض اینکه ورودی I 4.2 فعال می‌شود شمارنده ری‌ست خواهد شد. دستور A C 1 مقدار "۱" را در هنگام شمارش توسط شمارنده به بیت خروجی Q 2.4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

نماد مداری	نمودار زمانی
	

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

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

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



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

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

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

```
 PB 36

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

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

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

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

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

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

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

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

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

---

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

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

شمارهٔ شمارندهٔ مورداستفاده در برنامه

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

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

C 6 ACT : .... PAR : 30

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

۳-۲۳- تایمرها (Timers)

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

۱- تایمر پله‌ای SP (Pulse Timer)

۲- تایمر پله‌ای گسترده SE (Extended Pulse Timer)

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

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

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

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

```
 PB 37

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

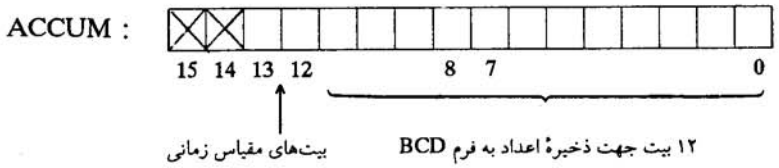
```
 PB 37

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

```
 PB 37

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

حال به توضیح در مورد ورودی‌ها و خروجی‌های تایمر می‌پردازیم:

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

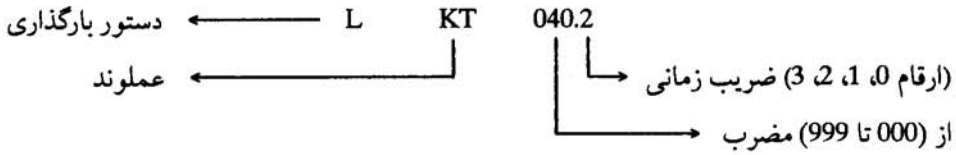
ورودی TV (Timer Value) : عددی است که پریود زمانی تایمر را معین می‌کند. معمولاً این عدد با فرمت ... KT L در تایمر بارگذاری می‌شود. پس از بارگذاری، مقدار KT در انبارک قرار می‌گیرد. KT در انباره‌ها شامل ۱۴ بیت بوده، به فرم BCD می‌باشد. دو بیت آخر در انبارک بدون استفاده می‌باشند. دو بیت باارزش بالاتر KT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



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

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

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



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

کمترین زمان ممکن : KT 001.0 ⟶ ۰/۰۱ ثانیه

بیشترین‌زمان‌ممکن : KT 999.3 ⟶ ۹۹۹۰ ثانیه

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

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

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

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

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

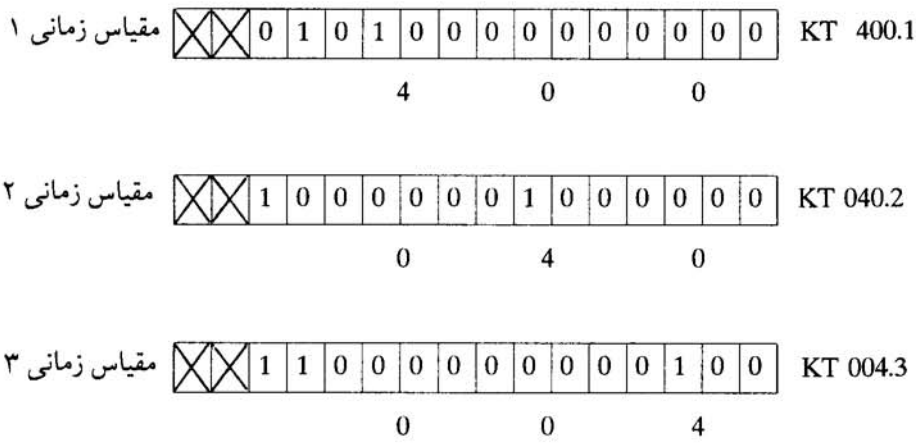
تولرانس	فاصلهٔ زمانی	دستور	
۰/۱ ثانیه	ثانیه ۰/۱±۴۰=۱/۰×۴۰۰	L KT 400.1	زمانهای ممکن استفاده شده‌برای‌بارگذاری‌زمان ۴۰ ثانیه در تایمر
۱ ثانیه	ثانیه ۱±۴۰=۱×۴۰	L KT 40.2	
۱۰ ثانیه	ثانیه ۱۰±۴۰=۱۰×۴	L KT 4.3	

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

---

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

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



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

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

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

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

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

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

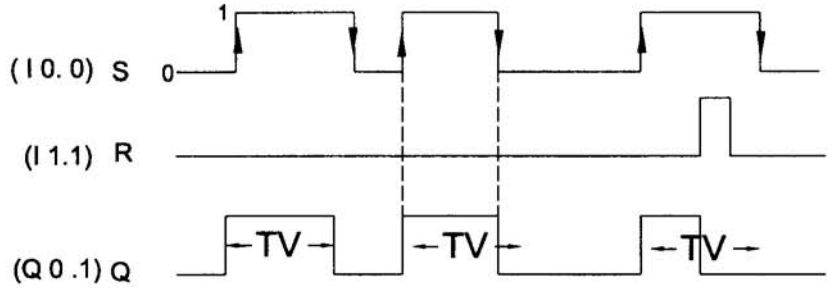
۳-۲۳-۱- تایمر پله‌ای (SP)[1]

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

```
 PB 11

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

1 - Pulse Timer

```
 PB 11

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



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

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

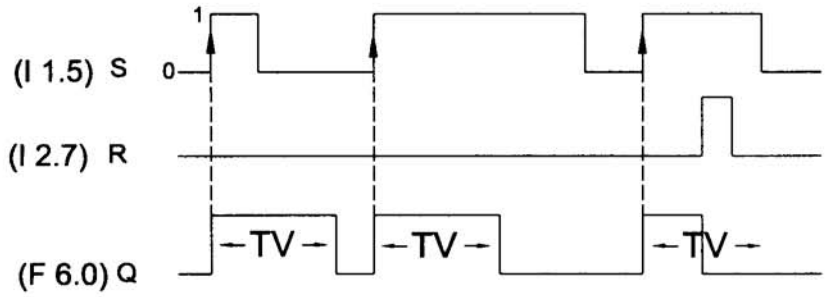
۲-۲۳-۲- تایمر پله‌ای گسترده (SE)[1]

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

```
 PB 12

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

```
 PB 12

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

1 - Extended Pulse Timer



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

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

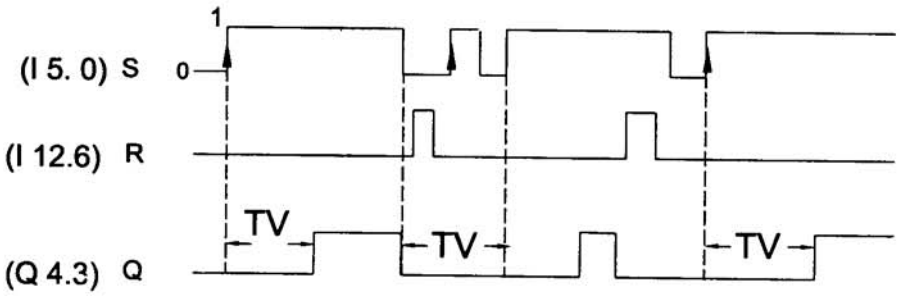
۳-۲۳-۳- تایمر با تأخیر روشن (SD)[1]

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

1 - On - Delay Timer

```
PB 13

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

```
 PB 13

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



۳-۲۳-۴- تایمر با تأخیر خاموش (SF)[1]

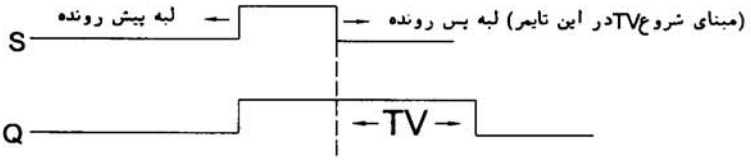
خروجی این تایمر با لبهٔ پیش‌روندهٔ ورودی S فعال و با لبهٔ پس‌روندهٔ ورودی S پس از TV ثانیه غیرفعال می‌گردد. برنامه‌های نوشته شده به روشهای STL و LAD به همراه شکل موجهای خروجی و ورودی و تأثیر ورودی‌های R و S بر خروجی در ادامه ارائه شده است.

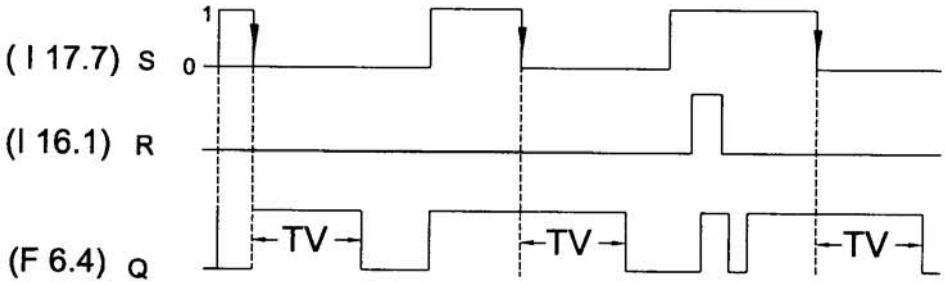
```
 PB 14

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

```
 PB 14

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

1 - Off - Delay Timer

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





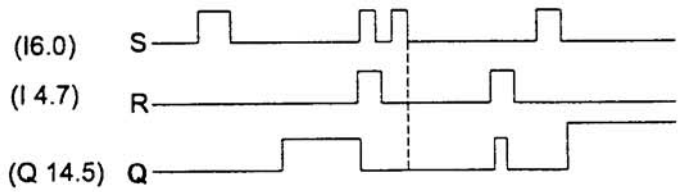
۳-۲۳-۵- تایمر تأخیر ماندگاری (SS)[1]

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

1 - Stored On - Delay Timer

```
 PB 15

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

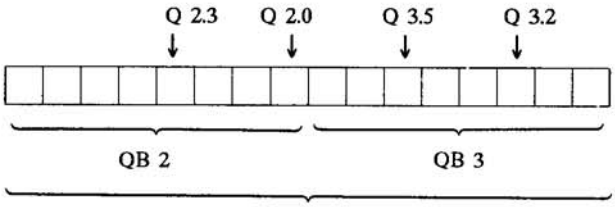
```
 PB 15

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



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

```
 PB 16

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

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



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

روشن و خاموش می‌شوند.

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

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

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

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

```
PB 17

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

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

به محض اینکه تایمر زمان TV را پشت سر گذاشت خروجی تایمر یعنی "۰" = 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 برنامهٔ تایمری را می‌نویسیم که دارای تولرانس ۱ ثانیه باشد. به عبارت دیگر مقیاس زمانی موجود در عدد شمارنده، ۲ باشد. این برنامه را در دو بخش می‌نویسیم به طوری که در بخش اول، تایمر فعال شده و عدد شمارش شده توسط تایمر به صورت باینری به یک کلمهٔ خروجی مثلاً 20 QW انتقال یابد. در بخش دوم بیت‌های این کلمهٔ خروجی را به دو چراغ اعمال می‌کنیم. واضح است که برای داشتن

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

```
 PB 18

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

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

۳-۲۴- دستورهای اعلام پایان برنامه[1]

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

---

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

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

جدول ۳-۸ : دستورات اعلام پایان برنامه

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

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

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

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

---

1 - Block End Uncoditional 2 - Block End Conditional

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

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

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

```
 FB 4

SEGMENT  1            0000
NAME :JUMP

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

روند اجرای این برنامه به صورت زیر است:

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

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

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

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

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

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


POWEREN.IR

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 راکه 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 پروسهٔ تحت کنترل اطلاعات کافی دارند، برنامه‌نویس را در جریان کلیهٔ جزئیات سیستم قرار می‌دهند.

۲- رسم فلوچارت[۱] برنامه

معمولاً در مورد کنترل پروژه‌های بزرگ و حتی پروژه‌های کوچک پس از تعریف پروژه، مرحلهٔ 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- تهیهٔ 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 شدهٔ قبلی یعنی رسم فلوچارت و ابزار مورد نیاز، برنامهٔ 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 ۱-۴ در نظر بگیرید. در این مخزن دو سنسور جهت تشخیص سطح مواد داخل مخزن در نظر گرفته شده است. I 0.0 معرف کمترین سطح ممکن (Low Level) و I 0.1 نشان دهندهٔ بالاترین سطح ممکن (High Level) می‌باشد. دو شیر (Valve) یکی برای

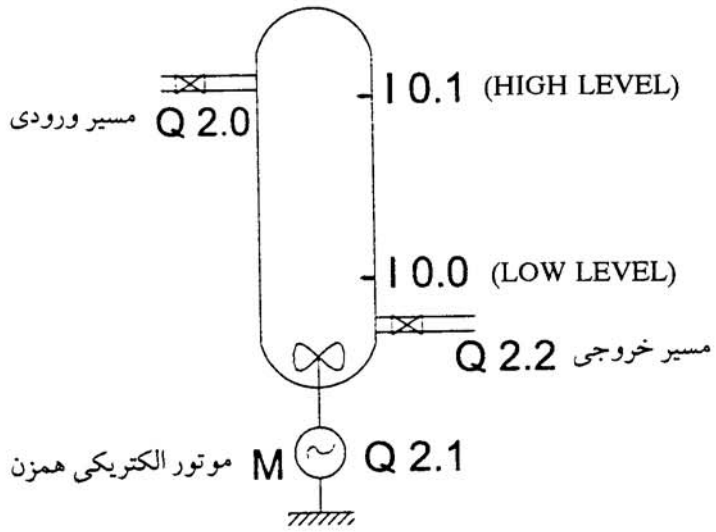
---

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 ("۱" = Q 2.0)، 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 ("۰" = Q 2.0)، موتور الکتریکی همزن شروع به چرخش نماید ("۱" = Q 2.1) پس از مدت زمانی که برای هم زدن این مواد تعریف می‌شود، موتور الکتریکی همزن خاموش و شیر خروجی باز می‌شود ("۱" = Q 2.2) تا اینکه مخزن از مواد موجود تخلیه گردد. پس از تخلیهٔ 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 پروسهٔ تحت کنترل جهت بررسی شرایط ایمنی

حال فرض کنید که بیت I 0.0 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 مرحلهٔ 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 I 0.0 برابر "۱" باشد ورودی I 0.1 که معرف High Level است نمی‌تواند 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.

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

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

نحوهٔ ارائهٔ 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[1] 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 : برنامهٔ چراغ راهنمایی:

در گروهی از برنامه‌ها با استفاده از مقایسه‌کننده‌های اعداد، فلگ‌های خاص، دستورات L ، T 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 برنامهٔ آن را ملاحظه می‌کنید.

در این مثال قصد داریم با نوشتن یک برنامه، پروسهٔ 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

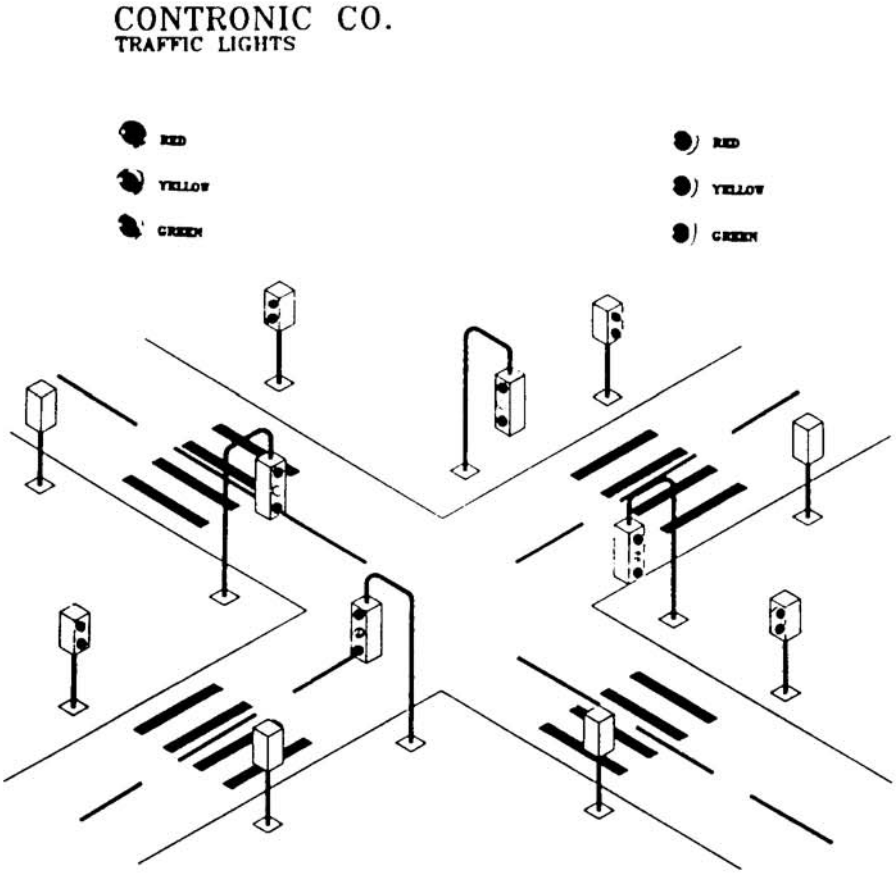
---

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.



شکل ۴-۲ : شبیه‌ساز برنامهٔ چراغ راهنمایی

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

۱- تعریف پروژه: پروژهٔ کنترل چراغ راهنمایی در ۶ مورد فوق تعریف و توصیف گردید.

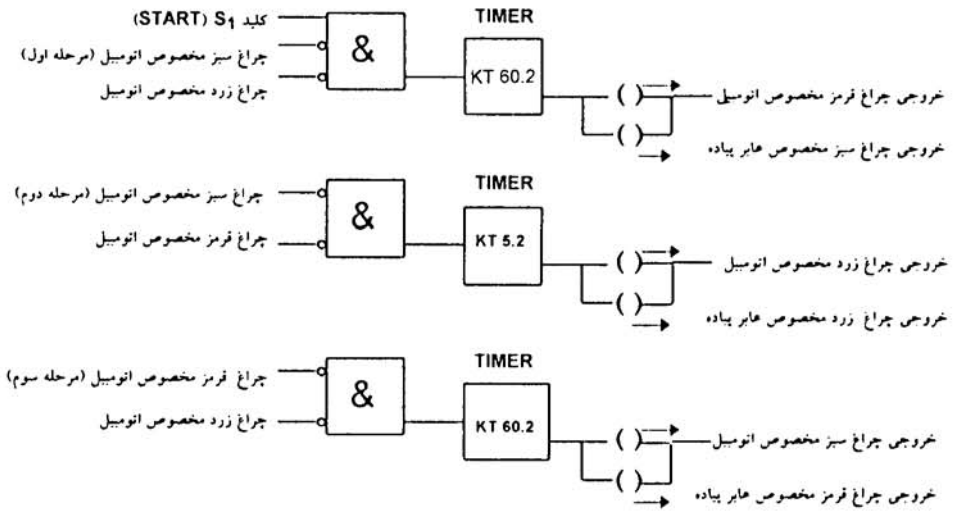
۲- رسم فلوچارت: از آنجایی که این پروژه شامل برنامه‌نویسی برای ۳ حالت می‌باشد برای رسم فلوچارت آن از سه مرحله استفاده می‌کنیم. این مراحل عبارتند از:

مرحلهٔ ۱: چراغ قرمز مخصوص اتومبیل به همراه چراغ سبز مخصوص عابر پیاده.

مرحلهٔ ۲: چراغ زرد مخصوص اتومبیل به همراه چراغ زرد مخصوص عابر پیاده.

مرحلهٔ ۳: چراغ سبز مخصوص اتومبیل به همراه چراغ قرمز مخصوص عابر پیاده.

در ادامه، فلوچارت هر سه مرحله آورده شده است.

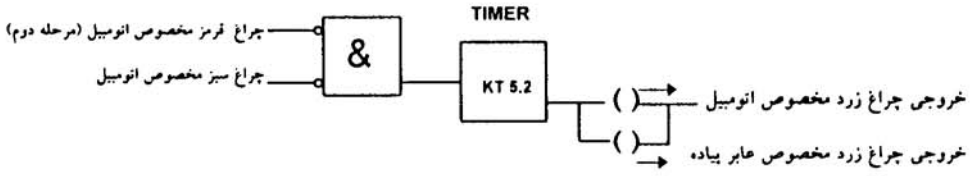


در رسم این فلوچارت، شرایط ایمنی نیز در نظر گرفته شده است. به عنوان مثال در مرحلهٔ اول برای به کار افتادن تایمر باید کلید استارت فعال باشد. این مورد تنها شرط لازم جهت ست نمودن تایمر نیست. همان‌گونه که در فلوچارت مرحلهٔ اول ملاحظه می‌کنید از نقیض خروجی مراحل دیگر یعنی نقیض خروجی چراغ سبز و چراغ زرد مخصوص 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 مرحلهٔ 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 مرحلهٔ 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 مرحلهٔ دوم به جای استفاده از فلوچارت رسم شدهٔ 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.

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

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 مرحلهٔ بعدی برنامه‌نویسی) به جای این ورودی و خروجی‌ها از دستور NOP 0 استفاده می‌کنیم.

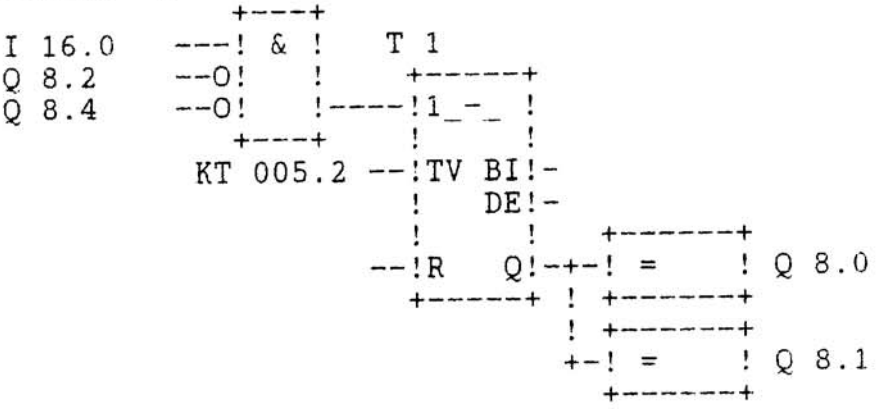
۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: در اینجا برای تمرین بیشتر از دو روش STL و CSF جهت برنامه‌نویسی این پروژه استفاده شده است. برنامهٔ مذکور در PB 10 و در سه بخش (Segment) نوشته شده است.

```
 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      :***
```

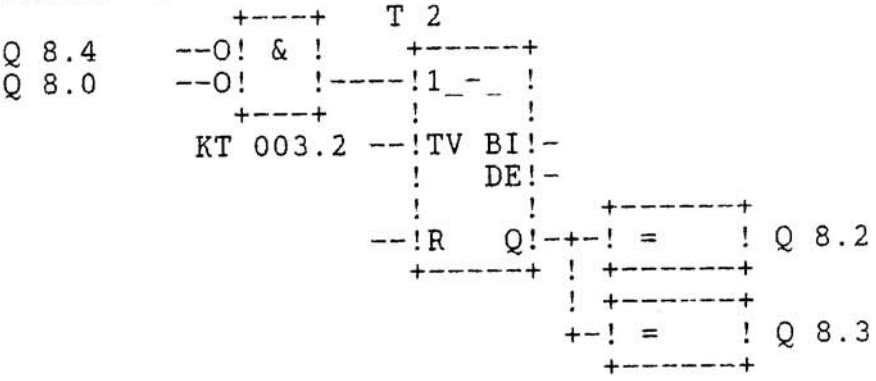
```
SEGMENT  3            0019
0019      :AN  Q     8.0
001A      :AN  Q     8.2
001B      :L   KT  005.2
001D      :SP  T     3
001E      :NOP 0
001F      :NOP 0
0020      :NOP 0
0021      :A   T     3
0022      :=   Q     8.4
0023      :=   Q     8.5
0024      :BE
```

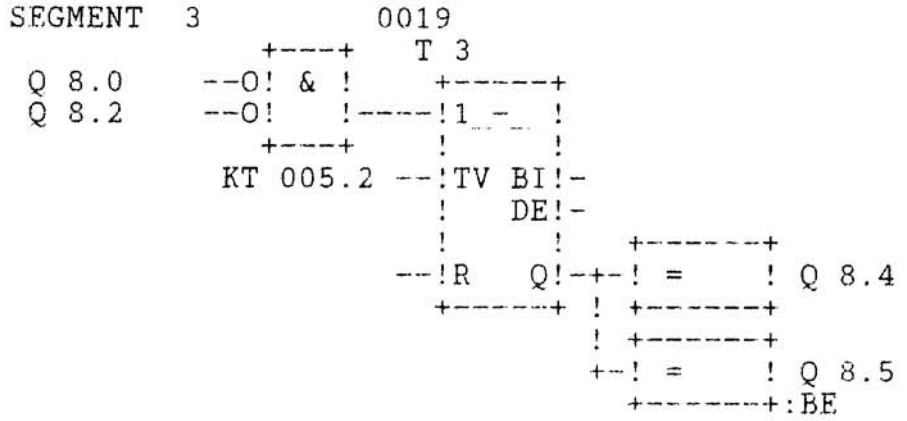
PB 10

SEGMENT 1 0000



SEGMENT 2 000D





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 ضریب زمانی 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 شمارهٔ آن ذکر شود مثلاً در مرحلهٔ سوم دستور SP T 3 ، تایمر شمارهٔ 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* ۴–۳**: فرض کنید که در یک پروسهٔ کنترلی و برای تعریف پروژه و برنامه‌نویسی به ۲۰ تایمر نیاز داریم اما PLC موجود و در دسترس تنها دارای ۱۶ تایمر (T0 - T15) است. برای رفع این شکل چه می‌کنید؟

برای حل این مشکل برنامهٔ 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. برنامهٔ PB 34 همان برنامهٔ مذکور است. در اینجا می‌توان با استفاده از تولید پالس توسط یک تایمر و یک شمارنده و ست و ری‌ست نمودن پی‌درپی، تایمرهای زیادی ایجاد نمود.

در برنامهٔ 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 شمارندهٔ ۵ اضافه می‌کند. حال اگر عدد موجود در شمارنده را با ۱۰ مقایسه نمائیم یعنی ۵۰ ثانیه (۵۰=۵×۱۰) 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 برنامهٔ 34 PB اضافه نمود.

در این برنامه شمارندهٔ ۵ بعد از هر ۹۹۹ بار شمارش یک بار صفر می‌شود. اگر از لبهٔ پائین‌روندهٔ شمارندهٔ ۵ به شمارندهٔ ۶ ارسال نماییم شمارندهٔ ۶ یک واحد افزایش خواهد یافت، بنابراین در صورتی که به عنوان مثال شمارندهٔ ۶ حاوی عدد ۱۰۰ باشد.

```
0019        :L    C     5
001A        :L    KF  +999
001C        :!=F
001D        :R    C     5
001E        :AN   C     5
001F        :CU   C     6
0020        :BE
```

بدین معنی است که به تعداد (۹۹۹×۱۰۰) 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 اِشکال در سیستم‌های کنترلی مدرن (سیستم‌های مونیتورینگ) پیغامهای خطایی نظیر TANK LEVEL LOW ، HIGH TEMPERATURE و ... بر روی صفحهٔ نمایش ظاهر

می‌شود. این‌گونه پیغامها در DB وجود دارد و PLC به هنگام ایجاد مشکل در سیستم، پیغامهای مناسب را از DB خوانده، به صفحهٔ مونیتور می‌فرستد. در PLC مورد بحث ما می‌توان ۲۵۶ بلوک اطلاعاتی (DB 0 - DB 255) تعریف نمود. هر DB می‌تواند 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) می‌گویند.

اطلاعاتی که می‌توان در DB قرار داد به یکی از صورتهای زیر می‌باشد:

۱- Data (مقادیر و پارامترها)

۲- Text (متن پیامها و ...)

۳- Bit Pattern (این اطلاعات شامل تعدادی بیت‌های "۰" و "۱" است که به صورت بایتی یا کلمه‌ای به‌خروجی ارسال‌شده، عمل‌سیگنالینگ یا فعال وغیرفعال نمودن‌خروجی‌هارا 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 می‌شوند می‌توان بلوک‌های PB را اجرا نمود. اما در مورد DBها فراخوانی اطلاعات با دستور دیگری انجام می‌گیرد. برای خواندن اطلاعاتِ هر DB ابتدا باید آن را در طول اجرای برنامه صدا[1] زد.

در مورد PBها ملاحظه شد که اجرای برنامه بدین صورت است که پردازنده از سطر اول، دستورات را به ترتیب اجرا می‌نماید تا اینکه به سطر انتهایی برنامه برسد اما در مورد DBها می‌توان به سطر دلخواه دست یافت. به عنوان مثال برای خواندن اطلاعات سطر صدم از DB 50 به ترتیب زیر عمل می‌کنیم.

ابتدا DB شمارهٔ ۵۰ را با استفاده از دستور C صدا زده، اطلاعات (DW) موجود در سطر صدم آن را با استفاده از دستور L در انبارک بارگذاری می‌کنیم:

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

1 - CALL

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

۱) KH : 16 BITS $(0000_{(H)} \longrightarrow FFFF_{(H)})$ برای اعداد در مبنای ۱۶

۲) KF : 16 BITS $(-32768 \longrightarrow +32768)$ برای اعداد در مبنای ۱۰

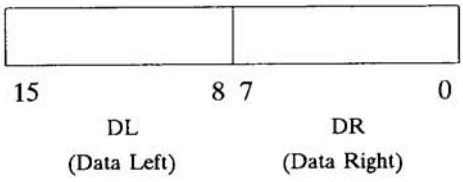
۳) KT : 14 BITS $(001.0 \longrightarrow 999.3)$ برای اعداد ثابت TV

۴) KC : 12 BITS $(000 \longrightarrow 999)$ 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 000 الی 255 را داشته باشد.



۶) KM : 16 BITS $\left((000 \ldots 00) \longrightarrow (111 \ldots 11)\right)$

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 ۰۹ ± ۱۵۲۴۶۷۰ را در نظر بگیرید، در PLC زیمنس این عدد ابتدا به 10^{6} تقسیم می‌شود. در مرحلهٔ بعد در صورت مثبت بودن علامت توان، PLC عدد حاصل را در 10^{9} و در صورت منفی بودنِ علامت توان، آن را در 10^{-9} ضرب می‌کند. محدودیتی که در این روش وجود دارد آن است که دو رقم مشخص کنندهٔ توان حداکثر ۳۸ می‌باشد.

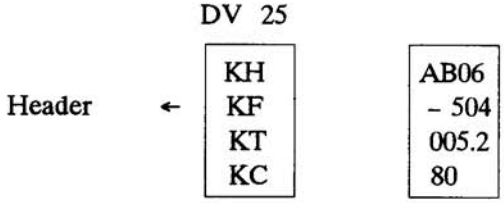
۸) KS : TANK LEVEL LOW.

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 ذخیرهٔ پیغام فوق ۱۵ بایت از حافظه اشغال می‌گردد. در این حالت هر حرف، علامت و حتی جای خالی و نقطهٔ 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 واردکردن 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 دیگر که 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 دستهٔ 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 برنامهٔ 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 C 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 حافظهٔ PLC ذخیره می‌شود تا هنگام بازگشت، اجرای برنامه روند قبلی خود را طی نماید:

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

۲- شمارهٔ سطری از برنامهٔ 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 ذخیرهٔ 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.

برنامهٔ 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 دادهٔ شمارهٔ 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

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

در انتهای برنامه، حاصل جمع دو عدد موجود در DW 0 و DW 1 به کلمهٔ خروجی شمارهٔ ۲ یعنی QW 2 فرستاده می‌شود. کلمهٔ خروجی ۲ می‌تواند به یک سری نشان دهنده مثلاً LED متصل شده باشد. در این صورت چگونگی افزایش محتویات DW 0 در هر سیکل اجرای برنامه با خاموش و روشن‌شدن LEDها قابل رویت می‌گردد. بیشترین عددی که در DW 0 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 DW 0 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 پردازندهٔ 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 QW 2 می‌باشد. بنابراین سرعت اجرای تعداد سیکل برنامه مثلاً ۲۰۰۰ مرتبه را اندازه‌گیری و سپس زمان به‌دست آمده را بر عدد ۲۰۰۰ تقسیم می‌کنیم.

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

برای نوشتن این برنامه از برنامهٔ نوشته شده در مثال ۴-۴ استفاده می‌کنیم. با این تفاوت که در DB 27، سطر دیگری را که همان KF = 2000 است وارد نموده، با ایجاد تغییراتی در PB 30 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 برنامهٔ PB 30 است با این تفاوت که در اجرای این برنامه و در هر سیکل، اطلاعات موجود در DW 2 یعنی عدد ۲۰۰۰ با حاصل جمع موجود در DW 0 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 به‌دست 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 Q 2.0 S در سطر هشتم برنامهٔ PB 31 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 لحظهٔ بسیار کوتاه، معادل با زمان لازم جهت مساوی شدن حاصل جمع عدد موجود در DW 0 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 Q 2.0 S جهت فعال نمودن خروجی، برای روشن شدن و روشن باقی ماندن LED تنها به لبهٔ پالس نیاز است و در لحظه‌ای که محتویات موجود در DW 0 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 OB 1 ساختار برنامهٔ استفاده کننده را مشخص می‌کند زیرا سیستم عامل در شروع هر سیکل برنامه به بلوک OB 1 رجوع می‌کند. بنابراین کلید معرف آغاز شمارش را در این بلوک تعریف می‌کنیم. برنامهٔ 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 OB 1 ، بلوک اطلاعاتی ۲۸ یعنی DB 28 فعال می‌گردد تا تمامی اطلاعات از این بلوک خوانده شود. در صورتی که کلید استارت شمارش یا I 0.0 برابر "۱" شود PB 31 که همان برنامهٔ قبلی است اجرا می‌شود. در این برنامه، شمارش از صفر شروع شده، تا ۲۰۰۰ ادامه می‌یابد. ولی اگر کلید استارت شمارش غیرفعال باشد PB 40 اجرا خواهد شد. در این برنامه ابتدا عدد صفر در DW 0 فرستاده می‌شود، در سطر سوم با ری‌ست نمودن بیت خروجی Q 2.0 امکان انجام این کار برای چندین بار عملی شده است.

بنابراین ملاحظه می‌کنید که در هر دو صورت، شمارش از صفر آغاز شده، تا ۲۰۰۰ ادامه می‌یابد. بنابراین زمان به‌دست آمده تا روشن شدن LED مربوط به Q 2.0 زمان پردازش ۲۰۰۰ سیکل برنامه می‌باشد.

بنا به دلایل مختلفی از جمله خطای چشم در مشاهدهٔ زمان روشن شدن بیت Q 2.0 و عوامل دیگر ممکن است در محاسبهٔ 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 به‌دست می‌آوریم.

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

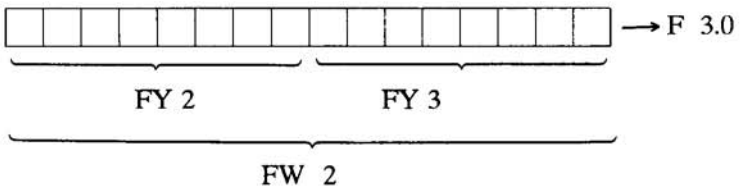
این برنامه به صورت زیر نوشته می‌شود.

```
 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 کلمهٔ فلگ ۲ و بیت‌های مربوط به آن

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

با اجرای این برنامه پس از ۲ ثانیه (KT 2.2) تعداد سیکل‌های انجام شده به‌دست می‌آید. این تعداد به دست آمده همان محتویات شمارندهٔ 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 برنامهٔ PB 32 انجام شده است. بنابراین مدت زمان اجرای یک سیکل از این برنامه $\frac{۱}{۱۵۱}$ ثانیه یا تقریباً ۶ میلی‌ثانیه است. همان‌گونه که قبلاً ذکر شد در مواردی که با فاصله‌های زمانی کوچک سروکار داریم و یا نیاز به دقت در محاسبهٔ زمان باشد از تولرانس‌های ۰ و ۱ استفاده می‌کنیم. بنابراین به جای استفاده از L KT 2.2 در این برنامه می‌توان از دستور L KT 20.1 و یا L KT 200.0 نیز استفاده‌نمود.

یکی از مواردی که در فرآیندهای صنعتی و خطوط تولید با آن روبرو می‌شویم محاسبهٔ زمان دقیق در مورد انجام هر قسمت از خط یا فرآیند است. می‌توان همین برنامه را در انتهای برنامهٔ کنترلی 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 رابطهٔ $\frac{x}{\text{تعداد سیکل‌های انجام شده در مدت x ثانیه}}$ به دست می‌آید.

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

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

برنامهٔ 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 PB 25:

```
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 OB 1، ابتدا DB 20 فعال و سپس دستور پرش به PB 25 (بلوک برنامه‌ای حاوی برنامهٔ کنترلی) صادر گردیده است. در PB 25 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

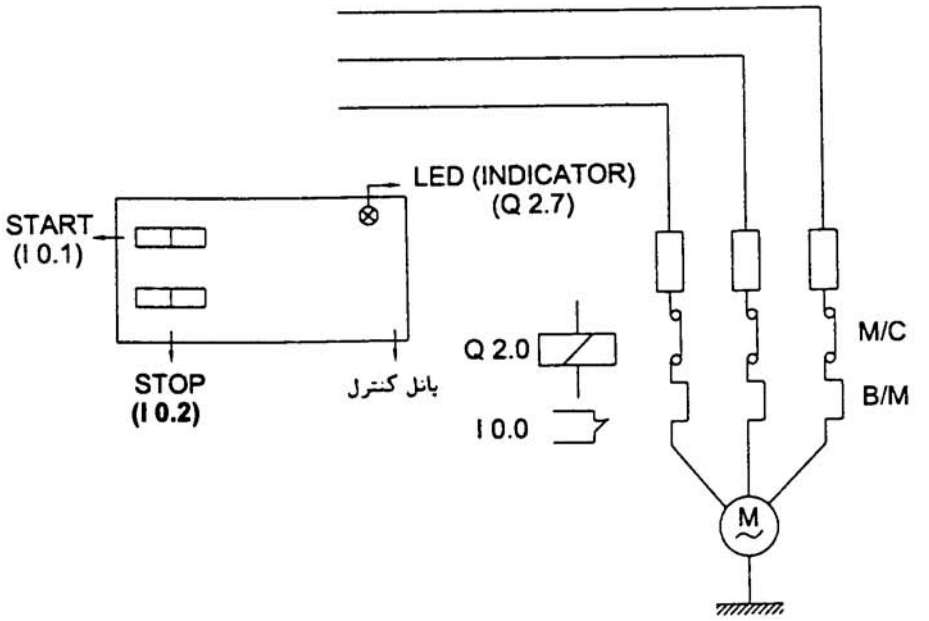
مخصوص اتومبیل و عابر پیاده فعال و سایر چراغها غیرفعال می‌گردند. روند اجرای مرحله سوم یعنی ارسال فرمان به خروجی‌های چراغهای سبز مخصوص اتومبیل و قرمز مخصوص عابر پیاده نیز مانند مراحل قبل است. در حقیقت زمان روشن بودن چراغهای سه مرحله به ترتیب 1500 و 1000 = (2500-1500) و 1500 = (4000-2500) برابر سیکل زمانی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

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

هنگامی که عدد DW 0 به هر دلیلی بیشتر از ۴۰۰۰ باشد پردازنده باید محتوای DW 0 را با عدد ۶۵۵۳۵ ($FFFF_H$) پر نموده، مجدداً صفر شود ولی با استفاده از I 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 دهندهٔ 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.)



برنامهٔ کنترل مورد نظر در ادامه آمده است:

```
PB 41

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 برنامهٔ 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 = "۱" باشد، و در این حالت موتور نیز روشن بماند F 10.0 به حالت "۱" ثابت می‌ماند ولی چنانچه کنتاکتور B/M قطع شده، موتور کار کند، فلگ F 21.2 که با فاصلهٔ 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 Q 2.7 ارسال می‌گردد.

۴-۳- بلوک‌های تابع‌ساز (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 برنامهٔ 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 تابع‌ساز 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 دفترچهٔ 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- بدنهٔ 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 مورداستفاده قرار می‌گیرند. به طورکلی می‌توان FB ها را به دو دستهٔ زیر تقسیم نمود:

۱- بلوک تابع‌ساز استاندارد (Standard FB)

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

۲- بلوک تابع‌ساز 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 ادامهٔ بحث درمورد برنامه‌نویسی FBها، مثالهایی از توابع استاندارد و انتسابی ارائه خواهدشد.

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 حاصلضرب 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:

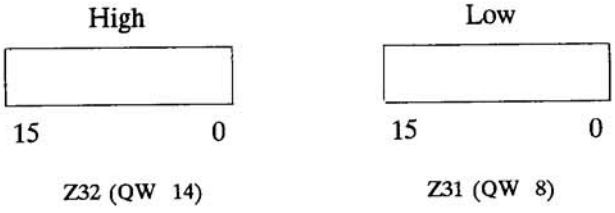
Z1 : عدد اول (IW 16) . این عدد به صورت کلمه‌ای به PLC داده می‌شود.

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

Z3 : در صورتی که نتیجهٔ 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 : نتیجهٔ حاصلضرب دو عدد Z1 و Z2 در یک کلمهٔ خروجی (QW 8) ظاهر می‌شود.

Z32 : در صورتی که نتیجهٔ حاصلضرب از یک کلمه یا ۱۶ بیت بیشتر شده باشد بیت‌های بعدی در یک کلمهٔ خروجی دیگر (QW 14) قرار می‌گیرد. با این تعاریف می‌توان گفت که نتیجهٔ حاصلضرب در دو کلمهٔ 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 باارزش پائین‌تر و Z32 یا QW 14 حاوی بیت‌هایی با ارزش بالاتر می‌باشد.

***مثال* ۴-۹** : به کمک بلوک FB 242 حاصلضرب دو عدد $۱۴_{(۱۰)}$ و $۲۵_{(۱۰)}$ 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.

$۱۴_{(۱۰)} = ۱۱۱۰_{(۲)}$ = ۰۰۰۰۰۰۰۰۰۰۰۱۱۱۰ ⟶ IW 16

$۲۵_{(۱۰)} = ۱۱۰۰۱_{(۲)}$ = ۰۰۰۰۰۰۰۰۰۰۰۱۱۰۰۱ ⟶ 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 Q 10.0 برابر "۰" می‌باشد.

Z3 = ۰ ⟶ Q 10.0

$Z31 = ۰۰۰۰۰۰۰۰۰۱۰۱۱۱۱۰_{(۲)}$ ⟶ QW 8

$= ۱\times۲^{۶} + ۱\times۲^{۴} + ۱\times۲^{۳} + ۱\times۲^{۲} + ۱\times۲^{۱} = ۳۵۰_{(۱۰)}$

$Z32 = ۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰_{(۲)}$ ⟶ 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 $۵۶۹۸_{(۱۰)}$ و $۳۲۶۵_{(۱۰)}$ را به دست آوریم همان روند قبلی را طی کرده، خواهیم داشت:

$۵۶۹۸_{(۱۰)} = ۰۰۰۱۰۱۱۰۰۱۰۰۰۰۱۰_{(۲)}$ ⟶ IW 16

$۳۲۶۵_{(۱۰)} = ۰۰۰۰۱۱۰۰۱۱۰۰۰۰۰۱_{(۲)}$ ⟶ IW 21

خروجی‌ها به صورت زیر می‌باشند:

Z3 = ۰

$Z31 = ۱۱۰۱۱۱۱۱۱۱۰۰۰۰۱۰_{(۲)}$ ⟶ QW 8

$Z32 = ۰۰۰۰۰۰۰۱۰۰۰۱۱۰۱۱_{(۲)}$ ⟶ QW 14

نتیجهٔ حاصلضرب = Z32 Z31 = ۰۰۰۰۰۰۰۱۰۰۰۱۱۰۱۱ ۱۱۰۱۱۱۱۱۱۱۰۰۰۰۱۰

$= ۱۸۶۰۳۹۷۰_{(۱۰)}$

همان‌گونه که ملاحظه می‌کنید نتیجهٔ حاصلضرب بسیار بزرگ بوده، تبدیل این عدد در مبنای ۲ به مبنای ۱۰ وقت‌گیر و مشکل است. برای حل این مشکل دو کلمهٔ موجود در خروجی (۳۲ بیت) را به ۸ دستهٔ ۴ بیتی تقسیم می‌نماییم که هر دسته معرف عددی در مبنای ۱۶ می‌باشد. سپس این عدد را بر مبنای ۱۶ به‌دست 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 دستهٔ ۴ بیتی

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

$$011BDFC2 = ۰\times۱۶^۷ + ۱\times۱۶^۶ + ۱\times۱۶^۵ + ۱۱\times۱۶^۴ + ۱۳\times۱۶^۳ + ۱۵\times۱۶^۲ + ۱۲\times۱۶^۱ + ۲\times۱۶^۰$$

$$= ۱۸۶۰۳۹۷۰_{(۱۰)}$$

اکنون به ذکر مثالی در مورد FB Assignable می‌پردازیم.

در این مثال به بررسی FB 20 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 I 0.1 و I 0.0 را در Q 2.0 قرار دهیم برنامهٔ 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
!
```

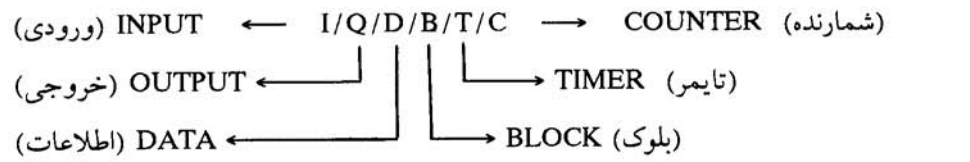
مثال ۴-۱۰ : با استفاده از تعریف تابع منطقی XOR در بلوک تابع‌ساز، برنامه‌ای بنویسید که هرگاه تنها یکی از دو موتور با شماره‌های ۱ و ۲ روشن باشد LED موجود بر روی پانل کنترل روشن شود. برای نوشتن این برنامه به دو بلوک FB و PB نیاز داریم، که باید در بلوک FB تابع XOR را به همراه پارامترهای موردنظر تعریف کنیم.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

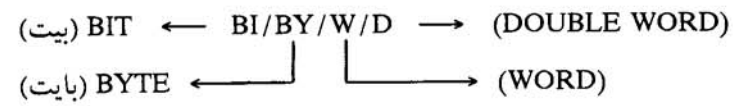
روند وارد نمودن اطلاعات به شرح زیر است:

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 مرحلهٔ بعد، نام پارامترها، نوع و ماهیت پارامترها و همچنین نحوهٔ ارسال و یا دریافت پارامترها از کاربر خواسته می‌شود. در مورد اولین ورودی، کاربر پارامتر MOT 1 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 ادامهٔ همین سطر چگونگی ارسال و یا دریافت پارامترها از استفاده‌کننده خواسته می‌شود. کاربر با انتخاب یکی از حالات BI/BY/W/D نحوهٔ 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 نحوهٔ 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 نحوهٔ 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 مرحلهٔ انتخاب نوع پارامتر، Q انتخاب شده است. نحوهٔ 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 OB 1 به صورت زیر می‌باشد:

```
 OB 1

SEGMENT 1 :

            : JU   PB   3

            : BE
```

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

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

از آنجایی که این بلوک‌های تابع‌ساز، انتسابی می‌باشند می‌توان در هر بار صدا زدن بلوک، عملوندهای متناظر با پارامترهای تعریف شده در بلوک FB را تغییر داد. حتی می‌توان در یک بلوک PB چندین بار بلوک FB خاصی را صدا زده، هر بار عملوندهای آن را تغییر داد. برنامهٔ نوشته شده در PB 6 این مطلب را روشن می‌سازد.

```
 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 PLCهای زیمنس دستورالعمل‌هایی به نام دستورات تکمیلی یا Supplementary وجود دارند که استفاده از آنها فقط در FBها مجاز است. در این قسمت به ذکر برخی از این دستورات مهم می‌پردازیم.

۴-۴-۱- دستور AW[1]

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 کلمهٔ ورودی (IW) قرار گرفته باشند و خروجی‌های متناظر با حاصل ترکیب عطفی دو ورودی در کلمهٔ 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.

---

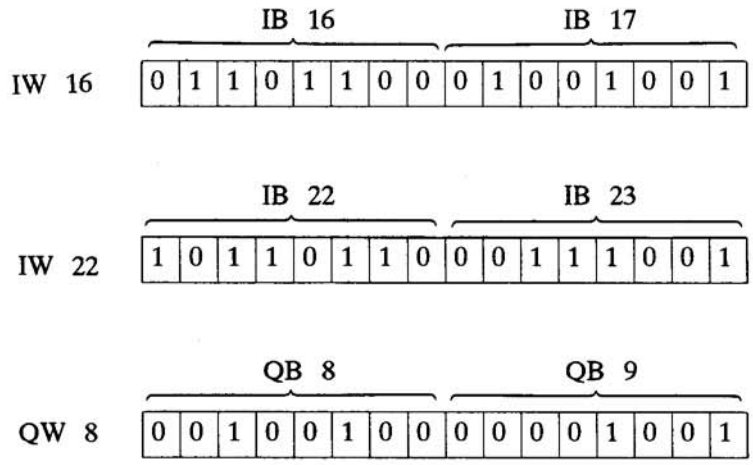
1 - AND WORD

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

در برنامهٔ 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 کلمهٔ خروجی، هم‌ارز با ترکیب عطفی بیت‌های متناظر در کلمات ورودی است. یعنی به عنوان مثال: I 23.3 · I 17.3 = Q 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 فرآیندها لازم است که بعضی از بیت‌های ورودی پوشانده و تنها برخی از آنها در

خروجی ظاهر شوند. به این عمل ماسک[1] کردن می‌گویند. چگونگی انجام این عمل را در مثال زیر می‌بینید.

مثال ۴-۱۱ : برنامه‌ای بنویسید که تنها بیت‌های با شمارهٔ فرد یک کلمهٔ ورودی را در خروجی ظاهر کند یا به عبارت دیگر بیت‌های با شمارهٔ 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 کلمهٔ ورودی ۱۴ (IW 14) ، به دست آورده، عدد حاصل را به یک کلمهٔ 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 کلمهٔ خروجی حاصل ترکیب عطفی (AND) بیت‌های متناظر در دو کلمهٔ تعریف شده در ورودی می‌باشد.

---

1 - Mask

۴-۴-۳- دستور OW[1]

عملکرد این دستور نیز مانند دستور AW است با این تفاوت که در اینجا حاصل ترکیب فصلی دو کلمهٔ ورودی، در یک کلمهٔ خروجی ظاهر می‌گردد، به صورتی که هر بیت کلمهٔ خروجی حاصل ترکیب فصلی بیت‌های متناظر در دو کلمهٔ 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[2] ورودی‌ها نیز گویند.

۴-۴-۳- دستور XOW[3]

در این دستور حاصل تابع منطقی XOR دو کلمهٔ ورودی در یک کلمهٔ 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
```

1 - OR WORD 2 - Combination 3 - XOR WORD

به کمک این دستور می‌توان اختلاف بین ورودی‌ها را در خروجی آشکار ساخت که به این عمل Detect نمودن می‌گویند. 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" می‌باشد.

