SABCO
BAREZ AUTOMATION CONTROL
SYSTEMS & ELECTRONIC INDUSTRIES

راهنمای جامع STEP 7

(جلد اول)

مشتمل بر :

- معرفی PLC های سری S7 زیمنس
- سخت افزار و تنظیم پارامترهای آن
- برنامه نویسی با زبانهای LAD,STL,FBD
- مثالهای کاربردی
- و .....

مؤلف:
محمد رضا ماهر

ناشر:
شرکت صابکو

بسم الله الرحمن الرحیم

راهنمای جامع STEP7
(جلد اول)

مولف : مهندس محمد رضا ماهر

ناشر : شرکت صابکو

پیشگفتار مولف

خداوند بزرگ را بخاطر توفیقی که در تدوین این کتاب بر من حقیر ارزانی داشت سپاسگزارم.
برخی دانش پژوهان که از کتاب قبلی اینجانب (*برنامه نویسی و کار بانرم افزار Step7 زیمنس*) استفاده کردند و بسیاری از عزیزانی که در کلاسهای آموزشی بنده شرکت نمودند خواهان تکمیل کتاب مزبور بصورت تفصیلی بعنوان راهنمای جامع Step7 بودند. درخواست بجای ایشان و تشویق های مکرر دوست عزیزم آقای مهندس علی بارزی مدیرعامل محترم شرکت صابکو اینجانب را بر تکمیل کتاب قبلی مصمم ساخت تا نهایتاً بخشی از نتیجه آن در قالب کتاب حاضر که به همت شرکت صابکو منتشر گردید در اختیار علاقمندان قرار گرفت. در تدوین کتاب فرض بر این بود که خواننده محترم با مبانی PLC مختصراً آشناست لذا مطالب مقدماتی در حد نیاز ذکر شد و از بیان تفصیلی آنها که قاعدتاً باید در مراجعی مانند طراحی دیجیتال جستجو شود خودداری گردید. این کتاب ابتدا تصویری کلی از PLC ترسیم میکند سپس بحث را بصورت تخصصی در موردپیکر بندی و برنامه نویسی PLC های سری S7 زیمنس دنبال میکند. پیکر بندی سخت افزار و برنامه نویسی به زبانهای LAD ، STL و FBD مفصلاً بیان شده اند ولی برنامه نویسی به زبانهای SFC و ST و همچنین بحث روی وقفه ها و کنترل کننده های PID و برخی مطالب دیگر به جلد دوم کتاب موکول شده است. پیکر بندی شبکه های صنعتی زیمنس با نرم افزار Step7 نیز موضوع جداگانه ای است که مباحث آن در حال تدوین است و بیاری خداوند در فاصله نه چندان زیادی بعداز این کتاب منتشر خواهد شد.
در تدوین این مجموعه منابع و مراجع بسیار بویژه از مدارک زیمنس مورد استفاده قرار گرفت و سعی شد تا مطالبی که بصورت پراکنده در آنها موجود بود به شکلی منسجم و کاربردی عرضه گردد.
امید فراوان دارم که دست اندرکاران اتوماسیون صنعتی و فارغ التحصیلان رشته های برق از این مجموعه بهره کافی ببرند و نظرات اصلاحی و تکمیلی خود را از طریق آدرس پست الکترونیکی reza.maher@gmail.com یا از طریق ناشر با اینجانب در میان بگذارند.

محمد رضا ماهر

بهمن ماه ۸۳

فهرست مطالب

٥- دستورات برنامه نویسی

٦- ارائه چند مثال برنامه نویسی

۱ – مقدمه

مشتمل بر :