۴-۴-۵- دستور CFW[1]

با استفاده از این دستور می‌توان مکمل ۱ یک کلمهٔ ورودی را در یک کلمهٔ خروجی قرار داد. در این دستور برخلاف دستورات قبلی تنها از یک کلمه در ورودی استفاده می‌شود.

همان‌گونه که می‌دانید مکمل "۰"، "۱" و مکمل "۱"، "۰" می‌باشد، بنابراین برای به دست آوردن مکمل ۱ یک کلمه ورودی کافی است ارزش بیت‌های 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 بیت‌های کلمهٔ ورودی ۱۶ در کلمهٔ خروجی ۸ قرار گرفته‌اند یا به عبارت دیگر ارزش بیت‌های ورودی عکس شده است.

---

1 - 1st Complement Word

۴-۴-۶- دستور CSW[1]

به کمک این دستور می‌توان مکمل ۲ یک کلمهٔ ورودی را در یک کلمهٔ خروجی قرار داد. در این دستور نیز تنها از یک کلمهٔ 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 IW 16 عدد $AAAA_{(H)}$ قرار گرفته باشد مکمل ۲ آن به صورت زیر به دست می‌آید:

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

$$\underline{\qquad\qquad 1 \qquad\qquad}$$

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

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	

---

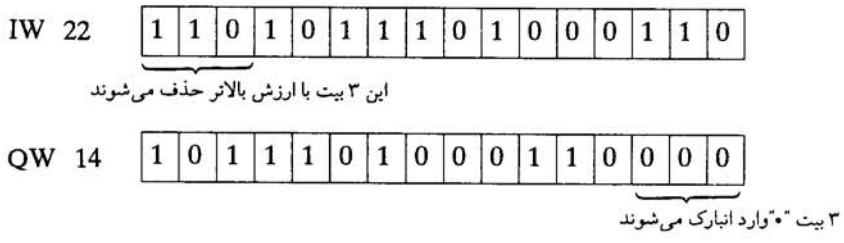
1 - 2^{nd} Complement Word

۴-۴-۷- دستور SLW[1]

به کمک این دستور می‌توان بیت‌های یک کلمهٔ 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 FB 16 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، کلمهٔ ورودی IW 22 در انبارک بارگذاری شده، سپس تک‌تک بیت‌های این کلمه به اندازهٔ 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 برنامهٔ فوق پس از اجرای برنامه، تمام بیت‌های IW 22 به اندازهٔ 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 کلمهٔ ورودی ۲۲ حذف می‌شوند. در ادامه، محتویات اولیهٔ IW 22 و همچنین محتویات QW 14 پس از انجام شیفت نشان داده شده است.



1 - Shift Left Word

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

مثال ۴-۱۲ : عدد $۲_{(۱۰)}$ را به عنوان کلمهٔ ورودی در IW 22 در نظر گرفته، دستورات SLW 1 ، SLW 2 و SLW 3 را در مورد آن اعمال کنید. پس از مقایسهٔ عدد حاصل در هر قسمت با عدد اولیه چه نتیجه‌ای می‌گیرید؟

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

$$۲_{(۱۰)} = ۱۰_{(۲)} = ۰۰۰۰۰۰۰۰۰۰۰۰۰۰۱۰_{(۲)} \longrightarrow \text{IW } 22$$

IW 22 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 | $۱۰_{(۲)} = ۲$

SLW 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 | $۱۰۰_{(۲)} = ۴$

SLW 2 | 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 | $۱۰۰۰_{(۲)} = ۸$

SLW 3 | 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 | $۱۰۰۰۰_{(۲)} = ۱۶$

همان‌گونه که دیده می‌شود با اجرای دستور SLW 1 بر روی IW 22 که حاوی عدد $۲_{(۱۰)}$ می‌باشد، عدد $۴_{(۱۰)}$ یعنی ۲ برابر مقدار اولیه به‌دست می‌آید. با اجرای دستور SLW 2 ، حاصل $۸_{(۱۰)}$ یعنی $۲^۲$ برابر عدد اول یعنی محتویات اولیهٔ IW 22 به‌دست می‌آید و در دستور SLW 3 نتیجهٔ حاصل شده $۲^۳$ برابر عدد اولیه خواهد بود. پس نتیجه می‌گیریم که با هر بار حرکت بیت‌ها به سمت چپ، عدد به‌دست آمده ۲ برابر می‌شود. بنابراین دستور SLW 5 عدد اولیهٔ موجود در IW 22 را در ۳۲ ($۲^۵$) ضرب می‌کند. اما جواب این حاصلضرب تنها هنگامی صحیح خواهد بود که عدد اول موجود در IW 22 به هنگام شیفت 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 SLW 1 ، SLW 2 ، SLW 3 ، SLW 4 و ... استفاده می‌شود به شرطی که پس از شیفت، بیت‌های با ارزش بالا با محتوای "۱" حذف نگردند.

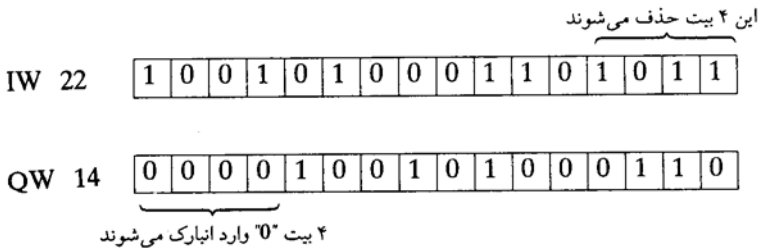
۴-۴-۸- دستور SRW[1]

عملکرد این دستور دقیقاً بر عکس دستور SLW است به این صورت که بیت‌های کلمهٔ ورودی را به سمت راست حرکت می‌دهد. تعداد انتقال بیت‌ها از ۰ تا ۱۵ قابل تغییر است. بلوک FB 17 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 برنامهٔ فوق پس از بارگذاری کلمهٔ ورودی ۲۲ در انبارک، تک‌تک بیت‌ها به اندازهٔ 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 اولیهٔ IW 22 و همچنین محتویات QW 14 پس از انجام عمل شیفت نشان داده شده است.



مثال ۴-۱۳ : عدد $۱۹۲_{(۱۰)}$ را به عنوان کلمهٔ ورودی ۲۲ در نظر گرفته، دستورات SRW 1، SRW 2 و SRW 3 را در مورد آن اعمال نمائید و پس از مقایسهٔ اعداد به‌دست آمده با عدد اول نتیجه را بیان کنید.

---

1 - Shift Right Word

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

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

IW 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | $۱۱۰۰۰۰۰۰_{(۲)} = ۱۹۲_{(۱۰)}$

SRW 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | $۱۱۰۰۰۰۰۰_{(۲)} = ۹۶_{(۱۰)}$

SRW 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | $۱۱۰۰۰۰_{(۲)} = ۴۸_{(۱۰)}$

SRW 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | $۱۱۰۰۰_{(۲)} = ۲۴_{(۱۰)}$

با مقایسهٔ اعداد به‌دست 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 $۲^۱$ تقسیم شده است. با اجرای دستور SRW 2 عدد به‌دست آمده برابر $\frac{۱}{۴}$ $(\frac{۱}{۲^۲})$ عدد اول است، با اجرای دستور SRW 3 به $\frac{۱}{۸}$ $(\frac{۱}{۲^۳})$ 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 مقایسهٔ عدد به‌دست آمده با عدد اول چه نتیجه‌ای می‌گیرید؟

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

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

IW 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | $۱۱۰۱۰۱۱۱_{(۲)} = ۲۱۵_{(۱۰)}$

SRW 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | $۱۱۰۱۰۱۱_{(۲)} = ۱۰۷_{(۱۰)}$

$\neq \frac{۲۱۵}{۲}$

با مقایسهٔ عدد به‌دست 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 SRW 1، SRW 2، SRW 3، SRW 4 و ... 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[1]

به‌کمک این دستور می‌توان حداکثر ۲۵۵ واحد معادل با یک بایت به کلمهٔ ورودی اضافه نمود. به بلوک FB 18 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 کلمهٔ 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 کلمهٔ ورودی نمایش می‌دهیم.

$۴۶۱_{(۱۰)} = ۱۱۱۰۰۱۱۰۱_{(۲)} = ۰۰۰۰۰۰۰۱۱۱۰۰۱۱۰۱_{(۲)} \longrightarrow$ IW 22

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

```
FB 19

SEGMENT  1              0000
NAME :PLUS1

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

1 - Increament

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

IW 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | $۱۱۱۰۰۱۱۰۱_{(۲)} = ۴۶۱_{(۱۰)}$

QW 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | $۱۱۱۰۱۱۱۰۰_{(۲)} = ۴۷۶_{(۱۰)}$

۴-۴-۱۰- دستور D[1]

به کمک این دستور می‌توان حداکثر ۲۵۵ واحد از کلمهٔ ورودی تفریق نمود. بلوک FB 20 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 برنامهٔ این بلوک از محتویات کلمهٔ ورودی ۲۲، ۱۲ واحد کسر می‌گردد.

مثال ۴-۱۶ : با استفاده از دستور D از عدد $۴۶۱_{(۱۰)}$، ۱۰۰ واحد کم کنید.

ابتدا عدد $۴۶۱_{(۱۰)}$ را به مبنای ۲ تبدیل نموده، آن را به صورت یک کلمهٔ ورودی در نظر می‌گیریم:

$۴۶۱_{(۱۰)} = ۱۱۱۰۰۱۱۰۱_{(۲)} = ۰۰۰۰۰۰۰۱۱۱۰۰۱۱۰۱_{(۲)} \longrightarrow$ IW 22

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

```
 FB 21

SEGMENT  1          0000
NAME :MINU1
0005       :L   IW  22
0006       :D      100
0007       :T   QW   8
0008       :BE
```

1 - Decreament

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

IW 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | $۱۱۱۰۰۱۱۰۱_{(۲)} = ۴۶۱_{(۱۰)}$

QW 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | $۱۰۱۱۰۱۰۰۱_{(۲)} = ۳۶۱_{(۱۰)}$

۴-۴-۱۱- دستور ADD

در مورد دستورات I و D خاطر نشان کردیم که حداکثر مقداری که می‌توان به کلمهٔ 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 فاصلهٔ [۳۲۷۶۸+ و ۳۲۷۶۸-] 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 FB 54 نحوهٔ 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 کلمهٔ ورودی استفاده می‌شود و پس از انجام عملیات، عدد حاصل به یک کلمهٔ خروجی فرستاده می‌شود. واضح است که می‌توان با اضافه نمودن عددی منفی به کلمهٔ ورودی، عملیات تفریق را نیز انجام داد. در بلوک FB 55 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[1]

این دستور در انجام برخی اعمال ریاضی قابل استفاده و اجرا خواهد بود. در بلوک 50 FB 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، بدنهٔ بلوک برنامه‌نویسی شده است.

در این برنامه ابتدا حاصل تفریق دو کلمهٔ 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 FB 14 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[1]

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

---

1 - Jump if Not Zero

```
 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 کلمهٔ 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 مقایسهٔ دو کلمه استفاده نمود. در بلوک FB 45 برنامهٔ مقایسهٔ دو کلمهٔ 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[1]

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 FB 46 این جایگزینی انجام شده است.

---

1 - Jump if Positive

```
 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[1]

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

1 - Jump if Minus

دیگر، پرش هنگامی صورت می‌گیرد که عدد اول یعنی IW 16 از عدد دوم یعنی IW 22 کوچکتر باشد. بنابراین از این دستور می‌توان به جای دستور مقایسه‌ای (F <) استفاده نمود. در بلوک FB 47 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 (PB 31، PB 40، DB 28، OB 1 و ...) اندازه‌گیری نمودیم. در اینجا قصد داریم تا با استفاده از بلوک تابع‌ساز، برنامهٔ 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 FB 10:

```
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
```

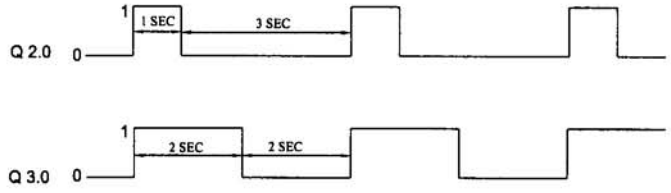
روند اجرای برنامه به ترتیب زیر است:

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 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 در 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.



برنامهٔ مطلوب در PB 40 نوشته شده است.

```
PB 40

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
```

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

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

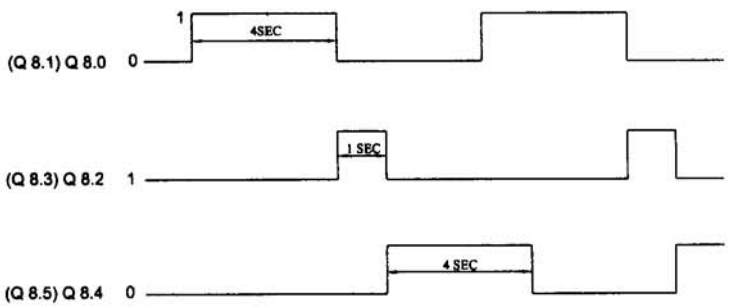
در صورتی که تایمر به 90 برسد خروجی Q 2.0 را ری‌ست می‌کند.

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 برنامهٔ 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. ادامهٔ این برنامه را به خوانندگان واگذار می‌کنیم.

در اکثر پروسه‌ها 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. برنامهٔ 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[1] 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 شمارهٔ 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 لبهٔ بالارونده ورودی است.

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

```
FB 100
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 KT
000D        :SEC  =TIM                نوع تایمر
000E        :L    =TIM          خروجی باینری تایمر
000F        :T    FW   20       و ارسال به FW 20
0010        :BE
```

1 - Blinking LED

بلوک OB 1 نیز به صورت زیر برنامه‌نویسی می‌گردد.

```
 OB 1

SEGMENT  1              0000
0000        :JU   FB 100
0001 NAME   :BLINK
0002 TIM    :     T     8
0003 DATA   :     KT  050.1
0004        :A    F    21.2
0005        :=    Q     8.0
0006        :A    F    21.3
0007        :=    Q     9.0
0008        :BE
```

انتخاب شمارهٔ تایمر

زمان موردنظر در تایمر به همراه تولرانس آن

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

فصل پنجم

شیوه‌های کنترل فرآیند

۵-۱- کنترل فرآیندها

در کنترل فرآیندها معمولاً با دو نوع برنامه مواجه هستیم:

۵-۱-۱- برنامه‌های ترکیبی (Combination Logic)

در این‌گونه برنامه‌ها شرایط موجود در مرحلهٔ فعلی، حرکت به سوی مرحلهٔ بعد را مشخص می‌نماید و هیچ‌گونه اطلاعاتی در مورد انجام مرحلهٔ بعد وجود ندارد، مانند پروسهٔ کنترلی راکتورها، بویلرها و ... . در حقیقت در برنامه‌های ترکیبی، شرایط فعلی موجود، اجرای مرحلهٔ بعد را تعریف می‌نمایند.

۵-۱-۲- برنامه‌های ترتیبی (Sequential Logic)

در این برنامه‌ها انجام یک مرحله، مشروط به انجام مرحله یا مراحل قبلی است یا به طور خلاصه در این برنامه‌ها انجام مراحل به ترتیب خاصی صورت می‌گیرد. مثلاً در سیستم آبکاری فلزات، ابتدا باید زنگ زدایی انجام شود و پس از اتمام این مرحله، مرحلهٔ بعدی انجام گیرد.

در برنامه‌های ترتیبی آغاز مرحله بعدی به دو صورت زیر مشخص می‌گردد.

۱- اتمام یک مرحله آغاز مرحلهٔ بعد را تعریف می‌کند که به این برنامه‌ها Process Dependent گویند.

۲- اتمام زمان انجام یک مرحله، آغاز مرحلهٔ بعد را مشخص می‌کند. به این برنامه‌ها Time - Controlled گویند.

در اکثر پروسه‌های صنعتی تلفیقی از دو برنامهٔ ترکیبی و ترتیبی در کنترل فرآیندها مورد استفاده قرار می‌گیرد.

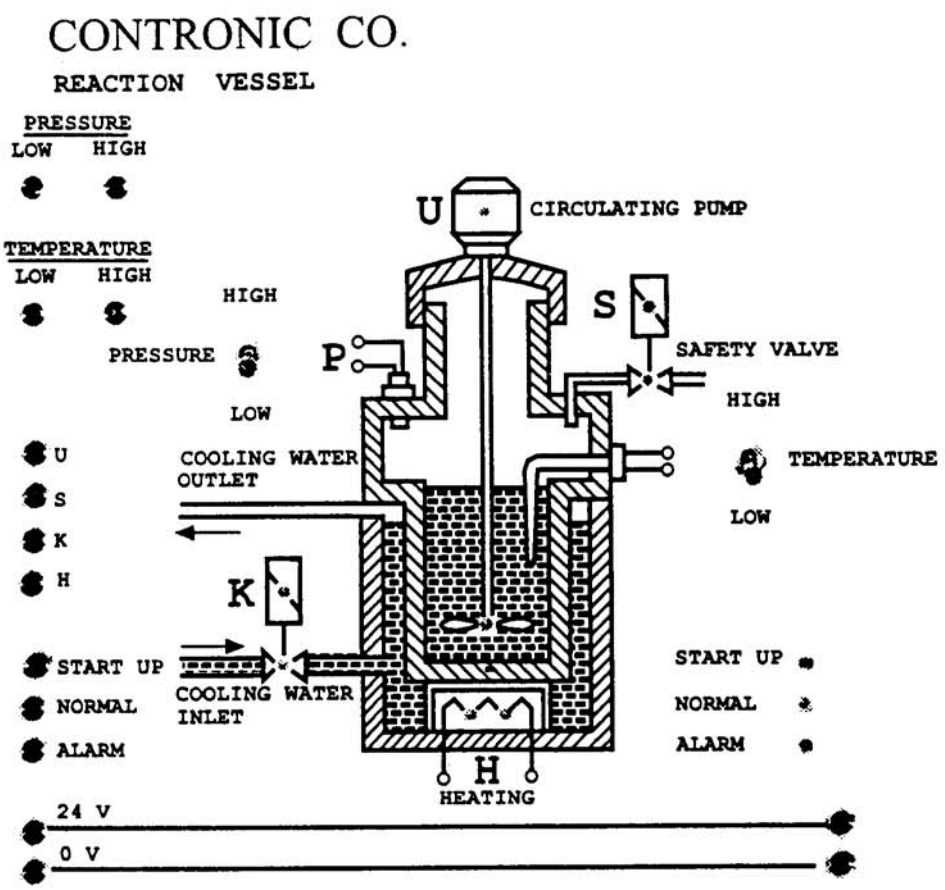
برای روشن شدن مطلب به ذکر یک مثال کاملاً عملی و کاربردی در مورد برنامه‌های ترکیبی می‌پردازیم.

مثال ۵-۱ : در کنترل فرآیندهایی نظیر راکتورها، مخازن تحت فشار کوره‌ها و ... رعایت مسائل ایمنی، صدور علائم خطر و انجام مانورهای مناسب به صورت اتوماتیک علاوه بر عملکرد نرمال سیستم کاملاً ضروری است. شبیه ساز "Reaction Vessel" برای آشنایی با چنین فرآیندهایی طراحی شده است. همان‌گونه که در شکل ۵-۱ دیده می‌شود راکتورهای مواد شیمیایی علاوه بر همزن معمولاً دارای فشارسنج و دماسنج برای اندازه‌گیری فشار و درجه حرارت داخل مخزن، همچنین سیستم گرم کننده و خنک کننده جهت کنترل درجه حرارت و شیر اطمینان جهت کنترل فشار و شرایط ایمنی می‌باشند. علاوه بر موارد مذکور سیستم‌های هشدار دهنده‌ای در راکتورها جهت اعلام نقص‌های احتمالی تعبیه شده‌اند. این سیستم‌ها چگونگی وضعیت عملکرد راکتورها را نشان می‌دهند مانند LEDهای نشان دهنده، آژیر یا ... .

در این مثال قصد داریم برنامهٔ کنترل راکتور را به گونه‌ای بنویسیم که:

۱- چنانچه درجه حرارت یا فشار داخلی سیستم پائین باشد و یا به عبارت دیگر آلارم‌های TEMPERATURE LOW یا PRESSURE LOW وجود داشته باشد سیستم گرم کننده (HEATING) و همزن شروع به کار نموده، LED مربوط به حالت START UP روشن شود.

۲- در صورتی که شرایط نرمال باشد یعنی فشار و درجه حرارت نه بالا و نه پائین باشد یا به عبارت دیگر هیچ یک از آلارم‌های PRESSURE LOW ، PRESSURE HIGH ، TEMPERATURE LOW ، TEMPERATURE HIGH وجود نداشته باشد، تنها LED مربوط به حالت NORMAL روشن شود.



شکل ۵-۱ : طرح ساده‌ای از شبیه‌ساز Reaction Vessel

۳- در صورتی که فشار داخلی سیستم بالا باشد و یا آلارم PRESSURE HIGH وجود داشته باشد شیر اطمینان یا SAFETY VALVE باز شده، LED مربوط به حالت ALARM نیز چشمک بزند.

۴- در صورتی که درجه حرارت داخلی سیستم بالا باشد و یا آلارم TEMPERATURE HIGH وجود داشته باشد فرمان باز شدن ورودی آب خنک کننده (COOLING WATER) صادر شده، LED مربوط به حالت ALARM چشمک بزند و همزن نیز شروع به کار کند.

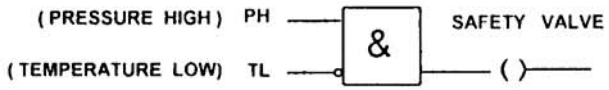
جهت نوشتن این برنامه تمام مراحل ذکر شده در روش برنامه‌نویسی فصل قبل را مرحله به مرحله دنبال می‌کنیم.

۱- **تعریف پروژه**: همانگونه که ذکر شد پروژهٔ کنترل راکتور در ۳ حالت START UP ، NORMAL و ALARM که هر یک معرف وجود شرایط خاصی هستند تعریف شده است.

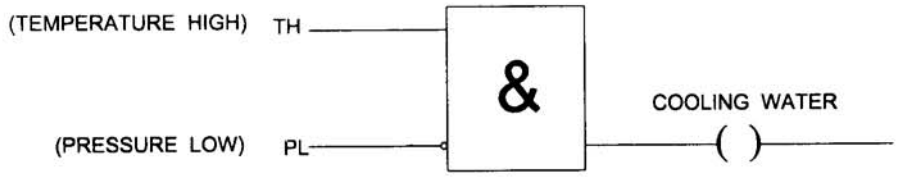
۲- **رسم فلوچارت برنامه**: برای رسم فلوچارت، در نظر گرفتن شرایط ایمنی و اعمال آنها در فلوچارت الزامی است. در رسم فلوچارت، ورودی‌ها و خروجی‌ها بر اساس اسامی آنها در سیستم نامگذاری شده‌اند. به عنوان مثال ورودی TH معرف حالت TEMPERATURE HIGH می‌باشد. به دلیل این که در رسم فلوچارت برنامه ملزم به در نظر گرفتن شرایط ایمنی هستیم لذا مرحلهٔ پنجم برنامه‌نویسی راکه همان بررسی شرایط ایمنی است در این قسمت توضیح خواهیم داد. اکنون شرایط ایمنی را با ذکر چند سؤال بررسی می‌کنیم.

سؤال اول: شیر اطمینان در چه شرایطی عمل می‌کند؟

افرادی که محیط‌های صنعتی و فرآیندهای کوچک و بزرگ را تجربه نموده‌اند به خوبی می‌دانند که شیر اطمینان در حالتی که فشار داخلی بالا بوده، درجه حرارت داخلی سیستم نیز پائین نباشد عمل می‌کند. زیرا در حالتی که فشار داخلی سیستم بالا می‌رود درجه حرارت نیز به نوبهٔ خود افزایش می‌یابد و در صورتی که در این حالت درجه حرارت داخلی سیستم پائین باشد باید به صحت عملکرد سنسورهای درجه حرارت شک نمود. بنابراین شرایط لازم جهت عملکرد شیر اطمینان را می‌توان به صورت زیر نشان داد:



سیستم آب خنک کننده نیز هنگامی شروع به کار می‌کند یا به عبارت دیگر فرمان باز شدن شیر آب خنک کننده هنگامی صادر می‌شود که درجه حرارت داخل سیستم بالا بوده، فشار داخلی نیز پائین نباشد.



شاید در ذهن خوانندگان این سؤال مطرح شود که ارتباط بین شروع به کار سیستم آب خنک‌کننده و پائین نبودن فشار داخل سیستم چیست؟ و یا به عبارت دیگر، آیا وجود شرط درجه حرارت بالا به تنهایی برای شروع به کار سیستم مذکور کافی است؟

در پاسخ به این سؤال باید گفت که بالا بودن درجه حرارت داخل سیستم همراه با بالا رفتن فشار داخلی آن می‌باشد و در صورت داشتن درجه حرارت بالا و فشار پائین احتمال معیوب بودن یکی از دو سنسور فشار و دما وجود دارد.

توجه داشته باشید که در دو مورد فوق یعنی باز شدن شیر اطمینان و به کار افتادن سیستم خنک‌کننده شرایط ایمنی نظیر احتمال معیوب بودن سنسورهای فشار و درجه حرارت نیز در نظر گرفته شده است.

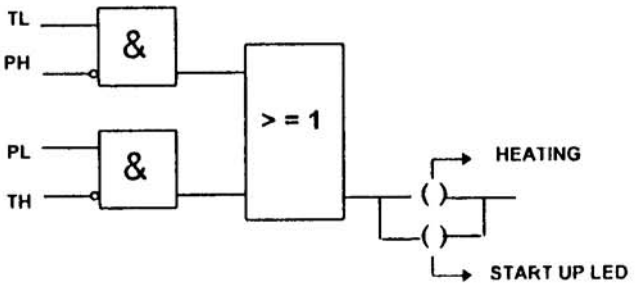
سؤال دوم: شرایط لازم جهت به کار افتادن سیستم گرم کننده (HEATING) چیست؟

سیستم گرم کننده در صورت برقراری لااقل یکی از دو شرط زیر به کار می‌افتد.

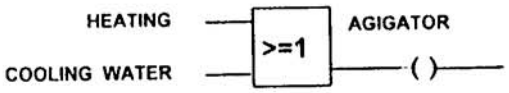
۱- در صورتی که درجه حرارت پائین بوده، فشار بالا نباشد. زیرا داشتن درجه حرارت پائین به همراه فشار بالا دو شرط متضاد و متناقض یکدیگرند که وجود یکی، عدم وجود دیگری را ثابت می‌کند. بنابراین در این حالت نیز احتمال معیوب بودن سنسورها در نظر گرفته شده است.

۲- در حالتی که فشار پائین بوده، درجه حرارت بالا نباشد. در این حالت نیز نظیر حالت قبلی احتمال معیوب بودن سنسورها در نظر گرفته شده است، زیرا در صورتی که فشار پائین باشد به طور منطقی نباید انتظار بالا بودن درجه حرارت را داشته باشیم. پس در صورتی که فشار پائین و درجه حرارت بالا باشد احتمالاً یکی از سنسورها معیوب است.

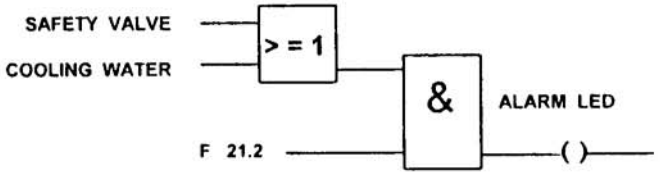
شرایط لازم جهت به کار افتادن سیستم گرم کننده در فلوچارت زیر نشان داده شده است. از آنجایی که به همراه روشن شدن سیستم گرم کننده لازم است که LED مربوط به حالت START UP نیزروشن شود فرمان روشن‌شدن این LED درفلوچارت زیر در نظر گرفته شده است.



روشن شدن همزن الکتریکی مستلزم روشن بودن و یا در حال کار بودن سیستم گرم کننده یا سیستم خنک کننده می‌باشد زیرا در حالتی که سیستم گرم کننده در حال کار باشد روشن بودن همزن الکتریکی جهت یکسان سازی درجه حرارت مواد داخل سیستم الزامی است. در مورد سیستم خنک‌کننده نیز همین استدلال را می‌توان بیان نمود.



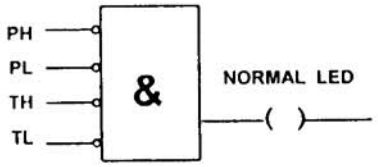
شرایط لازم برای چشمک زدن LED مربوط به حالت ALARM در فلوچارت زیر آمده است.



همانگونه که در فلوچارت فوق دیده می‌شود LED مربوط به حالت ALARM هنگامی چشمک می‌زند که شیر اطمینان باز شده باشد یا اینکه سیستم خنک کننده در حال کار باشد. F 21.2 سرعت چشمک زدن LED مربوطه را معین می‌کند. (در بلوک OB 1 در مورد این فلگ و چگونگی تعیین سرعت چشمک زدن به تفصیل سخن خواهیم گفت.)

سؤال سوم: شرایط لازم برای وضعیت NORMAL چیست؟

- در وضعیت NORMAL نباید فشار و درجه حرارت بالا یا پائین باشد به عبارت دیگر دما و فشار باید در محدودهٔ قابل قبول و تعریف شده برای سیستم وجود داشته باشد. در فلوچارت زیر این شرایط نشان داده شده‌اند.



همان‌گونه که قبلاً نیز ذکر شد در تمامی حالات فوق احتمال معیوب بودن سنسورهای درجه حرارت و فشار در نظر گرفته شده است.

از آنجایی که سیستم راکتورهای شیمیایی واقعاً حساس بوده و کنترل این‌گونه سیستم‌ها از اهمیت بالایی برخوردار است در برنامه‌نویسی بایستی احتمال معیوب بودن سنسورها نیز در نظر گرفته شود و برنامه به شیوه‌ای نوشته شود که در صورت معیوب بودن حتی یک سنسور، هیچ عملی انجام نشود.

۳- تهیه لیستی از ابزار مورد نیاز: جهت وارد نمودن اطلاعات در مورد ورودی‌ها و خروجی‌ها از عملوندهای زیر استفاده می‌کنیم:

ورودی‌ها
- PH : I 21.0
- TL : I 21.1
- TH : I 21.2
- PL : I 21.3

خروجی‌ها
- SAFETY VALVE : Q 14.0
- COOLINGWATER : Q 14.1
- HEATING : : Q 14.2
- START UP LED : Q 14.3
- AGIGATOR : Q 14.4
- ALARM LED : Q 14.5
- NORMAL LED : Q 14.6

۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: جهت نوشتن این برنامه از روش CSF استفاده شده است.

به دلیل اینکه کنترل راکتور شامل چندین مرحله است برنامهٔ نوشته شده در بلوک PB 20 شامل ۶ قسمت (SEGMENT) می‌باشد. برای اجرای این برنامه نیاز به تعریف بلوک OB 1 داریم و برای تعیین سرعت چشمک زدن LED هشدار دهنده از بلوک FB 100 که در مثال ۴-۲۰ برنامه‌نویسی شد استفاده می‌کنیم.

```
 OB 1

SEGMENT  1          0000
0000       :JU  FB 100
0001 NAME  :BLINK
0002 TIM   :     T     8
0003 DATA  :     KT 050.1
0004       :JU  PB  20
0005       :BE


 FB 100

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
000C       :LW   =DATA
000D       :SEC  =TIM
000E       :L    =TIM
000F       :T    FW  20
0010       :BE


 PB 20

SEGMENT  1          0000
             +---+
 I 21.0   ---! & !     +------+
 I 21.1   --O!   !--+-! =    ! Q 14.0
             +---+    +------+
```

```
SEGMENT  2          0004
            +---+
 I 21.2   ---! & !     +------+
 I 21.3   --O!   !--+-! =    ! Q 14.1
            +---+    +------+

SEGMENT  3          0008
            +---+
 I 21.1   ---! & !      +---+
 I 21.0   --O!   !------!>=1!
            +---+       !   !
            +---+       !   !
 I 21.3   ---! & !      !   !     +------+
 I 21.2   --O!   !------!   !--+-! =    ! Q 14.2
            +---+      +---+   ! +------+
                               ! +------+
                              +-! =    ! Q 14.3
                                +------+

SEGMENT  4          0010
            +---+
 Q 14.2   ---!>=1!     +------+
 Q 14.1   ---!   !--+-! =    ! Q 14.4
            +---+    +------+

SEGMENT  5          0014
            +---+
 Q 14.0   ---!>=1!      +---+
 Q 14.1   ---!   !------! & !
            +---+       !   !     +------+
          F 21.2   ---!   !--+-! =    ! Q 14.5
                      +---+    +------+

SEGMENT  6          001B
            +---+
 I 21.0   --O! & !
 I 21.1   --O!   !
 I 21.2   --O!   !     +------+
 I 21.3   --O!   !--+-! =    ! Q 14.6
            +---+    +------+:BE
```

روند اجرای برنامه به صورت زیر است:

سطر اول در OB 1 دستور پرش به برنامه بلوک FB 100 را صادر می‌کند. سپس در همین بلوک نام برنامه و شمارهٔ تایمری که در برنامه‌های دیگر استفاده نشده است از کاربر خواسته می‌شود. در قسمت DATA، کاربر، زمان تایمر و همچنین تولرانس و دقت چشمک زدن را مشخص می‌کند. همان‌گونه که می‌دانید ضریب زمانی ۱ سبب می‌شود که کم ارزش‌ترین بیت FW 20 یعنی F 21.0 با سرعت 0.1 ثانیه و F 21.1 با دو برابر سرعت بیت قبلی خود یعنی F 21.0 چشمک بزند و به همین ترتیب الی آخر. در این مثال از F 21.2 جهت تعیین سرعت چشمک زدن LED مربوط به حالت ALARM استفاده شده است یعنی سرعت چشمک زدن این LED، ۰/۴ ثانیه است. در سطر پنجم فرمان پرش به PB 20 صادر شده است. از این پس پردازنده دستورات موجود در PB 20 را سطر به سطر اجرا نموده، به محض رسیدن به انتهای بلوک به OB 1 باز می‌گردد و دستور BE موجود در OB 1 اتمام اجرای برنامه را اعلام می‌کند.

جهت ادامهٔ بحث نیاز به تعریف یک دستور جدید داریم.

۵-۲- دستور DO

این دستور یکی از دستورات تکمیلی است که تنها در FBها قابل استفاده است. این دستور را می‌توان در مورد DWها و FWها اعمال نمود.

اکنون فرض کنید قصد داریم در یک بلوک DB که شامل ۱۰۰ کلمه (DW) می‌باشد دستورات زیررا اجراکنیم. به عبارت دیگر محتویات تمام کلمه‌های موجود در این DB را با استفاده از دستور L بارگذاری کنیم.

```
: L  DW  0
: L  DW  1
: L  DW  2
     ⋮
: L  DW  99
```

جهت جلوگیری از طولانی شدن برنامه و همچنین صرفه‌جویی در زمان از دستور DO استفاده می‌کنیم. بدین ترتیب که آدرس اولین DW را خوانده، سپس با آدرس دهی‌های پیاپی اطلاعات موجود در این DWها را بارگذاری می‌کنیم. به این نوع آدرس دهی، آدرس‌دهی غیرمستقیم (Indirect Addressing) گویند.

مثال ۵-۲: قصد داریم در DB 30 و در کلمه‌های DW 15 الی DW 45 عدد صفر $(0000)_{(H)}$ را بارگذاری کنیم. در صورتی که از دستور L و T استفاده کنیم برنامهٔ مذکور به این صورت خواهد بود:

```
 FB 5

SEGMENT  1              0000
NAME :ZERO

0005        :C    DB  30
0006        :L    KF +0
0008        :T    DW  15
0009        :T    DW  16
000A        :T    DW  17
000B        :      :
000C        :      :
000D        :T    DW  44
000E        :T    DW  45
000F        :BE
```

ملاحظه می‌کنید که اجرای این برنامه در مورد بارگذاری ۳۰ کلمه تا چه اندازه طولانی شده است. واضح است که برای مقادیر بالاتر، برنامه طولانی‌تر خواهد شد.

حال با استفاده از دستور DO این عمل را انجام می‌دهیم. این برنامه در FB 6 به صورت زیر نوشته شده است.

```
 FB 6

SEGMENT  1              0000
NAME :DO

0005        :C    DB  30
0007        :L    KF +15
0008 M001   :T    FW  30
000A        :L    KF +0
000B        :DO   FW  30
000C        :T    DW   0
000D        :L    FW  30
000E        :I        1
000F        :T    FW  30
0011        :L    KF +45
0012        :<=F
0013        :JC   =M001
0014        :BE
```

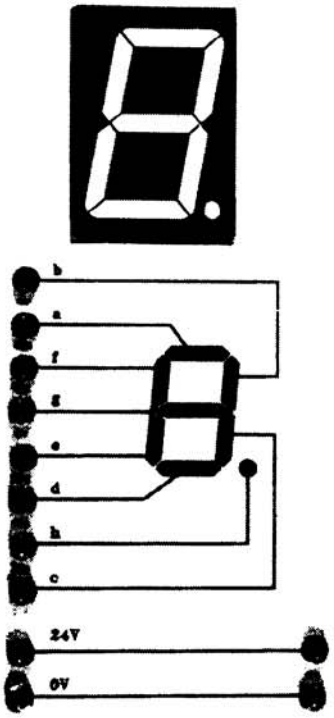
در سطر اول این برنامه از دستور C DB 30 به منظور صدا زدن و معتبر نمودن DB 30 استفاده شده است و این بدان معنی است که از این پس تمام اعمال L و T بر روی DWهای موجود در این DB انجام می‌گیرد. در سطر دوم و سوم با استفاده از دو دستور L و T عدد دهدهی ۱۵ در FW 30 قرار می‌گیرد. عدد ۱۵ آدرس شروع کلمه‌هایی است که قرار است بر روی آنها عملیاتی انجام گیرد. در سطر چهارم عدد دهدهی ۰ در انبارک بارگذاری می‌شود. در دستور DO FW 30، محتوای FW 30 (یا هر FW دیگری که بعد از دستور DO قرار می‌گیرد) آدرس کلمه‌ای از DB خواهد بود که باید اطلاعات به آنجا ارسال شوند. در دستور بعدی یعنی T DW 0 محتویات کلمهٔ شمارهٔ ۱۵ به کلمهٔ شماره ۰ انتقال می‌یابد. در دستور L FW 30 و I 1 به عدد ثابت ۱۵، یک واحد اضافه شده، حاصل مجدداً به FW 30 انتقال می‌یابد. سپس عدد حاصل با عدد ۴۵ مقایسه می‌شود و در صورتی که از ۴۵ کوچکتر و یا با آن مساوی باشد اجرای برنامه از سطری که با برچسب M001 مشخص شده، دنبال می‌گردد. اجرای این برنامه همچنان ادامه می‌یابد تا اینکه محتوای FB 30 که آدرس کلمه‌ای از DB است از ۴۵ تجاوز نماید در این حالت اجرای برنامه پایان می‌یابد و مجدداً از ابتدای برنامه، سیکل مذکور تکرار می‌شود.

همانگونه که ملاحظه نمودید با استفاده از دستور DO آدرس خطها یا کلمه‌های موجود در DB را تغییر داده، سپس دستورات L و T را بر روی آنها انجام می‌دهیم. در این قسمت به ذکر یک مثال کاملاً کاربردی و عملی در مورد چگونگی استفاده از دستور DO می‌پردازیم.

مثال ۵-۳: اکثر خوانندگان با LEDهای نورانی که هفت قسمتی بوده و Seven Segment نامیده می‌شوند آشنایی دارند. شکل ۵-۲ را که در واقع شبیه‌ساز طراحی شده برای برنامه‌نویسی این مثال می‌باشد در نظر بگیرید.

در این مثال قصد داریم برنامه‌ای بنویسیم که با استفاده از آن، اعداد ۰ تا ۹ با فاصلهٔ زمانی ۰/۸ ثانیه بر روی Seven Segment به نمایش در آیند. در گوشهٔ سمت راست Seven Segment یک نقطه (Point) نیز وجود دارد که با نمایش هر عدد، این نقطه باید روشن باشد.

با اندکی دقت در می‌یابیم که برای نوشتن این برنامه کافی است اعداد ۰ تا ۹ را به معادل هگزادسیمال تبدیل نموده، آنها را با فاصلهٔ زمانی ۰/۸ ثانیه بر روی صفحهٔ نمایش هفت قسمتی بفرستیم. واضح است که معادل هگز این اعداد باید در یک بلوک DB تعریف شوند.



شکل ۵-۲ : شمای یک نمایش دهندهٔ هفت قسمتی (7 Segment)

از آنجایی که برای روشن شدن هر یک از قسمتهای Seven Segment باید آنها را بـه یکـی از خروجی‌های PLC نسبت دهیم معادل با هر قسمت، یکی از خروجی‌های 14 QB راکه در ادامه نشان داده شده است در نظر می‌گیریم.

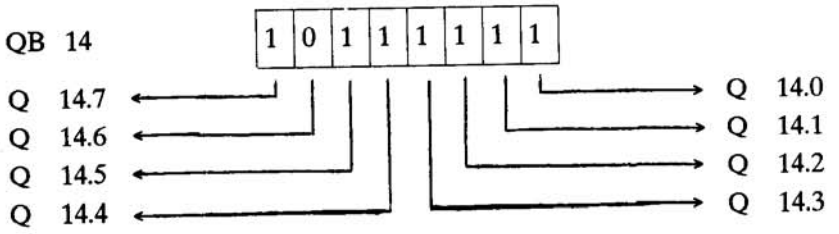
a ≡ Q 14.0	e ≡ Q 14.4
b ≡ Q 14.1	f ≡ Q 14.5
c ≡ Q 14.2	g ≡ Q 14.6
d ≡ Q 14.3	h ≡ Q 14.7

نسبت دادن هر یک از Segmentها به یک بیت در بایت خروجی ۱۴ بدین معنی است که به عنوان مثال برای روشن شدن قسمت a در Seven Segment باید خروجی Q 14.0 فعال شود و

برای روشن شدن Point کافی است که "۱" = Q 14.7 و ... . اکنون معادل هگز هر یک از اعداد ۰ تا ۹ را به دست می‌آوریم. برای مثال عدد ۰ را در نظر می‌گیریم. این عدد در این نمایش دهنده به صورت .0 به نمایش در می‌آید. برای نمایش این عدد کافی است که بیت‌های خروجی QB 14 به صورت زیر باشند:

a ≡ Q 14.0 = "۱"	e ≡ Q 14.4 = "۱"
b ≡ Q 14.1 = "۱"	f ≡ Q 14.5 = "۱"
c ≡ Q 14.2 = "۱"	g ≡ Q 14.6 = "۰"
d ≡ Q 14.3 = "۱"	h ≡ Q 14.7 = "۱"

پس برای نمایش .0 بر روی Seven Segment کافی است که QB 14 به صورت زیر باشد.

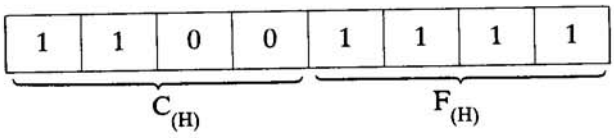


سپس مقدار موجود در این کلمهٔ خروجی یعنی $۱۰۱۱۱۱۱۱_{(۲)}$ را به مبنای ۱۶ تبدیل می‌کنیم.

$۱۰۱۱\ ۱۱۱۱_{(۲)} = BF_{(H)}$

بنابراین برای نمایش .0 کافی است که عدد $BF_{(H)}$ را در DW 0 قرار داده، سپس آن را به خروجی بفرستیم. برای نمایش .3 باید ارزش بیت‌های QB 14 به صورت زیر باشد:

a ≡ Q 14.0 = "۱"	e ≡ Q 14.4 = "۰"
b ≡ Q 14.1 = "۱"	f ≡ Q 14.5 = "۰"
c ≡ Q 14.2 = "۱"	g ≡ Q 14.6 = "۱"
d ≡ Q 14.3 = "۱"	h ≡ Q 14.7 = "۱"



به عبارت دیگر می‌توان مقدار CF_H را در DW 3 قرار داده، سپس آن را به خروجی فرستاد.
یافتن معادل هگز سایر اعداد را به عنوان تمرین به خواننده واگذار می‌کنیم.
مقادیر معادل هگز نمایش اعداد .0 تا .9 در ادامه آمده است.

0. ≡ $BF_{(H)}$	5. ≡ $ED_{(H)}$
1. ≡ $86_{(H)}$	6. ≡ $FD_{(H)}$
2. ≡ $DB_{(H)}$	7. ≡ $87_{(H)}$
3. ≡ $CF_{(H)}$	8. ≡ $FF_{(H)}$
4. ≡ $E6_{(H)}$	9. ≡ $EF_{(H)}$

برای تکمیل برنامه احتیاج به یک شمارنده داریم به طوری که هر ۰/۸ ثانیه یک بار عدد نمایش داده شده را یک واحد افزایش دهد. در مثال قبل دیدیم که با استفاده از دستور DO، آدرس‌ها را با دستور I 1، یک واحد افزایش دادیم. در اینجا مجاز به استفاده از دستور I 1 نیستیم زیرا سرعت انتقال اطلاعات به خروجی و تغییر وضعیت صفحهٔ نمایش هفت قسمتی به قدری سریع است که چشم انسان قدرت تشخیص و تمییز اعداد نمایش داده شده را از یکدیگر ندارد، چراکه عمل انتقال اطلاعات و تغییر وضعیت صفحهٔ نمایش با سرعت Cycle Time و در حدود چند میلی ثانیه انجام می‌گیرد. برای رفع این مشکل از برنامهٔ چشمک‌زن نوشته شده در FB 100 استفاده می‌کنیم.

همان‌گونه که قبلاً توضیح داده شد برنامهٔ FB 100 یکی از پرکاربردترین برنامه‌ها است که ابتدا شمارهٔ تایمری که در برنامه‌های دیگر استفاده نشده، سپس سرعت انتقال اطلاعات یا تولرانس تایمر را از کاربر سؤال می‌کند و استفاده‌کننده می‌تواند با انتخاب هر یک از بیت‌های FW 20 تعریف شده در FB 100 سرعت چشمک زدن را تغییر دهد. بنابراین برای نوشتن این برنامه ابتدا یک DB ایجاد نموده، معادل هگز تمام اعداد ۰ تا ۹ را در آن قرار می‌دهیم. سپس با تعریف یک FB مثلاً FB 35، DB ایجاد شده را معتبر می‌نماییم و با استفاده از دستور DO و آدرس‌دهی غیرمستقیم، DWهای موجود را به خروجی می‌فرستیم. در OB 1 که ساختار کلی برنامه را مشخص می‌کند دستور پرش به FB 35 را صادر و سپس با فراخوانی FB 100 سرعت چشمک زدن را مشخص می‌کنیم. برنامهٔ نوشته شده در ادامه آورده شده است.

(در اینجا از نوشتن FB 100 خودداری شده است زیرا همان‌طور که گفته شد بلوک‌های FB را تنها یک بار نوشته، در صورت نیاز، با فراخوانی آنها مقادیر پارامتر را وارد می‌کنیم.)

```
 OB 1

SEGMENT  1            0000
0000       :JU  FB  35
0001 NAME  :SEGMENT
0002       :JU  FB 100
0003 NAME  :BLINK
0004 TIM   :    T    8
0005 DATA  :    KT 050.1
0006       :BE


 FB 35

SEGMENT  1            0000
NAME :SEGMENT

0005       :C   DB  31
0006       :A   F   21.3
0007       :CU  C    5
0008       :L   C    5
0009       :T   FW  30
000A       :DO  FW  30
000B       :L   DW   0
000C       :T   QB  14
000D       :L   FW  30
000E       :L   KF +10
0010       :>=F
0011       :R   C    5
0012       :BE


DB31

    0:      KH = 00BF;
    1:      KH = 0086;
    2:      KH = 00DB;
    3:      KH = 00CF;
    4:      KH = 00E6;
    5:      KH = 00ED;
    6:      KH = 00FD;
    7:      KH = 0087;
    8:      KH = 00FF;
    9:      KH = 00EF;
   10:      KH = 0000;
```

ممکن است در ذهن خواننده این سؤال مطرح شود که دلیل استفاده از KH 00 در سطر یازدهم DB 31 چیست؟

پاسخ به این سؤال بسیار ساده است. در صورتی که DW 10 که همان KH 00 است در سطر آخر DB 31 گنجانده نشود آخرین عدد نمایش داده شده، با سرعت سیکل زمانی برنامه تغییر می‌کند. در این حالت قادر به مشاهدهٔ این عدد نخواهیم بود، اما با مقایسهٔ DW 10 برای فاصلهٔ زمانی یک سیکل می‌توان آخرین عدد را که 9 می‌باشد به مدت ۰/۸ ثانیه مشاهده نمود.

۵-۳- ارسال پیام‌های خطا بر روی صفحهٔ نمایش

افرادی که با سیستم‌های کنترل مونیتورینگ و پیشرفته آشنایی دارند به خوبی می‌دانند که در سیستم‌های کنترل امروزی پیامهای خطا و هشدار بر روی صفحهٔ مونیتور به نمایش در می‌آیند. این پیامها به حالت چشمک‌زن و یا ثابت بر روی صفحهٔ مونیتور ظاهر می‌شوند. همان‌گونه که در فصول گذشته ذکر شد پیامهای خطا و هشدار در بلوک DB نوشته شده است و به هنگام ایجاد هر گونه اشکال با آدرس‌دهی مناسب می‌توان با معتبر نمودن DB مذکور و بارگذاری DWهای مربوطه پیام دلخواه و متناسب با مشکل ایجاد شده در سیستم را در نقاط مختلف صفحهٔ نمایش ظاهر ساخت. در مثال زیر نحوهٔ ارسال اطلاعات و پیامها به طور مفصل آمده است.

مثال ۵-۴: فرض کنید که در یک سیستم کنترلی ۴ پیام و یا هشدار وجود دارد (خوانندگان می‌دانند که در سیستم‌های کنترلی عظیم ممکن است هزاران پیام بر روی صفحهٔ مونیتور ظاهر شود در این مثال برای جلوگیری از طولانی شدن برنامه تنها ۴ پیام برای سیستم در نظر گرفته شده است.) این پیامها در ۴ سطر از DB 32 نوشته شده‌اند. همان‌طور که قبلاً اشاره شد پیامها را با فرمت KS نمایش می‌دهیم. از آنجایی که پیامهای قابل نمایش بر روی صفحهٔ مونیتور در PLC زیمنس نباید از ۱۲ کلمه (۱۲ کلمه = ۲۴ بایت) تجاوز نماید بنابراین در ستون اول DB 32 که معرف شماره بایت‌های اشغال شده توسط پیامهاست مضاربی از عدد ۱۲ دیده می‌شود. در پیامها فاصلهٔ بین دو کلمه (Blank Space) نیز یک کاراکتر محسوب می‌شود. به دلیل اینکه هر کاراکتر مطابق کد ASCII ، ۸ بیت یا یک بایت از حافظه را اشغال می‌کند پس می‌توان از ۱۲ کلمه (۲۴ بایت) یا به عبارت دیگر ۲۴ کاراکتر در هر سطر استفاده نمود. در DB 32 سطرهای ۱ الی ۴ به پیامها اختصاص

داده شده است. برای مقاصد بعدی برنامه در سطرهای بعد، از کاراکترهای 0 استفاده شده است. در DB 32 چگونگی آرایش کاراکترها و پیامها را در این بلوک مشاهده می‌کنید.

```
DB32

  0:      KS =' tanks levels are stable';
 12:      KS =' motor no-6 is defective';
 24:      KS ='fan is out of order now ';
 36:      KS ='temperature is too high ';
 48:      KS ='000000000000000000000000';
 60:      KS ='000000000000000000000000.'
```

اکنون بلوک DB 33 را در نظر بگیرید که تمام سطرهای آن با کاراکترهای 0 پر شده‌اند.

```
DB33

  0:      KS ='000000000000000000000000';
 12:      KS ='000000000000000000000000';
 24:      KS ='000000000000000000000000';
 36:      KS ='000000000000000000000000';
 48:      KS ='000000000000000000000000';
 60:      KS ='000000000000000000000000';
```

اکنون ۴ ورودی را که به عنوان مثال سیگنال‌های ارسال شده از سنسورهای اندازه‌گیری فشار، درجه حرارت، لرزش و ... هستند در نظر بگیرید. در صورت فعال شدن هر ورودی یکی از پیامها (پیام متناظر با هر سنسور) از DB 32 به DB 33 انتقال می‌یابد.

نحوهٔ انتقال به گونه‌ای است که هر سطر پیام از DB 32 به سطر متناظر خود در DB 33 منتقل می‌شود.به عنوان مثال در صورت فعال شدن ورودی ناشی از سیگنال سنسور درجه حرارت، پیام "temperature is too high" از سطر چهارم DB 32 به سطر چهارم DB 33 انتقال می‌یابد. این پیام در DB 33 به جای کاراکترهای 0 قرار می‌گیرد. در حالت کنترل مونیتورینگ می‌توان تغییرات DB 33 را بر روی صفحهٔ نمایش مشاهده نمود و در صورت ارسال پیام و یا هشدار به رفع اشکال

سیستم پرداخت. در این مثال قصد داریم تا برنامه‌ای بنویسیم که این عمل یعنی انتقال اطلاعات پیام و علائم هشداردهنده را از DB 32 به DB 33 انجام دهد. البته انتقال و ارسال پیامها منوط به فعال شدن ورودی متناظر با هر پیام است.

با اندکی دقت در متن مثال در می‌یابیم که با استفاده از دستور DO می‌توان این برنامه را به راحتی نوشت. در حقیقت، فعال شدن هر ورودی می‌تواند مشخص کنندهٔ آدرس شروع بایت‌های اشغال شده توسط پیام متناظر با آن ورودی باشد. از آنجایی که استفاده از دستور DO تنها در FB مجاز است بنابراین برنامهٔ اصلی را در یک بلوک FB می‌نویسیم.

```
 FB 35

SEGMENT  1            0000
NAME :MESSAGE1

0005        :A    I    16.0      سنسور مربوط به پیام اول
0006        :JC   =M001
0007        :A    I    16.1      سنسور مربوط به پیام دوم
0008        :JC   =M002
0009        :A    I    16.2      سنسور مربوط به پیام سوم
000A        :JC   =M003
000B        :A    I    16.3      سنسور مربوط به پیام چهارم
000C        :JC   =M004
000D        :JU   =M005
000E        :BEU
000F M005   :C    DB   32        DB اولی که حاوی پیام‌هاست.
0010        :DO   FW   30        آدرس‌دهی غیرمستقیم
0011        :L    DW    0
0012        :C    DB   33        DB دومی که حاوی کاراکترهای 0 است.
0013        :DO   FW   40        آدرس‌دهی غیرمستقیم
0014        :T    DW    0
0015        :L    FW   30
0016        :I          1
0017        :T    FW   30
0018        :L    FW   40
0019        :I          1
001A        :T    FW   40
001B        :L    FW   30
001C        :L    KF  +12        انتهای ۱۲ کلمهٔ اول
001E        :!=F
001F        :JC   =M001
```

```
0020          :L    FW  30
0021          :L    KF +24      انتهای ۱۲ کلمهٔ دوم
0023          :!=F
0024          :JC   =M002
0025          :L    FW  30
0026          :L    KF +36      انتهای ۱۲ کلمهٔ سوم
0028          :!=F
0029          :JC   =M003
002A          :L    FW  30
002B          :L    KF +48      انتهای ۱۲ کلمهٔ چهارم
002D          :!=F
002E          :JC   =M004
002F          :BEU
0030 M001     :L    KF +0       آدرس شروع پیام اول
0032          :T    FW  30
0033          :T    FW  40
0034          :BEU
0035 M002     :L    KF +12      آدرس شروع پیام دوم
0037          :T    FW  30
0038          :T    FW  40
0039          :BEU
003A M003     :L    KF +24      آدرس شروع پیام سوم
003C          :T    FW  30
003D          :T    FW  40
003E          :BEU
003F M004     :L    KF +36      آدرس شروع پیام چهارم
0041          :T    FW  30
0042          :T    FW  40
0043          :BE
```

باید توجه داشت که در صورت انتخاب آدرس شروع هر پیام نیاز به برنامه‌ای است که از آدرس شروع، ۲۴ کاراکتر را دریافت و به DB 33 ارسال نماید. قسمتی از برنامه که با برچسب M005 مشخص شده است هر بار که آدرس شروع پیام را دریافت کند از آن آدرسی را با ۱۲ کلمه فراخوانده، انتهای هر ۱۲ کلمه را نیز با استفاده از مقایسه‌گرهای برنامه محاسبه و مشخص می‌کند. در ضمن اگر پیام در صفحهٔ مونیتور سیستم کنترل مونیتورینگ فقط یک بار به مونیتور فرستاده شود به دلیل سرعت پردازنده قابل مشاهده نخواهد بود. بنابراین می‌بایست عمل دریافت و ارسال اطلاعات از DBها تا رفع عیب و اشکال به صورت مداوم و پیوسته بر روی مونیتور ارسال شود.

به دلیل اینکه قسمت خاصی از صفحهٔ مونیتور به نمایش پیامها و علائم هشدار دهنده اختصاص داده شده است و در هر قسمت از مونیتور نمی‌توان پیامهای خطا را به نمایش در آورد. می‌خواهیم با ایجاد تغییراتی در برنامه با فعال و غیرفعال شدن هر سنسور پیغام متناظر با آن سنسور فقط به سطر اول از DB 33 منتقل شود. این برنامه در FB 36 به صورت زیر نوشته شده است.

```
 FB 36

SEGMENT  1           0000
NAME :MESSAGE1

0005       :A    I   16.0
0006       :JC   =M001
0007       :A    I   16.1
0008       :JC   =M002
0009       :A    I   16.2
000A       :JC   =M003
000B       :A    I   16.3
000C       :AN   I   16.0
000D       :AN   I   16.1
000E       :AN   I   16.2
000F       :JC   =M004
0010       :JU   =M005
0011       :BEU
0012 M005  :C    DB  32
0013       :DO   FW  30
0014       :L    DW   0
0015       :C    DB  33
0016       :DO   FW  40
0017       :T    DW   0
0018       :L    FW  30
0019       :I         1
001A       :T    FW  30
001B       :L    FW  40
001C       :I         1
001D       :T    FW  40
001E       :L    FW  30
001F       :L    KF +12
0021       :!=F
0022       :JC   =M001
0023       :L    FW  30
0024       :L    KF +24
```

```
0026        :!=F
0027        :JC   =M002
0028        :L    FW  30
0029        :L    KF +36
002B        :!=F
002C        :JC   =M003
002D        :L    FW  30
002E        :L    KF +48
0030        :!=F
0031        :JC   =M004
0032        :BEU
0033 M001   :L    KF +0
0035        :T    FW  30
0036        :T    FW  40
0037        :BEU
0038 M002   :L    KF +12
003A        :T    FW  30
003B        :L    KF +0
003D        :T    FW  40
003E        :BEU
003F M003   :L    KF +24
0041        :T    FW  30
0042        :L    KF +0
0044        :T    FW  40
0045        :BEU
0046 M004   :L    KF +36
0048        :T    FW  30
0049        :L    KF +0
004B        :T    FW  40
004C        :BE
```

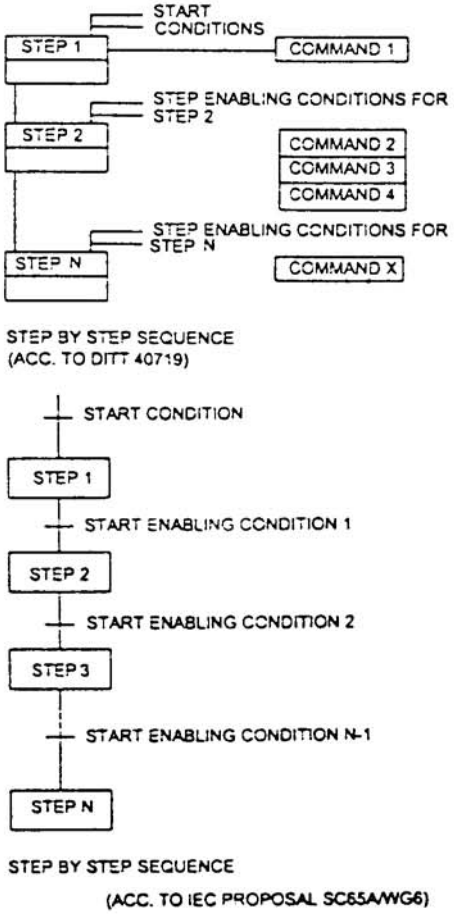
آدرس شروع سطر اول در DB دوم

در صورتی که برنامهٔ بلوک FB 36 را اجرا کنیم خواهیم دید که هنگام فعال و سپس غیرفعال شدن ورودی‌های سنسورها پیام خطای متناظر با هر ورودی بر روی سطر اول DB 33 منتقل می‌شود.

چنانچه در این برنامه بخواهیم در صورت عدم وجود خطا هیچ پیغامی بر روی صفحهٔ مونیتور ظاهر نشود می‌توان سطر پنجم از DB 32 را با کاراکترهای Blank Space پر نمود و با ایجاد تغییرات اندکی در برنامه، اطلاعات این سطر یعنی جای خالی را به سطر اول از DB 33 منتقل کرد. نوشتن این برنامه به عنوان تمرین به خواننده واگذار می‌شود.

۵-۴- ساختار برنامه‌های ترتیبی

برخلاف برنامه‌های ترکیبی که شرایط فعلی برنامه، اجرای مرحلهٔ بعدی را مشخص می‌نماید در برنامه‌های ترتیبی اجرای هر مرحله منوط به انجام زمان اجرای مرحلهٔ قبل و یا اتمام انجام پروسهٔ قبلی است. مشخصهٔ این برنامه‌ها، مرحله به مرحله‌ای بودن آن و برقراری شرایط خاصی برای اجرای مراحل است. در این‌گونه برنامه‌ها، عملیات کنترل پروسه شامل مراحلی جداگانه است که اجرای هر مرحله وابسته به برقراری شرایط تعریف شده‌ای در برنامه می‌باشد. هر مرحله دارای فرمانهای کنترل وشرایط فعال‌سازی‌است. درشکل ۵-۳ اجرای‌مرحله به‌مرحلهٔ یک‌برنامهٔ‌ترتیبی نشان‌داده‌شده است.



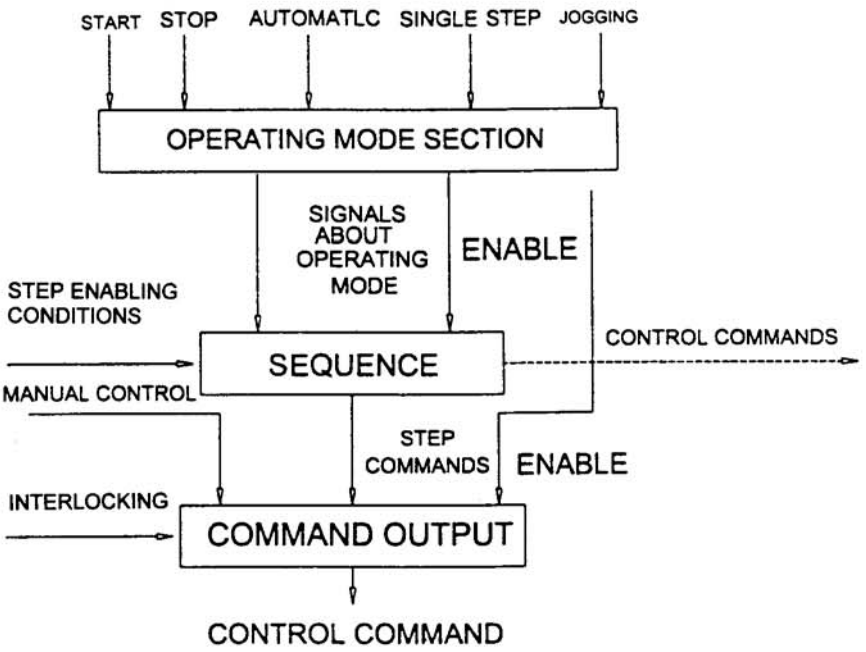
شکل ۵-۳ : اجرای مرحله به مرحلهٔ برنامهٔ ترتیبی

شرایط فعال‌سازی هر مرحله اجازهٔ ورود به مرحلهٔ بعدی و اجرای آن را صادر می‌نماید. فرمانهای هر مرحله شامل دستورات کنترلی برای واحدهای داخلی و خارجی است. به عنوان مثال ست نمودن فلگ‌ها، شروع نمودن زمان شمارش، سوئیچ نمودن المان‌ها و عناصر کنترلی و ...

به طور کلی هر برنامهٔ ترتیبی شامل سه بخش زیر است:

- انتخاب نحوهٔ اجرای مرحله (MODE SECTION)
- اجرای مرحله (SEQUENCE)
- بخش خروجی فرمانها (COMMAND OUTPUT)

در شکل ۵-۴ این قسمتها نشان داده شده است.

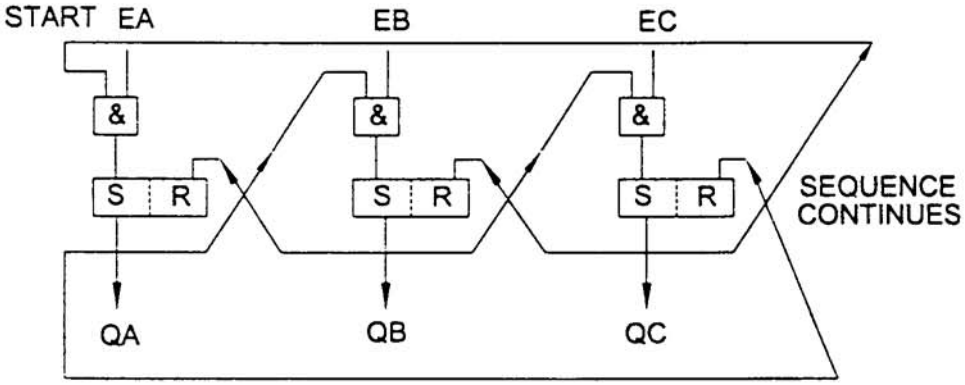


شکل ۵-۴ : ساختار یک کنترل‌کنندهٔ ترتیبی

- MODE SECTION : در این قسمت، پارامترهای اولیهٔ پیش‌نیاز (PRESET) جهت عملکرد مدهای مختلف اجرایی برنامه، پردازش می‌شوند. نتیجهٔ این قسمت به صورت سیگنال‌هایی به قسمت SEQUENCE و فرمانهای خروجی یا COMMAND OUTPUT

ارسال می‌گردد. این مرحله شامل فرمانها یا انتخاب‌های START ، STOP ،AUTOMATIC ، SINGLE STEP و JOGGING می‌باشد.

- در بخش SEQUENCE مراحل مختلف برنامه یکی پس از دیگری و مرحله به مرحله و البته منوط به برقراری شرایط فعال شدن هر مرحله، اجرا می‌شود. هر مرحله یا STEP را می‌توان معادل یا متناظر با قسمتی از برنامهٔ کنترل دانست که برای شروع و گذار از این مرحله نیاز به فرمانهای ست و ری‌ست دارد. این فرمانهای ست و ری‌ست ممکن است از فلیپ‌فلاپ، شمارنده، تایمر و یا ... صادر شوند. شکل زیر گویای مطلب عنوان شده است.



شکل ۵-۵ : نمونهٔ یک برنامهٔ ترتیبی با سه مرحله و چگونگی ست و ری ست شدن هر یک از مراحل

همانگونه که در شکل فوق ملاحظه می‌کنید از خروجی فلیپ‌فلاپ‌ها فرمانهایی صادر می‌شود که هر یک، فرمان آغاز مرحلهٔ بعد و پایان مرحلهٔ قبل است. اجرای مرحلهٔ فعلی در صورتی ادامه می‌یابد که شرایط فعال سازی آن (ENABLE) برقرار باشد. اگرچه فرمانهای کنترلی (CONTROL COMMAND) معمولاً از طریق قسمت خروجی فرمان به المان‌ها و عناصر کنترلی فرستاده می‌شوند اما این فرمانها می‌توانند مطابق شکل ۵-۵ مستقیماً در هر مرحله صادر شوند. برخلاف برنامه‌های ترکیبی که در برنامه‌نویسی آنها مجاز به استفاده از دستورات ست و ری‌ست (R و S) نبودیم در برنامه‌های ترتیبی اجرای مراحل توسط دستورات S و R به پیش می‌رود. این دستورات در هر مرحله و بنا به شرایط خاص هر مرحله ممکن است توسط یک

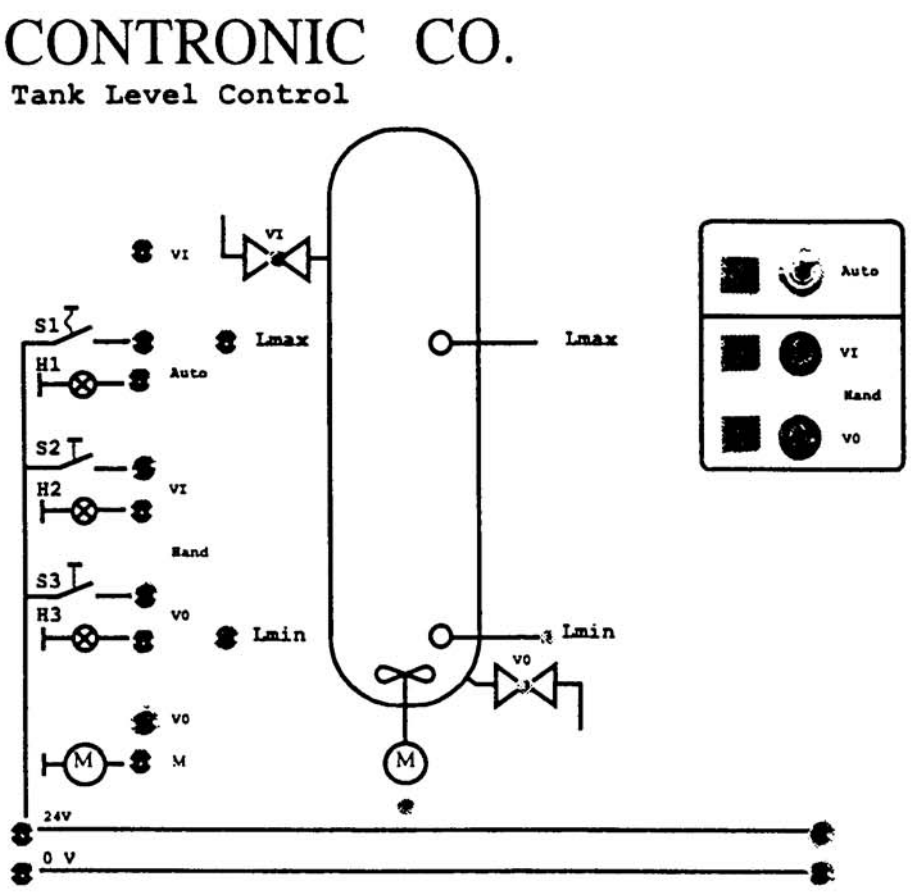
فلیپ‌فلاپ، تایمر یا شمارنده صادر شده باشد. همان‌گونه که دیده می‌شود برای ست‌شدن مرحلهٔ A باید دو سیگنال ENABLE (که در این مرحله همان سیگنال START است) و سیگنال گذار (TRANSITION) یا اتمام مرحلهٔ آخر یعنی C ، با یکدیگر AND شوند. پس از ست‌شدن مرحلهٔ A و ارسال فرمان Q_A در خروجی فلیپ‌فلاپ، این خروجی، سیگنال ENABLE مرحلهٔ بعدی یعنی B را فعال می‌سازد. برای ست‌شدن مرحلهٔ B نیز دو سیگنال ENABLE B و سیگنال اتمام مرحلهٔ A با یکدیگر AND شده، پس از ست‌شدن مرحلهٔ B و ارسال فرمان Q_B در خروجی، این فرمان، مرحلهٔ قبل یعنی مرحلهٔ A را ری‌ست می‌نماید و سیگنال ENABLE مرحلهٔ بعدی یعنی C را فعال می‌کند. برای ست‌شدن مرحلهٔ C نیز حاصل ترکیب عطفی دو سیگنال ENABLE C و سیگنال اتمام مرحلهٔ B لازم است. پس از ست‌شدن این مرحله، Q_C ، اجرای مرحلهٔ قبل یعنی B را ری‌ست نموده، سیگنال ENABLE مرحلهٔ بعد یعنی مرحلهٔ A را صادر می‌کند.

در بخش COMMAND OUTPUT فرمانهای هر مرحله به صورت منطقی با سیگنال‌های حاصل از بخش MODE SECTION و همچنین با سیگنال‌های START و STOP در ارتباط هستند. خروجی این قسمت فرمانهای ارسال شده به سوی عناصر کنترلی می‌باشد.

حال که با برنامه‌های ترتیبی و اجزای تشکیل‌دهندهٔ آنها آشنا شدیم به ذکر یک مثال می‌پردازیم. اگر به خاطر داشته باشید در فصل چهارم در مورد نحوهٔ روبرو شدن با یک پروژهٔ صنعتی به بحث پرداختیم و در نظر گرفتن شرایط ایمنی را به عنوان یکی از مهمترین مواردی که در برنامه‌نویسی باید مد نظر قرار داشت عنوان نموده، برای فهم بیشتر مطالب، مثال کنترل سطح مایع داخل مخزن را بررسی نمودیم. اکنون قصد داریم تا در ادامه، برنامهٔ کنترلی این مخزن را بنویسیم.

مثال ۵-۵: در بسیاری از پروسه‌های صنعتی عملکرد اتوماتیک و دستی سیستم تحت کنترل همراه با نیازمندیهای اساسی پروسه باید در برنامهٔ کنترل در نظر گرفته شود. شبیه‌ساز کنترل سطح مایع مخزن را که به منظور درک روند برنامه‌نویسی این‌گونه پروسه‌ها طراحی شده است در نظر بگیرید.

همان‌گونه که ملاحظه می‌شود علاوه بر شیرهای ورودی و خروجی (VI و VO) و سنسورهای نشان‌دهندهٔ سطح مایع (L min و L max) و موتور الکتریکی همزن (M) موارد دیگری نیز در نظر گرفته شده است. در این پروسه دو حالت کنترلی Auto و Hand وجود دارد که برای هر یک کلید جداگانه‌ای تعبیه شده است.



شکل ۵-۶ : شبیه‌ساز کنترل سطح مایع داخل مخزن

در صورتی که کلید Auto فعال باشد، سیستم به صورت خودکار مراحل زیر را انجام می‌دهد ولی در صورتی که کلید Hand فعال باشد کنترل به صورت دستی (Manual) انجام می‌شود. در حالت دستی دو کلید VI و VO وجود دارد که با فشار دادن هر یک، مرحلهٔ مربوط به همان کلید انجام می‌شود. مثلاً با فشار دادن کلید VO مراحل مربوط به باز شدن شیر خروجی دنبال می‌شود.

دراین پروسه در صورت خالی بودن مخزن و یا به عبارتی روشن شدن LED مربوط به L min، مواد شیمیایی از طریق شیر ورودی VI وارد مخزن شده، شیر ورودی تا زمان پر شدن مخزن یا فعال شدن LED مربوط به L max باز باقی می‌ماند. به محض پر شدن مخزن و روشن شدن LED

مربوط به L max شیر ورودی VI بسته و جهت تسریع واکنش، موتور الکتریکی همزن روشن می‌شود. مدت زمان روشن بودن موتور بستگی به نوع واکنش دارد. برای پروسه مذکور، این مدت را کوتاه و برابر ۵ ثانیه در نظر می‌گیریم . پس از خاموش شدن موتور همزن، شیر خروجی VO باز و مواد تخلیه می‌شوند. پس از تخلیه شدن، LED مربوط به L min روشن و سیکل مذکور مجدداً انجام می‌شود.

مواردی که ذکر شد عملکرد سیستم در حالت Auto است. قصد داریم در این حالت LED مربوط به حالت Auto نیز روشن باشد و هر زمان که کلید Auto غیرفعال شود LED مربوطه خاموش شده، کنترل به صورت دستی انجام گیرد.

در حالت کنترل دستی، برنامهٔ کنترلی به گونه‌ای نوشته شود که:

اگر کلید VI را فشار دهیم LED مربوطه روشن و شیر ورودی VI نیز باز شود و در صورتی که کلید VO را فشار دهیم باز هم LED مربوطه روشن و شیر خروجی VO نیز باز شود و در هر دو حالت فوق در صورت پر یا خالی شدن مخزن اجرای مرحلهٔ مربوطه متوقف گردد. در این حالت باید برنامه به گونه‌ای نوشته شود که وقتی هر یک از دو کلید VI و VO را رها نمودیم اجرای پروسه متوقف شود.

در مورد این پروسه روند برنامه‌نویسی را دنبال می‌کنیم.

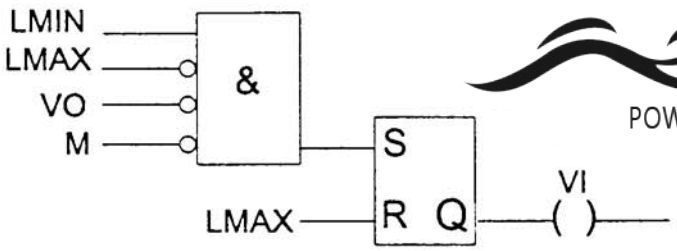
۱- **تعریف پروژه:** کنترل این پروسه به دو صورت AUTO و HAND به صورتی که تشریح شد انجام می‌گیرد.

۲- **رسم فلوچارت برنامه:** در رسم فلوچارت برنامه سعی شده تا در مورد ورودی‌ها و خروجی‌ها از اسامی متناظر آنها استفاده شود.

همان‌گونه که گفته شد کاربرد دستورات S و R در برنامه‌های ترتیبی بسیار زیاد است به طوری که تقریباً در فلوچارت هر مرحله، این دستورات نیز مشاهده می‌شوند. این دستورات ممکن است مربوط به فلیپ‌فلاپ، تایمر یا شمارنده باشد. در ادامه، فلوچارت هر مرحله در حالت AUTO به طور جداگانه آمده است. در رسم فلوچارت‌ها شرایط ایمنی نیز در نظر گرفته شده‌اند.

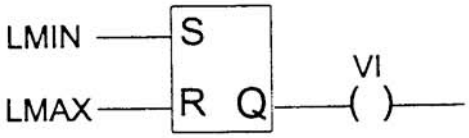
از خوانندگان تقاضا می‌شود به توضیحات ارائه شده در مورد هر فلوچارت دقت کافی مبذول دارند زیرا این روند برنامه‌نویسی کاملاً کلاسه شده بوده، نوشتن برنامهٔ پروژه‌های عظیم نیز با

به‌کارگیری این روش، بسیار ساده خواهد بود.



در فلوچارت مرحلهٔ اول که در انتها، فرمان باز شدن شیر ورودی VI را صادر می‌کند از یک فلیپ‌فلاپ SR استفاده شده است که ورودی S آن حاصل ترکیب عطفی سیگنال‌های L min ، $\overline{L\ max}$ ، $\overline{VO}$ و $\bar{M}$ بوده، ورودی R آن سیگنال L max باشد. حال به بررسی این فلوچارت می‌پردازیم.

در متن مسأله عنوان شده بود که هرگاه سطح مایع درون مخزن به L min برسد فرمان باز شدن و هنگامی که سطح مایع به L max برسد فرمان بسته شدن شیر ورودی صادر می‌شود. در صورتی که فرمان باز و بسته شدن شیر ورودی را با ست و ری‌ست شدن یک فلیپ‌فلاپ نمایش دهیم باید فلوچارت زیر را داشته باشیم:



اما در فلوچارت رسم شده برای این مرحله دیدیم که ورودی ست این فلیپ‌فلاپ حاصل ترکیب عطفی چندین سیگنال می‌باشد. برای روشن شدن مطلب و دلیل استفاده از چنین ترکیبی در ورودی S ، فرض کنید که به عنوان مثال در حالتی هستیم که سطح مایع داخل مخزن به L min رسیده و فرمان باز شدن شیر ورودی VI صادر شده باشد. حال اگر در این زمان شیر خروجی VO نیز باز باشد واضح است که تمام مواد داخل شده از شیر ورودی VI از طریق شیر خروجی VO بدون مخلوط

شدن به وسیلهٔ همزن الکتریکی خارج شده، عملاً در این پروسه از همزن استفاده نمی‌شود. در ضمن از آنجایی که شیر خروجی باز است هیچ‌گاه مخزن پر نمی‌شود و سیگنال ناشی از L max همیشه در حالت "۰" باقی مانده، خروجی فلیپ‌فلاپ را ری‌ست نمی‌کند. به عبارت دیگر شیر ورودی برای همیشه باز باقی خواهد ماند. بنابراین در ورودی S فلیپ‌فلاپ علاوه بر L min باید از نقیض VO نیز استفاده نمود و این بدان معنی است که برای باز شدن شیر ورودی لازم است که سطح مایع داخل مخزن به L min رسیده، علاوه بر آن شیر خروجی VO نیز بسته باشد. دلیل استفاده از سیگنال‌های دیگر نیز در ادامه آمده است:

$\overline{\text{L max}}$: واضح است که چنانچه سیگنال L min ظاهر شود و یا به عبارت دیگر "۱" = L min باشد باید برای سیگنال L max مقدار "۰" را داشته باشیم و در صورتی که L min و L max هر دو در یک زمان مقدار "۱" داشته باشند احتمال معیوب بودن یکی از سنسورها وجود دارد. بنابراین در حالتی که "۱" = L min است، "۰" = L max خواهد بود و در نتیجه "۱" = $\overline{\text{L max}}$ می‌باشد. دلیل استفاده از $\overline{\text{L max}}$ در نظر گرفتن شرایط ایمنی در این پروژه است و در صورت پر بودن مخزن نیز شیر ورودی بسته خواهد شد.

$\bar{M}$: از آنجایی که در مدت زمان پر شدن مخزن بایستی موتور الکتریکی همزن خاموش باشد از سیگنال $\bar{M}$ در ورودی S همراه با سیگنال‌های دیگر استفاده شده است. در حالتی که موتور خاموش است "۰" = M بوده، نقیض M دارای ارزش "۱" می‌باشد.

بنابراین خروجی فلیپ‌فلاپ نشان داده شده در این فلوچارت هنگامی ست می‌شود که:

اولاً "۱" = L min بوده، یا به عبارت دیگر سطح مایع داخل مخزن به کمترین مقدار خود برسد.

ثانیاً "۰" = L max (یا "۱" = $\overline{\text{L max}}$) باشد زیرا در حالتی که سطح مخزن به پائین‌ترین مقدار خود رسیده باشد داشتن مقدار "۱" در L max کاملاً غیرمنطقی است.

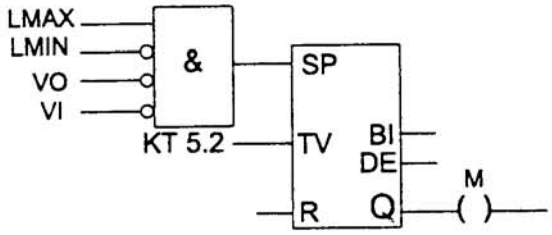
ثالثاً "۰" = VO (یا "۱" = $\overline{\text{VO}}$) باشد. یعنی برای ست شدن خروجی فلیپ‌فلاپ یا به عبارت دیگر برای باز شدن شیر ورودی VI باید شیر خروجی VO بسته باشد.

رابعاً "۰" = M (یا "۱" = $\bar{M}$) باشد. یعنی برای باز شدن شیر ورودی می‌بایست موتور الکتریکی همزن خاموش باشد. پس در ورودی S فلیپ‌فلاپ از ترکیب عطفی سیگنال L min و نقیض سیگنال‌های L max، VO و M استفاده می‌کنیم. حال ورودی R این فلیپ‌فلاپ را بررسی می‌کنیم.

در متن مسأله ذکر شد که اگر سطح مایع داخل مخزن به L max برسد شیر ورودی بسته خواهد شد. پس ورودی R تنها سیگنال L max می‌باشد.

سؤال: دلیل استفاده از فلیپ فلاپ SR در این مرحله چیست یا به عبارت دیگر در صورتی که از فلیپ‌فلاپ RS استفاده شود چه اشکال یا تغییری در برنامه بوجود می‌آید؟

همان‌گونه که در فصول گذشته عنوان شد در فلیپ‌فلاپ‌ها همواره از نظر اجرایی ارجحیت با ورودی دوم است، یعنی در فلیپ‌فلاپ SR به دلیل نزدیکتر بودن ورودی R به انتهای برنامه، این ورودی ارجح خواهد بود. بنابراین سیگنال L max را در ورودی R یک فلیپ‌فلاپ SR قرار می‌دهیم تا نسبت به ورودی S ارجح باشد، زیرا روند اجرای برنامه به گونه‌ای است که ابتدا شیر ورودی باز و فرمان ست صادر می‌شود و پس از رسیدن سطح مایع به L max شیر ورودی بسته و یا خروجی فلیپ‌فلاپ ری‌ست می‌شود. اکنون فلوچارت مرحلهٔ دوم یعنی روشن شدن موتور الکتریکی همزن را بررسی می‌کنیم.



همان‌گونه که ملاحظه می‌کنید جهت ست شدن تایمر استفاده شده در این مرحله از حاصل ترکیب عطفی سیگنال‌های L max ، $\overline{\text{L min}}$ ، $\overline{\text{VO}}$ و $\overline{\text{VI}}$ استفاده شده است. این بدان معنی است که برای روشن ماندن موتور الکتریکی همزن برای مدت زمان خواسته شده بایستی تمامی شرایط زیر برقرار باشند:

۱- L max = "۱"، یا به عبارت دیگر سطح مایع داخل مخزن در بالاترین حد ممکن خود باشد.

۲- L min = "۰" (یا $\overline{\text{L min}}$ = "۱")، واضح است که چنانچه L max = "۱" باشد L min نمی‌تواند مقدار "۱" داشته باشد. (در نظر گرفتن شرایط ایمنی)

۳- VO = "۰" (یا $\overline{\text{VO}}$ = "۱")، یعنی شیر خروجی بسته باشد.

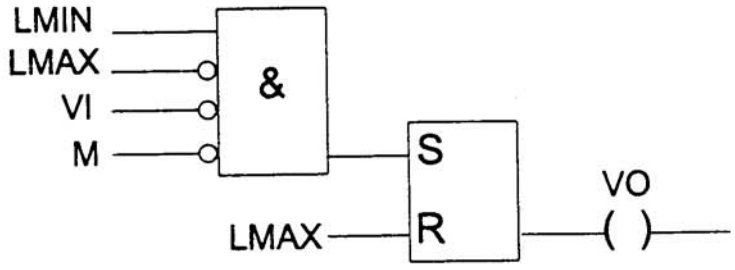
۴- "۰" = VI (یا "۱" = $\overline{VI}$)، یعنی شیر ورودی بسته باشد.

در این فلوچارت برای روشن ماندن موتور در مدت زمان خواسته شده در متن مسأله (۵ ثانیه) از یک تایمر استفاده شده است حال ممکن است در ذهن خواننده این سؤال مطرح شود که:

سؤال: برای انجام این مرحله از پروژه مجاز به استفاده از چه نوع تایمری هستیم؟

در این پروسه به دلیل وابسته بودن خروجی به ورودی باید از تایمر SP استفاده نمائیم. زیرا جهت ست شدن تایمر باید تمامی شرایط مذکور برقرار باشد و در صورت قطع شدن یکی از شرایط باید خروجی تایمر غیرفعال شود، و از آنجایی که در تایمر SP فعال بودن خروجی وابسته به فعال بودن ورودی S می‌باشد از این نوع تایمر استفاده نموده‌ایم. در این تایمر برای پارامتر TV از KT 5.2 استفاده شده است. برای دستیابی به دقت بیشتر و تولرانس یا خطای کمتر می‌توان از KT 50.1 نیز استفاده نمود.

همان‌گونه که در فلوچارت دیده می‌شود از ورودی R و خروجی‌های BI و DE تایمر، استفاده نشده است و تنها خروجی تایمر، جهت فرمان دادن به موتور الکتریکی همزن به کار برده شده است. دلیل عدم استفاده از خروجی‌های BI و DE را می‌توان عدم نیاز به آنها و دلیل عدم استفاده از ورودی R را می‌توان استفاده از تایمر SP دانست. زیرا در تایمر SP وضعیت خروجی کاملاً وابسته به ورودی S است و در این پروسه ورودی S حاصل ترکیب عطفی شرایط و سیگنال‌هایی است که عدم برقراری حداقل یکی از آنها موجب ری‌ست شدن تایمر مذکور شده، خروجی "۰" خواهد شد. بنابراین عدم برقراری حداقل یکی از شرایط باعث ری‌ست شدن خروجی خواهد شد. در اینجا نیازی به تعریف ورودی R جهت ری‌ست کردن خروجی Q نداریم. حال فلوچارت مرحلهٔ سوم یعنی باز شدن شیر خروجی VO را رسم می‌کنیم.



در این مرحله جهت فرمان باز شدن شیر خروجی VO از یک فلیپ‌فلاپ SR استفاده شده است که ورودی S آن حاصل ترکیب عطفی L max و $\overline{L\ min}$ ، $\overline{VI}$ و $\bar{M}$ بوده، ورودی R آن L min می‌باشد. این بدان معنی است که برای باز شدن شیر خروجی باید تمامی شرایط زیر برقرار باشند:

۱- سیگنال "۱" = L max باشد یعنی سطح مایع داخل مخزن در بالاترین مقدار ممکن خود قرار گیرد.

۲- سیگنال "۰" = L min باشد (یا "۱" = $\overline{L\ min}$)، این شرط به دلیل در نظر گرفتن شرایط ایمنی در مورد معیوب بودن سنسورهاست.

۳- سیگنال "۰" = VI باشد (یا "۱" = $\overline{VI}$)، به عبارت دیگر شیر ورودی بسته باشد.

۴- سیگنال "۰" = M باشد (یا "۱" = $\bar{M}$)، به عبارتی موتور الکتریکی همزن خاموش باشد.

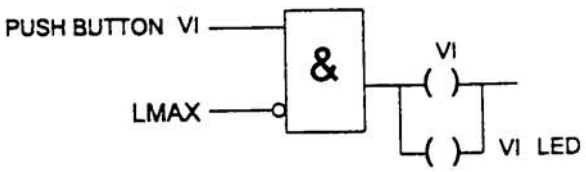
واضح است که برای ری‌ست یا بسته شدن شیر خروجی کافی است سیگنال "۱" = L min بوده، و یا به عبارت دیگر سطح مایع به پائین‌ترین مقدار خود رسیده باشد.

سه مرحلهٔ مذکور مربوط به حالتی است که سیستم به صورت اتوماتیک (AUTO) کنترل می‌شود. حال مراحل مربوط به حالت HAND را بررسی می‌کنیم. همان‌گونه که گفته شد در این حالت می‌خواهیم پروسه به گونه‌ای عمل نماید که:

- چنانچه پوش باتن (Push Button) مربوط به VI دستی را فشار دهیم لامپ یا LED مربوطه روشن و شیر ورودی نیز باز شود.

- چنانچه پوش باتن مربوط به VO دستی را فشار دهیم LED مربوطه روشن و شیر خروجی نیز باز شود.

توجه داشته باشید که در نظر گرفتن شرایط ایمنی در هر دو حالت AUTO و HAND الزامی است. در حالت HAND و برای مرحلهٔ VI دستی، فلوچارت زیر را در نظر بگیرید.



در این مرحله از ترکیب عطفی دو سیگنال پوش باتن VI دستی و $\overline{L\ max}$ استفاده شده است. در اینجا برخلاف حالت AUTO از فلیپ‌فلاپ و یا در حالت کلی از ست و ری‌ست استفاده نشده است. دلیل عدم استفاده از ست و ری‌ست پس از توضیح روند پروسه کاملاً روشن خواهد شد. روند اجرای این مرجله بدین صورت است که:

در صورت فشار دادن و نگه داشتن پوش باتن VI دستی و پائین‌تر بودن سطح مخزن از بالاترین مقدار ممکن خود، شیر ورودی باز و LED مربوطه نیز روشن خواهد شد. در صورتی که دست را از روی پوش باتن برداریم و یا در حالتی که مخزن پر شده و یا به عبارت دیگر "۱" = L max شود ("۰" = $\overline{L\ max}$) حاصل ترکیب عطفی این دو سیگنال "۰" و شیر ورودی بسته می‌شود. سپس LED مربوط به VI دستی نیز خاموش خواهد شد. در اینجا علت عدم استفاده از S و R مشخص می‌گردد. اگر به خاطر داشته باشید در فصل سوم تفاوت بین دو دستور ست (S) و هم‌ارزی (=) را بیان کردیم. دو برنامهٔ زیر را در نظر بگیرید.

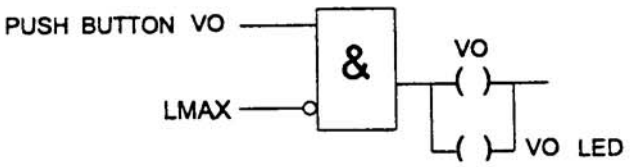
(۱)
```
:A  I  0.0
:S  Q  14.0
```

(۲)
```
:A  I  0.0
:=  Q  14.0
```

در برنامهٔ (۱) فعال شدن (ست شدن) خروجی Q 14.0 تنها مستلزم وجود یک لبهٔ بالارونده در ورودی I 0.0 است و در صورتی که I 0.0 در مدت کوتاهی فعال و سپس غیرفعال شود خروجی Q 14.0 در حالت فعال باقی می‌ماند و برای ری‌ست شدن احتیاج به یک ورودی R دارد.

در برنامهٔ (۲)، خروجی کاملاً وابسته به ورودی است و در صورتی که ورودی I 0.0 فعال باشد خروجی Q 14.0 نیزفعال است ودرصورتی که "۰" = I 0.0 باشد خروجی نیز غیرفعال خواهد بود.

در مرحلهٔ بعدی حالت HAND، فلوچارت پروسهٔ خالی شدن مخزن را بررسی می‌کنیم. مطابق فلوچارت زیر، شیر خروجی VO هنگامی باز خواهد شد که پوش باتن VO در حالت فعال نگه داشته شده، سطح مخزن نیز به L min نرسیده باشد.



۳- تهیهٔ لیستی از ابزار مورد نیاز: ورودی‌ها، خروجی‌ها و به طور کلی عملوندهای مورد استفاده برای اجرای این پروسه به شرح زیر می‌باشد:

ورودی‌ها:
- L_{min} : I 21.0
- L_{max} : I 21.1
- کلید AUTO : I 21.2
- کلید دستی VI : I 21.3
- کلیددستی VO : I 21.4

خروجی‌ها:
- VO : Q 14.0
- M : Q 14.1
- VI : Q 14.2
- AUTO LED : Q 14.3
- VI LED دستی : Q 14.4
- VO LED دستی : Q 14.5

۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: در این پروسه، برنامهٔ اجرای حالت AUTO را در یک بلوک PB و اجرای حالت HAND را در PB دیگر قرار می‌دهیم. اگر به خاطر داشته باشید در فصل سوم در یکی از مثالهای اولیه مسأله‌ای به صورت زیر مطرح شد که:

می‌خواهیم برنامه‌ای بنویسیم که با فعال شدن یک کلید، بلوک خاصی به اجرا در آید و در صورت غیرفعال بودن همان کلید، بلوک دیگری اجرا گردد. در انتهای برنامه خاطرنشان کردیم که این کلید می‌تواند کلید مشخص کنندهٔ حالت AUTO و HAND (MANUAL) باشد.

در این پروژه نیز کلید AUTO یا ورودی I 21.2 را به عنوان کلید مشخص کنندهٔ حالت اجرای برنامه در یکی از دو وضعیت AUTO یا HAND تعریف می‌کنیم. اگر برنامهٔ حالت AUTO را در PB 20 و حالت HAND را در PB 21 بنویسیم در OB 1 که ساختار کلی اجرای برنامه را نشان می‌دهد خواهیم داشت:

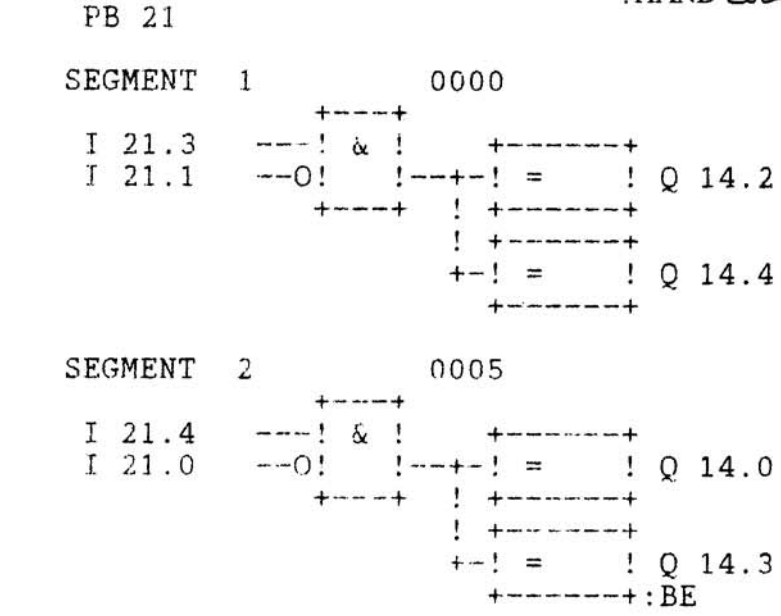
```
OB 1