۱-۱ PLC در یک نگاه

۱-۲ استاندارد IEC1131

۱-۳ PLC های مختلف زیمنس

۱-٤ خانواده S7

۱-٥ Step7 و نسخه های مختلف آن

۱-٦ مزیتهای Step7 نسبت به Step5

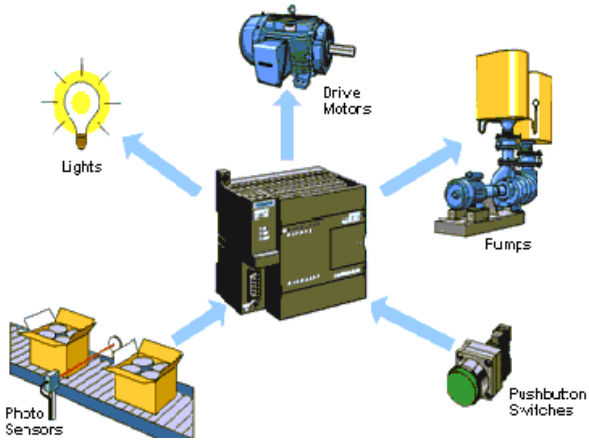
۱-۷ ملزومات نصب Step7

۱-۸ نرم افزارهای جنبی و مرتبط با Step7

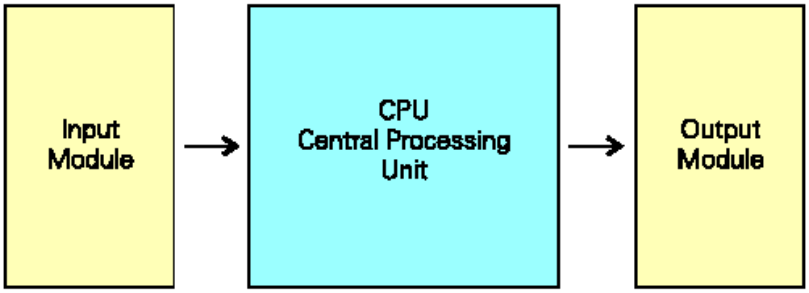
۱-۹ جایگاه Step7 در سیستم کنترل

۱-۱ PLC در یک نگاه

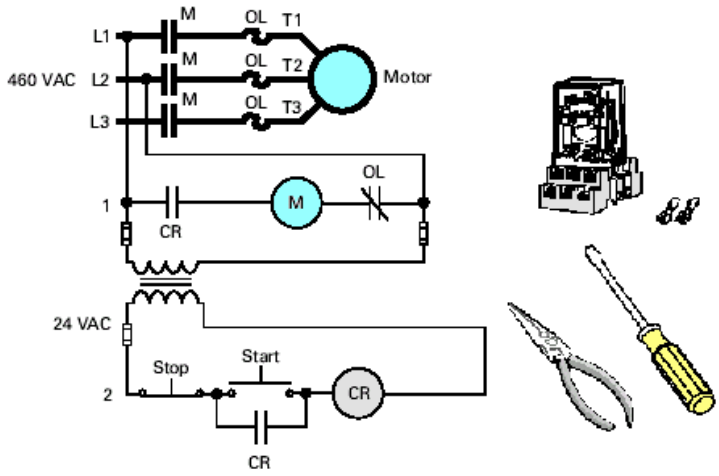
PLC یا Programmable Logic Controller که به نام Programmable Controller نیز شناخته میشود کنترل کننده برنامه پذیری است که از خانواده کامپیوتر ها بشمار می آید. این کنترل کننده که عمدتاً در مقاصد صنعتی بکار میرود ورودی ها را میگیرد و بر اساس برنامه ای که در حافظه آن نوشته شده خروجی های لازم را برای ماشین یا فرآیندی که تحت کنترل آنست صادر مینماید.



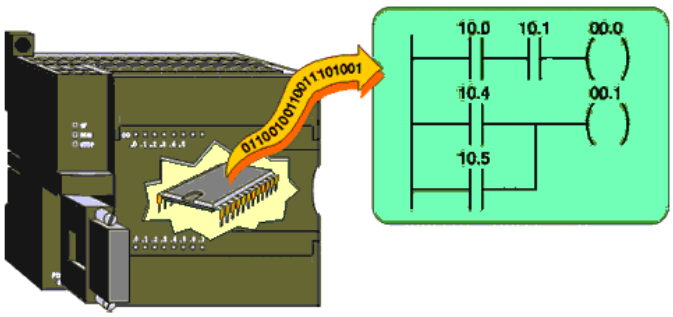
بنا بر این در نگاه اول PLC از سه قسمت اصلی یعنی مدولهای ورودی ، CPU و مدولهای خروجی تشکیل شده است.مدول ورودی سیگنالهای متنوع دیجیتال یا آنالوگ را از Field قبول میکند وسپس آنها را به سیگنالهای منطقی (0 و 1) که برای CPU قابل پردازش باشد تبدیل می نماید. CPU مطابق با برنامه ای که قبلاً کاربر در حافظه آن ذخیره کرده است دستورات کنترلی را اجرا کرده و خروجی لازم را بصورت سیگنالهای منطقی به مدولهای خروجی میفرستد. این مدولها سیگنال های مزبور را به فرم دیجیتال یا با تبدیل به آنالوگ به تجهیزات Field مانند عملگرها(Actuators) ارسال مینمایند.