SEGMENT  1            0000
0000       :A    I    21.2
0001       :=    Q    14.3
0002       :JC   PB   20
0003       :AN   I    21.2
0004       :JC   PB   21
0005       :BE
```

بلوک‌های PB 20 و PB 21 نیز به صورت زیر نوشته می‌شوند.

بلوک برنامه در حالت AUTO:

```
 PB 20

SEGMENT  1          0000
            +---+
 I 21.0   ---! & !
 I 21.1   --O!   !   Q 14.2
 Q 14.0   --O!   !    +------+
 Q 14.1   --O!   !----!S     !
            +---+    !      !   +------+
           I 21.1   --!R    Q!-+-! =    ! Q 14.2
                      +------+   +------+

SEGMENT  2          000A
            +---+
 I 21.1   ---! & !
 I 21.0   --O!   !   T 2
 Q 14.0   --O!   !    +------+
 Q 14.2   --O!   !----!1_-_  !
            +---+    !       !
          KT 005.2 --!TV  BI!-
                     !    DE!-
                     !       !   +------+
                   --!R    Q!-+-! =    ! Q 14.1
                     +------+   +------+

SEGMENT  3          0017
            +---+
 I 21.1   ---! & !
 I 21.0   --O!   !   Q 14.0
 Q 14.2   --O!   !    +------+
 Q 14.1   --O!   !----!S     !
            +---+    !      !   +------+
           I 21.0   --!R    Q!-+-! =    ! Q 14.0
                      +------+   +------+:BE
```

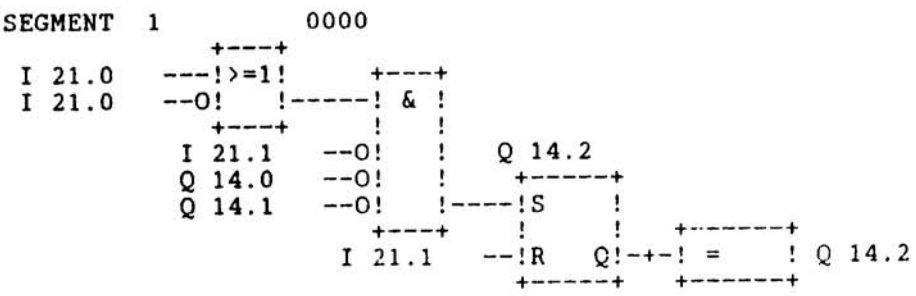
بلوک برنامه در حالت HAND:



۵- **بررسی شرایط ایمنی**: در بخش رسم فلوچارت، شرایط ایمنی نیز در نظر گرفته شده است. در این قسمت قصد داریم که با ایجاد تغییراتی در برنامه، پروسه را به نحوی کنترل نمائیم که به عنوان مثال:

۱- در حالت AUTO و در صورت قطع برق و وصل مجدد آن اجرای پروسه از حالت پر شدن مخزن شروع شود.

برای انجام این عمل کافی است با ایجاد تغییرات اندکی در PB 20 (بلوک برنامهٔ حالت AUTO) باعث شویم که همواره "۱" = L min باشد. برای این عمل می‌توان ترکیب فصلی I 21.0 و $\overline{\text{I 21.0}}$ (Lmin و $\overline{\text{L min}}$) را در ورودی S فلیپ‌فلاپ قرار داد.



۲- فرض کنید که پروسه در حالت AUTO و در مرحلهٔ روشن بودن موتور الکتریکی همزن است. قصد داریم با ایجاد تغییراتی در برنامه باعث شویم که در صورت خاموش شدن موتور الکتریکی همزن، انجام پروسه به مدت ۵ ثانیه متوقف شود تا مواد شیمیایی داخل مخزن از تلاطم باز ایستد و پس از گذشت این زمان، شیر خروجی VO باز شود.

خوانندگان توجه داشته باشند که معمولاً در نظر گرفتن این مدت زمان تأخیر بین دو مرحله از یک پروسه الزامی است، زیرا در صورتی که بلافاصله پس از خاموش شدن همزن الکتریکی، شیر خروجی باز شود مواد شیمیایی متلاطم به هنگام خروج از لولهٔ خروجی به اطراف پاشیده می‌شود. بنابراین در طراحی سیستم و برنامه‌نویسی، در نظر گرفتن چنین مواردی موجب اجرای مطلوب‌تر پروسه خواهد شد.

بـه ایـن مـدت زمـان تأخیر کـه در یک پـروسه و بـین دو مـرحـله در نـظر گـرفته مـی‌شود Waiting Time می‌گویند. این زمان تأخیر یکـی از پـرکاربردترین مـوارد در پـروسه‌های کـنترلی محسوب می‌شود.

بنابراین برای داشتن چـنین حـالتی کـافی است در حـالت AUTO (در بـلوک PB 20) و در SEGMENT 3 از یک تایمر SD استفاده کنیم. دلیل استفاده از تایمر SD، نیاز به تأخیر (Delay) در این پروسه است.

```
SEGMENT 3          001A
          +---+
I 21.1  ---! & !
I 21.0  --O!   !    T 3
Q 14.2  --O!   !     +------+
Q 14.1  --O!   !-----!T!-!0!
          +---+      !      !
        KT 005.2 --!TV BI!-
                   !   DE!-
                   !      !  Q 14.0
                   !      !    +------+
                 --!R    Q!---!S     !
                   +------+    !      !    +-------+
                  I 21.0    --!R    Q!-+-! =     ! Q 14.0
                               +------+    +-------+:BE
```

در ادامه، برنامهٔ کلی با ایجاد دو تغییر یاد شده به روش STL آمده است.

بلوک برنامه در حالت AUTO:

```
 PB 20

SEGMENT  1              0000
0000      :A(
0001      :O    I     21.0
0002      :ON   I     21.0
0003      :)
0004      :AN   I     21.1
0005      :AN   Q     14.0
0006      :AN   Q     14.1
0007      :S    Q     14.2
0008      :A    I     21.1
0009      :R    Q     14.2
000A      :A    Q     14.2
000B      :=    Q     14.2
000C      :***


SEGMENT  2              000D
000D      :A    I     21.1
000E      :AN   I     21.0
000F      :AN   Q     14.0
0010      :AN   Q     14.2
0011      :L    KT  005.2
0013      :SP   T      2
0014      :NOP  0
0015      :NOP  0
0016      :NOP  0
0017      :A    T      2
0018      :=    Q     14.1
0019      :***
```

```
SEGMENT  3              001A
001A      :A    I     21.1
001B      :AN   I     21.0
001C      :AN   Q     14.2
001D      :AN   Q     14.1
001E      :L    KT  005.2
0020      :SD   T      3
0021      :NOP  0
0022      :NOP  0
0023      :NOP  0
0024      :A    T      3
0025      :S    Q     14.0
0026      :A    I     21.0
0027      :R    Q     14.0
0028      :A    Q     14.0
0029      :=    Q     14.0
002A      :BE
```

بلوک برنامه در حالت HAND:

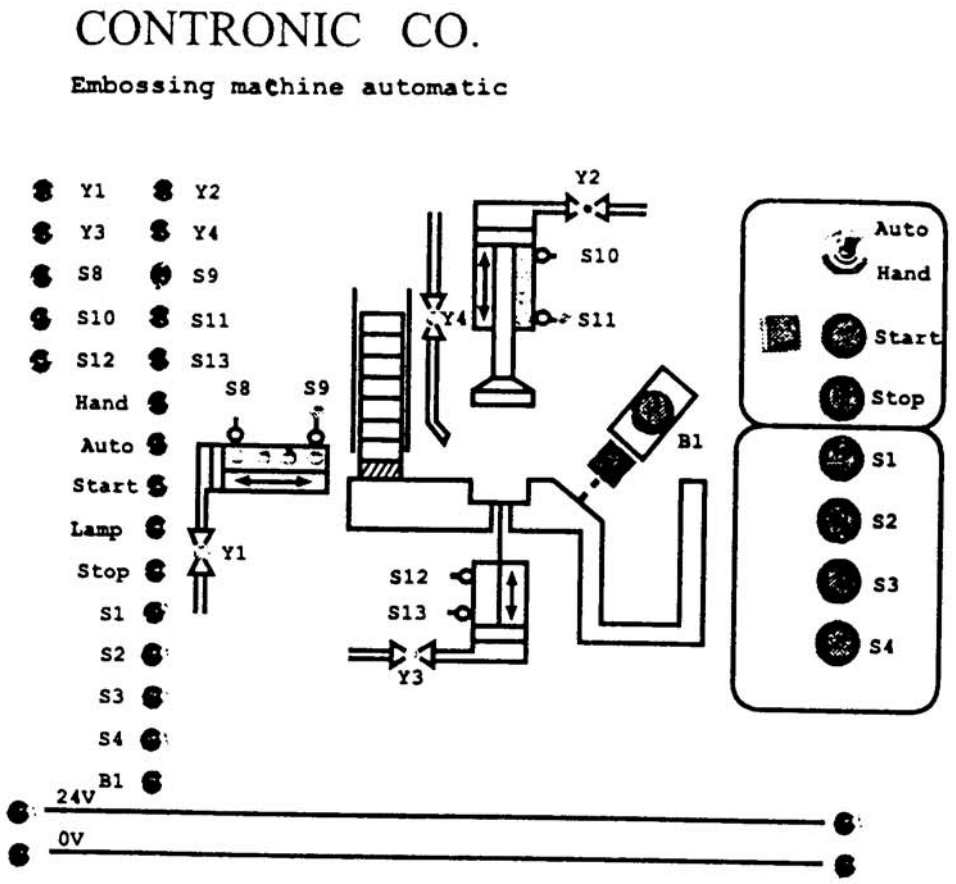
```
 PB 21

SEGMENT  1              0000
0000      :A    I     21.3
0001      :AN   I     21.1
0002      :=    Q     14.2
0003      :=    Q     14.4
0004      :***


SEGMENT  2              0005
0005      :A    I     21.4
0006      :AN   I     21.0
0007      :=    Q     14.0
0008      :=    Q     14.3
0009      :BE
```

اکنون به منظور درک بهتر برنامه‌نویسی فرآیندهای ترتیبی به توضیح عملکرد شبیه‌ساز دیگری می‌پردازیم:

مثال ۵-۶ : برای کنترل فرآیندهای ترتیبی (Sequential) ، هنگامی که چند عمل یکی پس از دیگری و تحت شرایط خاصی صورت می‌گیرند ساختار منطق کنترلی از سازمان مشخصی پیروی

می‌کند. برای کسب تجربه و برای کنترل چنین فرآیندهایی شبیه‌ساز ماشین پرس طراحی گردیده است.

شیرهای هیدرولیک، لیمیت سوئیچ‌ها، لامپ‌ها (LED) ، سوئیچ‌های فرمان متعدد، سلکتورهای انتخاب حالت دستی، اتوماتیک و تک مرحله‌ای برای شبیه‌سازی چنین فرآیندهایی ایده‌آل است. فرآیندهای پیچیده‌تر در حقیقت تکرار متناسب همان مراحلی است که برنامهٔ آن برای این شبیه‌ساز طراحی شده است. در شکل ۵-۷ این شبیه‌ساز را به همراه تمام عناصر کنترلی موجود ملاحظه می‌کنید.



شکل ۵-۷ : شبیه‌ساز ماشین پرس و عناصر کنترلی آن

۵-۵- نقطهٔ شروع یا حالت اولیه (Initial State)

در برنامه‌های ترتیبی لازم است که کنترل فرآیند از نقطه‌ای خاص شروع شود. مسلماً آغاز عملیات کنترل در این نقطه شرایط خاص خود را دارد. به این شرایط کنترلی که در برنامه‌های ترتیبی آغازکنندهٔ اجرای پروسه می‌باشند حالت اولیه یا Initial State گویند.

در این پروسه، حالت اولیه به گونه‌ای است که جهت آغاز اجرای پروسه باید تمام سیلندرها عقب بوده، هیچ قطعه‌ای در خط تولید وجود نداشته باشد. در صورت برقراری دو شرط مذکور و فشرده‌شدن کلید START، شیر ورودی روغن Y1 مربوط به سیلندر هیدرولیک اول باز خواهد شد. روند اجرای پروسه به شرح زیر است:

قطعات ورق خام در نردبان عمودی موجود در شکل بر روی یکدیگر قرار گرفته‌اند. با باز شدن شیر Y1 سیلندر شمارهٔ ۱ به جلو آمده، این حرکت رو به جلو توسط سنسورهایی که در این شبیه‌ساز با LED نمایش داده شده‌اند حس می‌شود. با جلو آمدن بازوی سیلندر شمارهٔ ۱، تنها یک قطعه ورق درون قالب قرار می‌گیرد که قرارگرفتن قطعه درون قالب توسط سنسور S9 حس می‌شود. (در حالتی که سیلندر حرکت کرده و سنسور S9 تحریک گردد قطعهٔ مورد نظر درون قالب قرار می‌گیرد.) در این مرحله شیر ورودی روغن سیلندر شمارهٔ ۲ یعنی Y2 باز و Y1 بسته می‌شود. سیلندر شمارهٔ ۲ به صورت عمودی پائین آمده، به محض تحریک نمودن سنسور S11، به آخرین حد مجاز حرکت خود می‌رسد. در اینجا قطعهٔ موجود در قالب کاملاً به شکل مورد نظر در آمده است. با تحریک شدن سنسور S11 شیر ورودی Y2 بسته و شیر ورودی Y3 (مربوط به بازوی هیدرولیک شمارهٔ ۳) باز خواهد شد. این بازو، قطعهٔ شکل داده شده را از قالب بیرون می‌اندازد. حرکت این بازو نیز به صورت عمودی و از پائین به بالا می‌باشد. به محض تحریک شدن سنسور S12 شیر ورودی Y3 بسته و Y4 باز می‌شود. با باز شدن Y4 ، هوای با فشار زیاد قطعهٔ موجود در خط را به محلی که جهت جمع‌آوری قطعات آماده درنظر گرفته‌شده است پرتاب می‌کند. عبور قطعه وافتادن آن در محل مذکور توسط سنسور B1 حس می‌شود. پس‌ازاین مرحله قطعهٔ بعدی وارد خط شده، سیکل مجدداً تکرار می‌شود.

نکاتی که در نوشتن برنامه جهت کنترل این پروسه بایستی مدنظر قرار گیرند عبارتند از:

۱- در صورت وجود قطعه‌ای در خط تولید، مجاز به واردکردن قطعهٔ دیگری نیستیم. به عبارت دیگر در هر زمان باید تنها یک قطعه در خط قرار داشته باشد.

۲- در صورت قطع برق و وصل مجدد آن، اجرای پروسه از همان مرحله‌ای آغاز شود که سیستم قبل از قطع برق در آن حالت قرار داشته است و یا در حالتی که نیاز به توقف خط و اجرای پروسه بوده، به عبارتی کلید STOP توسط کاربر استفاده شده باشد پس از شروع به کار مجدد، اجرای پروسه از همان مرحلهٔ قبل از توقف (STOP) آغاز گردد.

در این پروسه نیز همانند مثالهای قبلی دو حالت کنترل AUTO و HAND وجود دارد. در صورت انتخاب حالت کنترلی HAND می‌توان توسط کلیدهای S1 الی S4 که در شکل نشان داده شده‌اند اجرای مرحله به مرحله نیز داشته باشیم. در اجرای مرحله به مرحله، فشار هر کلید موجب به اجرا در آمدن مرحلهٔ متناظر با شمارهٔ کلید خواهد شد. همانگونه که عنوان شد قصد داریم در صورت توقف برنامه در هر دو حالت AUTO و HAND اجرای برنامه از مرحلهٔ قبل از توقف برنامه (STOP) آغاز شود. برای نوشتن این برنامه از تعدادی فلگ و فلیپ‌فلاپ استفاده خواهیم کرد. اگر به یاد داشته باشید در فصل دوم بیان شد که می‌توان برخی شرایط و حالات را در فلگ قرار داد و در صورت نیاز می‌توانیم اطلاعات را با فراخوانی فلگ‌ها از حافظه بخوانیم.

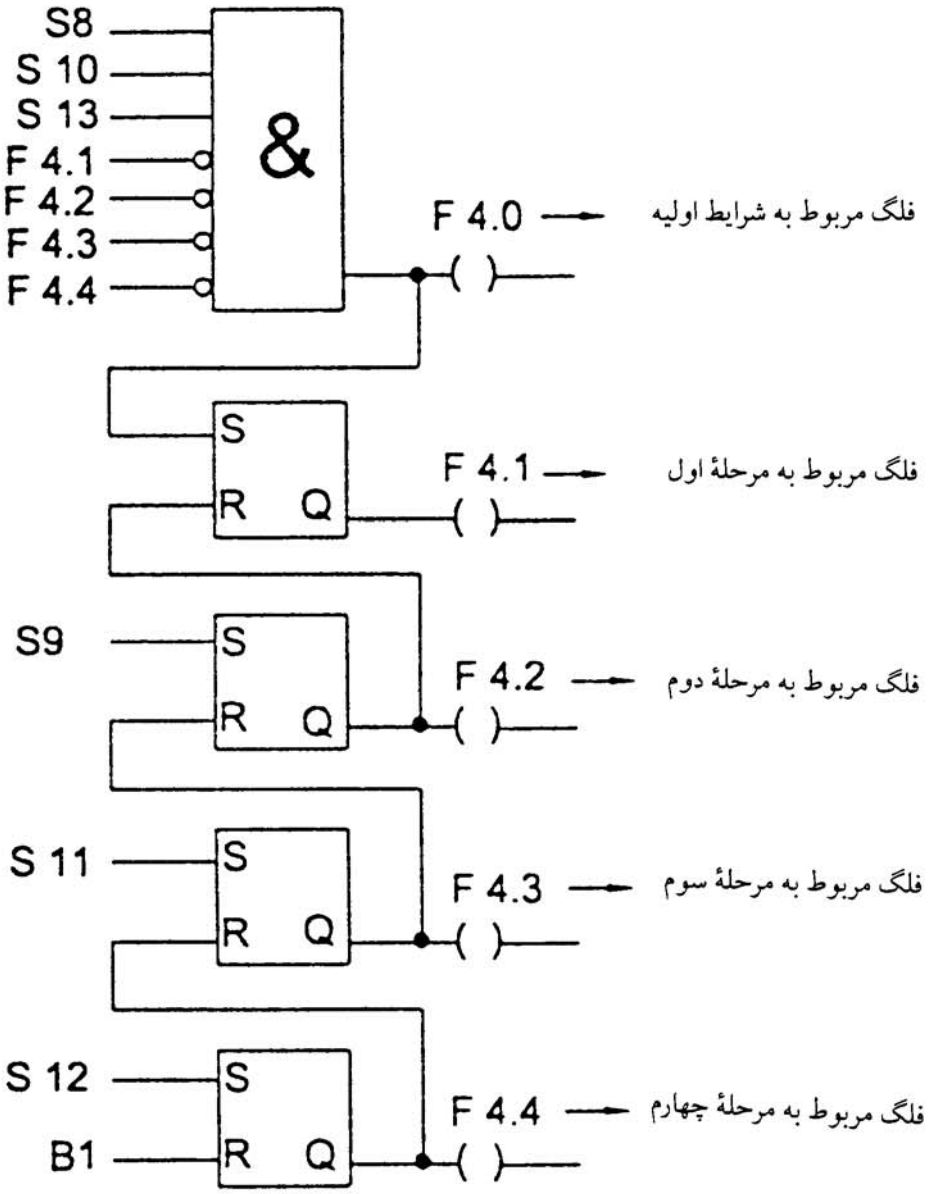
توجه داشته باشید که اطلاعات و شرایط مذکور بایستی در فلگ‌های پایدار (Retentive) قرار گیرند زیرا همانگونه که در فصل دوم گفته شد فلگ‌های پایدار پس از قطع منبع تغذیه اطلاعات خود را حفظ می‌کنند و از آنجایی که در برخی پروسه‌های حساس مانند این پروسه، اجرای برنامه پس از قطع جریان برق باید از همان مرحلهٔ قبل از قطع منبع تغذیه دنبال شود و نیاز به اطلاعات و آگاهی از شرایط موجود در آن مرحله می‌باشد، این شرایط و حالات را باید در فلگ‌های پایدار قرار داد. در PLC مورد بحث ما ۲۵۶ فلگ بایت وجود دارد (FY 0 - FY 255) که نیمه اول فلگ‌ها یعنی (FY 0 - FY 127) پایدار و بقیه ناپایدار می‌باشند. بنابراین از بیت‌های FW 4 (FY 4 , FY 5) جهت ذخیرهٔ شرایط مذکور استفاده شده است.

در مورد این پروسه نیز روند برنامه‌نویسی را به روش کلاسیک دنبال خواهیم نمود.

۱- **تعریف پروژه:** همانطور که ذکر شد این پروسه در دو حالت AUTO و HAND انجام‌پذیر است که توضیحات مربوط به چگونگی اجرای هر حالت ارائه شد.

۲- **رسم فلوچارت:** برای رسم فلوچارت ابتدا مرحلهٔ اولیه یا Initial State را بررسی می‌کنیم. شرایط اولیه به گونه‌ای است که تمام سیلندرها عقب بوده، قطعه‌ای در خط قرار نداشته باشد. عقب

بودن سیلندرها توسط سنسورهای S8 ، S10 و S13 حس می‌شود و عدم وجود قطعه در خط را می‌توان با فراخوانی اطلاعات موجود در فلگ‌های هر مرحله دریافت. در ادامه فلوچارت تمام مراحل رسم شده است.

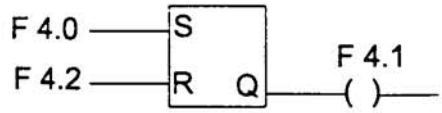


حال به توضیح هر یک از مراحل فوق می‌پردازیم:

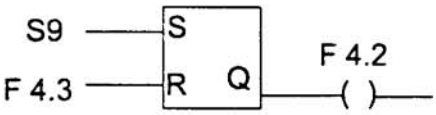
حالت اولیه یا F 4.0 : همان‌گونه که در متن مسأله ملاحظه نمودید حالت Initial State شامل شرایطی است که تمامی سیلندرها عقب بوده، هیچ قطعه‌ای درون خط نباشد. عقب بودن سیلندرها توسط سنسورهای S8 ، S10 و S13 حس می‌شوند. حال عدم وجود قطعه در هر یک از مراحل را بررسی می‌کنیم.

حالتی را در نظر بگیرید که قطعه‌ای در مرحلهٔ سوم وجود داشته باشد پس فلیپ‌فلاپ مربوط به این مرحله ست بوده و داریم: "۱" = F 4.3 . بنابراین عدم وجود قطعه در این مرحله را می‌توان با نقیض F 4.3 نشان داد. زیرا در صورتی که قطعه‌ای در این مرحله نباشد "۰" = F 4.3 بوده، نقیض این فلگ مقدار "۱" را خواهد داشت. بنابراین عدم وجود قطعه در مراحل چهارگانه را می‌توان با حاصل ترکیب عطفی نقیض فلگ‌های مربوط به هر چهار مرحله نشان داد. پس F 4.0 که معرف حالت اولیه یا Initial State می‌باشد هنگامی "۱" خواهد شد که حاصل ترکیب عطفی نشان داده شده در فلوچارت زیر "۱" باشد.

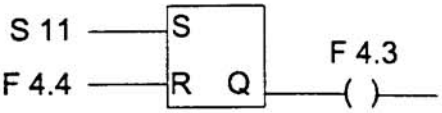
مرحلهٔ اول یا F 4.1 : همان‌گونه که در فلوچارت کلی ملاحظه می‌شود جهت فعال و غیرفعال نمودن این مرحله از یک فلیپ‌فلاپ SR استفاده شده است. در متن مسأله دیدیم که شرط انجام این مرحله وجود شرایط اولیه یا Initial State می‌باشد. بنابراین برای ست نمودن فلگ F 4.1 که معرف انجام مرحلهٔ اول است از F 4.0 استفاده شده است. جهت ری‌ست نمودن این فلگ، فلگ F 4.2 یعنی انجام مرحلهٔ بعدی به کار برده شده است، زیرا همان‌طور که قبلاً گفته شد در برنامه‌های ترتیبی انجام هر مرحله، مرحلهٔ قبلی را ملغی (ری‌ست) می‌کند. در اینجا نیز به محض فعال شدن F 4.2 یعنی آغاز مرحلهٔ دوم، F 4.1 ری‌ست خواهد شد و این بدان معنی است که با آغاز مرحلهٔ دوم، مرحلهٔ اول نیز پایان یافته است. در زیر فلوچارت مربوط به این مرحله جداگانه رسم شده است.



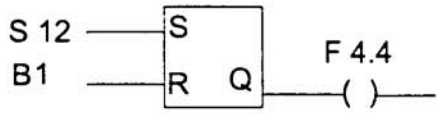
مرحله دوم F 4.2: همان‌طور که در فلوچارت زیر مشاهده می‌شود شرط اجرای این مرحله تحریک سنسور S9 می‌باشد. به محض اینکه بازوی هیدرولیک شمارهٔ ۱ به منتهی‌الیه سمت راست خود یعنی سنسور S9 برسد F 4.2 ست می‌شود. ورودی R فلیپ‌فلاپ به کار برده شده در این مرحله، F 4.3 یعنی اجرای مرحلهٔ سوم خواهد بود.



مرحله سوم F 4.3: شرط ورود به این مرحله تحریک سنسور S11 می‌باشد. در صورتی که بازوی هیدرولیک شمارهٔ ۲ به پایین‌ترین نقطهٔ ممکن خود برسد یا به عبارت دیگر سنسور S11 تحریک گردد فلیپ‌فلاپ مربوط به این مرحله ست می‌شود و خواهیم داشت "۱" = F 4.3. ورودی R این فلیپ‌فلاپ F 4.4 یعنی فلگ مربوط به مرحلهٔ بعدی است، بدین ترتیب که در صورت فعال شدن این فلگ ، فلگ مرحلهٔ قبل یعنی F 4.3 ری‌ست می‌شود.



مرحلهٔ چهارم F 4.4: همان‌طور که در فلوچارت زیر مشاهده می‌کنید شرط ورود به این مرحله تحریک شدن سنسور S12 می‌باشد. این بدان معنی است که بازوی هیدرولیک شمارهٔ ۳ به بالاترین حد مجاز خود رسیده است. ورودی ری‌ست فلیپ‌فلاپ این مرحله سنسور B_1 می‌باشد. عبور قطعه از این مرحله فلگ F 4.4 را ری‌ست می‌کند و پس از این مرحله، سیکل مجدداً تکرار می‌شود.

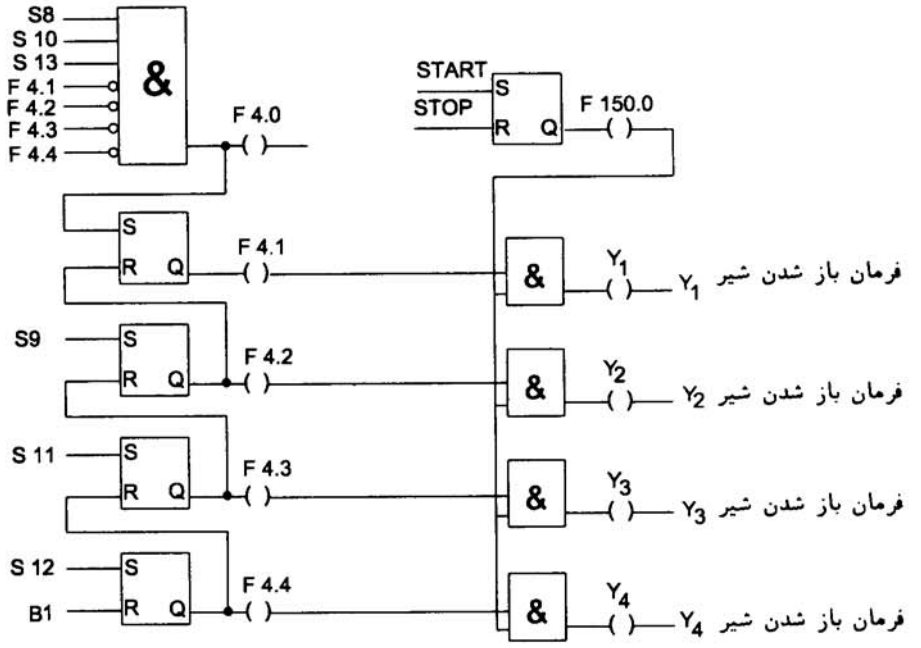


حال چگونگی ارتباط کلیدهای START و STOP را با برنامهٔ نوشته شده (فلوچارت رسم شده) بررسی می‌کنیم:

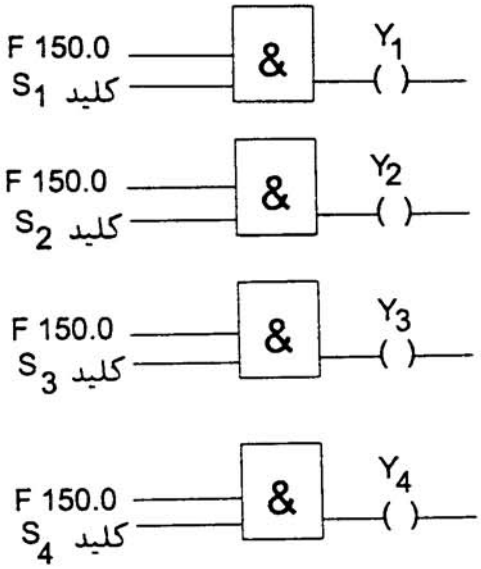
در صورتی که به خاطر داشته باشید در قسمت قبل بیان شد که به دلیل نیاز به اطلاعات در مورد شرایط و حالات سیستم پس از قطع جریان برق از بیت‌های فلگ پایدار استفاده نمودیم ولی در مورد START و STOP باید از یک فلگ ناپایدار استفاده نماییم. دلیل این امر کاملاً روشن است، زیرا به عنوان مثال، در صورت توقف سیستم (فشرده شدن کلید STOP) مجدداً نیاز به راه‌اندازی سیستم (فشرده شدن کلید START) داریم و باید از فلگ‌های ناپایدار که در هنگام قطع جریان برق اطلاعات خود را از دست می‌دهند استفاده کنیم.

حال که محل ذخیرهٔ اطلاعات مربوط به START و STOP مشخص گردید این سؤال مطرح است که این اطلاعات (به عنوان مثال F 150.0) چگونه با سایر فلگ‌ها ارتباط دارد؟

کاملاً واضح است که در مورد این قسمت نیز باید از یک فلیپ‌فلاپ SR استفاده کنیم که ورودی S آن کلید START و ورودی R آن کلید STOP باشد. خروجی این فلیپ‌فلاپ یعنی F 150.0 باید با تمامی فلگ‌های مربوط به مراحل چهارگانه، AND شود، زیرا تنها در صورت روشن بودن و در حال کار بودن سیستم (F 150.0 = "۱" یا START = "۱") مراحل چهارگانه قابل اجرا هستند. در فلوچارت زیر این مطلب به روشنی دیده می‌شود.



تا این قسمت از مثال در مورد حالت AUTO بحث نمودیم. اکنون حالت HAND را بررسی می‌کنیم. همان‌طور که قبلاً ذکر شد در این حالت چهار کلید S1، ...، S4 برای اجرای هر مرحله وجود دارد. در ادامه، فلوچارت این حالت رسم شده است.



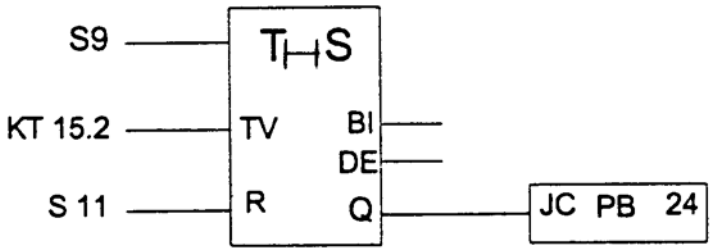
اکنون فرض کنید که قصد داریم در برنامهٔ حالت AUTO تغییری ایجاد کنیم به این صورت که اگر به عنوان مثال در مرحلهٔ دوم یعنی پایین آمدن پرس هیدرولیک شمارهٔ ۲ بنا به دلایلی مثلاً قطع شدن لولهٔ متصل به تانک حاوی روغن مشکلی بوجود آید اجرای این مرحله با تأخیر اندکی انجام شود این تأخیر در این مرحله ۱۵ ثانیه در نظر گرفته می‌شود و با ایجاد تغییری دیگر در برنامه به PLC فرمان می‌دهیم که در صورت عدم فعال شدن خروجی در طول این مدت (عدم رفع اشکال مذکور) اجرای برنامه متوقف شود. به این زمان تأخیر که معمولاً جهت مسائل ایمنی در نظر گرفته می‌شود Monitoring Time می‌گویند.

در برخی از پروسه‌های می‌توان برای هر مرحله و یا حتی برای کل پروسه Monitoring Time و Waiting Time تعریف نمود. مثلاً برای تعریف کل پروسه می‌توان از F 4.0 به عنوان شرط شروع (ست) و از تحریک سنسور B1 به عنوان شرط اتمام انجام پروسه (ری‌ست) استفاده نمود. در صورتی که برای هر مرحله از پروسه هم Waiting Time و هم Monitoring Time در نظر گرفته شده باشد رابطهٔ زیر را خواهیم داشت.

$$\left(\text{Monitoring Time}_{\text{هر مرحله}}\right) = \left(\text{Waiting Time}_{\text{هر مرحله}}\right) + \left(\text{Tolerance}_{\text{هر مرحله}}\right) + \left(\text{زمان اجرای}_{\text{هر مرحله}}\right)$$

یا (زمان اجرای مرحله + ٪ + TW = TM)

برای ایجاد تأخیر مذکور لازم است در برنامهٔ حالت AUTO بخش دیگری اضافه و در آن از تایمر SS جهت ایجاد این تأخیر استفاده کنیم.



و در PB 24 فرمان STP که PLC را به حالت STOP می‌برد استفاده شده است.

```
PB 24

SEGMENT  1              0000
0000       :STP
0001       :BE
```

۳- تهیهٔ لیستی از ابزار مورد نیاز: در این پروسه ۱۵ ورودی ۵ خروجی و ۱ تایمر و تعدادی فلگ خواهیم داشت که در ادامه آورده شده‌اند.

ورودی‌ها:

I 22.0 : کلید STOP
I 22.1 : کلید AUTO
I 22.2 : کلید HAND
I 22.3 : کلید S1
I 22.4 : کلید S2
I 22.5 : کلید S3
I 22.6 : کلید S4

I 21.0 : سنسور S8
I 21.1 : سنسور S10
I 21.2 : سنسور S13
I 21.3 : سنسور S9
I 21.4 : سنسور S11
I 21.5 : سنسور S12
I 21.6 : سنسور B1
I 21.7 : کلید START

خروجی‌ها:

Q 14.0 : فرمان باز شدن شیر Y_1 (حرکت بازوی هیدرولیک ۱)
Q 14.1 : فرمان باز شدن شیر Y_2 (حرکت بازوی هیدرولیک ۲)
Q 14.2 : فرمان باز شدن شیر Y_3 (حرکت بازوی هیدرولیک ۳)
Q 14.3 : فرمان باز شدن شیر Y_4 (حرکت بازوی هیدرولیک ۴)
Q 14.4 : LED مربوط به حالت START

فلگ‌ها:

F 4.0 (پایدار) : فلگ مربوط به حالت Initial State
F 4.1 (پایدار) : فلگ مربوط به مرحلهٔ اول
F 4.2 (پایدار) : فلگ مربوط به مرحلهٔ دوم
F 4.3 (پایدار) : فلگ مربوط به مرحلهٔ سوم
F 4.4 (پایدار) : فلگ مربوط به مرحلهٔ چهارم
F 20.0 (پایدار) : مربوط به حالت AUTO / HAND
F 150.0 (ناپایدار) : مربوط به حالت START / STOP

T_4 : برای در نظر گرفتن Monitoring Time

۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: در ادامه، تمام بلوکها آورده شده‌اند.

```
OB 1

SEGMENT  1              0000
0000       :A    T      4
0001       :JC   PB   24
0002       :A    I    21.7
0003       :S    F   150.0
0004       :A    I    22.0
0005       :R    F   150.0
0006       :A    F   150.0
0007       :=    Q    14.4
0008       :A    I    22.1
0009       :S    F    20.0
000A       :A    I    22.2
000B       :R    F    20.0
000C       :A    F    20.0
000D       :JC   PB   22
000E       :AN   F    20.0
000F       :JC   PB   23
0010       :BE
```

بلوک برنامهٔ حالت HAND:

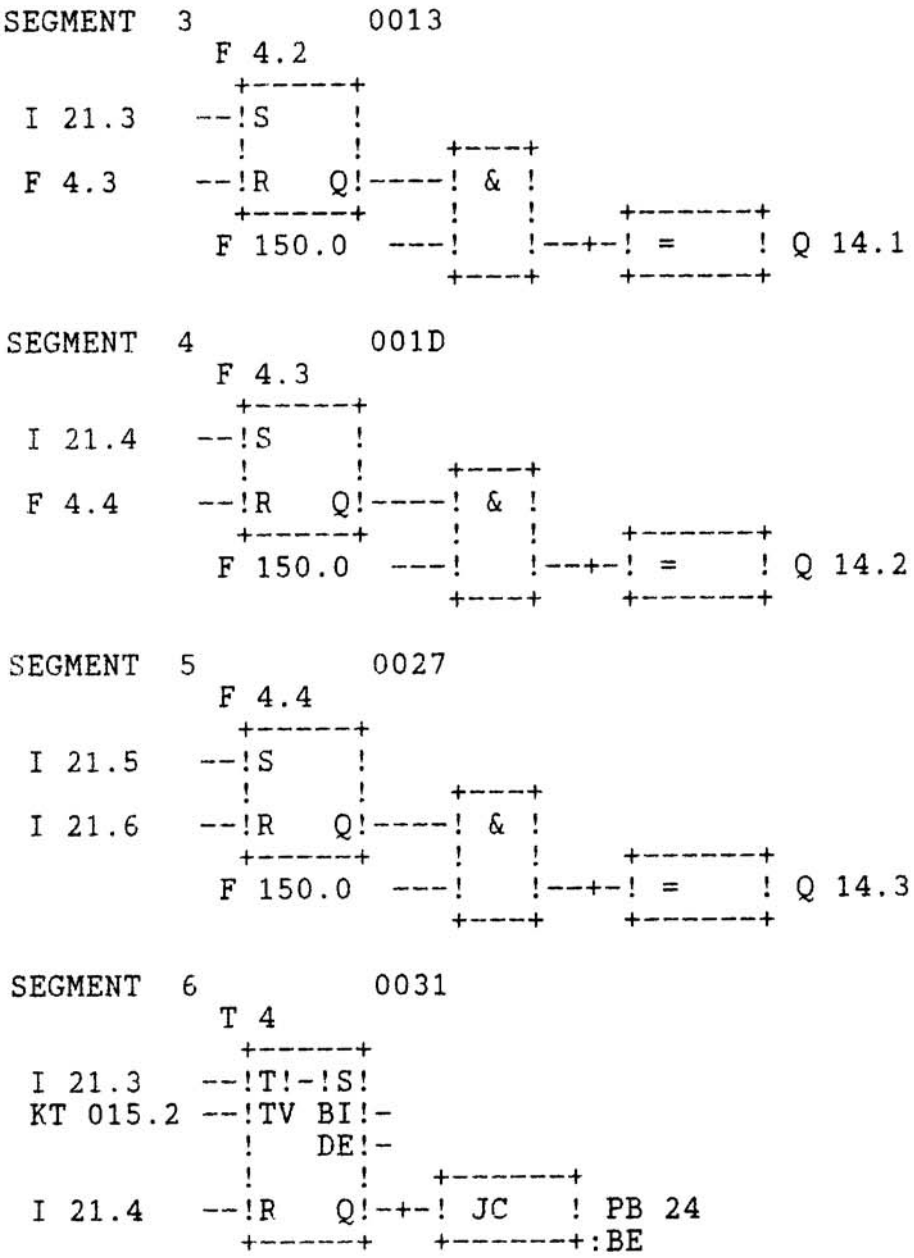
```
 PB 23

SEGMENT  1              0000
                +---+
 F 150.0  ---! & !     +-------+
 I 22.3   ---!   !--+-! =     ! Q 14.0
                +---+  +-------+

SEGMENT  2              0004
                +---+
 F 150.0  ---! & !     +-------+
 I 22.4   ---!   !--+-! =     ! Q 14.1
                +---+  +-------+
```

```
SEGMENT  3          0008
              +---+
 F 150.0  ---! & !     +------+
 I 22.5   ---!   !--+-! =    ! Q 14.2
              +---+     +------+

SEGMENT  4          000C
              +---+
 F 150.0  ---! & !     +------+
 I 22.5   ---!   !--+-! =    ! Q 14.2
              +---+     +------+

SEGMENT  5          0010
              +---+
 F 150.0  ---! & !     +------+
 I 22.6   ---!   !--+-! =    ! Q 14.3
              +---+     +------+:BE
```

بلوک برنامهٔ حالت AUTO:

```
 PB 22

SEGMENT  1          0000
              +---+
 I 21.0   ---! & !
 I 21.1   ---!   !
 I 21.2   ---!   !
 F 4.1    --O!   !
 F 4.2    --O!   !
 F 4.3    --O!   !     +------+
 F 4.4    --O!   !--+-! =    ! F 4.0
              +---+     +------+

SEGMENT  2          0009
            F 4.1
             +-----+
 F 4.0    --!S    !
            !     !    +---+
 F 4.2    --!R   Q!----! & !
             +-----+    !   !     +------+
            F 150.0  ---!   !--+-! =    ! Q 14.0
                        +---+     +------+
```

ادامهٔ بلوک PB 22:



در این قسمت قصد داریم برای درک بهتر و همچنین آشنایی با تکنیک‌های برنامه‌نویسی در PLC ، بلوک برنامهٔ حالت AUTO را تنها با استفاده از یک شمارنده بازنویسی کنیم. البته در این قسمت از فلگ استفاده نمی‌شود و شمارندهٔ مذکور باید همانند فلگ‌های استفاده شده در بلوک PB 22 از نوع پایدار انتخاب شود.

```
PB 25

SEGMENT  1              0000
0000      :A    I     21.0
0001      :A    I     21.1
0002      :A    I     21.2
0003      :A    I     21.6
0004      :R    C      5
0005      :O    I     21.3
0006      :O    I     21.4
0007      :O    I     21.5
0008      :CU   C      5
0009      :L    C      5
000A      :L    KF +0
000C      :!=F
000D      :A    F   150.0
000E      :=    Q    14.0
000F      :L    C      5
0010      :L    KF +1
0012      :!=F
0013      :A    F   150.0
0014      :=    Q    14.1
0015      :L    C      5
0016      :L    KF +2
0018      :!=F
0019      :A    F   150.0
001A      :=    Q    14.2
001B      :L    KF +3
001D      :L    C      5
001E      :!=F
001F      :A    F   150.0
0020      :=    Q    14.3
0021      :BE
```

با ایجاد تغییرات اندکی در بلوک حالت AUTO باید در OB 1 نیز تغییرات را به صورت زیر اعمال کنیم:

```
 OB 1

SEGMENT  1            0000
0000       :A    I    21.7
0001       :S    F   150.0
0002       :A    I    22.0
0003       :R    F   150.0
0004       :A    I    22.1
0005       :S    F    20.0
0006       :A    I    22.2
0007       :R    F    20.0
0008       :A    F    20.0
0009       :JC   PB   25
000A       :AN   F    20.0
000B       :JC   PB   23
000C       :BE
```

حال قصد آن داریم تا با ایجاد تغییری در برنامه، پروسه به صورت مرحله به مرحله (SINGLE STEP) انجام شود. در این برنامه می‌خواهیم روند اجرای برنامه به گونه‌ای باشد که با فشار دادن کلید (۱ + STEP) مربوط به هر مرحله تنها پروسهٔ همان مرحله انجام شود. این برنامه در PB 32 به صورت زیر آمده است. در این برنامه کلیدی اضافه می‌کنیم که در صورت فعال شدن آن، مرحلهٔ پس از مرحلهٔ فعلی انجام گیرد. به این کلید، (۱ + STEP) می‌گوئیم. این کلید با I 16.1 نشان داده شده است.

```
 PB 32

SEGMENT  1            0000
                  +---+
 I 21.0     ---! & !
 I 21.1     ---!   !
 I 21.2     ---!   !
 F 4.1      --O!   !
 F 4.2      --O!   !
 F 4.3      --O!   !     +-------+
 F 4.4      --O!   !--+-! =     ! F 4.0
                  +---+     +-------+
```

ادامهٔ بلوک PB 32:

```
SEGMENT  2          0009
            +---+    F 4.1
 F 4.0    ---! & !    +-----+
 I 16.1   ---!   !----!S    !
            +---+    !     !    +---+
           F 4.2     --!R   Q!----! & !
                       +-----+    !   !    +------+
                      F 150.0  ---!   !--+-! =    ! Q 14.0
                                 +---+    +------+

SEGMENT  3          0014
            +---+    F 4.2
 I 21.3   ---! & !    +-----+
 I 16.1   ---!   !----!S    !
            +---+    !     !    +---+
           F 4.3     --!R   Q!----! & !
                       +-----+    !   !    +------+
                      F 150.0  ---!   !--+-! =    ! Q 14.1
                                 +---+    +------+

SEGMENT  4          001F
            +---+    F 4.3
 I 21.4   ---! & !    +-----+
 I 16.1   ---!   !----!S    !
            +---+    !     !    +---+
           F 4.4     --!R   Q!----! & !
                       +-----+    !   !    +------+
                      F 150.0  ---!   !--+-! =    ! Q 14.2
                                 +---+    +------+

SEGMENT  5          002A
            +---+    F 4.4
 I 21.5   ---! & !    +-----+
 I 16.1   ---!   !----!S    !
            +---+    !     !    +---+
           I 21.6    --!R   Q!----! & !
                       +-----+    !   !    +------+
                      F 150.0  ---!   !--+-! =    ! Q 14.3
                                 +---+    +------+

SEGMENT  6          0035
           T 4
            +------+
 I 21.3   --!T!-!S!
 KT 015.2 --!TV BI!-
            !   DE!-
            !     !    +-------+
 I 21.4   --!R   Q!-+-! JC    ! PB 24
            +-----+    +-------+:BE
```

فصل ششم

رفع اشکال نرم‌افزاری

۶-۱- تشخیص اشکالات برنامه‌نویسی

یکی از مهمترین مواردی که همواره مد نظر برنامه نویسان بوده و هست تشخیص اشکالات نرم‌افزاری و نحوهٔ رفع اشکال در برنامه است. همانگونه که می‌دانید پس از نوشتن یک برنامهٔ کامپیوتری به هر زبان نیاز به کامپایل نمودن[1] برنامه و رفع اشکالات احتمالی آن است.

در پروگرامرهای PLC کامپایلری جهت این کار وجود ندارد، اما وجود اشکالات نرم‌افزاری را می‌توان در بخشی از حافظه به نام ISTACK[2] و BSTACK[3] جستجو نمود. این بخش، حافظهٔ داخلی CPU می‌باشد و محلی است که اطلاعاتی در مورد دستور نادرست یا به عبارت دیگر دستوری که باعث ایجاد وقفه در اجرای برنامه شده به دست می‌دهد. در این حالت PLC به حالت STOP[4] رفته، اطلاعات آخرین دستوری که باعث توقف برنامه شده در ISTACK باقی می‌ماند.

باید توجه داشت که تنها در حالتی می‌توان به ISTACK دست یافت که PLC در حالت RUN بوده، سپس به حالت STOP رفته باشد. برای عیب‌یابی نرم‌افزاری نیاز به یک سری اطلاعات است

---

1 - COMPILE 2 - INTERRUPT STACK 3 - BLOCK STACK

4 - STOP MODE

که توسط PLC در اختیار ما قرار می‌گیرد.

به محض اینکه یک بلوک توسط برنامه‌نویس ایجاد می‌شود (... ، FB ، PB ، OB) در حافظهٔ PLC، ۵ کلمه (WORD) توسط زبان STEP 5 اشغال می‌شود.

- در کلمهٔ اول عدد 7070_H قرار می‌گیرد، که به آن، کلمهٔ همزمان کننده یا Synchronization Word می‌گویند. بنابراین با دستیابی به Memory Map یا نقشهٔ حافظه و مشاهدهٔ عدد 7070_H در می‌یابیم که این نقطه، محل ایجاد و باز شدن یک بلوک است.

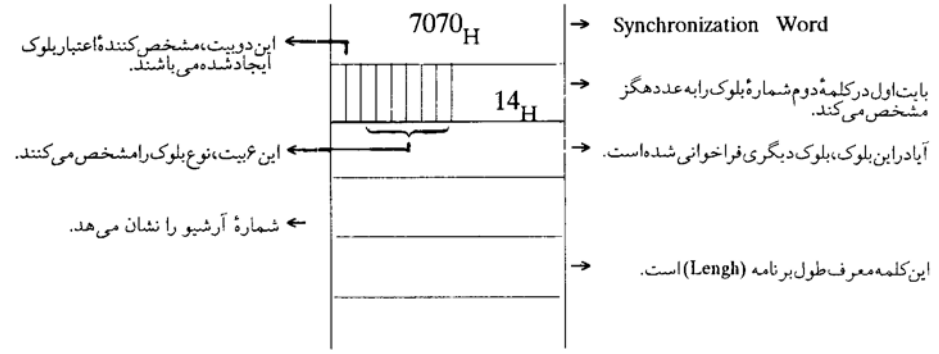
- در کلمهٔ دوم شمارهٔ بلوک، نوع و اعتبار بلوک قرار می‌گیرد. در این کلمه، دو بایت وجود دارد که بایت اول معرف شمارهٔ بلوک به عدد هگز می‌باشد و بایت بعدی را به بیت‌های جداگانه‌ای اختصاص می‌دهد. در دو بیت اول این بایت اعتبار بلوک و در ۶ بیت دیگر نوع بلوک مشخص می‌شود. به عنوان مثال اگر در ۶ بیت مذکور عدد $۴_{(۱۰)}$ را به هگز قرار دهد بدین معنی است که بلوک ایجاد شده از نوع PB است.

- کلمهٔ سوم مشخص کنندهٔ آن است که آیا در بلوک ایجاد شده، بلوک دیگری که معمولاً از نوع FB است فراخوانی شده است یا خیر؟

- کلمهٔ چهارم معرف آن است که آیا این بلوک دارای شمارهٔ آرشیو است یا خیر؟ زیرا برای هر بلوک می‌توان شمارهٔ آرشیو مشخص نمود.

- کلمهٔ پنجم طول برنامه را برحسب بایت مشخص می‌کند.

به این ۵ کلمه که توسط S5 در حافظه اشغال می‌شوند Block Header می‌گویند. از کلمهٔ ششم به بعد دستورات موجود در بلوک ایجاد شده که به کد ماشین تبدیل شده‌اند دیده می‌شوند. در شکل ۶-۱ نمونه‌ای از Block Header را می‌بینید:



شکل ۶-۱ : نقشه حافظه و Block Header ایجاد شده در آن

به دلیل اینکه دستورالعمل نادرست در ISTACK و BSTACK به صورت کد ماشین به برنامه‌نویس ارائه می‌شود ابتدا به بررسی چگونگی تبدیل کد ماشین به دستورات S5 می‌پردازیم.

۶-۲- تبدیل کد ماشین به دستورات STEP 5

در جدول ضمیمهٔ ۳، کدهای ماشین استفاده شده در PLC زیمنس آورده شده است. این کدها را می‌توان به چند دستهٔ عمومی تقسیم نمود.

- دسته‌ای از کدها که احتیاج به تغییر و تبدیل ندارند مانند کدهای 6500 و 6501 و ... که به ترتیب معرف دستورات BE و BEU می‌باشند. در این دستورات هیچ عملوندی یافت نمی‌شود.

- دسته‌ای از کدها که نیاز به تغییر و تبدیل به عملکرد و عملوند و همچنین آدرس عملوند دارند. روش برخورد با این کدها به صورت زیر است:

با جستجو در جدول حاوی کدهای ماشین، بزرگترین کد موجود در جدول را که به هنگام تفاضل از کد مورد نظر نیاز به بیت قرضی[1] نداشته باشد یافته، حاصل تفریق را به دست می‌آوریم. کُد متناظر موجود در جدول شامل قسمتی از دستورالعمل نادرست است و به عنوان مثال به صورت A I ... می‌باشد ولی شماره و آدرس عملوند در آن یافت نمی‌شود. حاصل تفریق، مشخص کنندهٔ آدرس عملوند است. در مثالهای زیر سعی شده تا تمام موارد ممکن بررسی شود تا خواننده در عیب‌یابی نرم‌افزاری و تبدیل کدهای ماشین به دستورات S5 تسلط کافی پیدا کند.

مثال ۶-۱ : کد ماشین 6500 را به دستورالعمل S5 تبدیل کنید.

با مقایسهٔ کد داده شده و کدهای موجود در جدول ملاحظه می‌کنید که این کد مربوط به دستورالعمل BE است. از آنجایی که این دستورالعمل فاقد عملوند می‌باشد در جدول بدون هیچگونه عملوند دیده می‌شود. در صورتی که این کد، کد دستورالعمل نادرست باشد بدین معنی است که احتمالاً در حین برنامه‌نویسی این دستور از قلم افتاده است.

مثال ۶-۲ : کُد ماشین C000 را به دستورالعمل S5 تبدیل کنید.

همان‌طور که گفته شد با جستجو در جدول ضمیمهٔ ۳ بزرگترین کُد ماشین را که به هنگام تفریق

---

1 - Borrow Bit

از کد C000 نیاز به بیت قرضی نداشته باشد می‌یابیم. سپس با انجام عملیات تفریق و به دست آوردن حاصل تفریق، شمارهٔ عملوند و آدرس آن را به دست می‌آوریم. بزرگترین کد ماشین موجود در جدول که به هنگام تفاضل نیاز به بیت قرضی نداشته باشد همان کُد C000 است که معرف دستور A I ... می‌باشد. آدرس عملوند از حاصل تفریق به دست می‌آید.

C000 -
C000 ⟶ A I 0.0 ← مقایسه در جدول و یافتن کد مذکور
00.00
شماره بایت ← ← شماره‌بیت

بنابراین دستورالعمل نادرست A I 0.0 بوده است.

مثال ۶-۳: دستورالعمل متناظر با کد ماشین C010 را پیدا کنید.

بزرگترین کد موجود در جدول که به هنگام تفاضل از کد C010 نیاز به بیت قرضی ندارد C000 است. آدرس عملوند نیز از حاصل تفریق به دست می‌آید.

C010 -
C000 ⟶ A I 16.0 ← مقایسه در جدول و یافتن کد مذکور
00.10
شماره‌بیت ← ← شمارهٔ بایت به هگز: $1 \times 16 + 0 = 16$

مثال ۶-۴: دستورالعمل متناظر با کد ماشین CA85 کدام است؟

با مقایسه در جدول ملاحظه می‌کنید که کد C880 بزرگترین کد ماشین است که به هنگام تفاضل از کد CA85 نیاز به بیت قرضی ندارد. این کد معرف ... O Q می‌باشد.

CA85 -
C880 ⟶ O Q 5.2 ← مقایسه در جدول و یافتن کد مذکور
02.05
شماره بایت به هگز: $0 \times 16 + 5 = 5$ ← ← شماره‌بیت به هگز: $0 \times 16 + 2 = 2$

مثال ۶-۵: کد ماشین 343B را به دستورالعمل متناظر آن در S5 تبدیل کنید.

بزرگترین کد موجود در جدول کد 3400 بوده که متناظر با دستور ... T SP می‌باشد. شمارهٔ تایمر از حاصل تفریق به دست می‌آید.

343B -
3400 ⟵ مقایسه در جدول و یافتن کد مذکور ⟵ SP T 59
003B
3 × 16 + 11 = 59

مثال ۶-۶: دستورالعمل متناظر با کد ماشین 7582 را بیابید.

با مقایسه در جدول به کد 7500 دست خواهیم یافت که معرف پرش غیرشرطی به بلوک برنامه به صورت ... JU PB می‌باشد. شمارهٔ PB مورد نظر نیز از حاصل تفریق به دست می‌آید.

7582 -
7500 ⟵ مقایسه در جدول و یافتن کد مذکور ⟵ JU PB 178
0082
8 × 16 + 2 = 178

مثال ۶-۷: کد ماشین F823 را به دستورالعمل متناظر آن در S5 تبدیل کنید.

کد F800 بزرگترین کد ماشین است که به هنگام تفاضل از کد F823 نیاز به بیت قرضی ندارد. این کُد معرف دستور ... A T می‌باشد و شمارهٔ تایمر نیز از حاصل تفریق به دست می‌آید.

F823 -
F800 ⟵ مقایسه در جدول و یافتن کد مذکور ⟵ A T 35
0023
2 × 16 + 3 = 35

مثال ۶-۸: دستورالعمل متناظر با کد ماشین 861E را بیابید.

با مقایسه در جدول به کد 8000 دست خواهیم یافت. این که معرف دستورالعمل ... A F می‌باشد که آدرس فلگ از حاصل تفریق به دست می‌آید.

861E -
8000 ⟵ مقایسه در جدول و یافتن کد مذکور ⟵ A F 30.6
06.1E
30 = 14 + 16 × 1 : شمارهٔ بایت به هگز ⟵ ⟵ شمارهٔ بیت به هگز : 6 = 6 + 16 × 0

حال که با روش تبدیل کد ماشین به دستورالعمل متناظر با آن در S5 آشنا شدیم به بررسی ISTACK و BLOCK STACK می‌پردازیم. در پروگرامرهای مختلف، صفحهٔ نمایش ISTACK نیز متفاوت می‌باشد ولی اصول کار همگی آنها تقریباً یکسان است. در جداول ۶-۱ و ۶-۲ صفحهٔ

نمایش ISTACK در پروگرامر PG 605 و PG 615U نشان داده شده است. قسمتهایی که با خطوط پر رنگ‌تر مشخص شده‌اند نشان دهندهٔ علت ایجاد اشکال و همچنین آدرس دستورالعمل نادرست می‌باشد. در جداول ضمیمهٔ ۴ این اشکالات به صورت مشروح آورده شده است.

جدول ۶-۱ : صفحهٔ نمایش ISTACK در پروگرامرهای PG 615 U , PG 605

Bit / Byte	7	6	5	4	3	2	1	0	Absolute addr.	System data word (RS)
1			BST SCH	SCH TAE	ADR BAU	SPA BBR			EA0A	SD 5
2	CA-DA	CE-DA		REM AN					EA0B	
3	STO ZUS	STO ANZ	NEU STA		BAT PUF		BARB	BARB END	EA0C	SD 6
4		UA FEHL				AF			EA0D	
5	ASP NEP	ASP NRA	KOPF NI		ASP NEEP				EA0E	SD 7
6	KEIN AS	SYN FEH	NINEU					UR LAD	EA0F	
7	IRRELEVANT									
8	IRRELEVANT									
9	STOPS		SUF	TRAF	NNN	STS	STUEB	FEST	EBAC	SD 214 (UAW)
10	NAU	QVZ	KOLIF	ZYK	SYSFE	PEU	BAU	ASPFA	EBAD	
11									EBAA	SD 213
12	ANZ1	ANZ0	OVFL		OR	STA TUS	VKE	ERAB	EBA8	
13	6th nesting level					OR	VKE	FKT	EBA8	SD 212
14	IRRELEVANT								EBA9	
15	4th nesting level					OR	VKE	FKT	EBA6	SD 211
16	5th nesting level					OR	VKE	FKT	EBA7	

جدول ۶-۲ : صفحهٔ نمایش ISTACK در پروگرامرهای PG 605 , PG 615 U

(ادامه جدول ۶-۱)

Bit / Byte	7	6	5	4	3	2	1	0	Abso-lute addr.	System data word (SD)
17	2nd nesting level					OR	VKE	FKT	EBA4	SD 210
18	3rd nesting level					OR	VKE	FKT	EBA5	
19	Nesting depth (0 to 6)								EBA2	SD 209
20	1st nesting level					OR	VKE	FKT	EBA3	
21	Start address of the data block (high)								EBA0	SD 208
22	Start address of the data block (low)								EBA1	
23	Block stack pointer (high)								EB9E	SD 207
24	Block stack pointer (low)								EB9F	
25	Step address counter (high)								EB9C	SD 206
26	Step address counter (low)								EB9D	
27	Statement register (high)								EB9A	SD 205
28	Statement register (low)								EB9B	
29	ACCUM 2 (high)								EB98	SD 204
30	ACCUM 2 (low)								EB99	
31	ACCUM 1 (high)								EB96	SD 203
32	ACCUM 1 (low)								EB97	

در جداول زیر صفحات نمایش ISTACK و بیت‌های کنترلی را بر روی پروگرامر مشاهده می‌کنید. در صفحهٔ نمایش CONTROL BITS قسمتهایی که با حروف پر رنگ‌تر مشخص شده‌اند نشان دهندهٔ یکی از احتمالات وجود خطا می‌باشد.

جدول ۶-۳ : صفحهٔ نمایش بیت‌های کنترلی در پروگرامرهای مذکور

CONTROL BITS								Absolute address	System data word
NB	PBSSCH	BSTSCH	SCHTAE	ADRBAU	SPABBR	NAUAS	QUITT	EA0A	SD5
CA-DA	CE-DE	NB	REMAN	NB	NB	NB	NB	EA0B	
STOZUS	STOANZ	NEUSTA	NB	BATPUF	NB	BARB	BARBEND	EA0C	SD6
NB	UAFEHL	MAFEHL	EOVH	NB	AF	NB	NB	EA0D	
ASPNEP	ASPNRA	KOPFNI	PROEND	ASPNEEP	PADRFE	ASPLUE	RAMADFE	EA0E	SD7
KEINAS	SYNFEH	NINEU	NB	NB	NB	SUMF	URLAD	EA0F	

جدول ۶-۴ : صفحهٔ نمایش ISTACK در پروگرامرهای مذکور

INTERRUPT STACK														Abso-lute addr.	System data word
DEPTH: 01															
BEF-REG: 0000 BST-STP: EB07		SAZ: E30A Baust.-NR.: 1 REL-SAZ: 0000			DB-ADR: 0000 DB-NR.:									EB9A -EBA0	SD205-208
AKKU1: FFF1		AKKU2: 00FF												EB96-EB98	SD203-204
BRACKETS:		KE1	000	KE2	000	KE3	000	KE4	000	KE5	000	KE6	000	EBA2-EBA8	SD209-212
RESULT DISPLAY:		ANZ1	ANZ0	OVFL	CARRY			ODER	STATUS	VKE	ERAB			EBAA	SD213
CAUSE OF FAULT:		STOPS		SUF	TRAF	NNN		STS	STUEB					EBAC	SD214 (UAW)
		NAU	QVZ		ZYK			PEU	BAU	ASPFA				EBAD	

در جداول ۶-۵ و ۶-۶ نیز صفحات نمایش بیت‌های کنترلی و ISTACK در پروگرامر 685U آورده شده است. به شباهتهای موجود بین جداول مذکور و جداول ۶-۳ و ۶-۴ توجه کنید.

جدول ۶-۵ : صفحهٔ نمایش بیت‌های کنترلی در پروگرامر 685U

C O N T R O L BITS

NB	PBSSCH	BSTSCH	SCHTAE	ABRBAU	SPABBR	NAUAS	QUITT
NB	NB	NB	REMAN	NB	NB	NB	NB
STOZUS X	STOANZ X	NEUSTA	NB	BATPUF X	NB	BARB	BARBEND
NB	UAFEHL	MAFEHL	EOVH	NB	AF	NB	NB
ASPNEP	ASPNRA	KOPFNI	PROEND	ASPNEEP	PADRFE	ASPLUE	RAMADFE
KEINAS	SYNFEH	NINEU	NB	NB	NB	SUMF	URLAD

جدول ۶-۶ : صفحهٔ نمایش ISTACK در پروگرامر 685U

I N T E R R U P T S T A C K

D E P T H: 01

OP-REG :	3200	SAC:	9066	DB-ADD:	0000
BLK-STP:	EB07	PB-NO :	25	DB-NO.:	
		REL-SAC:	0004		

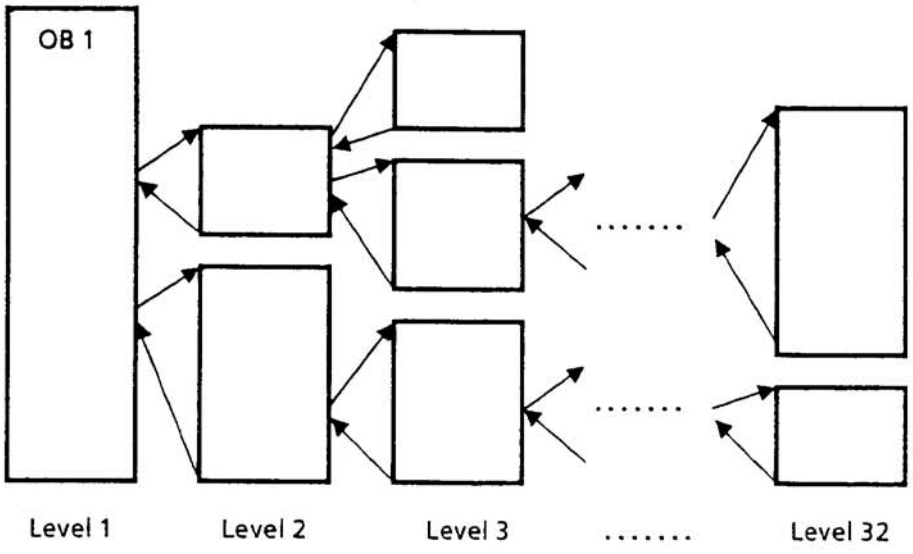
ACCU1: 1388 ACCU2: 05D8

CONDITION CODE :	CC1	CC0	OVFL	CARRY	OR	ERAB X
	STATUS	RLO X				
CAUSE OF INTERR.:	STOPS	NB	SUF	TRAF X	NNN	STS
	STUEB	NAU	QVZ	ZYK	PEU	BAU
	ASPFA					

اکنون به توضیح برخی از بخشهای مهم در جدول ۶-۴ می‌پردازیم. لازم به ذکر است که توضیحات ارائه شده برای موارد مشابه در جدول ۶-۶ نیز صادق است.

DEPTH : شماره‌ای که مقابل این بخش از صفحهٔ ISTACK قرار می‌گیرد معرف شمارهٔ سطح (LEVEL) بلوک‌ها در یک برنامه است. در جدول ۶-۴ منظور از DEPTH 1 اولین بلوک در اجرای برنامه یعنی OB 1 می‌باشد، بنابراین دستورالعمل نادرست در OB 1 وجود دارد. در شکل ۶-۲ سطوح (LEVEL) مختلف در برنامه‌نویسی نشان داده شده است.

لازم به ذکر است که تعداد سطوح یا تعداد پرشهای افقی در یک برنامه در PLCهای گوناگون، متفاوت است. در اینجا فرض بر آن است که آخرین سطح پرش 32 LEVEL می‌باشد.



شکل ۶-۲ : تعداد سطوح یا پرشهای افقی مجاز در یک برنامه

BEF-REG : کُد دستورالعمل نادرست را به صورت کد ماشین نشان می‌دهد. این کُد ماشین متناظر با دستورالعمل نادرستی است که باعث ایجاد توقف در اجرای برنامه شده است.

SAZ: آدرس محلی از حافظه است که دستورالعمل نادرست در آن قسمت از حافظه قرار گرفته است.

DB-NR , DB-ADR : در صورتی که در بلوکی که حاوی دستورالعمل نادرست است اطلاعاتی از بلوک DB فراخوانده شده باشد DB و آدرس DB مذکور در حافظه را نشان می‌دهد.

AKKU 1 , AKKU 2 :محتویات دو انبارک را در لحظهٔ ایجاد وقفه (STOP) نشان می‌دهد.

۶-۳- دستیابی به دستورالعمل نادرست با استفاده از ISTACK

همانگونه که عنوان شد در جدول ISTACK بخش BEF-REG حاوی کُد دستورالعمل نادرست می‌باشد که به صورت کُد ماشین بر روی صفحهٔ نمایش ظاهر می‌گردد. اکنون برای درک بهتر چگونگی دستیابی به دستورالعمل نادرست، جدول ۶-۴ را در نظر می‌گیریم. در بخش

DEPTH شمارهٔ 01 درج شده است و این شماره نشان دهندهٔ شمارهٔ سطح بلوکی است که در آن بلوک دستورالعمل نادرست وجود دارد. با توجه به شمارهٔ 01 و قرار داشتن بلوک OB 1 در این سطح می‌توان گفت که دستورالعمل نادرست در OB 1 قرار دارد.

در بخش BEF-REG کد 0000 دیده می‌شود. با مراجعه به جداول موجود در ضمیمهٔ ۳ ملاحظه می‌کنیم که کد مذکور مربوط به دستورالعمل NOP 0 است. بنابراین دستور نادرست NOP 0 در بلوک OB 1 باعث ایجاد وقفه شده است.

با دقت در جدول ۶-۴ ملاحظه می‌کنید که در بخش DB-ADR و DB-NR عدد 0000 وجود دارد که نشان دهندهٔ عدم فراخوانی بلوک DB در برنامه می‌باشد.

در جدول ۶-۶ نیز در بخش DEPTH عدد 01 دیده می‌شود. بنابراین دستورالعمل نادرست در OB 1 قرار دارد. در بخش OP-REG که متناظر با بخش BEF-REG در جدول ۶-۴ می‌باشد کد 3200 وجود دارد که با مراجعه به جدول ضمیمه ملاحظه می‌کنید که مربوط به دستورالعمل ... L DW می‌باشد. با توجه به توضیحات ارائه شده، شمارهٔ DW به روش زیر به دست می‌آید.

کد موجود در OP-REG←3200 -
کد موجود در جدول←3200

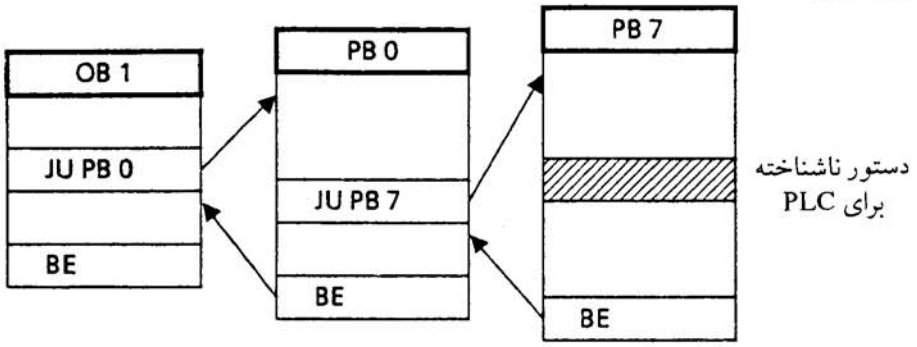
0000 ⟶ شمارهٔ DW

بنابراین دستورالعمل نادرست L DW 0 ، می‌باشد که در OB 1 قرار دارد. با دقت در جدول ۶-۶ ملاحظه می‌کنید که در بخش DB-ADD و DB-NO عدد 0000 وجود دارد که نشان دهندهٔ عدم فراخوانی بلوک DB در این برنامه است. بنابراین واضح است که وقفه بوجود آمده به دلیل بارگذاری یک کلمه (DW) از بلوکی است که در این برنامه فراخوانی نشده است.

۶-۴- روش دیگربرای دستیابی به دستورنادرست با استفاده از ISTACK

در توضیحات جداول ۶-۴ و ۶-۶ عنوان شد که بخش SAZ یا SAC در صفحهٔ نمایش ISTACK حاوی آدرس محلی از حافظه است که PLC قبل از رفتن به حالت STOP در آن محل قرار داشته است. بنابراین به کمک این آدرس می‌توان دستور نادرست را یافت. مثال زیر این مطلب را روشن می‌سازد.

مثال ۶-۹ : در شکل زیر یک برنامه راکه شامل ۳ بلوک PB 7 و PB 0 و OB 1 می‌باشد مشاهده می‌کنید. در PB 7 یک دستور ناشناخته برای PLC وجود دارد که باعث ایجاد وقفه در اجرای برنامه شده است.

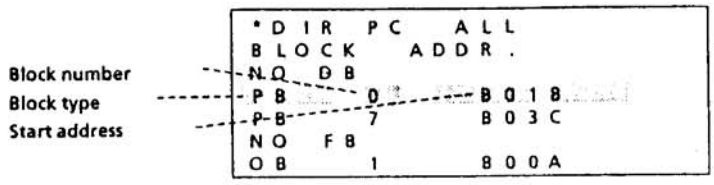


شکل ۶-۳ : این برنامه شامل سه بلوک است که یکی از بلوک‌ها حاوی دستوری ناشناخته برای PLC می‌باشد

در حین اجرای برنامه به محض اینکه PLC به دستور نادرست موجود در PB 7 برسد اجرای برنامه متوقف شده، PLC به حالت STOP می‌رود. در این حالت پیغام NNN ظاهر می‌گردد. این پیغام خطا معرف آن است که این سطر قابل پردازش و تعریف توسط CPU نمی‌باشد.

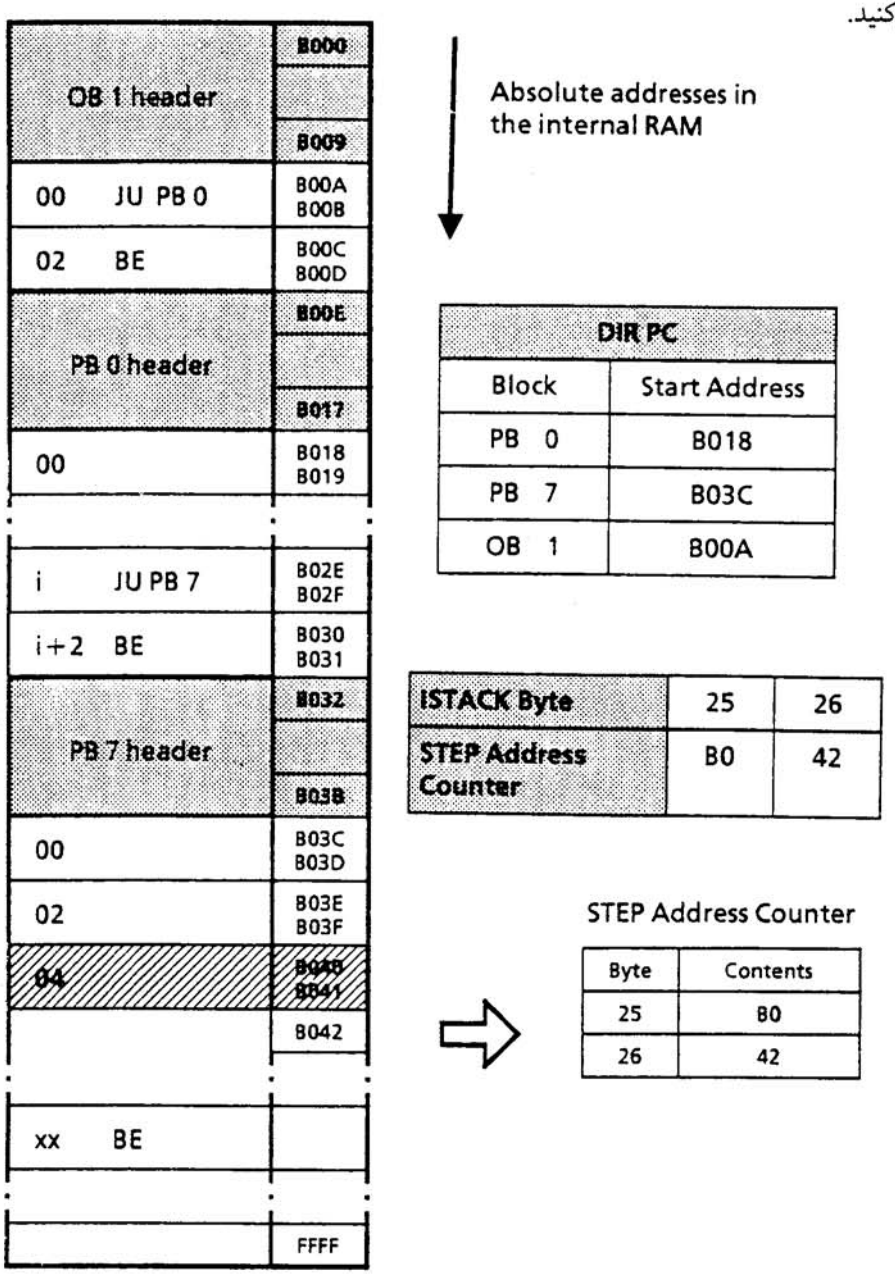
در بخش SAZ آدرس دستور نادرست در حافظه داده می‌شود. در زبان برنامه‌نویسی S5 دستور دیگری وجود دارد که به صورت PC DIR بوده، با اجرای آن می‌توان آدرس شروع هر بلوک در برنامه را بر روی صفحهٔ PG مشاهده نمود. بنابراین می‌توان با به دست آوردن تفاضل آدرس SAZ و آدرس شروع بلوکی که در آن، دستور نادرست وجود دارد به سطری که حاوی دستورالعمل نادرست می‌باشد دست یافت. البته این روش تنها در مورد پروگرامر 605U PG معتبر است.

در شکل ۶-۴ صفحهٔ نمایش دستور PC DIR بر روی پروگرامر 615U PG دیده می‌شود.



شکل ۶-۴ : صفحهٔ نمایش PC DIR در پروگرامر PG 615U

در شکل ۶-۵ نحوهٔ دستیابی به سطری از برنامه که حاوی دستور نادرست می‌باشد را ملاحظه می‌کنید.



شکل ۶-۵ : نحوهٔ دستیابی به شمارهٔ سطر محتوی دستورالعمل نادرست

با مقایسهٔ دو آدرس موجود در بخش SAZ یعنی B042 و آدرس شروع بلوک PB 7 یعنی B03C در می‌یابیم که دستورالعمل نادرست در این بلوک قرار دارد.

B042 – : محاسبهٔ شمارهٔ سطر محتوی دستورالعمل نادرست
B03C
0006 ⟵ سطرششم ازبرنامهٔ PB 7 باعث توقف اجرای برنامه شده‌است.

۶-۵- دستیابی به دستورالعمل نادرست با استفاده از BSTACK

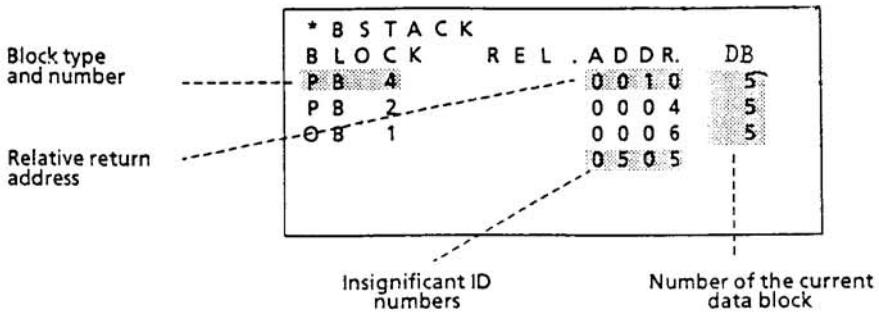
همان‌گونه که در فصول گذشته عنوان شد به هنگام پرش از یک بلوک به بلوک دیگر سه دسته اطلاعات زیر در BSTACK ذخیره می‌شوند:

- شمارهٔ بلوک DB که قبل از JUMP یا STOP معتبر بوده است.
- آدرس سطری از برنامه که هنگام بازگشت بایستی اجرای برنامه از آن سطر دنبال گردد.
- شماره و نوع بلوکی که از آن JUMP یا STOP انجام گرفته است.

با استفاده از BSTACK می‌توان به اطلاعات مذکور دست یافت. BSTACK حاوی وضعیت BLOCK STACK در زمانی است که در اجرای برنامه وقفه‌ای ایجاد شده است.

مثال ۶-۱۰: در طول اجرای یک برنامه، وجود دستوری در FB 2 باعث ایجاد وقفه و رفتن PLC به حالت STOP شده است. در این حالت بر روی صفحهٔ PG پیغام خطای TRAF ظاهر می‌شود که مربوط به فراخوانی نادرست اطلاعات از DB 5 می‌باشد.

در شکل زیر صفحهٔ نمایش BSTACK را در پروگرامر PG 615 می‌بینید.

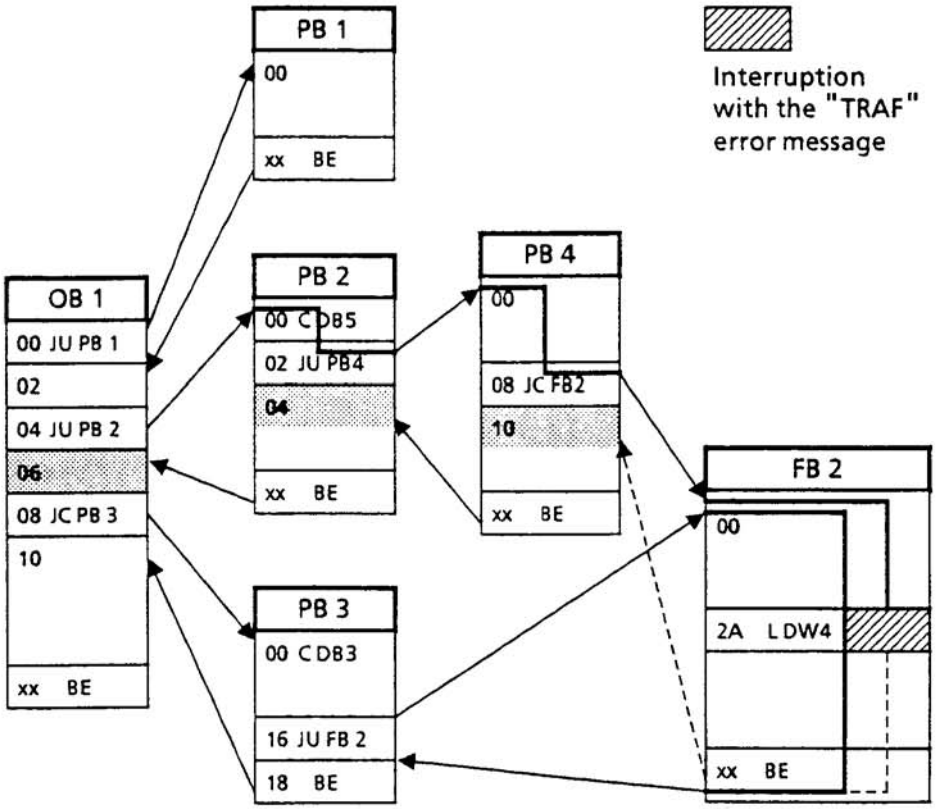


شکل ۶-۶: صفحهٔ نمایش BSTACK در پروگرامر PG 615

در شکل ۶-۶ ملاحظه نمودید که در طول اجرای برنامه به صورت دنبال OB 1→ PB 2→ PB 4، اطلاعات بلوک DB 5 به صورت نادرست خوانده شده است و یا اینکه در DB 5 اشکال ساختاری وجود دارد.

البته روش ذکر شده در مورد تمام پروگرامرها به استثنای PG 605 معتبر است.

در شکل ۶-۷ چگونگی روند اجرای برنامه تا زمانی که PLC به حالت STOP می‌رود نشان داده شده است.



شکل ۶-۷ : روند اجرای برنامهٔ مثال ۶-۱۰ تا زمان ایجاد وقفه در اجرای برنامه

فصل هفتم

برنامه‌نویسی به زبان CSTL

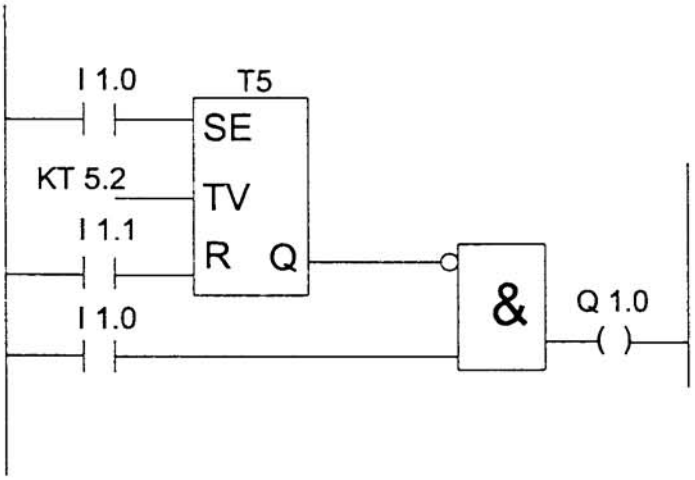
زبان PLC شرکت کنترونیک که CSTL نام دارد و مخفف عبارت "CONTRONIC STATEMENT LIST" است از همان قواعد کلی مربوط به زبان PLCهای زیمنس پیروی می‌کند با این تفاوت که در این زبان کوشیده شده تا آنجا که ممکن است استفاده از این قواعد ساده‌تر و نوشتن برنامه، خلاصه‌تر گردد. فهرست دستورات‌این زبان در ضمیمهٔ ۵ آمده است.

البته بین دو زبان S5 و CSTLتفاوتهای اندکی وجود دارد. به عنوان مثال در تعریف تایمرها و همچنین دستور DO بین این دو زبان تفاوتهایی به چشم می‌خورد. در این فصل به ذکر چند نمونه از تفاوتهای مذکور پرداخته، سپس با استفاده از زبان CSTL به برنامه‌نویسی برخی از مثالهای حل شده در فصول گذشته که به زبان S5 برنامه‌نویسی شده‌اند خواهیم پرداخت تا خواننده با مقایسهٔ این برنامه‌ها، با شباهتهای این دو زبان برنامه‌نویسی نیز آشنا گردد.

۷-۱- تایمرها

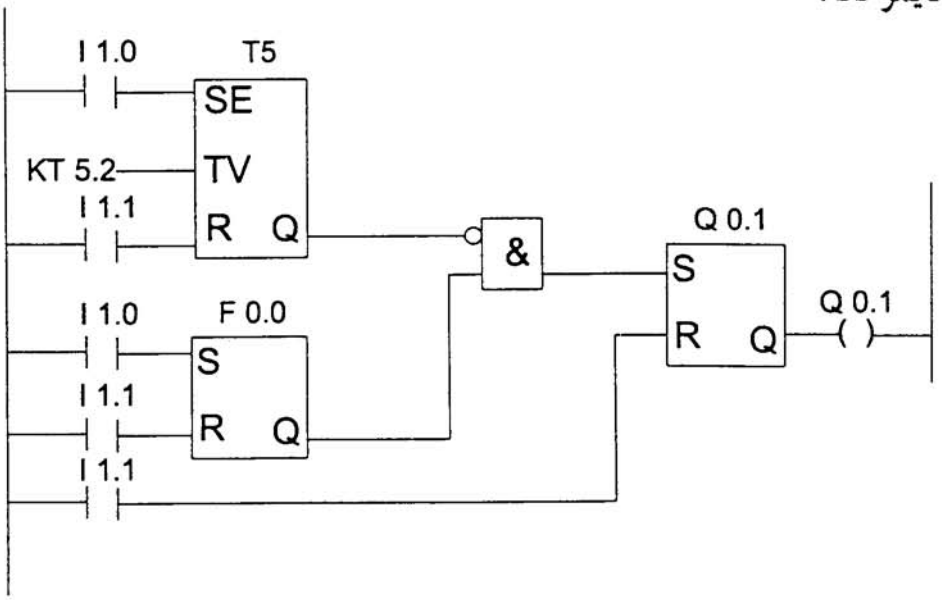
در PLCهای کنترونیک، سه نوع تایمر SE و SP و SF موجودند و تایمرهای دیگر یعنی SD و SS را باید نرم‌افزاری ساخت. در ادامه، چگونگی تعریف تایمرهای SD و SS به کمک تایمر SE نشان داده شده است.

تایمر SD :




```
0001          :A     I   1.0
0002          :L     KT  5.2
0003          :SE    T   5
0004          :A     I   1.1
0005          :R     T   5
0006          :AN    T   5
0007          :A     I   1.0
0008          :=     Q   1.0
0009          :BE
```

تایمر SS :




```
0001                :A      I    1.0
0002                :L     KT   5.2
0003                :S      F    0.0
0004                :SE     T    5
0005                :A      F    0.0
0006                :AN     T    5
0007                :S      Q    0.1
0008                :A      I    1.1
0009                :R      T    5
0010                :R      Q    0.1
0011                :R      F    0.0
0012                :BE
```

۷-۲- آدرس‌دهی غیر مستقیم (Indirect Addressing)

این روش آدرس‌دهی برای خواندن یا بارگذاری تعدادی پارامتر از نقطه‌ای و ارسال به نقطهٔ دیگر مورد استفاده قرار می‌گیرد. طرز عملکرد این روش را در زبان CSTL با ذکر یک مثال بررسی می‌کنیم.

مثال ۷-۱ : برنامهٔ روشن شدن نمایش دهندهٔ هفت قسمتی (7-Segment) را با استفاده از روش آدرس‌دهی غیرمستقیم در CSTL بازنویسی کنید.

چنانچه به یاد داشته باشید در برنامهٔ مذکور اعداد ۰ تا ۹ با فاصلهٔ زمانی معینی بر روی نمایش دهنده ظاهر می‌شوند و این روند به همین ترتیب ادامه می‌یافت. در ادامه، برنامهٔ بازنویسی شده با استفاده از روش آدرس‌دهی غیرمستقیم در CSTL (معادل با دستور DO در S5) آمده است. در این برنامه فاصلهٔ زمانی نمایش هر عدد و عدد بعدی ۰/۸ ثانیه در نظر گرفته شده است.