قبل از اینکه PLC در صنعت مورد استفاده قرار گیرد مدارهای کنترلی کاملاً سخت افزاری (Hard-wired) بودند این مدارهای بر اساس رله ها طراحی و سپس سیم بندی میشدند. بزرگترین عیب این روش آن بود که کوچکترین تغییری در سیستم کنترل مستلزم تغییر سخت افزار و سیم کشی بودکه علاوه بر هزینه زیاد زمان زیادی را نیز برای اجرا نیاز داشت بعلاوه در هنگام بروز خطا کار عیب یابی (Troubleshooting) این مدارها چندان ساده نبود.



سیستم جدید یعنی PLC مسایل فوق را بهمراه نداشت . بسادگی قابل برنامه ریزی بود و تغییر در سیستم کنترل با تغییر در نرم افزار برنامه کنترل بسهولت امکان پذیر میشد.



مزایتهای فوق همراه با مزایای دیگری چون کوچکتر شدن ابعاد سیستم کنترل ، عیب یابی سریعتر ، خرابی کمتر ، توانایی اجرای فانکشن های پیچیده ، توانایی تبادل اطلاعات با سیستمهای دیگر و ... موجب شد که مدارهای رله ای بسرعت میدان را برای حضور PLC ها خالی کنند.

بیان تاریخچه PLC از حوصله این کتاب خارج است همینقدر میتوان اشاره کرد که اولین PLC ها در سال ۱۹٦۸ ساخته شدند. در دهه ۷۰ قابلیت برقراری ارتباط (Communication) به آنها اضافه شد . در دهه ۸۰ پروتکل های ارتباطی استاندارد شد و بالاخره در دهه ۹۰ استاندارد زبانهای برنامه نویسی PLC یعنی استاندارد IEC1131 ارائه گردید این استاندارد در صفحات بعد توضیح داده شده است.