```
0001            :L     FB  10    indirect add flag byte
0002            :==    KD  10
0003            :O     I   0.0
0004            :JC    = AAAAA
0005            :AN    F   20.0
0006            :L     KT  8.1
0007            :SE    T   1
0008            :A     T   1
0009            :=     F   20.0
0010            :A     F   20.0  0.8 sec flag=0
0011            :BEC
0012            :L     FB  10
0013            :+     KD  1
0014            :T     FB  10
0015            :L     KH  000BF hex parameter for zero
0016            :T     MB  1     memory word add
0017            :L     KH  00086 hex parameter for one
0018            :T     MB  2     memory word add
0019            :L     KH  000DB hex parameter for two
0020            :T     MB  3     memory word add
0021            :L     KH  000CF hex parameter for three
0022            :T     MB  4
0023            :L     KH  000E6 hex parameter for four
0024            :T     MB  5
0025            :L     KH  000ED hex parameter for five
0026            :T     MB  6
0027            :L     KH  000FD hex parameter for six
0028            :T     MB  7
0029            :L     KH  00087 hex parameter for seven
0030            :T     MB  8
0031            :L     KH  000FF hex parameter for eight
0032            :T     MB  9
0033            :L     KH  000EF hex parameter for nine
0034            :T     MB  10
0035            :L     MW (FB10) FB10 VARIABLE ADDRESS
0036            :T     QB  2
0037            :BEU
0038      AAAAA :L     KD  0     START POINT
0039            :T     FB  10
0040            :BE
```

روند اجرای این برنامه به صورت زیر است:

در سطرهای اول و دوم مجتویات FB 10[1] با عدد دهدهی ۱۰ مقایسه می‌شود در صورت تساوی این دو مقدار و یا "۱" بودن مقدار ورودی I 0.0 (کلید ورودی I 0.0 جهت کنترل دستی و شمارش از 0 به کار برده شده است) پردازنده به سطری که با برچسب AAAAA مشخص شده است پرش نموده، عدد 0 را در FB 10 قرار می‌دهد.

تایمر تعریف شده هر ۰/۸ ثانیه یک بار روشن و خاموش ("۱" و "۰") شده، در صورتی که "۱" = F 20.0 باشد، دستوراتی که پس از دستور BEC قرار دارند اجرا نخواهند شد. به محض اینکه خروجی تایمر "۰" شود ("۰" = F 20.0) پردازنده دستوراتی را که پس از دستور BEC قرار دارند اجرا نموده، هر بار یک واحد به محتویات FB 10 اضافه می‌کند، ضمن اینکه پارامترهای هگزادسیمال معادل 0 تا 9 را در MBهای[2] تعریف شده ثبت خواهد کرد. در انتهای برنامه، MB ای را که آدرس آن در FB 10 تعریف شده است را از حافظه خوانده و به QB 2 ارسال می‌کند. بنابراین اعداد 0 تا 9 در 7-Segment به فاصلهٔ زمانی هر ۰/۸ ثانیه ظاهر خواهد شد.

۷-۳- قابلیت استفاده از دستورات تکمیلی در تمامی بلوک‌ها

در صورتی که به خاطر داشته باشید در فصول گذشته عنوان شد که دستورات تکمیلی (Supplementary) در S5 تنها در FBها (بلوک‌های تابع‌ساز) قابل استفاده هستند. برخلاف PLC های زیمنس در PLC های کنترونیک و در زبان CSTL استفاده از این دستورات در تمامی بلوک‌ها مجاز است. بنابراین یکی از مزایای زبان CSTL نسبت به S5 امکان استفاده از دستورات تکمیلی در تمامی بلوک‌ها است.

اکنون که با تفاوتهای نرم‌افزاری جزئی موجود بین دو زبان CSTL و S5 آشنا شدیم به بررسی سخت‌افزار یکی از مدلهای PLC ساخت شرکت کنترونیک یعنی PLC 500 می‌پردازیم.

---

۱ - در CSTL ، FB معادل باFY در S5 است (FB = Flag Byte)

۲ - MBها (Memory Byte) در واقع بایت‌هایی از حافظه جهت ذخیرهٔ پارامترها هستند از MWها (Memory Word) نیز می‌توان به صورت بیت استفاده نمود.

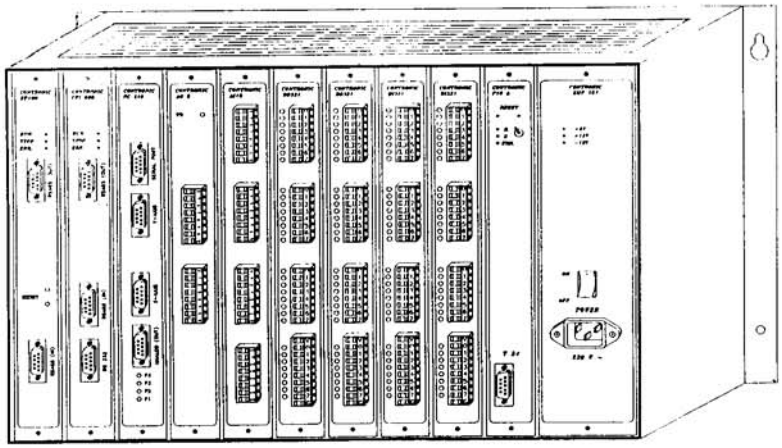
PLC 500 -۴-۷

PLCهای سری 500 (PLC 500) شرکت کنترونیک، کنترل کننده‌هایی هستند که دارای چند پروسسور بوده، برای انجام امور اتوماسیون در سطوح متوسط و بزرگ طراحی شده‌اند. سری PLC 500 راه حل جامع و ساده‌ای برای انجام وظایف کنترلی در مواردی مانند:

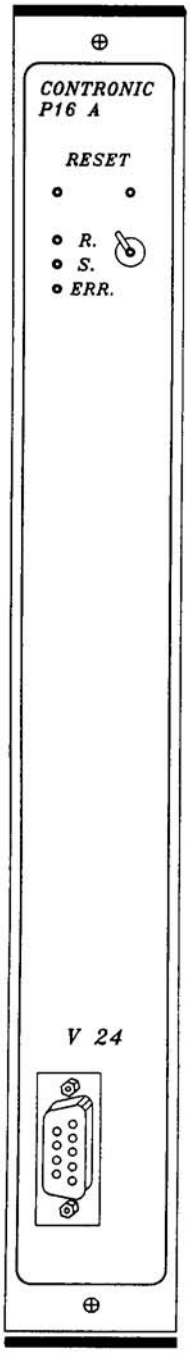
- سیستم‌های کنترل ماشین‌ها - اتوماسیون فرآیندها - مونیتورینگ فرآیندها

ارائه می‌دهد. استفاده از تکنولوژی استاندارد در ساخت سخت‌افزار و طرح مدولار این PLCها کاربرد ساده، سریع و مطمئن آنها را تضمین می‌کند. امکان نصب تعداد زیادی ورودی و خروجی، پردازش سیگنال‌های آنالوگ و انجام عملیات توابع پیچیده کنترلی از دیگر مزایای سری PLC 500 کنترونیک می‌باشد.

در سری PLC 500 کنترونیک تعداد زیادی از اعمال وقت‌گیر و پیچیدهٔ جانبی توسط مدول‌های هوشمند انجام می‌شود در نتیجه پردازشگر اصلی از سرعت پردازش بالا برخوردار است. به خاطر دارا بودن کارت‌های جانبی هوشمند متعدد، این PLC به راحتی قابل گسترش بوده و وظایف کنترلی نظیر کنترل مکان، پردازش سیگنال‌های آنالوگ و ... را به راحتی انجام می‌دهد.
برنامه‌نویسی PLC توسط کامپیوترهای سازگار با IBM انجام شده، برنامه‌ها به صورت سریال به PLC ارسال می‌شوند. در ادامه، به معرفی سخت‌افزار این PLC می‌پردازیم.



شکل ۷-۱ : نمای شماتیکی PLC 500



۷-۴-۱- کارت پردازشگر مرکزی (P16 A)

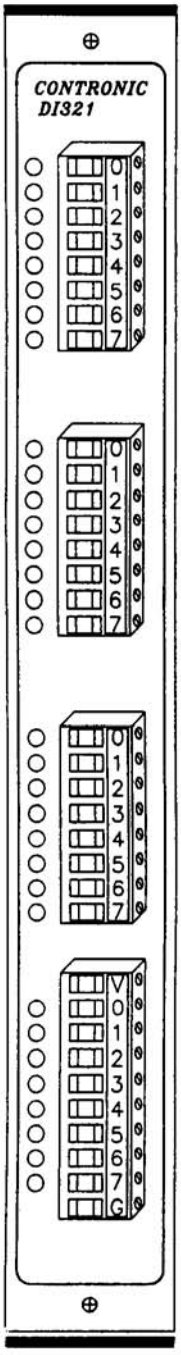
در PLCهای سری 500 دو مدل CPU متناسب با نوع پروسه کنترل قابل به‌کارگیری است.

کارت پروسسور در دو مدل 160C با میکروپروسسور ۱۶ بیتی و مدول 80C با میکروپروسسور ۸ بیتی برای پردازش عملیات بیتی، بایتی، کلمه‌ای و روتین‌های داخلی مانند تایمرها و شمارنده‌ها به کار برده می‌شوند. حافظه موجود عبارت است از: حافظه RAM برای فلگ‌ها، تایمرها، شمارنده‌ها و ...، حافظه داخلی RAM برای برنامه کاربر (User Program) و حافظه داخلی EEPROM برای نگهداری همیشگی برنامه کاربر.

برنامه‌ریزی از طریق پورت ارتباط سریال با کامپیوتر پروگرامر صورت می‌پذیرد و برنامه سیستم عامل جهت اجرای سیکلیک برنامه‌ها و پردازش برنامه‌های دارای وقفه و نیز انجام امور مختلف شامل ارتباط با مدول‌های مختلف و ارتباط با پروگرامر به‌کار برده می‌شود.

در صفحه جلوی کارت CPU کلید RUN/STOP ، کلید Reset ، پورت ارتباط سریال و تعدادی علائم نوری جهت گزارش عملکرد کارت CPU قرار دارد.

۷-۴-۲- کارت ورودی دیجیتال (DI321)



ورودی دیجیتال DI321 سیگنال‌های رسیده از فرآیند را به سطح سیگنال‌های مورد قبول PLC تبدیل می‌کند. وضعیت سیگنال‌ها توسط دیودهای نوری سبز رنگ که در جلوی مدول نصب شده‌اند به راحتی قابل مشاهده است.

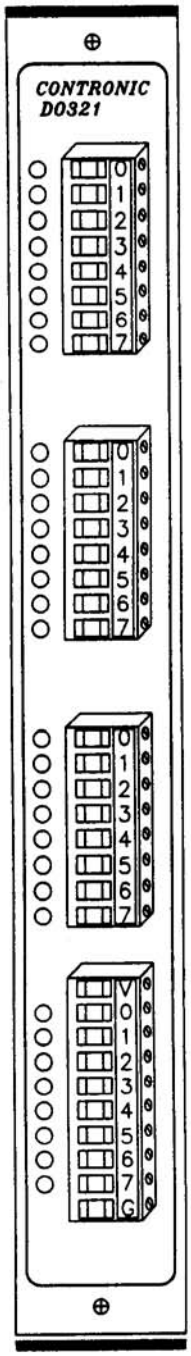
آدرس هر مدول ورودی توسط تعدادی سوئیچ DIP تنظیم می‌شود.

برای اتصال کابل‌های ورودی از کانکتورهای ۸ پین استفاده شده است.

سطح ولتاژ سیگنال‌های ورودی ۰ یا ۲۴ V DC بوده، به کمک قطعات زوج نوری از بقیه اجزای داخلی PLC ایزوله گردیده‌اند. بنابراین با هر گونه اتصال و یا اضافه ولتاژ در اتصالات، فرآیند نمی‌تواند آسیبی به سایر واحدهای PLC وارد کند.

DI321 دارای ۳۲ ورودی دیجیتال می‌باشد و جریان مصرفی هر ورودی در حالت فعال حدود ۱۰ mA می‌باشد.

۷-۴-۳- کارت خروجی دیجیتال (DO321)

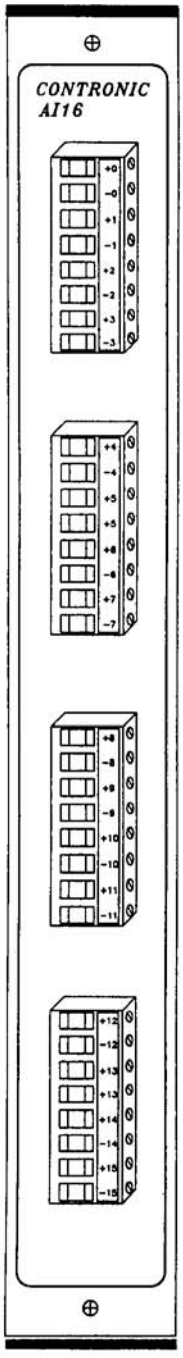


مدول‌های خروجی دیجیتال DO321 سطوح سیگنال‌های داخلی PLC را به سطح ۲۴ V DC می‌رساند. وضعیت سیگنال‌های خروجی توسط دیودهای نوری زرد رنگ که در جلوی مدول نصب شده‌اند به راحتی قابل مشاهده است.

آدرس هر مدول خروجی دیجیتال DO321 توسط تعدادی سوئیچ DIP تنظیم می‌شود.

سطح ولتاژ سیگنال‌های خروجی ۰ یا ۲۴ V DC بوده، از بقیه اجزای داخلی PLC ایزوله گردیده‌اند.

DO321 دارای ۳۲ کانال خروجی دیجیتال می‌باشد و جریان مصرفی معادل ۰/۵ آمپر را با سطح ولتاژ ۲۴ V DC تأمین می‌کند.



۷-۴-۴- کارت ورودی آنالوگ (AI16)

مدول ورودی آنالوگ AI16 سیگنال‌های پیوسته آنالوگ رسیده از فرآیند را به مقادیر دیجیتال تبدیل می‌کند. مقادیر دیجیتال حاصل، از آن پس می‌توانند توسط PLC پردازش شود.

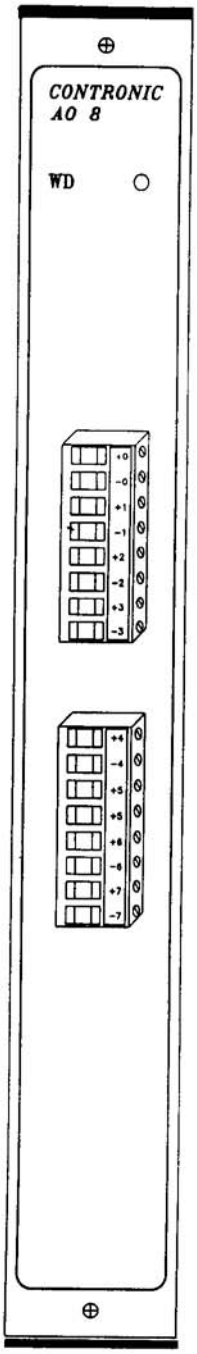
کاربرد مدول‌های آنالوگ در مواردی مانند:

- مونیتورینگ فرآیند
- اندازه‌گیری مقادیر فیزیکی مانند حرارت، فشار، سرعت و ...
- کنترل

می‌باشد.

AI16 دارای ۱۶ کانال بوده، هر دسته ۸ تایی آن به صورت مجزا نسبت به سایر اجزای PLC ایزوله می‌باشد.

ورودی‌های استاندارد ۱۰ V - ۰، ۲۰ mA - ۴، ۲۰ mA - ۰ و ۱۰ V ± - ۰ مستقیماً به ورودی‌های AI16 متصل می‌شوند.



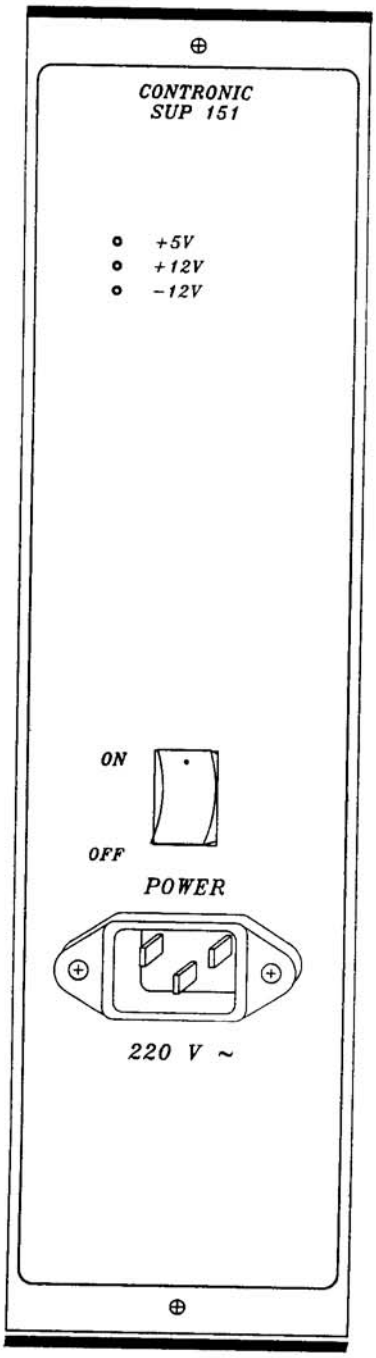
۷-۴-۵- کارت خروجی آنالوگ (AO 8)

مدول خروجی آنالوگ AO 8 مقادیر دیجیتال پردازش شده توسط PLC را به سیگنال‌های پیوسته (آنالوگ) مورد نیاز فرآیند تبدیل می‌کند.

آدرس مدول توسط تعدادی سوئیچ DIP بر روی آن تنظیم می‌شود.

هر مدول AO 8 دارای ۸ کانال خروجی ایزوله است بنابراین هرگونه اتصال و یا اضافه ولتاژ در اتصالات فرآیند نمی‌تواند آسیبی به سایر واحدهای PLC وارد کند.

سطح خروجی این مدول می‌تواند یکی از استانداردهای ۰ - ± ۱۰ V، ۰ - ۲۰ mA و یا ۴ - ۲۰ mA باشد.



۷-۴-۶- کارت منبع تغذیه (SUP 151)

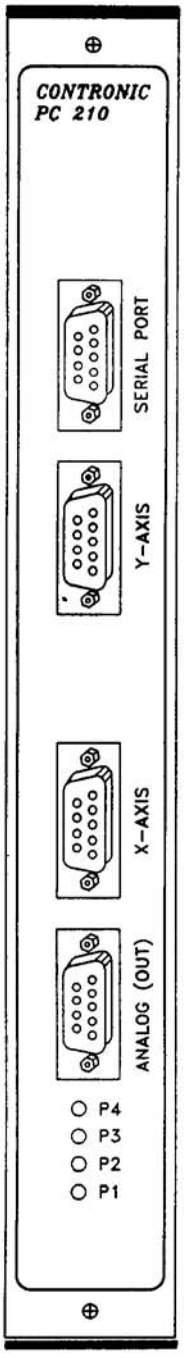
مدول منبع تغذیه SUP 151 با دریافت برق شهر سطوح ولتاژ مورد نیاز PLC را تأمین می‌کند.

از جمله ولتاژهای مورد نیاز ۵+ V DC و ۱۲ V DC ± می‌باشد.

طرح مدارات این کارت به صورت Switching بوده، از پایداری بسیار بالا و راندمان بیش از ۹۵٪ برخوردار می‌باشد.

مدارات ایمنی در این مدول بر سطح ولتاژهای تولید شده نظارت داشته، تنها در صورتی که سطوح ولتاژ مورد قبول باشند به سایر اجزای PLC اعمال می‌شوند.

تغییرات پیوسته برق شهر بین ۱۵۰ V تا ۲۸۰ V بر عملکرد این مدول تأثیری نداشته، در مقابل نوسانات آنی شدیدتر پایداری مناسبی دارد.

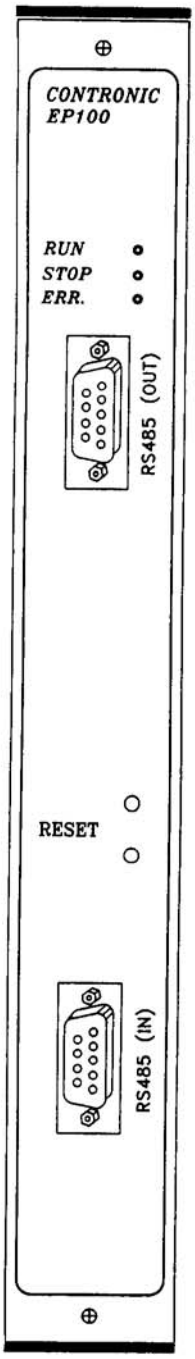


۷-۴-۷- کارت کنترل مکان (PC 210)

مدول PC 210 از جمله مدول‌هایی است که کلیه عملیات مربوط به کنترل مکان را خود مستقلاً انجام می‌دهد و مجهز به یک پردازنده مستقل می‌باشد.تنها ارتباط آن با مدول CPU مربوط به دریافت بعضی مقادیر و متغیرها و ارسال گزارشهای خود به CPU می‌باشد.

اطلاعات مربوط به مکان توسط رمزکننده‌های نوری (OPTICAL ENCODER) مربوط به دو محور به PC 210 وارد می‌شود. مدول PC 210 با ایجاد سیگنال آنالوگ خروجی به‌طور متناسب فرمانهایی را جهت تصحیح مکان به سیستم انتقالِ حرکت اعمال می‌نماید. این مدول جهت تطبیق حلقه‌های کنترلی تعدادی پارامتر را از PLC دریافت می‌کند. همچنین اطلاعاتی نظیر مختصات مقصد، سرعت پیشروی، شتاب و ... از جمله داده‌هایی است که باید از PLC به این مدول داده شود.

۷-۴-۸- کارت گسترش (EP 100)

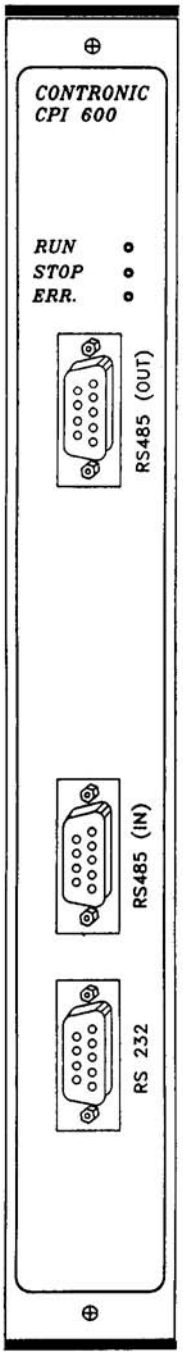


راک اصلی PLC (MAIN RACK) در دو نوع با دوازده و هفت شیار ساخته شده که دو شیار مربوط به منبع تغذیه و یکی مربوط به CPU 80C است. اگر از CPU 160C استفاده شود باید دو شیار برای CPU در نظر گرفته شود در نتیجه می‌تواند تعداد ورودی و خروجی بیشتری را به خوبی پشتیبانی نماید لیکن گاهی فرآیند تولید به تعداد بیشتری از ورودی‌ها و خروجی‌ها و سایر مدول‌های دیگر احتیاج دارد. در چنین مواردی باید از مدول‌های EP 100 استفاده کرد.

EP 100 مدولی است که MAIN RACK را به EXPANSION RACK های دیگر متصل می‌کند، در نتیجه می‌توان توانایی PLC را گسترش داد. EP 100 تمامی سیگنال‌های لازم جهت جمع‌آوری اطلاعات کارت‌های نصب شده در EXPANSION RACK را دارا می‌باشد.

عملکرد EP 100 توسط سیستم عامل PLC 500 پشتیبانی می‌شود و نیاز به هیچ کار اضافی توسط کاربر نمی‌باشد.

۷-۴-۹- کارت شبکه (CPI 600)



در خطوط تولید و فرآیندهای گسترده صنعتی، واحدهای مختلف کنترلی علاوه بر انجام عملیات محلی، با دیگر واحدهای کارخانه در تماس هستند.

هر واحد کنترلی PLC از چگونگی انجام مراحل فرآیند توسط واحدهای دیگر گزارش گرفته، یا به آنها گزارش ارسال می‌کند. با استفاده از این گزارشها، عملکرد هر واحد کنترلی می‌تواند متأثر از وضعیت عملکرد واحدهای دیگر (متناسب با برنامه) باشد.

مدول CPI 600 برای پاسخ به چنین نیازی طراحی شده است. این مدول با دارا بودن پردازنده و حافظه داخلی، توانایی ارتباط با PLC های دیگر در فرآیند و یا کامپیوتر مرکزی را دارا می‌باشد.

ارتباط استاندارد RS485 و سرعت 19600 baud rate شبکه ارتباطی بین PLCها را تشکیل می‌دهد.

در شبکه‌ای که بدین صورت شکل می‌گیرد یکی از PLCها و یا یک کامپیوتر مرکزی نقش اصلی (MASTER) و دیگر PLCها نقش فرعی (SLAVE) را به عهده خواهند داشت که در نتیجه کلیه مبادلات اطلاعات توسط MASTER سازماندهی خواهد شد.

فصل هشتم

برنامه‌نویسی به روش GRAPH 5

طراحی و برنامه‌نویسی یک سیستم کنترلی پیچیده با روشهای گفته شده در فصول قبل تا حدی برای برنامه‌نویس وقت‌گیر بوده و ممکن است مشکلاتی را برای وی بوجود آورد. علاوه بر این در صورت پیچیده‌تر شدن مراحل ترتیبی موجود در برنامه مثلاً پرش به بلوک‌های دیگر و انشعاب شاخه‌ای در بلوکها، مشکلات مذکور دو چندان خواهد شد.

همانگونه که می‌دانید ساختار برنامه‌های ترتیبی با بلوکهای ترتیبی یا مرحله‌ای پیاده‌سازی می‌شود. بنابراین می‌توان با تقسیم‌بندی و شکستن یک برنامه پیچیده به بلوک‌ها و مراحل کوچکتر تا حدی درصدد رفع مشکلات یاد شده برآمد.

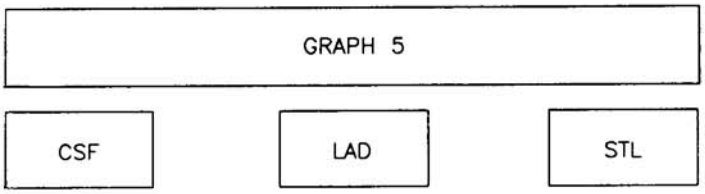
برای این کار روش برنامه‌نویسی GRAPH 5 که مختص زبان STEP 5 می‌باشد و به صورت بستهٔ نرم‌افزاری در اختیار برنامه‌نویس قرار می‌گیرد ابداع شده است. در این روش، برنامه‌نویس با استفاده از روش گرافیکی، برنامهٔ کنترلی را بسیار ساده و بدون مشکل دنبال می‌نماید.

در این روش کافی است که برنامهٔ ترتیبی به صورت گرافیکی ترسیم شده، سپس شرایط گذار (TRANSITION) و عملیات هر مرحله (ACTION) به یکی از سه روش STL، LAD یا CSF برنامه‌نویسی گردد. در روش مذکور نیز می‌توان برای هر مرحله یا برای کل برنامه MONITORING TIME و WAITING TIME تعریف نمود.

۸-۱- GRAPH 5 چیست؟

همانگونه که ذکر شد GRAPH 5 یک بستهٔ نرم‌افزاری است که جهت طراحی و برنامه‌نویسی کنترل‌کننده‌های ترتیبی به شیوهٔ گرافیکی مورد استفاده قرار می‌گیرد. این روش قابلیت‌های زبان STEP 5 را گسترش می‌دهد.

با استفاده از این روش می‌توان برنامه را به صورت مرحله به مرحله تقسیم نمود. برنامه نویسی هر یک از مراحل و شرایط گذار نیز به هر یک از روشهای STL ، CFS یا LAD امکان‌پذیر است.



شکل ۸-۱ : نمایش شماتیکی ارتباط GRAPH 5 با سه روش برنامه‌نویسی STL ، LAD و CSF

۸-۲- قابلیت‌های روش GRAPH 5

از قابلیت‌های مهم این روش می‌توان موارد زیر را نام برد:

- طراحی (PLANNING AND DESIGN)
- برنامه‌نویسی (PROGRAMMING)
- امکان ارائهٔ توضیحات در هر مرحله و برای هریک‌از شرایط گذار (DOCUMENTATION)
- امکان تست و اجرای برنامه (TESTING)

در برنامه‌نویسی با این روش FB های استاندارد مورد استفاده قرار می‌گیرند.

یک برنامه کنترل ترتیبی به روش GRAPH 5 در دو بخش نوشته می‌شود:

۱- OVERVIEW LEVEL

۲- ZOOM – IN LEVEL

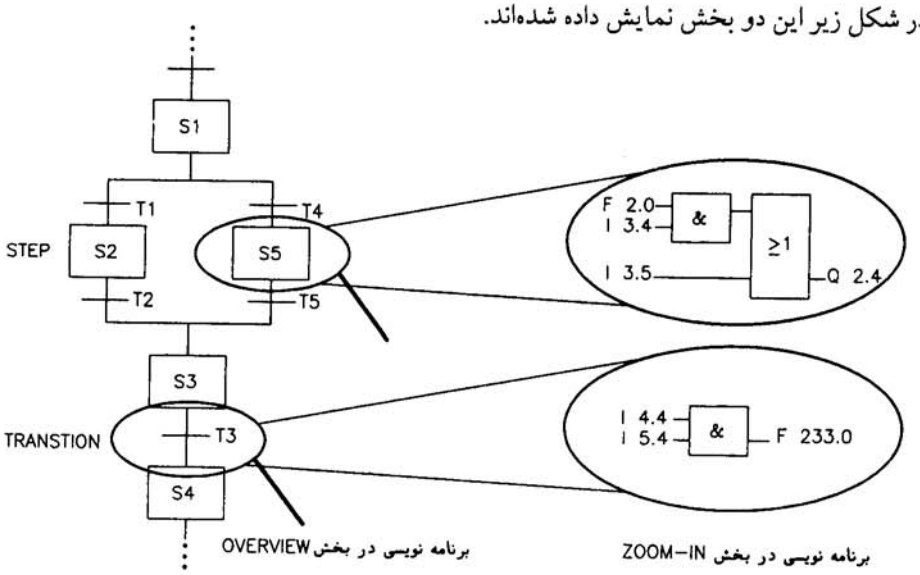
۸-۲-۱- OVERVIEW LEVEL

در این قسمت ساختار ترتیبی برنامه مشخص می‌گردد. مراحل و شرایط گذار هر مرحله،

شاخه‌ها، ارتباطات همزمان و غیرهمزمان و پرشها در این بخش مشخص می‌گردند. در این قسمت می‌توان برای هر مرحله در صورت نیاز MONITORING TIME و WAITING TIME تعریف نمود.

۸-۲-۲-ZOOM – IN LEVEL

در این قسمت محتویات برنامه‌ایِ هر مرحله و شرایط‌گذار (شرایط عبور از یک مرحله به مرحله دیگر) مشخص می‌گردد. در این بخش، عملیات و شرایط‌گذار هر مرحله برنامه‌نویسی می‌شوند. در شکل زیر این دو بخش نمایش داده شده‌اند.



شکل ۸-۲ : دو بخش ZOOM – IN و OVERVIEW

علاوه بر قابلیت‌های طراحی و برنامه‌نویسی در روش GRAPH 5 می‌توان برنامهٔ نوشته شده را تست‌نمود. همچنین‌می‌توان‌درمورد هر یک از مراحل و شرایط‌گذار در هر دو بخش OVERVIEW و ZOOM – IN توضیح داد.

توضیحات ارائه شده می‌تواند به صورت‌های زیر باشد:

- توضیح در مورد هر مرحله و شرط‌گذار

- توضیح در مورد عنوان SEGMENT ها
- توضیح در مورد هر سطر برنامه (STATEMENT)

در این روش همچنین می‌توان به جای عملوندها از سمبل‌های شناخته شده و مأنوس نیز استفاده کرد. به عنوان مثال فرض کنید در یک برنامهٔ کنترلی ورودی I 21.7 کلید START باشد. به جای وارد نمودن دستور A I 21.7 می‌توان از دستور A START استفاده نمود.

۸-۳- روش برنامه‌نویسی GRAPH 5

همانگونه که ذکر شد در این روش برنامهٔ کنترلی به مراحل جداگانه‌ای تقسیم‌بندی می‌شود که اجرای هر مرحله منوط به برقراری شرایط گذار همان مرحله می‌باشد. در ادامهٔ بحث به توضیح در مورد اصطلاحات مرحله (STEP) و شرط گذار (TRANSITION) می‌پردازیم.

مرحله (STEP) : در این بخش، عملیات قسمت‌های کوچکتر برنامه مورد بررسی قرار می‌گیرد. در برنامه‌نویسی این قسمت در بخش ZOOM – IN حتماً بایستی از یک فلگ خاص یعنی F 233.0 به عنوان سیگنال فعال‌ساز (ENABLE) در مورد SB های استفاده شده در روش GRAPH 5 استفاده شود. مقدار این فلگ در حالتی که STEP فعال باشد "1" خواهد بود.

شرط گذار (TRANSITION): در این بخش، شرایط فعال‌سازی مرحله یا مراحل بعد مشخص می‌شود. برنامه‌نویسی این بخش نیز در ZOOM – IN و باز با استفاده از F 233.0 جهت فعال‌سازی مرحله یا مراحل بعدی انجام می‌گیرد.

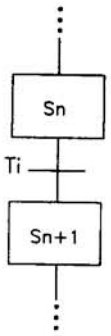
در روش GRAPH 5 استفاده از FB های استاندارد جهت راه‌اندازی و بکارگیری مدهای برنامهٔ کنترلی ترتیبی الزامی است. در این روش ابتدا FB ها را فراخوانی نموده، سپس با تخصیص مقادیر پارامترها و ... برنامه‌نویسی را دنبال می‌کنیم.

۸-۴- المانها و عناصر موجود در روش GRAPH 5

برنامهٔ نوشته شده در این روش شامل مراحل و شرایط گذار می‌باشد. به صورتی که:

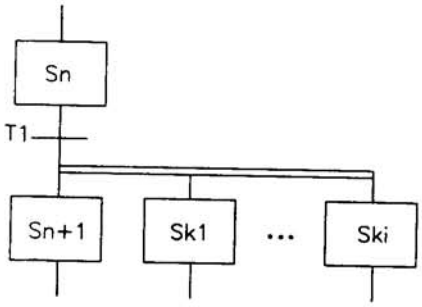
- هر شرط گذار بایستی پس از اجرای یک مرحله قرار گیرد.
- هر مرحله نیز باید پس از یک شرط گذار قرار داشته باشد.

حال المانهای استفاده شده در این روش را مورد بررسی قرار می‌دهیم.



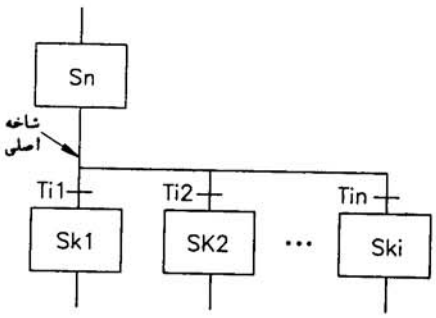
۱ - دنبالهٔ خطی (LINEAR SEQUENCE) : همانگونه که در شکل مقابل ملاحظه می‌کنید روند اجرای برنامه بدین صورت است که ابتدا مرحلهٔ Sn انجام شده، سپس مرحلهٔ Sn+۱ انجام خواهد شد. به هنگام تغییر وضعیت Ti (شرط گذار)، مرحلهٔ Sn+۱ فعال و مرحلهٔ Sn غیرفعال می‌گردد. در دنباله‌های خطی، مراحل یکی پس از دیگری و در صورت برقرار شدن شرایط گذار انجام می‌شوند. توجه داشته باشید که هر مرحله باید پس از یک شرط گذار بوده و هر شرط گذار نیز باید پس از یک مرحله قرار گرفته باشد.

۲ - شاخه‌های همزمان (SIMULTANEOUS BRANCHES) : در این حالت در صورت برقراری شرط گذار، چندین مرحله به طور همزمان با یکدیگر فعال می‌شوند.



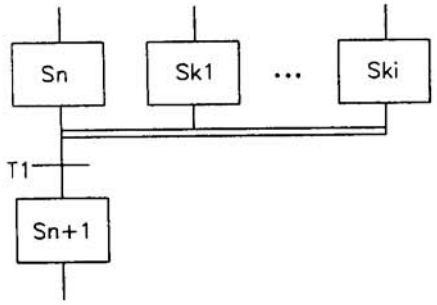
در شکل مقابل به محض اینکه T1 تغییر وضعیت دهد مراحل Sn+۱ و Ski , ... , Skl فعال شده و Sn غیرفعال خواهد شد. (این عمل متناظر با انجام عمل AND می‌باشد) همانگونه که ملاحظه می‌کنید شرط گذار برای تمامی مراحل بعدی مشترک است.

۳ - شاخه‌های غیرهمزمان یا متناوب (ALTERNATIVE BRANCHES) : در شکل مقابل به محض فعال شدن مرحلهٔ Sn ، تمامی شرایط گذار نشان داده شده یعنی Tin , ... , Ti1 فعال خواهند شد. توجه داشته باشید که در برنامه‌نویسی بهتر است در صورت امکان، شرایط مربوط به هر مرحله انحصاری باشد. در صورتی که شرایط گذار تمام مراحل، همزمان با یکدیگر فعال شده باشند اولویتِ انجام با مرحله‌ای خواهد بود که به شاخهٔ اصلی نزدیکتر باشد.

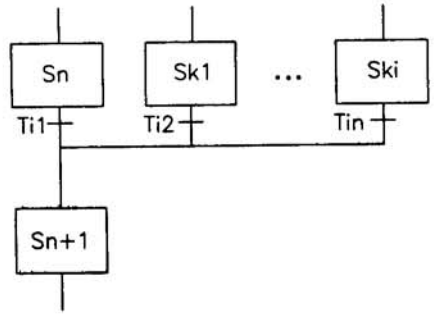


۴ - همزمانی مراحل در اتصال (SYNCHRONIZATION) : شاخه‌های موازی موجود در برنامه می‌توانند با یکدیگر همزمان شده و به یکدیگرمتصل شوند. روند اجرای برنامهٔ

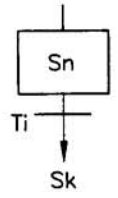
کنترلی نشان داده شده در شکل بدین ترتیب است که ابتدا تمامی مراحل Sn و Sk1 , ... , Ski انجام شده، سپس مرحلهٔ Sn+۱ انجام خواهد شد. در این حالت شرط‌گذار در صورتی معتبر خواهد شد که تمام مراحل مذکور انجام شده باشند. سپس با تغییر وضعیت شرط‌گذار Tl، این مراحل غیرفعال شده و Sn+۱ فعال می‌شود.



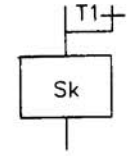
۵- اتصال غیرهمزمانی یا متناوب شاخه‌ها (ALTERNATIVE JUNCTION) : در شکل مقابل شاخه‌های غیرهمزمان به یکدیگر متصل شده‌اند. در این حالت مرحلهٔ Sn+۱ در صورتی انجام خواهد شد که حداقل یکی از شرایط‌گذار Ti1 , ... , Tin تغییر وضعیت دهد. به عنوان مثال در صورتی که شرط‌گذار Til سریعتر از بقیهٔ شرایط‌گذار برقرار گردد روند اجرای برنامه به صورت Sn+۱ ⟵ Skl خواهد بود.



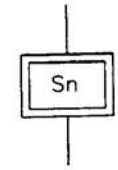
۶- پرش (JUMP) : مطابق شکل مقابل پس از انجام مرحلهٔ Sn مرحلهٔ Sk انجام خواهد شد که Sk می‌تواند هر یک از مراحل کنترل باشد البته به شرط $k \neq n$. پس از تغییر وضعیت شرط‌گذار Tl مرحلهٔ Sn غیرفعال و Sk فعال می‌گردد.

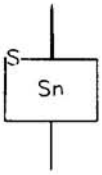


۷- اتصال به یک پرش مرحله‌ای (JUNCION OF A JUMP) : در شکل روبرو پرش به مرحلهٔ Sk در صورتی انجام خواهد شد که شرط‌گذار Tl تغییر وضعیت دهد. همانگونه که ملاحظه می‌شود هیچگونه ارتباط فیزیکی بین شرط‌گذار Tl و Sk وجود ندارد.



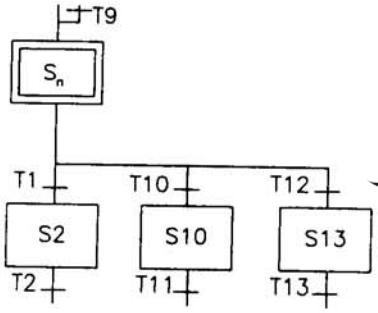
۸- مرحلهٔ آغازین / انتخابی (INITIAL/SELECTIVE STEP) : مرحلهٔ آغازین یا ابتدایی در ابتدای دنبالهٔ کنترل قرار گرفته و برای اجرای این مرحله نیاز به شرایط‌گذار نیست. به تفاوت شکل روبرو در این مرحله با مراحل دیگر توجه کنید.





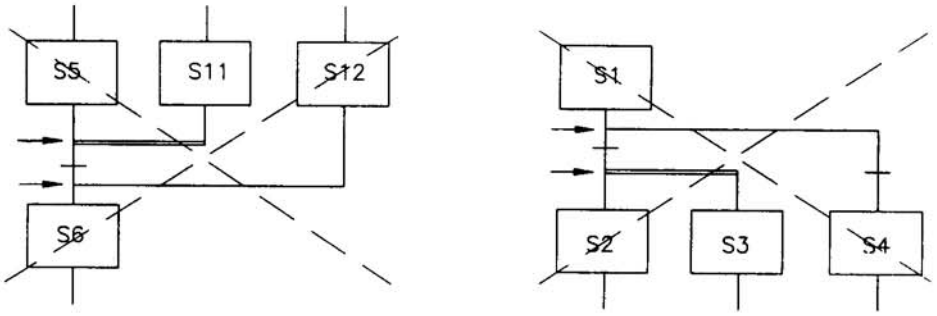
حال به بررسی مفهوم مرحلهٔ انتخابی می‌پردازیم. همانگونه که گفته شد روند اجرای برنامه‌های ترتیبی به گونه‌ای است که مراحل یکی پس از دیگری انجام می‌شوند. در برخی از پروسه‌ها می‌توان بعضی از مراحل میانی را حذف نمود. در اجرای این برنامه‌ها پردازنده از مراحل انتخاب شده توسط استفاده‌کننده پرش نموده و آنها را نادیده فرض می‌کند. به حرف S که در گوشهٔ سمت چپ مرحلهٔ انتخابی وجود دارد توجه کنید.

نمونه‌ای از برنامه‌های ترتیبی در شکل زیر آمده است. در این برنامه سعی شده تا از تمامی المانهای مذکور استفاده گردد. توضیح در مورد روند اجرای پروسه به عنوان تمرین به خوانندگان واگذار می‌شود.



شکل ۸-۳: نمونهٔ یک برنامه ترتیبی با استفاده از روش GRAPH 5

توجه داشته باشید که داشتن دو شاخهٔ بدون وجود مرحله‌ای بین آنها مجاز نیست. این مطلب در مورد اتصال دو شاخه نیز صادق است در شکل ۸-۴ این مطلب به وضوح نشان داده شده است.



شکل ۸-۴ : حالات غیرمجاز در مورد دو شاخهٔ غیرهمزمان بدون وجود مرحلهٔ واسطه و همچنین اتصال دو شاخه

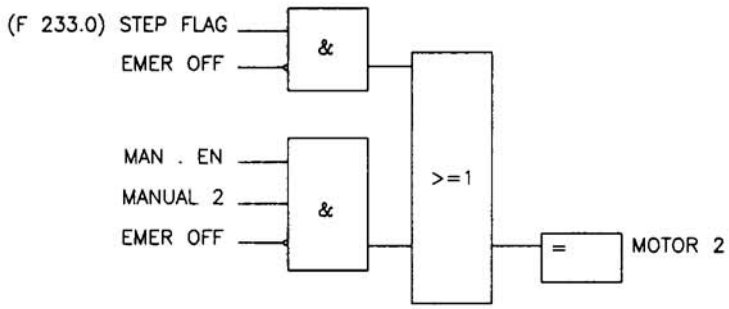
۸-۵- برنامه‌نویسی در بخش ZOOM – IN

تا اینجا در مورد برنامه‌نویسی در بخش OVERVIEW به بحث پرداختیم. اکنون برنامه‌نویسی در قسمت ZOOM – IN را مورد بررسی قرار می‌دهیم. همانگونه که ذکر شد مراحل مختلف یک برنامهٔ ترتیبی و شرایط‌گذار در این قسمت برنامه‌نویسی می‌شوند. برنامه‌نویسی در این بخش می‌تواند به یکی از سه روش STL ، LAD یا CSF صورت گیرد.

۸-۶- برنامه‌نویسی مراحل مختلف برنامه (PROGRAMMING STEPS)

در نوشتن برنامهٔ هر مرحله می‌توان از فرامینی نظیر بارگذاری و انتقال (L و T) ، فرمان فعال نمودن ACTUATOR ها ، فرمان شروع به کار تایمرها، شمارنده‌ها و ... استفاده نمود. همچنین در برنامه‌نویسی مراحل مختلف می‌توان FB ها را نیز فراخوانی کرد.

در ادامه، برنامه‌ای به روش CSF آمده که در این برنامه به جای ورودی‌ها و خروجی‌ها متعارفی که تاکنون به بحث در مورد آنها می‌پرداختیم از سمبل‌های مأنوس و شناخته شده استفاده شده است.



۸-۷- برنامه‌نویسی شرایط گذار (PROGRAMMING TRANSITIONS)

همانگونه که ذکر شد شرایط‌گذار، شرایط لازم جهت فعال نمودن یک مرحله یا مراحل مختلف یک برنامهٔ ترتیبی می‌باشند. در برنامه‌نویسی این شرایط مسائلی از قبیل شرایط فعال نمودن هر مرحله یا مراحل بعد باید مدنظر قرار گیرند. در ادامه، یک نمونه برنامهٔ شرط‌گذار به روش STL آمده است.

```
: A    SENSOR 4
: AN   LIMIT 2
: =    STEP FLAG
```

۸-۸- در نظر گرفتن MONITORING TIME و WAITING TIME

در فصل پنجم خاطر نشان کردیم که برای هر یک از مراحل مختلف یک برنامه می‌توان WAITING TIME , MONITORING TIME تعریف نمود. در برخی از پروسه‌ها نیز می‌توان برای کل پروسه این دو زمان را در نظر گرفت.

۸-۸-۱- زمان مراقبت یا نظارت TM (MONITORING TIME)

TM زمان در نظر گرفته شده برای اجرای یک مرحله با تولرانس محدود در صورت عدم اجرای آن مرحله در زمان مقرر می‌باشد. در صورت سپری شدن TM و عدم اجرای مرحلهٔ بعدی، خطای TIME OUT توسط PLC صادر خواهد شد. به عبارت دیگر در TM زمان اجرای یک مرحله اندازه‌گیری شده و در صورت پیش آمدن هرگونه خطا و طولانی شدن مدت زمان مذکور تایمر مورد نظر فعال شده، واکنش مطلوب را انجام خواهد داد.

۸-۸-۲- زمان انتظار TW (WAITING TIME)

TW زمان تأخیر برای عبور از یک مرحله به مرحلهٔ دیگر است و اجرای مرحلهٔ بعدی هنگامی آغاز خواهد شد که این زمان پایان یافته باشد. در حقیقت زمان TW معرف یک نوع تأخیر برای هر مرحله است.

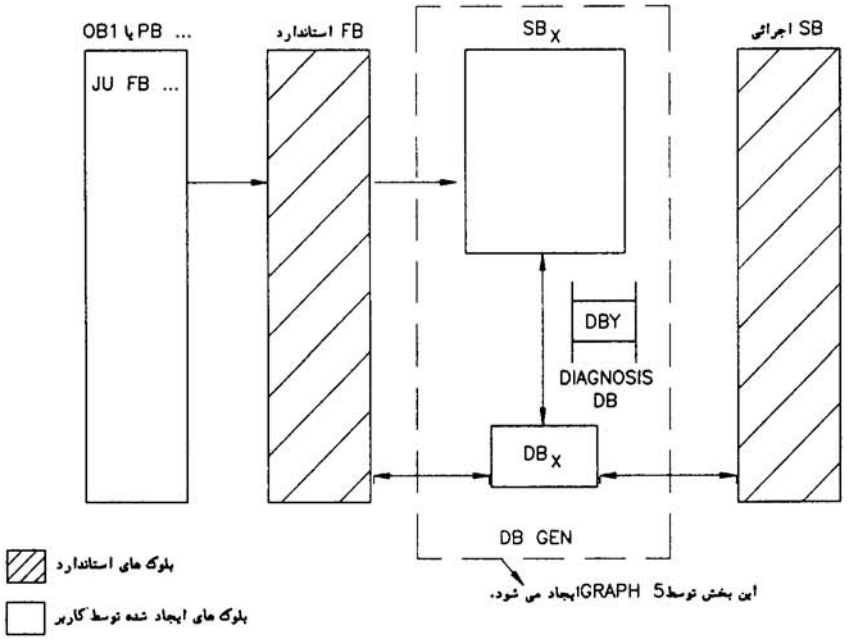
۸-۹- ارائهٔ توضیحات در روش GRAPH 5 (COMMENTS)

در روش GRAPH 5 می‌توان در مورد هر مرحله و شرط‌گذار و همچنین در مورد هر SEGMENT و ... در هر دو بخش ZOOM – IN و OVERVIEW توضیح داد.

۸-۱۰- ساختار برنامه در روش GRAPH 5

برای برنامه‌نویسی به این روش لازم است که برنامه‌نویس یک SB و DB ایجاد نموده و با فراخوانی یکی از بلوکهای FB استاندارد و تخصیص پارامتر به عنوان ورودی‌ها و خروجی‌های آن، برنامه‌نویسی را دنبال نماید. در اجرای یک برنامه به روش GRAPH 5 یک SB دیگر به نام

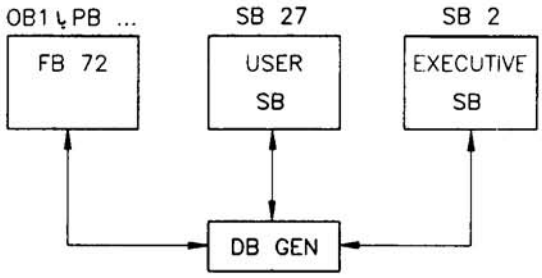
SB اجرایی (EXECUTIVE SB) نیز وجود دارد که به همراه فراخوانی FB در PLC بارگذاری می‌شود. هر یک از FB ها دارای SB اجرایی خاص خود می‌باشند. به عنوان مثال به هنگام فراخوانی FB 72 حتماً SB اجرایی شمارهٔ ۲ یعنی SB 2 بارگذاری می‌شود. DB های ایجاد شده توسط کاربر یعنی USER DB و DIAGNOSIS DB نیازی به برنامه‌نویسی نداشته و توسط دستور DB GEN(DB GENERATE) به PLC بارگذاری می‌شوند. در شکل ۸-۵ چگونگی ساختار برنامه در روش GRAPH 5 نشان داده شده است.



شکل ۸-۵: بلوک‌های استاندارد و بلوک‌های ایجاد شده توسط کاربر در روش GRAPH 5

روند برنامه‌نویسی بدین صورت خواهد بود که کاربر ابتدا یک SB مثلاً SB 27 ایجاد نموده و در آن تمام مراحل، شرایط‌گذار و ... را در بخش OVERVIEW مشخص می‌کند. سپس یک DB با همان شمارهٔ SB یعنی DB 27 ایجاد می‌کند. به SB و DB ایجاد شده توسط استفاده‌کننده USER SB و USER DB گویند.

فرض کنید که برای برنامه‌نویسی یک پروسه احتیاج به فراخوانی بلوک استاندارد FB 72 داشته باشیم. در این صورت PLC به صورت اتوماتیک SB 2 که همان SB اجرایی مختص FB 72 می‌باشد را بارگذاری می‌کند. در شکل ۸-۶ این مطلب نشان داده شده است.



شکل ۸-۶: ارتباط بین FB استاندارد، SB اجرایی و USER SB با DB GEN

۸-۱۱- لیست بلوک‌های لازم در برنامه‌نویسی به روش GRAPH 5

بلوک‌های FB استاندارد و همچنین SB های اجرائی متناظر با آنها در زیر آمده است:

FB 70: برای دنبالهٔ اصلی (MAIN SEQUENCE)

FB 71: برای دنبالهٔ ثانویه یا فرعی البته در صورت لزوم (SECONDARY SEQUENCE)

SB 0: بلوک اجرائی متناظر با FB های مذکور (EXECUTIVE BLOCK)

FB 72: برای دنبالهٔ اصلی

FB 74: برای روشهای عملیاتی در صورت لزوم (OPERATING MODES)

SB 2: بلوک اجرائی متناظر با FB های مذکور

FB 73: برای دنبالهٔ اصلی

FB 74: برای مدهای عملیاتی

SB 3: بلوک اجرائی متناظر با FB های مذکور

بلوک‌های دیگر مورد استفاده در روش GRAPH 5 به صورت زیر می‌باشند.

X SB: بلوک دنباله‌ای ایجاد شده توسط کاربر برای دنبالهٔ اصلی (USER SB)

X DB: بلوک اطلاعاتی ایجاد شده توسط کاربر (USER DB)

Y DB: (DIAGNOSIS DB)

و بلوکهای دیگری از قبیل : DB ، FB ، PB و OB

به علت اینکه موارد استفادهٔ FB 72 بسیار زیاد بوده و در برنامه‌نویسی اکثر پروسه‌ها به روش GRAPH 5 از این بلوک استفاده می‌شود قبل از برنامه‌نویسی، در مورد این بلوک و چگونگی وارد کردن پارامترهای آن توضیحاتی ارائه می‌کنیم.

این بلوک جهت اجرای برنامه‌های دنباله‌ای که دارای حالت‌های START/STOP ، AUTO/HAND ، SINGLE STEP MODE و ... می‌باشند فراخوانی و استفاده می‌شود. برای فراخوانی این بلوک ابتدا لازم است که در بلوک OB 1 یا یک بلوک PB دیگر مثلاً PB 40 دستور پرش به FB 72 را صادر کنیم در زیر پارامترها و توضیح در مورد هر یک آمده است:

```
PB 40
        :  JU   FB    72
NAME    :  ABC
SBNR    :  KF   27
AUS     :  I    16.0
A/H     :  I    16.1
TIPP    :  I    16.2
T+1     :  I    16.3
QIT     :  I    16.4
STO     :  Q    8.0
        :  BE
```

در این سطر کاربر به دلخواه نامی برای بلوک FB 72 انتخاب می‌کند.

استفاده‌کننده شمارهٔ USER SB را با فرمت KF وارد می‌نماید.

این ورودی معرف کلید START/STOP بوده و در صورتی که "1" باشد سیستم متوقف خواهد شد.

این ورودی معرف کلید AUTO/HAND بوده و در صورتی که "1" باشد سیستم به صورت AUTO عمل خواهد نمود.

این ورودی معرف کلید SINGLE STEP MODE بوده و در صورتی که "1" باشد سیستم، مرحله به مرحله عمل خواهد نمود.

این ورودی کلید (STEP + 1) بوده و در صورتی که "1" باشد مرحلهٔ بعدی انجام خواهد شد.

در صورت بروز اشکال و رفع شدن آن برای راه‌اندازی مجدد با فعال نمودن این ورودی، سیستم را RESET می‌کنیم.

در صورت بروز هرگونه اشکال این بیت خروجی "1" خواهد شد و نشان‌دهندهٔ وجود ERROR در سیستم می‌باشد.

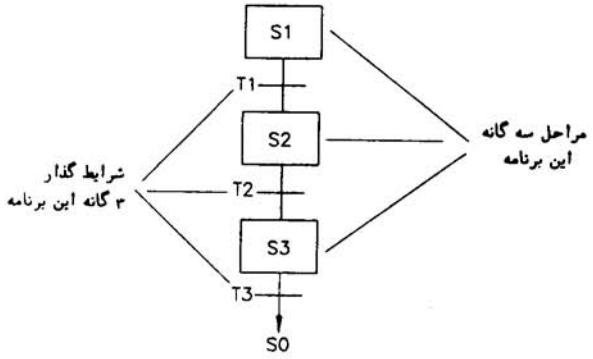
در صورتی که به یاد داشته باشید در فصل چهارم، برنامهٔ کنترل چراغ راهنمایی را بررسی

نمودیم. حال قصد آن داریم تا به روش GRAPH 5 این برنامه را بازنویسی کنیم.

مثال ۸-۱: برنامهٔ کنترل چراغ راهنمایی را به روش GRAPH 5 بازنویسی کنید.

این پروسه شامل ۳ مرحله و ۳ شرط‌گذار می‌باشد. پس در یک SB مثلاً SB 27 این مراحل را در بخش OVERVIEW مشخص می‌کنیم.

SB 27



سپس در بخش ZOOM – IN به برنامه‌نویسی شکل هر یک از مراحل و شرایط‌گذار می‌پردازیم:

STEP 1/1

مرحلهٔ اول : روشن ماندن چراغ قرمز مخصوص اتومبیل
و چراغ سبز مخصوص عابر پیاده به مدت ۶۰ ثانیه

```
: A     F    233.0
: L     KT   60.2
: SP    T    1
: NOP   0
: NOP   0
: NOP   0
: A     T    1
: =     Q    8.0
: =     Q    8.1
: BE
```

STEP 2/1

مرحلهٔ دوم: روشن ماندن چراغ زرد مخصوص اتومبیل
و عابر پیاده به مدت ۵ ثانیه

```
: A     F    233.0
: L     KT   5.2
: SP    T    2
: NOP   0
: NOP   0
: NOP   0
: A     T    2
: =     Q    8.2
: =     Q    8.3
: BE
```

STEP 3/1

مرحلهٔ سوم : روشن ماندن چراغ سبز مخصوص اتومبیل
و چراغ قرمز مخصوص عابر پیاده به مدت ۶۰ ثانیه

```
: A     F    233.0
: L     KT   60.2
: SP    T    3
: NOP   0
: NOP   0
: NOP   0
: A     T    3
: =     Q    8.4
: =     Q    8.5
: BE
```

حال شرایط گذار را تعیین می‌کنیم:

شرط گذار T1 TRANSITION 1/1

شرط عدم روشن بودن چراغ قرمز مخصوص اتومبیل

یا به عبارت دیگر اتمام مرحلهٔ اول : : AN Q 8.0

: BE

شرط گذار T2 TRANSITION 2/1

شرط عدم روشن بودن چراغ زرد مخصوص اتومبیل

یا به عبارت دیگر اتمام مرحلهٔ دوم: : AN Q 8.2

: BE

شرط گذار T3 TRANSITION 3/1

شرط عدم روشن بودن چراغ سبز مخصوص اتومبیل

یا به عبارت دیگر اتمام مرحلهٔ سوم : AN Q 8.4

: BE

پس از تعریف مراحل و شرایط گذار با ایجاد یک بلوک DB (USER DB) تعداد مراحل و شرایط گذار را وارد می‌کنیم. همانطور که گفته شد شمارهٔ این DB نباید با شمارهٔ USER SB یکسان باشد. زیرا PLC خود یک بلوک DB با شمارهٔ USER SB ایجاد می‌کند. سپس در بلوک OB 1 دستورات زیر را وارد نموده و برنامه را اجرا می‌کنیم.

```
OB 1
    : JU    PB   40
    : BE
```

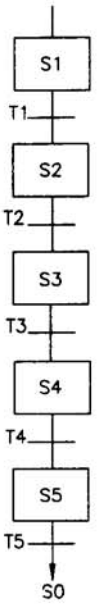
حال در این قسمت قصد داریم تا پروسهٔ پرس هیدرولیک فصل پنجم را با روش GRAPH 5 بازنویسی کنیم. در اینجا برای سهولت کار تنها حالت AUTO را بررسی می‌کنیم.

مثال ۸-۲: حالت عملکرد AUTO مربوط به پرس هیدرولیک مثال ۵-۶ را به روش GRAPH 5 برنامه‌نویسی کنید.

در صورتی که به خاطر داشته باشید پروسهٔ حالت AUTO شامل ۵ مرحله بود. البته این ۵ مرحله بدون در نظر گرفتن SEGNMENT 6 می‌باشد. این SEGMENT جهت در نظر گرفتن WAITING TIME به برنامهٔ اصلی اضافه شده بود.

ابتدا در یک بلوک SB مثلاً SB 50، مراحل و شرایط گذار را در قسمت OVERVIEW به صورت زیر وارد می‌کنیم.

SB 50



سپس به برنامه‌نویسی هر یک از مراحل و شرایط گذار در بخش ZOOM – IN می‌پردازیم:

```
STEP   1/1
      : A   F   233.0      سیگنال ENABLE
      : A   I   21.0       فعال بودن سنسور S8 یا عقب بودن بازوی هیدرولیک ۱
      : A   I   21.1       فعال بودن سنسور S10 یا عقب بودن بازوی هیدرولیک ۲
      : A   I   21.2       فعال بودن سنسور S13 یا عقب بودن بازوی هیدرولیک ۳
      : =   F   4.0        فلگ مربوط به INITIAL STATE
      : BE
```

```
STEP  2/1
        : A   F   233.0
        : =   Q   14.0          فرمان فعال نمودن Y1
        : BE

STEP  3/1
        : A   F   233.0
        : =   Q   14.1          فرمان فعال نمودن Y2
        : BE

STEP  4/1
        : A   F   233.0
        : =   Q   14.2          فرمان فعال نمودن Y3
        : BE

STEP  5/1
        : A   F   233.0
        : =   Q   14.3          فرمان فعال نمودن Y4
        : BE
```

برنامه‌نویسی شرایط‌گذار :

```
TRANSITION    1/1
        : A   F   4.0           وجود شرط INITIAL STATE
        : BE

TRANSITION    2/1
        : A   I   21.3          شرط فعال بودن سنسور S9 یا قرارگرفتن هیدرولیک ۱ در
        : BE                    منتهی‌الیه سمت راست یا به عبارت دیگر شرط پایان یافتن مرحلهٔ اول

TRANSITION    3/1
        : A   I   21.4          شرط فعال بودن سنسور S11 یا اتمام مرحلهٔ دوم
        : BE
```

```
TRANSITION    4/1
        : A   I    21.5        شرط فعال بودن سنسور S12 یا اتمام مرحلهٔ سوم
        : BE
```

```
TRANSITION    5/1
        : A   I    21.6        شرط فعال بودن سنسور B1 یا اتمام مرحلهٔ چهارم
        : BE
```

با توجه به اینکه حالات START/STOP و AUTO/HAND در این شبیه‌سازی با چهار ورودی (برای هر حالت، یک ورودی) تعریف شده است بنابراین از دو فلگ برای این منظور استفاده می‌کنیم. F 150.0 که فلگ ناپایدار بوده و برای حالت START/STOP در نظر گرفته می‌شود و دیگری F 20.0 که یک فلگ پایدار بوده و برای حالت AUTO/HAND مورد استفاده قرار می‌گیرد.

سپس در بلوک PB 25 ، FB 72 را فراخوانی نموده، به پارامترهای آن ورودی‌ها و خروجی‌های پروسه را اختصاص می‌دهیم. در OB 1 نیز دستوراتی را به صورت زیر وارد می‌کنیم.

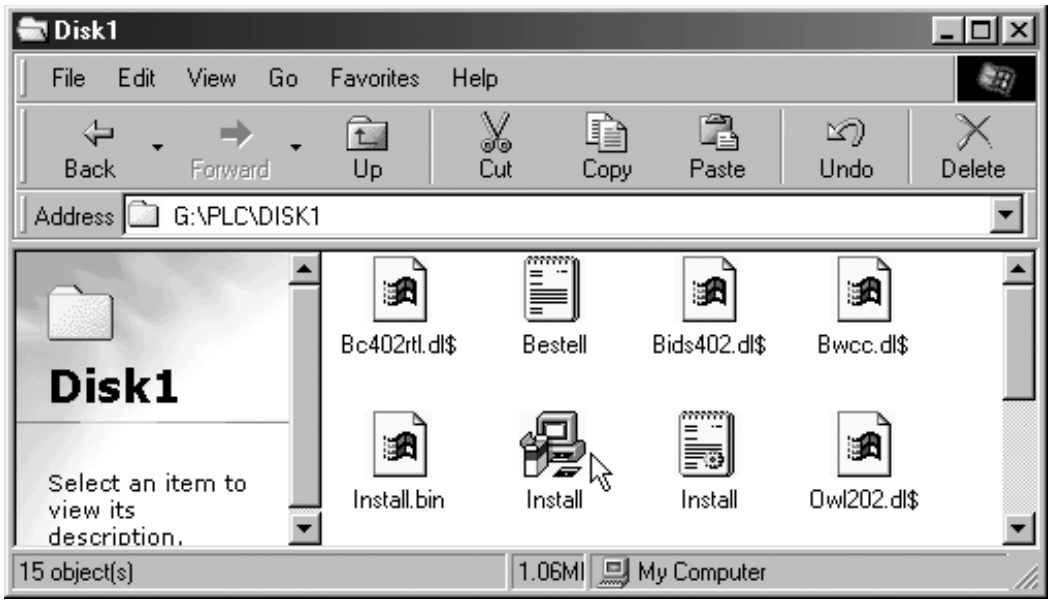
```
OB 1
        : A     I     21.7
        : S     F     150.0
        : A     I     22.0
        : R     F     150.0
        : A     I     22.1
        : S     F     20.0
        : A     I     22.2
        : R     F     20.0
        : A     F     20.0
        : JU    PB    25
        : BE
```

برنامه‌نویسی بلوک PB 25 به صورت زیر می‌باشد.