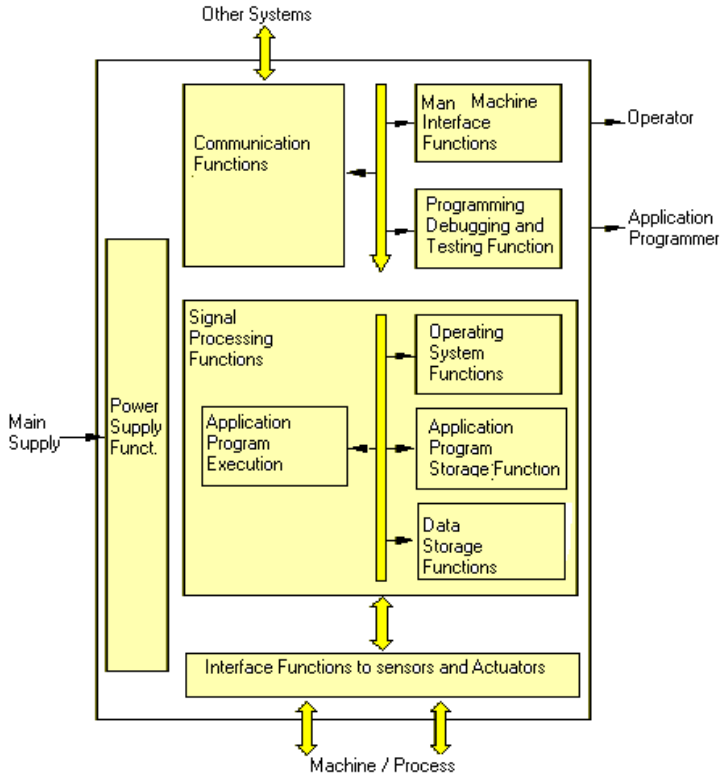
۱-۲ استاندارد IEC1131

در سال ۱۹۷۹ یک گروه متخصص در IEC (International Elctro-technical Commission) کار بررسی جامع PLC ها را شامل سخت افزار ، برنامه نویسی و ارتباطات بعهده گرفت. هدف این گروه تدوین روشهای استانداردی بود که موارد فوق را پوشش دهد و توسط سازندگان PLC بکار گرفته شود. این کار حدود ۱۲ سال بطول انجامید ونهایتاً پس از بحثهای موافق و مخالفی که انجام شد استاندارد IEC1131 شکل گرفت و جنبه های مختلف این وسیله از طراحی سخت افزار گرفته تا نصب ، تست ، برنامه ریزی و ارتباطات آن را زیر پوشش قرار داد. خواننده محترم با مشاهده لیست بخشهای مختلف این استاندارد که در ضمیمه ۱ آورده شده میتواند تصور بهتری از تلاشی که در این زمینه صورت گرفته داشته باشد.این استاندارد که با همکاری برخی از سازندگان بزرگ PLC از جمله زیمنس شکل گرفته بود از آن به بعد توسط ایشان به کار گرفته شد و سعی نمودند محصولات خود را با آن منطبق سازند.

استاندارد IEC1131 از بخشهای زیر تشکیل شده است :

بخش ۱ - اطلاعات کلی (General Information)

این بخش ضمن تعریف بخشهای مختلف PLC و وسایل جانبی آن (مانند وسایل برنامه ریزی ، تجهیزات HMI و ...) عملکرد هر قسمت مانند CPU، منبع تغذیه ، ورودیها و خروجیها و ... را تشریح کرده و یک ساختار کلی را بعنوان الگو مطابق شکل زیر ارایه نموده است



بخش ۲ – ملزومات سخت افزاری و آزمایشها (Equipment Requirements and Tests)

ایـن بخـش حداقـل ملزومات برای ساخت ، سرویس ، انبار کردن ، حمل و نقل ، عملکرد و ایمنیPLC هاو وسایل جنبی آنها را بیان کرده و تسـتهای کاربـردی مـربوطه را توضیح مـیدهد . در ایـن بخـش پیش فرض آنست که PLC و متعلقات آن در محیط های صنعتی بکار گرفته میشوند.

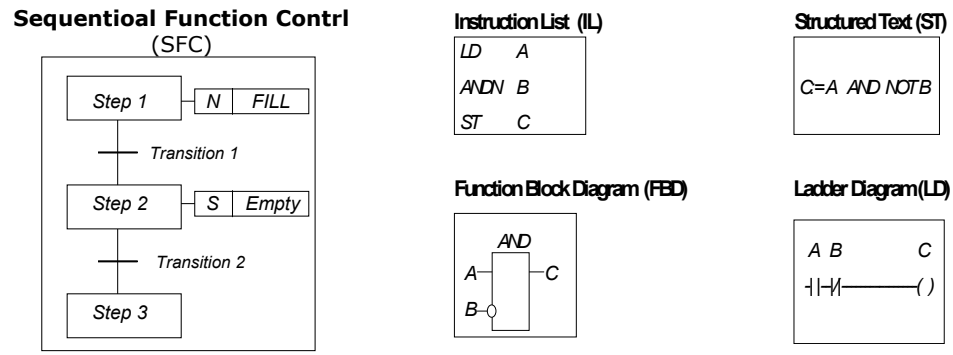
بخش ۳ - زبانهای برنامه نویسی (Programming Languages)

در این بخش انواع دیتاهایی که میتوانند در برنامه نویسی استفاده شوندمانندBool ، Byte، Time , Date , Word ,Real, Integer تعریف شده اند . همچنین POU ها یعنی (Program Organization Units) مانند فانکشن (FC) و فانکشن بلاک (FB) مشخص گردیده اند . وجه تمایز FB از FC اینگونه تعیین شده که FB علاوه بر الگوریتم برنامه ، دیتاها را نیز شامل میشود.