```
PB  25
        : JU    FB    72
NAME    : ABC
SNBR    : KF    +50
AUS     : F     150.0
A/H     : F     20.0
TIPP    : I     16.0
T+1     : I     16.1
QIT     : I     16.3
STO     : Q     8.0
        : BE
```

ضمیمه ۶: روش نصب و راه‌اندازی نرم افزار شبیه‌ساز PLC

نرم‌افزار s5w-demo ساختار سخت‌افزار و نرم‌افزار یک سیستم PLC زیمنس را شبیه‌سازی می‌کند. برای نصب این نرم‌افزار استفاده از رایانه‌های شخصی 486 به بالا توصیه می‌شود. نرم‌افزار مذکور در حدود ۳ مگابایت از ظرفیت دیسک سخت را اشغال می‌کند. برای نصب این نرم‌افزار لوح فشردهٔ همراه این کتاب را در داخل گردانندهٔ CD قرار دهید و مطابق شکل ۱، فایل Install را از مسیر G:\PLC\Disk1 اجرا کنید.



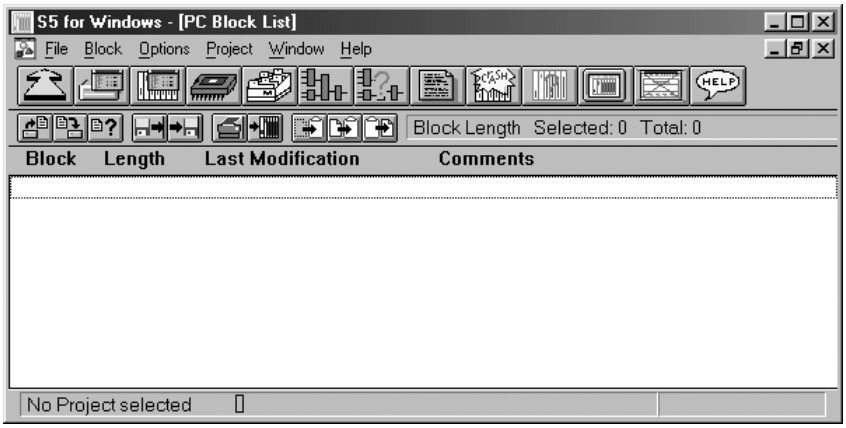
شکل ۱: نحوهٔ نصب نرم‌افزار s5w-demo

این نرم‌افزار بر روی ۳ عدد فلاپی ذخیره شده است. در هنگام نصب، به جای قرار دادن فلاپی‌های شماره ۲ و ۳، آدرس قرار گرفتن آنها را از لوح فشرده تعیین کنید.

این نرم‌افزار به طور کامل بر روی CD ذخیره نشده است. در مرحلهٔ پایانی نصب نرم‌افزار، سیستم عامل رایانه در یک پنجرهٔ محاوره‌ای اعلام می‌کند که یکی از مثال‌های موجود در مسیر G:\plc\disk3\example\example.s5 بر روی لوح فشرده وجود ندارد. بنابراین در مرحلهٔ پایانی بر روی کلید Cancel در پنجرهٔ محاوره‌ای کلیک کنید. پس از این مرحله، سیستم عامل رایانه پیغامی مبنی بر عدم نصب کامل این نرم‌افزار اعلام می‌کند. شایان ذکر است که عدم وجود این مثال، در عملکرد نرم‌افزار شبیه‌ساز تأثیری ندارد؛ بنابراین در مورد صحت عملکرد این نرم‌افزار نگران نباشید.

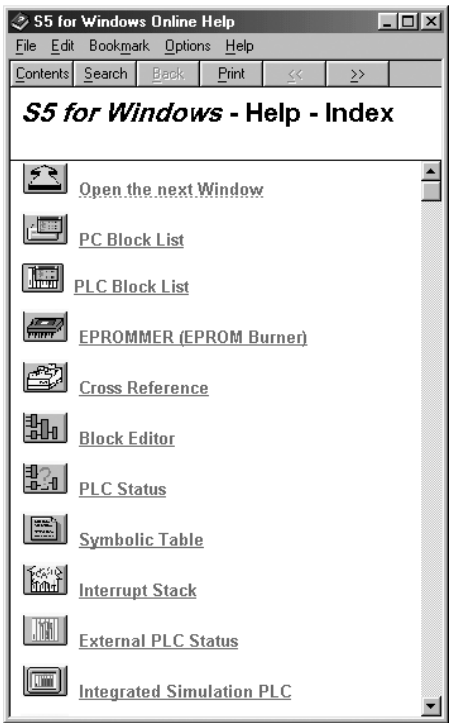
اکنون این نرم‌افزار را بارگذاری کنید. در این حالت پس از ظاهر شدن دو پنجره که

اطلاعاتی در مورد این نرم‌افزار در اختیار شما قرار می‌دهند، پنجرهٔ اصلی را با عنوان S5 for Windows - [PC Block List] مشاهده خواهید کرد.



شکل ۲: پنجرهٔ اصلی در محیط برنامه‌نویسی که در آن فهرست بلوک‌های ایجاد شده به‌نمایش در می آیند.

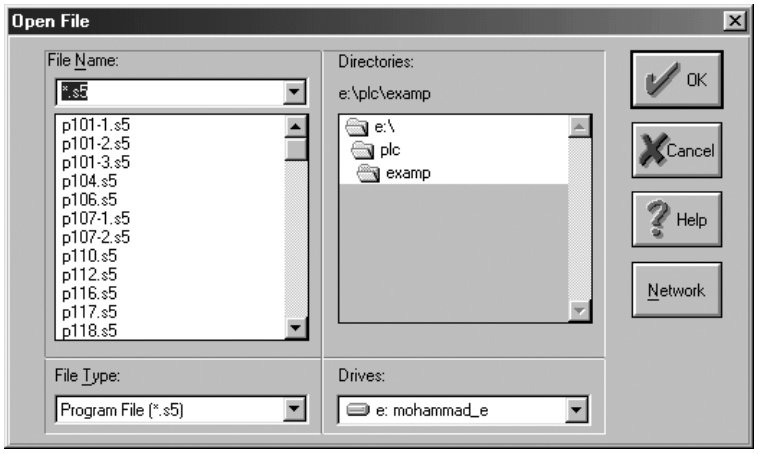
در این پنجره، آیکن ها و کلیدهای بسیاری وجود دارند. با استفاده از گزینهٔ Contents از منوی Help به پنجره Online Help مراجعه کنید و در مورد هر یک از کلیدهای موجود اطلاعاتی به دست آورید. به شما توصیه می کنیم قبل از آغاز برنامه نویسی، با کلیدها و عملکرد آنها به خوبی آشنا شوید و حتماً مطالب عنوان شده در پنجرهٔ Online Help را به طور کامل مطالعه کنید.



شکل ۳: پنجرهٔ Online Help

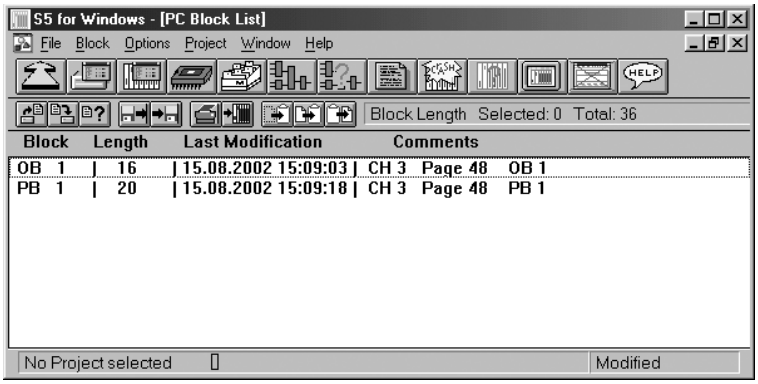
توجه: از این به بعد به عنوان یک قرارداد، انتخاب گزینهٔ Contents از منوی Help را با Help >> Contents نمایش می‌دهیم.

قبل از آغاز برنامه‌نویسی لازم است اجرای برنامه‌ها و مثال‌های موجود در کتاب را ملاحظه کنید. توصیه می‌کنیم که فهرست Examp را که حاوی مثال‌ها و برنامه‌های کتاب است از لوح فشردهٔ همراه این کتاب بر روی دیسک سخت رایانهٔ خود در فهرست s5w-demo کپی کنید. سپس گزینهٔ File >> Open را در پنجرهٔ اصلی برنامه انتخاب نمایید. در این حالت یک پنجرهٔ محاوره‌ای مطابق شکل ۴ ظاهر خواهد شد که می‌توانید تمامی برنامه‌ها و مثال‌های موجود در کتاب را در آن ملاحظه کنید.



شکل ۴: پنجرهٔ محاوره‌ای Open File

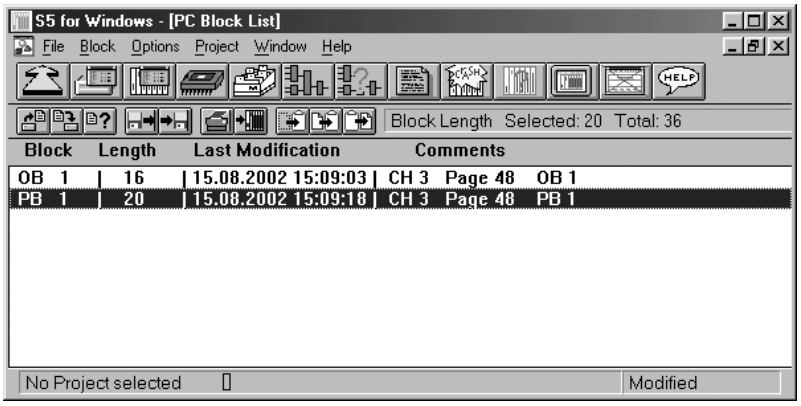
همان گونه که ملاحظه می‌کنید تمامی فایل‌های موجود در فهرست Examp، با پسوند s5. ذخیره شده‌اند. مطابق شکل ۴، نام تمامی فایل‌های موجود شامل حرف p و یک عدد است. این عدد شماره صفحه‌ای از کتاب می‌باشد که برنامهٔ اصلی در آن ذخیره شده است. حرف p مخفف کلمهٔ page است. به عنوان مثال فایل p48.s5 شامل برنامه یا برنامه‌هایی است که در صفحهٔ ۴۸ کتاب نوشته شده‌اند. اکنون این فایل را انتخاب کرده، آن را باز کنید. در این حالت پنجره‌ای مطابق شکل ۵ ظاهر خواهد شد.



شکل ۵: فهرست بلوک‌های موجود در فایل p48.s5

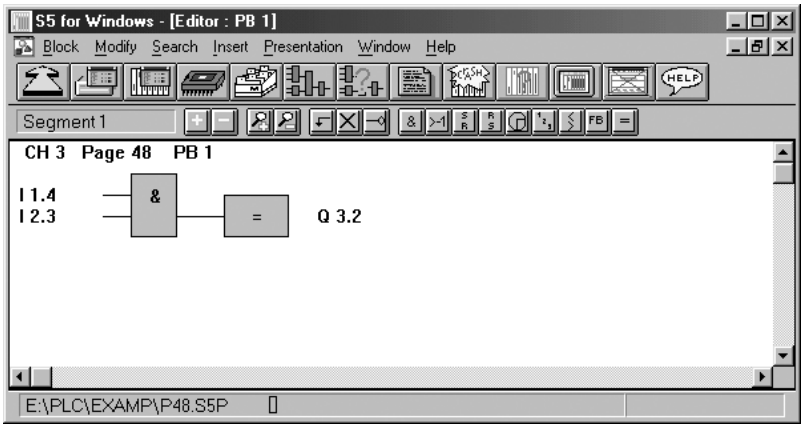
در این فایل دو برنامه یا به عبارت دیگر دو بلوک OB 1 و PB 1 ذخیره شده‌اند. در مورد هر یک از این برنامه‌ها اطلاعاتی از قبیل نوع بلوک، تاریخ اعمال آخرین تغییرات و همچنین توضیحات مختصری در مورد آن در اختیار کاربر قرار می‌گیرد. به عنوان مثال توضیحات ارائه شده در مورد بلوک PB 1 در این فایل نشان می‌دهد که این بلوک مربوط به برنامهٔ موجود در صفحهٔ ۴۸ از فصل سوم کتاب با نام PB 1 می‌باشد و ۲۰ بایت از حافظهٔ دیسک سخت را به خود اختصاص داده است.

با کلیک کردن بر روی هر یک از برنامه‌های موجود، اطلاعاتی در این پنجره ظاهر می‌شود. به عنوان مثال پس از کلیک کردن بر روی بلوک PB 1 در این پنجره، این بلوک فعال شده و مطابق شکل ۶ اطلاعاتی در مورد حجم حافظهٔ اشغال شده توسط این بلوک و همچنین حجم حافظهٔ اشغال شده توسط تمامی برنامه‌های موجود در این پنجره در اختیار کاربر قرار می‌گیرد.



شکل ۶: نمایش بلوک PB1 به روش CSF

با دوبار کلیک کردن بر روی این بلوک، پنجره‌ای مطابق شکل ۷ ظاهر شده و برنامهٔ نوشته شده به یکی از سه روش STL، CSF یا LAD به نمایش در می‌آید[۱].



شکل ۷

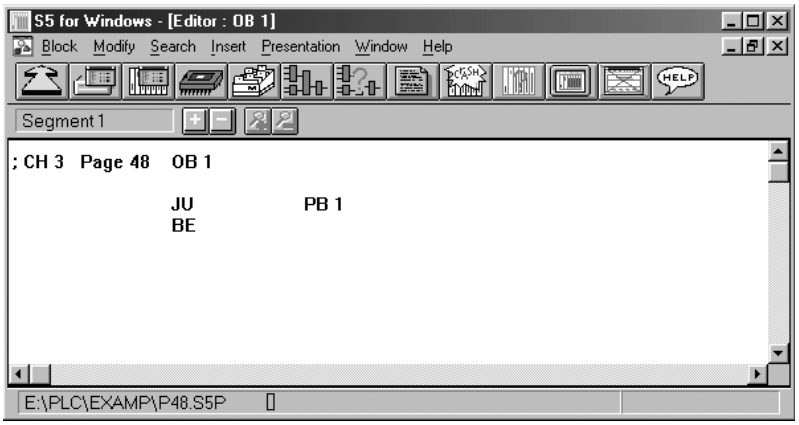
۱- در صورتی که برنامهٔ انتخاب شده به روش CSF قابل نمایش باشد، همواره این روش به صورت پیش‌فرض برای به نمایش در آوردن برنامه انتخاب می‌شود.

همان گونه که در شکل ۷ ملاحظه می‌کنید، در این حالت منوهای موجود در این پنجره تغییر می‌کنند. حال با انتخاب گزینه‌های موجود در منوی Presentation، برنامهٔ نوشته شده را به روش‌های مختلف به نمایش در آورید. به عنوان مثال مشاهده می‌کنید که این بلوک در مورد یک برنامهٔ بسیار ساده و ابتدایی است که در آن، خروجی Q 3.2 از ترکیب عطفی دو ورودی I 1.4 و I 2.3 به‌دست آمده است.

توضیح ارائه شده در مورد این برنامه نیز در بالای صفحهٔ برنامهٔ این بلوک و در هر سه روش STL، CSF و LAD دیده می‌شود. به دلیل اینکه برنامه‌ها در محیط Edit باز می‌شوند، در این حالت می‌توانید هر گونه تغییراتی را در مورد این برنامه اعمال کنید. برای ذخیرهٔ تغییرات انجام شده در مورد هر بلوک گزینهٔ Block >> Save را انتخاب کنید.


PC Block List Button

پس از مشاهدهٔ حالات و روش‌های مختلف نمایش این بلوک، با کلیک کردن بر روی دکمهٔ PC Block List، مجدداً به پنجرهٔ قبلی بازگردید. اکنون بلوک OB 1 را بررسی کنید. ملاحظه می‌کنید که برای این بلوک، تنها حالت نمایش STL فعال است و به نمایش در آوردن این بلوک به روش CSF یا LAD امکان‌پذیر نیست[1].

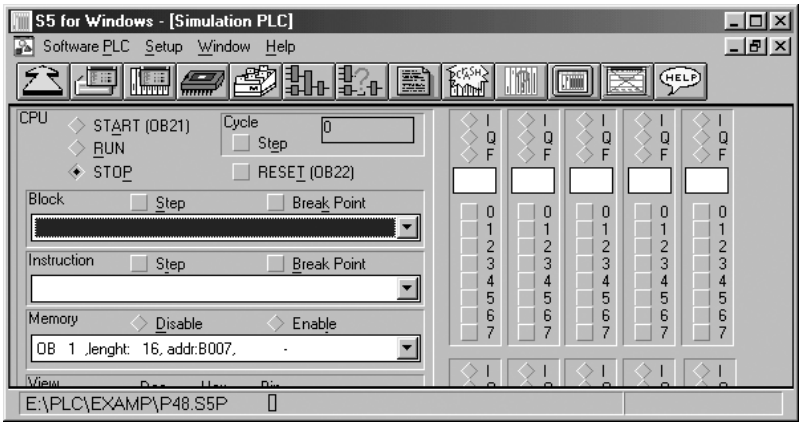


شکل ۸: نمایش بلوک OB 1 به روش STL


Integrated Simulation PLC Button

همان گونه که در شکل ۸ مشاهده می‌کنید در این بلوک دستور پرش غیر شرطی به بلوک PB1 صادر شده است. برای اجرای بلوک PB 1 ابتدا آن را انتخاب نموده، سپس بر روی دکمهٔ Simulation Integrated PLC کلیک کنید. با کلیک کردن بر روی این دکمه، پنجره‌ای مطابق شکل ۹ ظاهر می‌گردد.

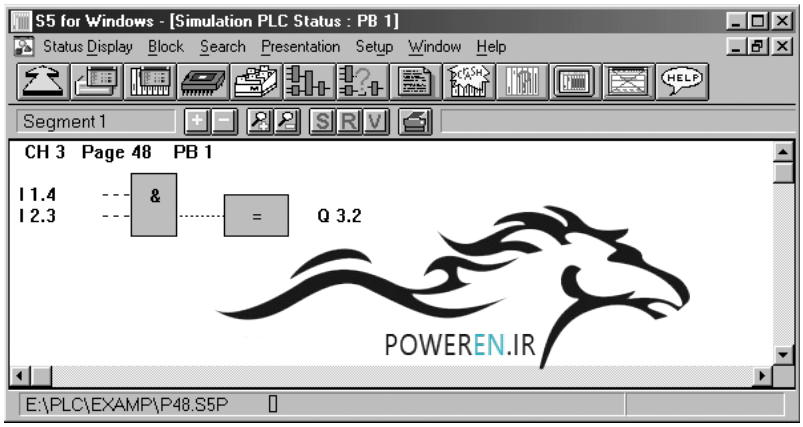
۱- در فصل سوم کتاب عنوان شده است که تمامی برنامه‌های نوشته شده به روش LAD و CSF، به روش STL نیز به نمایش در می‌آیند؛ اما عکس این مطلب همواره صادق نیست.



شکل ۹: پنجرهٔ محاوره‌ای Simulation PLC

در این صفحه عملکرد PLC شبیه سازی شده است و همان گونه که ملاحظه می کنید دکمه های STOP، RUN، START و RESET در این پنجره تعبیه شده اند. در ضمن در این پنجره تعداد سیکل های طی شده در بخش Cycle به نمایش در می آید. حال بر روی دکمهٔ RUN در این پنجره کلیک کرده، و بدین روش PLC را روشن کنید.

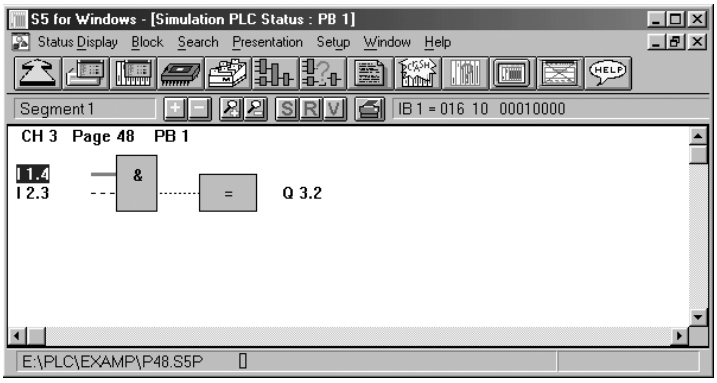
در این حالت مجدداً بر روی دکمهٔ PC Block List کلیک کنید و به پنجرهٔ اصلی بازگردید. با کلیک کردن بر روی دکمهٔ PLC Status پنجره‌ای مطابق شکل ۱۰ ظاهر می‌شود.



شکل ۱۰: نمایش وضعیت ورودی‌ها و خروجی‌های بلوک PB 1 در روش CSF

مطابق شکل ۱۰، دو ورودی I 1.4 و I 2.3 با خطوط نقطه‌چین شده به بلوک عطفی AND اتصال یافته‌اند. این طرز نمایش بیانگر غیر فعال بودن ورودی‌هاست. برای فعال یا غیر فعال ساختن هر یک از ورودی‌ها ابتدا ورودی مورد نظر را با کلیک کردن بر روی

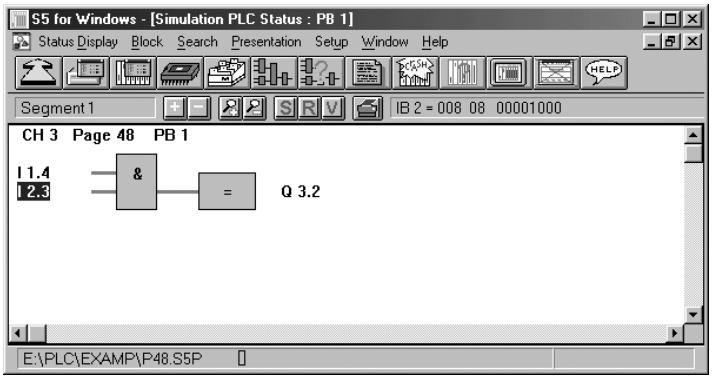
آن انتخاب نموده، سپس یکی از دو گزینهٔ Set Operand یا Reset Operand را از منوی Status Display انتخاب کنید. به عنوان مثال در شکل ۱۱، ورودی I 1.4 فعال شده است.



شکل ۱۱: نمایش وضعیت ورودی‌ها و خروجی‌های بلوک PB 1 در روش LAD

همان گونه که ملاحظه می کنید در این حالت، اتصال این ورودی به بلوک عطفی AND با خط توپر نشان داده می شود.

حال به همین ترتیب ورودی I 2.3 را نیز فعال کنید. مطابق شکل ۱۲ ملاحظه می کنید که هر دو ورودی I 2.3 و I 1.4 با خطوط توپر به بلوک AND اتصال یافته‌اند. همچنین بلوک عطفی AND نیز با یک خط توپر به خروجی مرتبط شده است.



شکل ۱۲

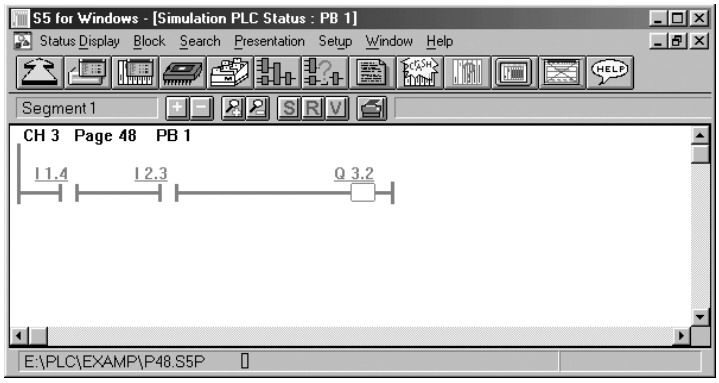
اکنون در همین پنجره بر روی ورودی I 1.4 کلیک کرده، اطلاعات مندرج در شکل ۱۳ را مشاهده کنید.

شکل ۱۳

این اطلاعات، وضعیت هر یک از بیت های موجود در IB 1 را به ترتیب از چپ به راست در سه مبنای ده، شانزده و دو نشان می دهد. به عنوان مثال بیت های 00010000 نشان می دهند که در حال

حاضر تنها بیت چهارم از بایت ورودی اول یا به‌عبارت دیگر، ورودی I 1.4 فعال است.

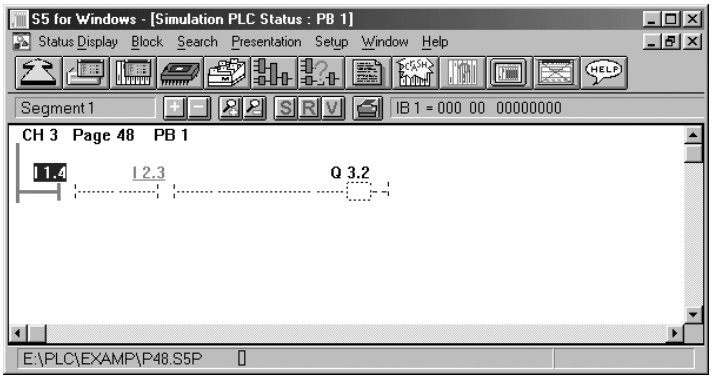
با کلیک کردن بر روی ورودی I 2.3 و خروجی Q 3.2 نیز اطلاعاتی در مورد بایت‌های ورودی و خروجی IB 2 و QB 3 به دست آورید. اکنون گزینهٔ Presentation >> Ladder Diagram (LAD) را انتخاب نمایید. در این حالت پنجره‌ای مطابق شکل ۱۴ ظاهر خواهد شد.



شکل ۱۴

در این پنجره، فعال بودن هر یک از ورودی‌ها و خروجی‌ها با وجود یک خط در زیر آن‌ها نشان داده می‌شود. حال با کلیک کردن بر روی ورودی I 1.4 و انتخاب گزینهٔ Status Display >> Reset Operand این ورودی را غیر فعال کنید.

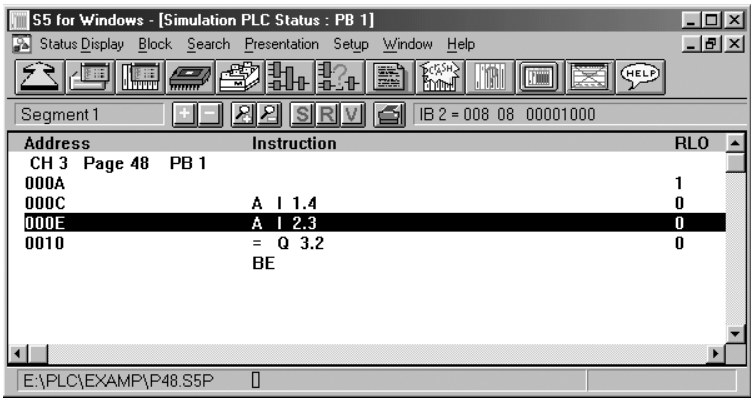
در این حالت صفحهٔ برنامه مطابق شکل ۱۵ تغییر می‌کند و نشانگر فعال بودن ورودی I 1.4 و غیر فعال بودن ورودی I 2.3 است. به خط افقی موجود در زیر ورودی I 2.3 توجه کنید.



شکل ۱۵

اکنون اطلاعات مربوط به IB 1 را در پنجرهٔ برنامه ملاحظه کنید و ببینید که ورودی I 1.4 غیر فعال شده است. حال با انتخاب گزینهٔ Presentation >> Statement List (STL)، این برنامه را به روش STL به نمایش در آورید.

در این صفحه نیز وضعیت هر یک از بیت‌های ورودی و خروجی را با کلیک کردن بر روی ورودی یا خروجی مورد نظر به دست آورید. در این پنجره مراقب باشید که مفهوم بیت RLO را با وضعیت ورودی یا خروجی برنامه اشتباه نگیرید. به عنوان مثال در شکل ۱۶، بیت RLO مربوط به سطر شمارهٔ 000E در برنامه یعنی A I 2.3 در وضعیت 0 بوده، در حالی که اطلاعات موجود در بالای پنجره یعنی 00001000 نشان می‌دهد که در حال حاضر ورودی I 2.3 فعال است.

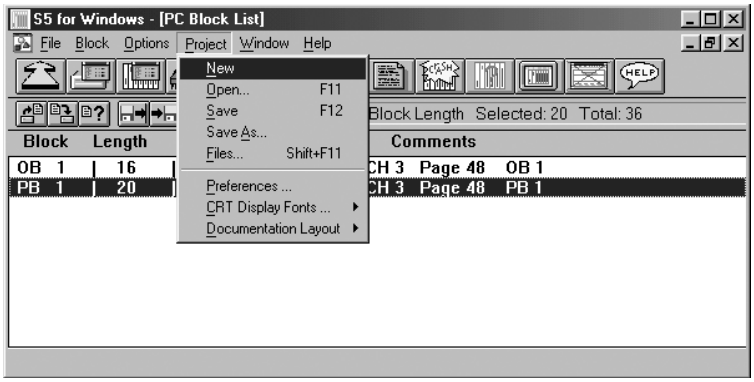


شکل ۱۶: نمایش وضعیت ورودی‌ها و خروجی‌های بلوک PB 1 در روش STL

اکنون سعی کنید که این برنامه را به هر یک از روش‌های STL، CSF و LAD به نمایش در آورده و با اعمال تغییر در ورودی‌ها، وضعیت خروجی را مشاهده کنید.

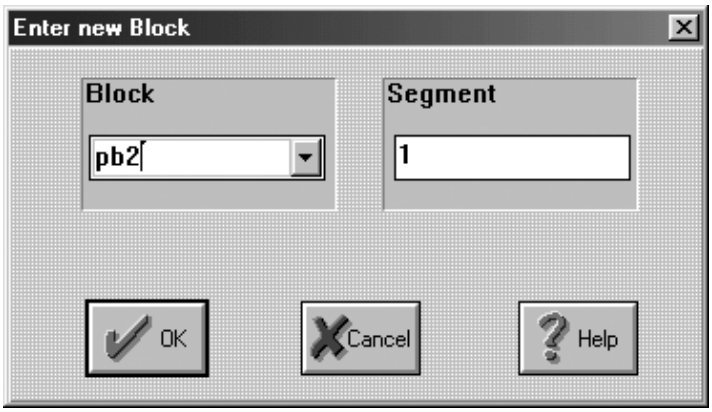
برنامه‌نویسی در نرم‌افزار شبیه‌ساز PLC

برای نوشتن برنامه ابتدا با انتخاب گزینهٔ Project >> New یک پروژهٔ جدید باز کنید.



شکل ۱۷: روش ایجاد یک پروژهٔ جدید

سپس با انتخاب گزینهٔ Block >> New Block... نوع و شمارهٔ بلوک مورد نظر را وارد کنید. به عنوان مثال در شکل ۱۸ ملاحظه می‌کنید که عبارت pb2 در این پنجرهٔ محاوره‌ای وارد شده است.



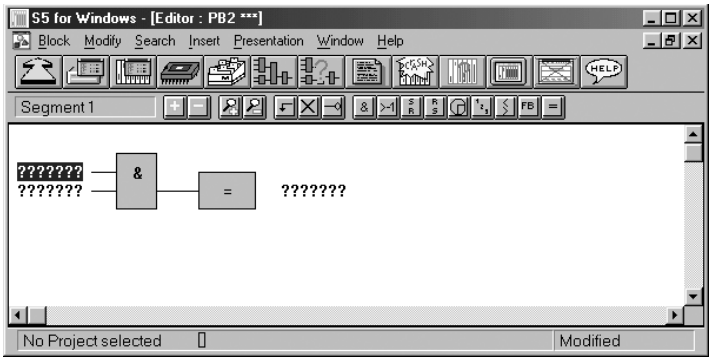
شکل ۱۸: پنجرهٔ محاوره‌ای Enter new Block برای وارد کردن نوع و شمارهٔ بلوک مورد نظر

همان گونه که ملاحظه می‌کنید برای وارد کردن نوع و شمارهٔ بلوک، استفاده از حروف بزرگ و در نظر گرفتن فاصله بین نوع بلوک و شماره آن الزامی نیست. کامپایلر این نرم‌افزار، عملیات مذکور را به صورت خودکار انجام می‌دهد. بدین مفهوم که برای نام‌گذاری بلوک از حروف بزرگ استفاده نموده، بین نام بلوک و شمارهٔ آن نیز فاصله‌ای ایجاد می‌کند.

این نرم‌افزار همواره روش CSF را برای ایجاد یک برنامه به صورت پیش‌فرض انتخاب می‌کند. برای نوشتن یک برنامه به روش‌های دیگر، گزینهٔ مورد نظر را از منوی Presentation انتخاب نمایید.

حال فرض کنید که قصد داریم برنامهٔ موجود در صفحهٔ ۴۹ کتاب یا PB 2 را به روش CSF بازنویسی کنیم.

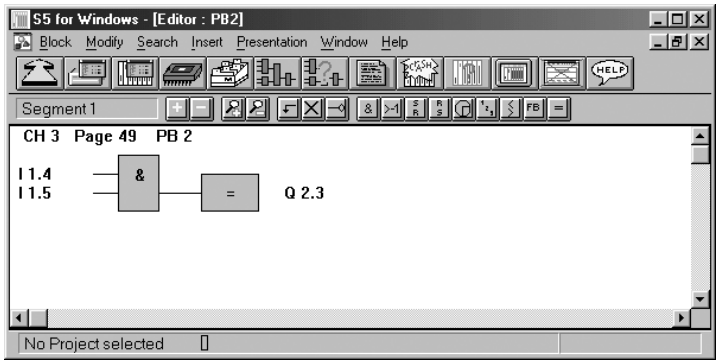
با انتخاب گزینهٔ And << Insert یا با کلیک کردن بر روی دکمهٔ  در پنجرهٔ برنامه، یک بلوک AND در صفحه قرار دهید.



شکل ۱۹: روش ایجاد بلوک عطفی در برنامهٔ PB 2

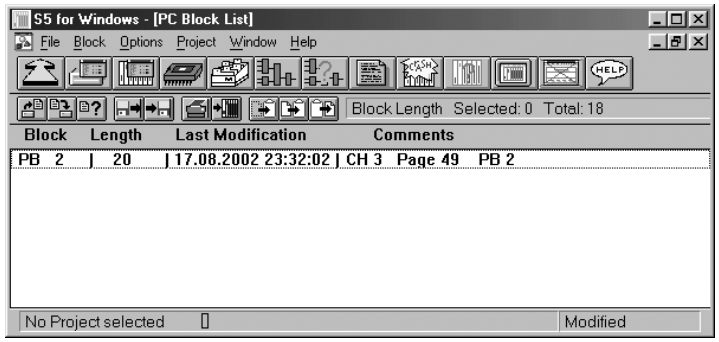
اکنون ورودی‌های مورد نظر یعنی I 1.4 و I 1.5 را به بلوک AND اعمال کنید. در این حالت، در نظر گرفتن فاصله بین عملوند و آدرس یا شمارهٔ آن الزامی نیست. پس از فشردن کلید <Enter> این عمل به

صورت خودکار انجام می گیرد. سپس خروجی Q 2.3 را به بلوک AND نسبت دهید. در این حالت پنجرهٔ برنامه باید به صورت شکل ۲۰ ظاهر گردد. در صورت تمایل می توانید در سطر اول برنامه توضیحاتی در مورد عملکرد برنامه و یا موارد دیگر وارد نمایید. همان گونه که در شکل ۲۰ ملاحظه می کنید در سطر اول برنامه، محل و نام این بلوک در کتاب، به عنوان توضیح درج شده است.

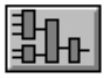


شکل ۲۰: انتساب ورودی‌ها و خروجی‌های بلوک PB 2 در روش CSF

جهت ذخیرهٔ تغییرات اعمال شده گزینهٔ Block >> Save را انتخاب کنید. اکنون با کلیک کردن بر روی دکمهٔ PC Block List مجدداً به پنجرهٔ اصلی پروژه بازگردید. در این حالت صفحه‌ای مطابق شکل ۲۱ ظاهر می شود که در آن، تنها بلوک PB 2 را ملاحظه می کنید.



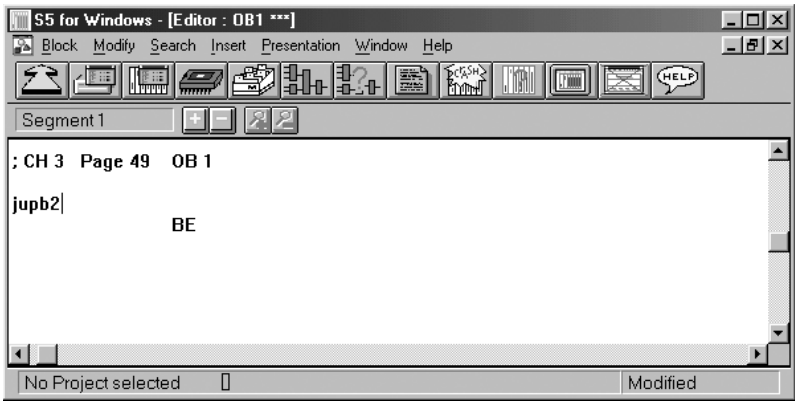
شکل ۲۱



با کلیک کردن بر روی دکمهٔ Block Editor یا با دوبار کلیک کردن بر روی عنوان این بلوک می توانید وارد محیط Edit شوید و تغییرات دلخواه را در مورد آن اعمال کنید. در این حالت می توانید بلوک مورد نظر را به روش های مختلف به نمایش در آورید.

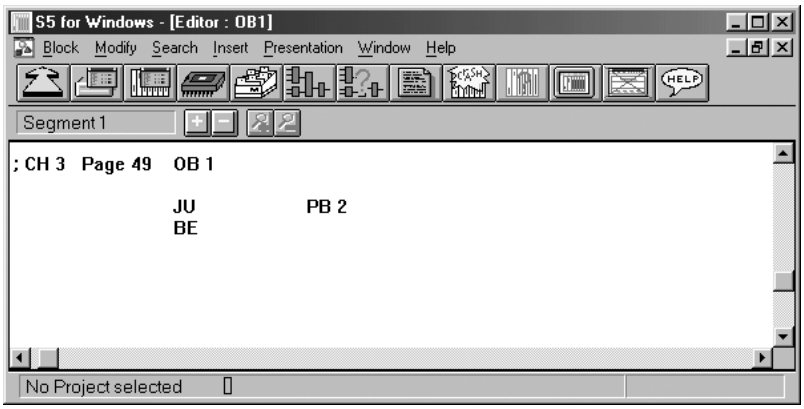
برای اجرا نمودن این برنامه لازم است ابتدا یک بلوک OB 1 ایجاد نمایید و در آن،

دستور پرش غیرشرطی به بلوک PB 2 را صادر کنید. به دلیل اینکه در بـلوک OB 1 معمولاً از دستوراتی استفاده می‌شود که تنها به حالت STL مورد استفاده قرار می‌گیرند، با انتخاب گزینهٔ Presentation >> Statement List (STL) بلوک OB 1 را مطابق شکل ۲۲ به روش STL ایجاد کنید.



شکل ۲۲: نحوهٔ ایجاد بلوک PB 2 به روش STL

همان گونه که ملاحظه می‌کنید برای وارد کردن برنامه (و نه توضیحات مربوط به آن) استفاده از حروف بزرگ و در نظر گرفتن فاصله بین حروف الزامی نیست. پس از انتخاب گزینهٔ Format >> Block و یا فشار دادن کلید <F9> این عمل به طور خودکار انجام شده و بلوک OB 1 به صورت زیر به نمایش درمی‌آید.

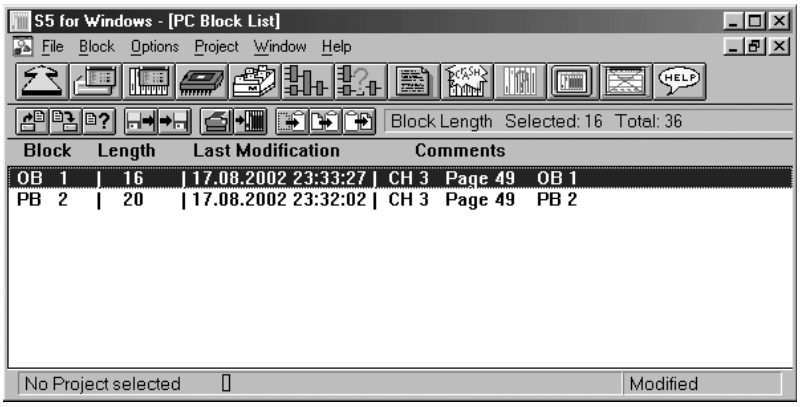


شکل ۲۳: نمایش بلوک OB 1 به روش STL پس از فشردن کلید <F9> یا انتخاب گزینهٔ Format >> Block

پس از وارد کردن برنامه، با انتخاب گزینهٔ Block >> Save این بلوک را ذخیره کنید. حال گزینهٔ Presentation >> Control System Flowchart (CSF) را انتخاب نمایید. ملاحظه می‌کنید که بلوک OB 1 به روش CSF یا LAD نمایش داده نمی‌شود و نمایش این بلوک تنها به روش STL امکان‌پذیر است.

با کلیک کردن بر روی دکمهٔ PC Block List به پنجرهٔ اصلی پروژه بازگردید. حال دو بلوک با

عناوین OB 1 و PB 2 را به صورت زیر در این پنجره ملاحظه می کنید.



شکل ۲۴

برای اجرای این برنامه بر روی دکمهٔ PLC Simulation کلیک کنید و مراحل ذکر شده در مورد بلوک PB 1 را در مورد بلوک PB 2 تکرار کنید.

اکنون برای ذخیرهٔ تمامی بلوک های ایجاد شده در این فایل پروژه، گزینهٔ Project >> Save را انتخاب کنید. توصیه می کنیم برای دسترسی سریع و آسان به برنامه های ایجاد شده، تمامی فایل های پروژهٔ خود را در یک فهرست ذخیره نمایید.

همان گونه که ملاحظه می کنید تمامی فایل ها در این بخش با پسوند s5p. ذخیره می شوند حرف p نمایانگر واژهٔ Project است و در هنگام ذخیرهٔ هر فایل پروژه دو فایل دیگر نیز با همین نام اما با پسوندهای s5. و seq. ذخیره می شوند.

شایان ذکر است که برای نام گذاری فایل ها در این نرم افزار، باید از قواعد نام گذاری فایل در Windows 3.1 پیروی کنید. سه فایل مذکور با پسوندهای s5p.، s5. و seq. به ترتیب مربوط به فایل پروژه[1]، فایل برنامه[2] و نمادهای[3] استفاده شده در برنامه هستند. این سه فایل، همنام هستند ولی با پسوندهای متفاوتی ذخیره می شوند. برای کسب اطلاعات بیشتر در مورد این فایل ها، مطالب مندرج در پنجرهٔ Online Help را مطالعه کنید.

حال قصد داریم برای انجام تمرین بیشتر، مثال ۳-۷ در صفحهٔ ۶۵ کتاب را بررسی نموده، مراحل ایجاد این برنامه را قدم به قدم دنبال کنیم. در این برنامه از چهار ورودی استفاده شده است. همان گونه که ملاحظه می کنید ورودی های این برنامه دوبه دو با یکدیگر OR شده، سپس حاصل آنها با یکدیگر AND می شود. قصد داریم ابتدا این برنامه را به روش CSF باز نویسی کنیم. بنابراین با

1- Project file
2- Program file
3- Symbolic file

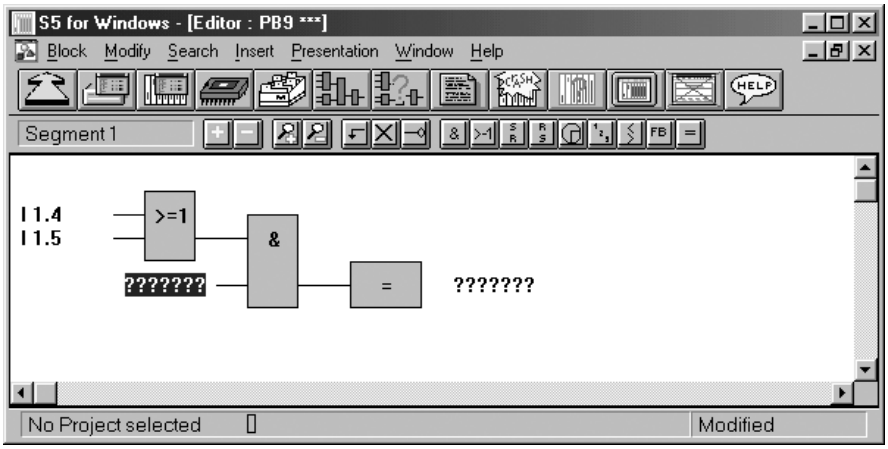
انتخاب گزینهٔ Project >> New یک فایل پروژهٔ جدید ایجاد کنید. حال گزینهٔ Block >> New Block... را انتخاب نموده، در پنجرهٔ محاوره‌ای ظاهر شده، نام PB 9 را وارد کنید.

همان گونه که قبلاً اشاره شد پس از انتخاب گزینهٔ Block >> New Block... و وارد کردن یکی از بلوک‌های OB، PB یا FB، حالت نمایش CSF به صورت پیش‌فرض برای ایجاد بلوک جدید در نظر گرفته می‌شود.

برنامهٔ موجود در صفحهٔ ۶۵ را ملاحظه کنید. ابتدا با کلیک کردن بر روی دکمهٔ مربوط به بلوک OR یعنی  و یا با انتخاب گزینهٔ Insert >> Or، بلوک OR را بر روی صفحه قرار دهید و تنها ورودی‌های آن را اعمال کنید. در این مرحله، از انتساب خروجی خودداری کنید.

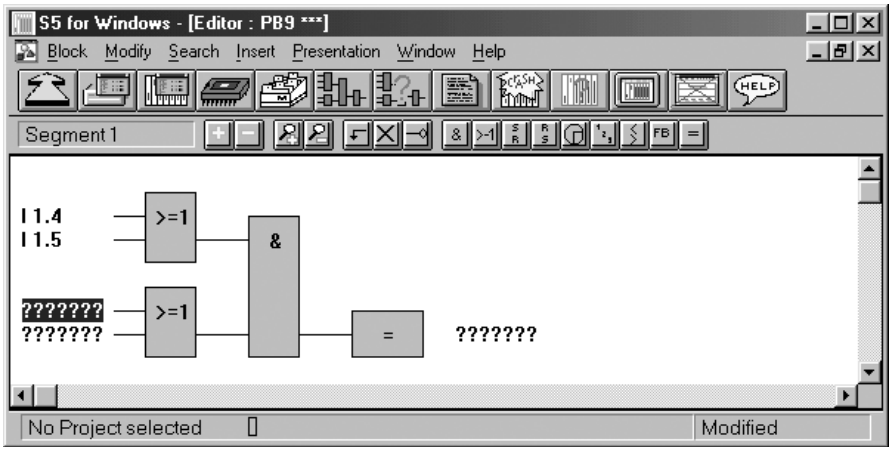
همان گونه که عنوان شد در این مرحله در نظر گرفتن فاصله بین عملوند و آدرس یا شمارهٔ آن الزامی نیست. پس از فشردن کلید <Enter> این عمل به صورت خودکار انجام می‌گیرد.

در این حالت بر روی بلوک OR کلیک کنید تا این بلوک فعال گردد و با رنگ آبی نمایش داده شود. با این عمل در حقیقت این بلوک را برای انتساب به یک بلوک دیگر آماده می‌سازید. حال بر روی دکمهٔ مربوط به بلوک AND یعنی کلیک کرده، یا گزینهٔ Insert >> And را انتخاب کنید تا پنجره‌ای مطابق شکل ۲۵ ظاهر شود.



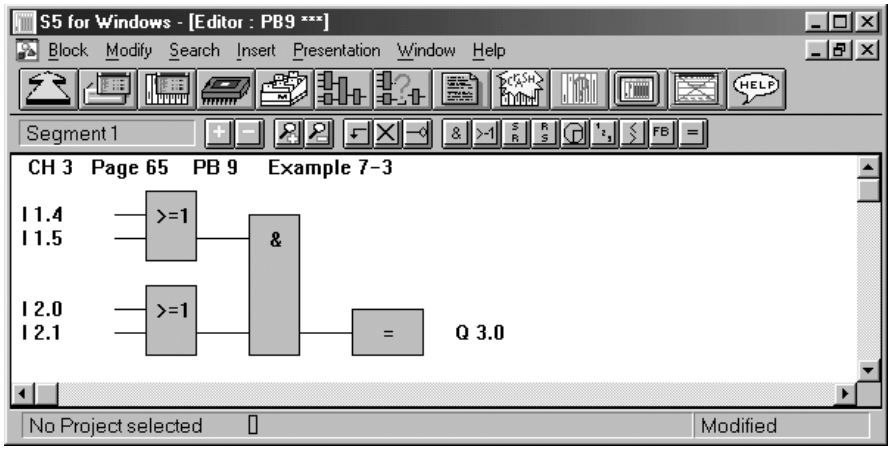
شکل ۲۵

در شکل ۲۵ ملاحظه می‌کنید که در خروجی بلوک OR، یک بلوک AND قرار گرفته است. حال ورودی دوم بلوک AND را با کلیک کردن بر روی آن فعال نموده، بلوک OR را با کلیک کردن بر روی دکمهٔ  یا انتخاب گزینهٔ Insert >> Or در این بخش وارد کنید. در این حالت شکل ۲۶ ظاهر می‌گردد.



شکل ۲۶

اکنون می‌توانید ورودی‌های I 2.0 و I 2.1 را به دومین بلوک OR اعمال کنید و در مرحلهٔ آخر نیز بلوک AND را به خروجی Q 3.0 نسبت دهید. در صورت تمایل می‌توانید توضیحات مندرج در شکل ۲۷ را وارد کنید.



شکل ۲۷: بلوک تکمیل شدهٔ PB 9 در روش CSF

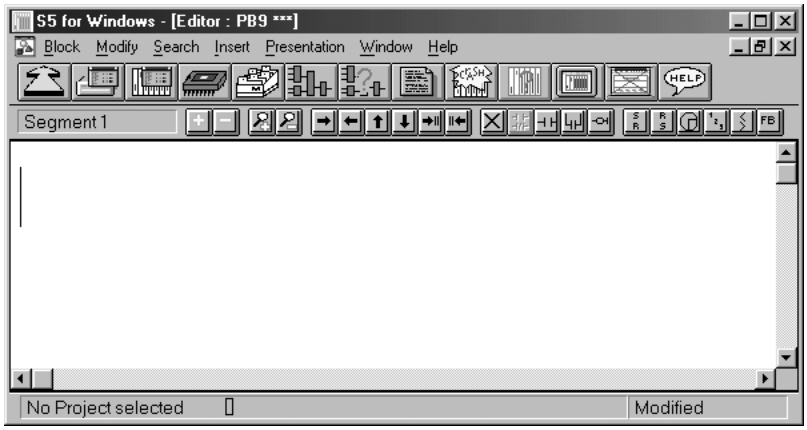
برای اجرای این برنامه ابتدا یک بلوک OB 1 ایجاد کنید و در آن، دستور پرش غیرشرطی به PB 9 را صادر نمایید. مراحل اجرای برنامه نیز شبیه به موارد قبلی است.

ایجاد یک برنامه به روش LAD

در این بخش قصد داریم تا همین بلوک یعنی PB 9 را به روش LAD بازنویسی کنیم. بنابراین با انتخاب گزینهٔ New << Projec یک فایل پروژهٔ جدید ایجاد کنید. سپس گزینهٔ Block >> New Block...

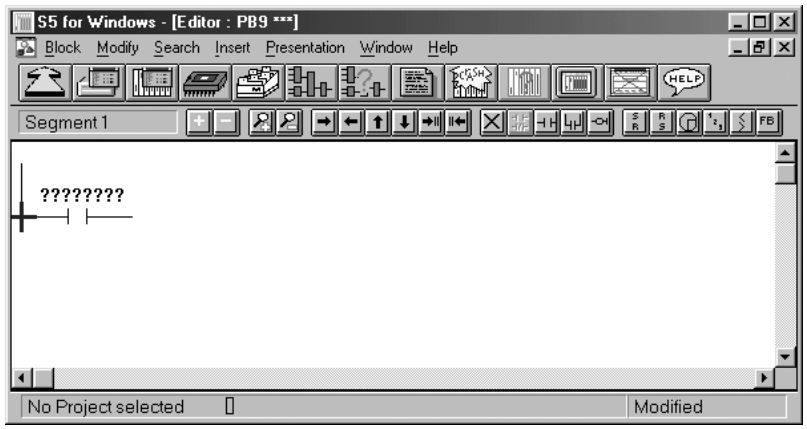
را انتخاب نمایید و در پنجرهٔ محاوره‌ای ظاهر شده، عبارت PB 9 را وارد کنید.

حال با انتخاب گزینهٔ Presentation >> Ladder Diagram (LAD) این بلوک را به روش LAD بازنویسی کنید. ملاحظه می‌کنید که در این حالت دکمه‌های موجود در پنجرهٔ برنامه به صورت زیر تغییر می‌کنند.



شکل ۲۸: ایجاد بلوک PB 9 به روش LAD

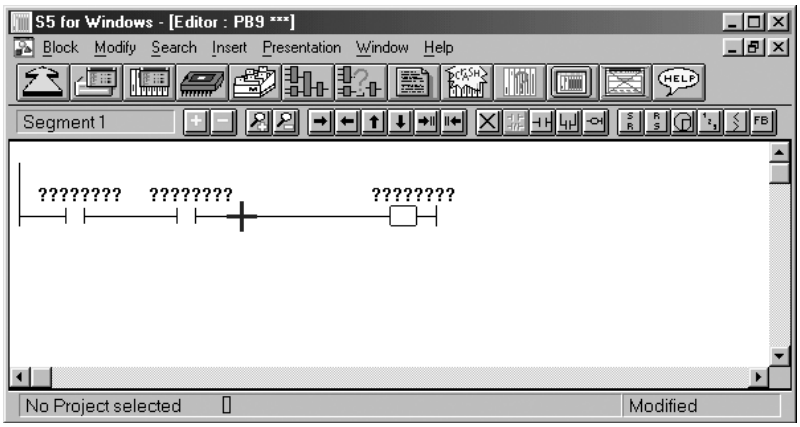
با کلیک کردن بر روی خط عمودی موجود در شکل ۲۸ خود را برای برنامه‌نویسی آماده کنید. در این حالت یک علامت به صورت + در پایین خط مذکور ایجاد می‌شود. با کلیک کردن بر روی دکمهٔ یا انتخاب گزینهٔ Insert >> Contact in Row اولین ورودی را بر روی صفحه قرار دهید.



شکل ۲۹

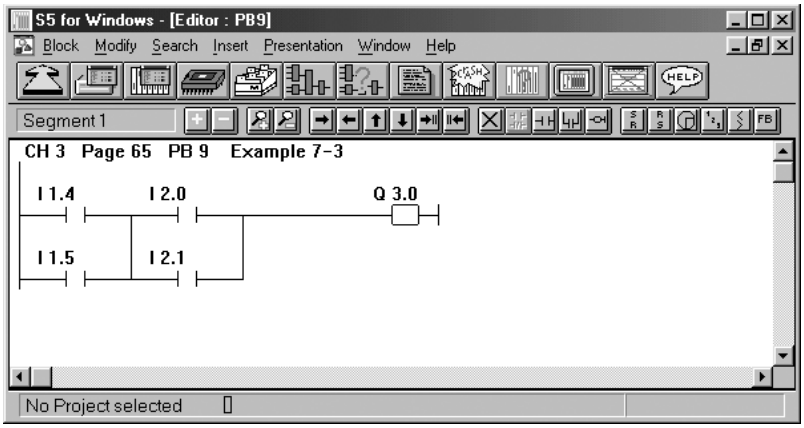
همان گونه که ملاحظه می‌کنید ورودی‌های بلوک در روش LAD به صورت کنتاکتور به نمایش در می‌آیند. در این مرحله می‌توانید نام این ورودی یعنی I 6.0 را وارد کنید. اما ترجیحاً ابتدا مدار را تکمیل کرده، سپس نام ورودی‌ها و خروجی بلوک را وارد نمایید. حال بر روی آخرین نقطهٔ اتصال در شکل ۲۹ کلیک کنید و با انتخاب گزینهٔ

Insert >> Contact in Row یا با کلیک کردن بر روی دکمهٔ دومین ورودی را به صورت سری با ورودی اول قرار دهید. در مرحلهٔ بعد با کلیک کردن بر روی دکمهٔ یا انتخاب گزینهٔ Insert >> Assignment Coil برای این بلوک، یک خروجی در نظر بگیرید. پس از طی مراحل مذکور، پنجرهٔ برنامه به صورت شکل ۳۰ ظاهر می‌گردد.



شکل ۳۰

سپس علامت + را در محل نشان داده شده در شکل ۳۰ قرار دهید و با استفاده از کلیدهای ، و مدار را تکمیل کنید. در پایان نیز نام ورودی ها و خروجی بلوک را وارد نمایید. در صورت تمایل می‌توانید توضیحات مندرج در شکل ۳۱ را نیز برای راهنمایی کاربران دیگر وارد کنید.



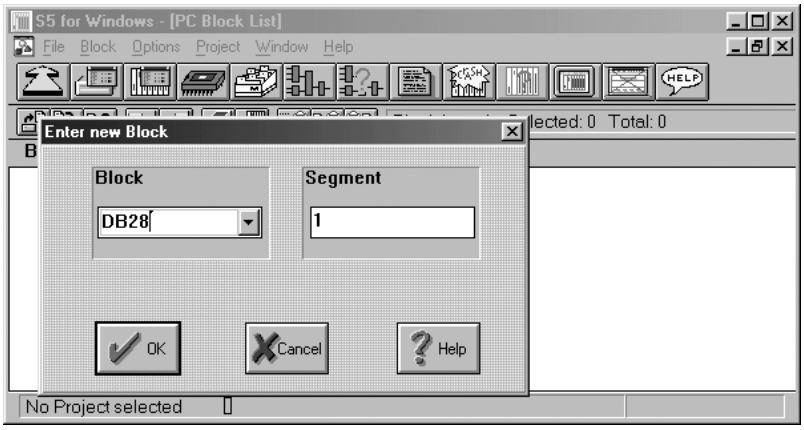
شکل ۳۱: بلوک تکمیل شدهٔ PB 9 در روش LAD

حال با انتخاب گزینهٔ Block >> Save این بلوک را ذخیره کنید.

روش ایجاد بلوک‌های اطلاعاتی (DB)

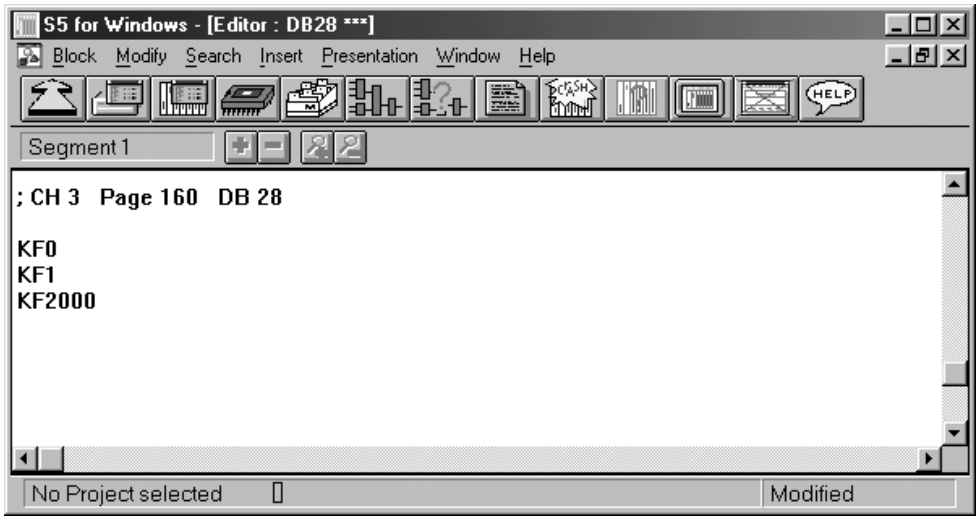
در این بخش قصد داریم بلوک DB 28 در صفحهٔ ۱۶۰ کتاب را ایجاد کنیم. برای ایجاد این بلوک اطلاعاتی در یک فایل پروژه، گزینهٔ Block >> New Block... را انتخاب نموده، در پنجرهٔ محاوره‌ای

ظاهر شده عبارت DB 28 را وارد کنید.



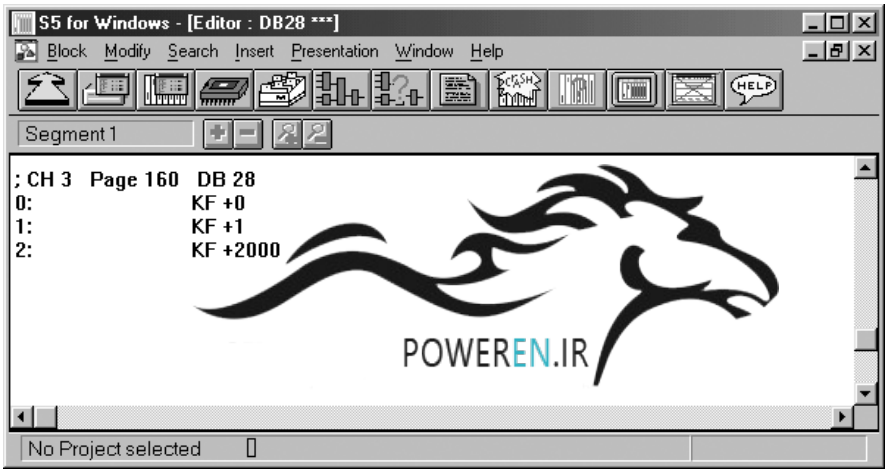
شکل ۳۲

به دلیل اینکه بلوک DB از نوع بلوک‌های اطلاعاتی است، بنابراین تنها به روش STL نوشته می‌شود. با این وجود منوی Presentation را باز نمایید و ببینید که سایر روش‌های نمایش یعنی CSF و LAD برای این بلوک غیرفعال هستند. سپس در پنجرهٔ این بلوک، متن نشان داده شده در شکل ۳۳ را به همین صورت تایپ کنید.



شکل ۳۳: روش ایجاد بلوک DB 28

سپس برای ایجاد بلوک DB 28 به صورت نشان داده شده در صفحهٔ ۱۶۰ کتاب، گزینهٔ Format >> Black را انتخاب نموده، یا کلید <F9> را فشار دهید. در این حالت پنجرهٔ بلوک DB 28 به صورت شکل ۳۴ تغییر می‌کند.



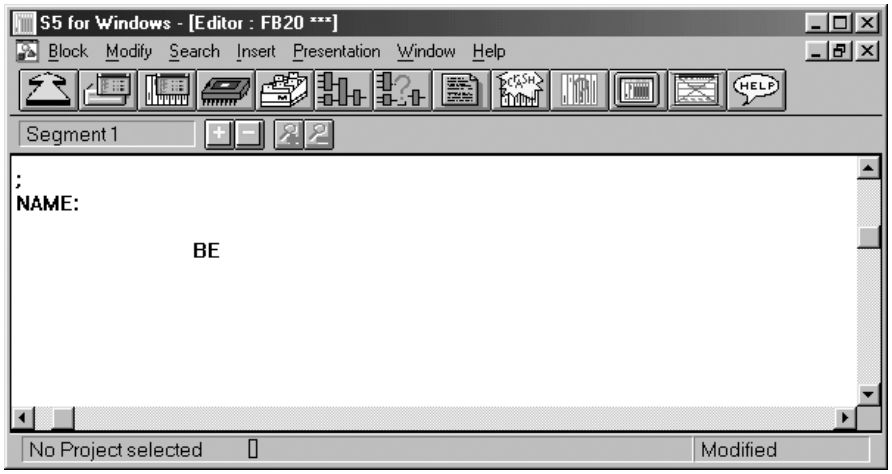
شکل ۳۴: نمایش بلوک DB 28 پس از فشردن کلید <F9> یا انتخاب گزینهٔ Block >> Format

حال با انتخاب گزینهٔ Block >> Save این بلوک را ذخیره نمایید.

روش ایجاد بلوک‌های تابع ساز (FB)

به دلیل اینکه نحوهٔ تعریف و وارد نمودن پارامترهای‌انتسابی یک بلوک FB کمی سؤال‌برانگیز می‌باشد و با روش ایجاد بلوک‌های OB، PB و DB متفاوت است، در این بخش قصد داریم بلوک FB 20 و به دنبال آن PB 3 (در صفحهٔ ۱۷۴ کتاب) را که بلوک مذکور را فراخوانی می کند بازنویسی کنیم.

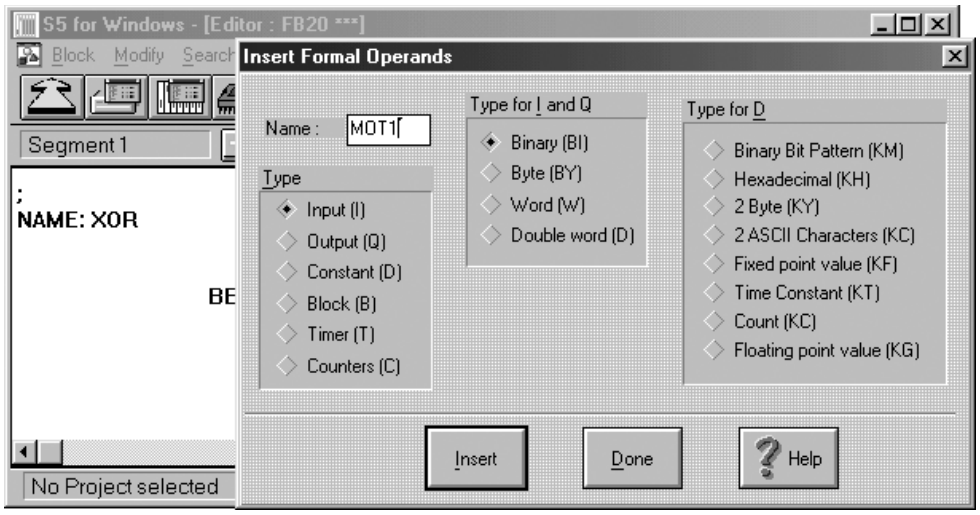
برای ایجاد بلوک FB 20 در یک فایل پروژهٔ جدید، گزینهٔ Block >> New Block... را از منوی پنجرهٔ برنامه باز نموده، در پنجرهٔ محاوره‌ای ظاهر شده، عبارت FB 20 را وارد کنید. در این حالت پنجره‌ای مطابق شکل ۳۵ ظاهر می شود.



شکل ۳۵

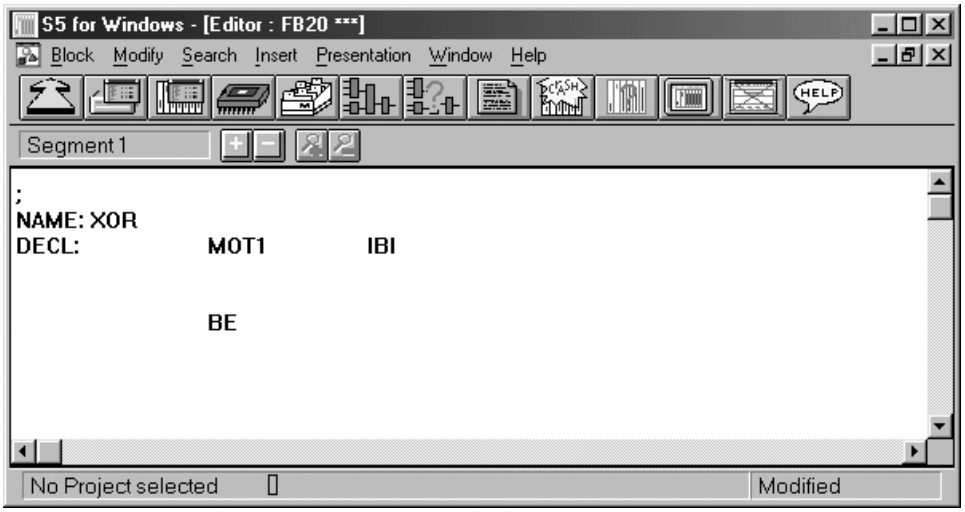
همان گونه که در شکل ۳۵ ملاحظه می‌کنید در اولین سطر این بلوک از کاربر درخواست می‌شود نامی دلخواه برای این بلوک تابع‌ساز انتخاب کند. با مراجعه به پنجرهٔ Online Help درمی‌یابیم که حداکثر تعداد حروف قابل استفاده برای نام‌گذاری یک متغیر، ۴ حرف می‌باشد. کاراکترهای نام انتخاب شده به طور خودکار به صورت حروف بزرگ نوشته می‌شوند. در این قسمت نام XOR را وارد نمایید. حال برای تعریف متغیرها و پارامترهای این بلوک، گزینهٔ Insert >> FB/FX - Formal Operand را انتخاب کنید.

در این حالت یک پنجرهٔ محاوره‌ای برای تعیین نام و نوع پارامتر مورد نظر باز می‌شود. مطابق شکل ۳۶ در بخش Name عبارت MOT1 را وارد نموده، برای تعیین نوع پارامتر در دو بخش Type و Type for I and Q، به ترتیب دو گزینهٔ Input (I) و Binary (BI) را انتخاب کنید. سپس برای وارد کردن این پارامتر در بلوک، بر روی کلید Insert در پایین این پنجره کلیک کنید. برای بازگشت به پنجرهٔ اصلی بر روی دکمهٔ Done کلیک کنید.



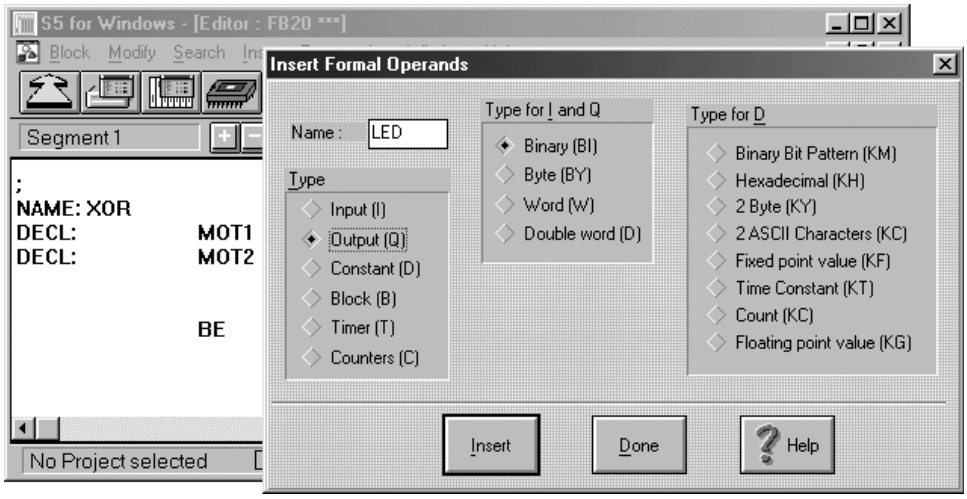
شکل ۳۶: پنجرهٔ محاوره‌ای Insert Formal Operands برای انتساب پارامتر ورودی بلوک FB 20

در این حالت پنجرهٔ برنامه به صورت شکل ۳۷ در می‌آید. ملاحظه می‌کنید که برای اولین پارامتر این بلوک، نام MOT1 در نظر گرفته شده است. عبارت IBI که از دو بخش I و BI تشکیل شده است نشان می‌دهد که این پارامتر از نوع ورودی است و به صورت یک بیت در این بلوک مورد استفاده قرار می‌گیرد.



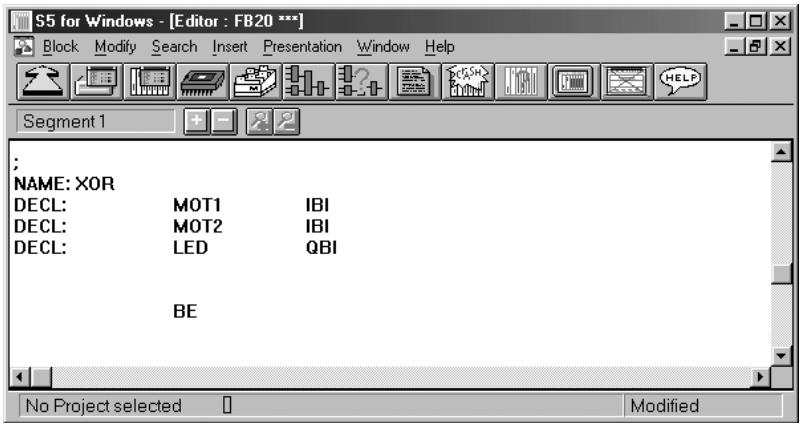
شکل ۳۷

حال برای تعریف دومین پارامتر نیز همین مراحل را تکرار کنید. با این تفاوت که در این قسمت در بخش Name عبارت MOT2 را وارد نمایید. برای تعریف سومین پارامتر، گزینهٔ Insert >> FB/FX - Formal Operand را انتخاب کنید. به دلیل اینکه پارامتر سوم در این بلوک از نوع خروجی است، در پنجرهٔ محاوره‌ای ظاهر شده اطلاعات زیر را وارد کنید.



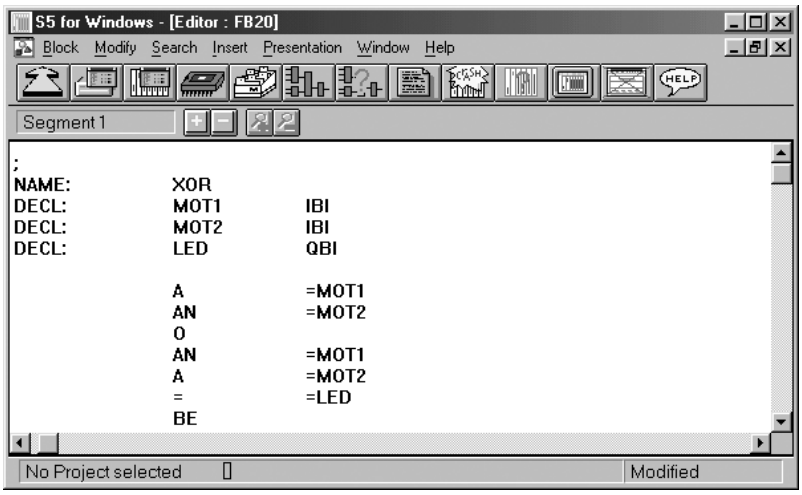
شکل ۳۸: پنجرهٔ محاوره‌ای Insert Formal Operands برای انتساب پارامتر خروجی بلوک FB 20

در این حالت پنجرهٔ برنامه به صورت شکل ۳۹ در می‌آید. در تعریف متغیر سوم، عبارت LED بیانگر نام این پارامتر بوده، QBI نیز نشان‌دهندهٔ آن است که این متغیر از نوع خروجی است و به‌صورت یک بیت در این بلوک مورد استفاده قرار می‌گیرد.



شکل ۳۹

حال می‌توانید سطرهای دیگر موجود در بلوک را تایپ کنید. البته در این بخش در نظر گرفتن فاصله بین عملکرد و پارامترهای منسوب به عملوندهای برنامه الزامی نیست. پس‌از تایپ نمودن سطرهای برنامه، برای ایجاد فاصله بین عملکردها و پارامترهای موجود در برنامه، گزینهٔ Format >> Black را انتخاب کنید یا کلید <F9> را فشار دهید. در این حالت پنجرهٔ برنامه به صورت شکل ۴۰ ظاهر می‌شود.

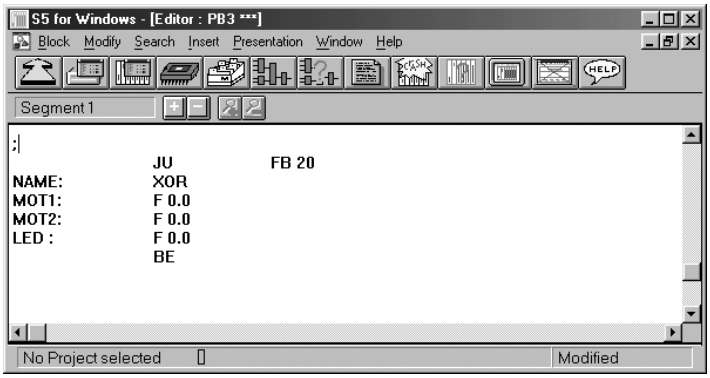


شکل ۴۰: بلوک تکمیل شدهٔ FB 20

حال گزینهٔ Block >> Save را برای ذخیرهٔ این بلوک انتخاب کنید.

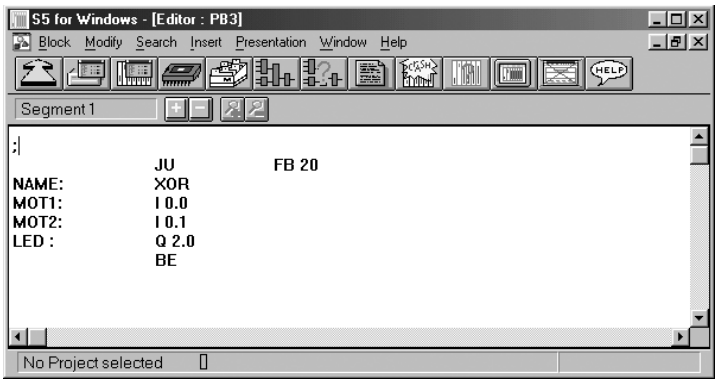
در این بخش قصد داریم تا بلوک PB 3 را نیز که در صفحهٔ ۱۷۴ کتاب، بلوک FB 20 را فراخوانی نموده است بازنویسی کنیم. جهت ایجاد این بلوک مراحل لازم را طی کنید تا پنجرهٔ بلوک PB 3 برای نوشتن برنامه آماده گردد. با انتخاب گزینهٔ Presentation >> Statement List (STL) این بلوک را به روش STL ایجاد نمایید.

در صورت تمایل توضیحات دلخواه را در سطر اول برنامه و پس از کاراکتر «;» وارد کنید. در غیر این صورت کلید <Enter> را فشار دهید و در سطر بعدی، عبارت JUFB20 را وارد کنید. سپس کلید <F9> را فشار دهید. در این حالت، پنجرهٔ برنامه به صورت شکل ۴۱ ظاهر می‌شود.



شکل ۴۱

همان گونه که ملاحظه می‌کنید نام در نظر گرفته شده برای بلوک FB 20 یعنی XOR به همراه سه پارامتر تعریف شده در آن، بر روی صفحه ظاهر می‌شوند. پس از فراخوانی هر بلوک FB، کامپایلر به صورت پیش فرض علموندهایی برای متغیرهای استفاده شده در بلوک در نظر می‌گیرد. به عنوان مثال در این بلوک برای انتساب هر سه متغیر، از F 0.0 استفاده شده است. اکنون عملوندهای نسبت داده شده به این متغیرها را وارد کنید تا صفحهٔ برنامه به صورت زیر در آید. سپس با انتخاب گزینهٔ Block >> Save این بلوک را ذخیره کنید.



شکل ۴۲

سایر روش‌ها و مطالب لازم برای ایجاد برنامه‌های گوناگون را به مرور زمان پس از انجام تمرین با این نرم‌افزار کاربردی و بسیار سودمند به سرعت فرا خواهید گرفت. در صورت بروز هر گونه اشکال، با فشار دادن کلید <F1> از پنجرهٔ Online Help کمک بگیرید.