IEC در این بخش چهار زبان برنامه نویسی که قبلاً نیز بکار میرفت را انتخاب کرده و یک زبان جدید نیز بر آن افزوده و جمعاً ۵ زبان برنامه نویسی PLC ها را بعنوان استاندارد ارائه نموده است. این زبانها عبارتند از:

- **IL** یا Instruction List یک زبان سطح پایین و از زبان های قبلی PLC است که بصورت متنی میباشد . این زبان بیشتر شبیه زبان اسمبلر های میکروپروسسور است.
- **FBD** یا Function Block Diagram زبان گرافیکی است که قبلاً نیز مورد استفاده قرار میگرفت . در FBD برنامه نویسی توسط یک سری بلوکهای پایه که در کنار هم قرارمیگیرند انجام میشود.
- **LD** یا Ladder Diagram روش گرافیکی است که قبلاً نیز استفاده میشد ولی بصورت پیشرفته تر عرضه شده است. در روش جدید LD و FBD میتوانند بصورت توام در برنامه بکار روند.
- **ST** یا Structured Text زبان جدیدی است که IEC به ٤ زبان قبلی افزوده است. ST یک زبان سطح بالا شبیه C و پاسکال است و کاربردی عالی بویژه در الگوریتم های پیچیده ریاضی را داراست.
- **SFC** یا Sequential Function Control نیز روش جدیدی است .در این روش برنامه به مراحلی که ترتیب الگوریتم های کنترلی را نشان میدهد تقسیم میگردد و شامل Step های مختلف برنامه است هرگاه شرایطی که در بخش Transition مشخص شده برآورده گردید Step قبلی غیر فعال و Step بعدی فعال میگردد. .

در شکل ، چهار زبان بصورت ساده مورد مقایسه قرار گرفته اند. اینکه چه زبانی انتخاب شود بستگی به ساختار سیستم کنترل و نیز تاحدودی بستگی به سلیقه استفاده کننده دارد.

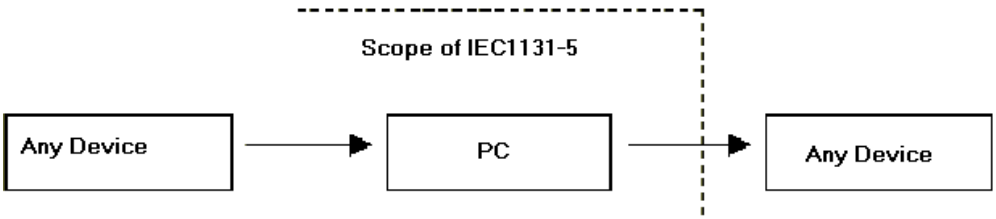


بخش ٤ - راهنمای کاربران (User Guidelines)

بخش چهارم راهنمای کاربر نهایی (End User) برای انتخاب و مشخص کردن ملزومات سیستمی است که سخت افزار ، نرم افزار و ارتباطات در آن منطبق با استاندارد IEC1131 باشد.

بخش ٥ – ارتباطات (Communications)

در این بخش جنبه های ارتباطی از دیدگاه کنترل کننده تشریح شده است . شکل زیر حوزه ای که این استاندارد برای بخش ارتباطات کنترلر تعیین کرده است را نشان میدهد.



بخش ٦ – این بخش خالی است و برای استفاده در آینده رزرو شده است.

بخش ٧- برنامه نویسی کنترل فازی (Fuzzy Control Programming)

این بخش که در سال ٢٠٠١ به استاندارد اضافه شد برنامه نویسی کنترل فازی را معرفی مینماید و برای کاربرانی که بخوبی با بخش سوم استاندارد آشنا باشند قابل استفاده است.

بخش ٨ – راهنمای کاربرد زبانهای برنامه نویسی (Guidelines for the application of programming languages)

در بخش ٤ مجموعه ای برای راهنمایی کاربران ارائه شده بود که جنبه های مختلف PLC را پوشش میداد ولی بخش ٨ صرفاً راهنمای کاربران برای استفاده از زبانهای برنامه نویسی است که در بخش ٣ معرفی شده اند.

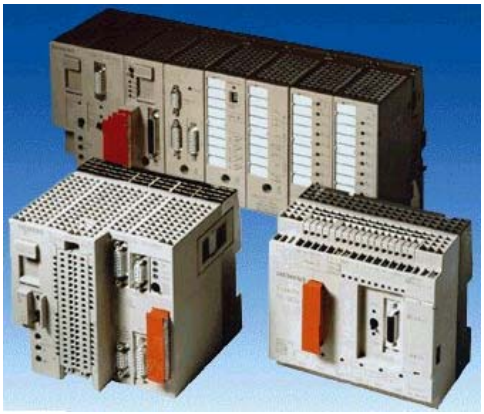
لیست بخشهای مختلف استاندارد IEC1131 در ضمیمه ١ آورده شده است.

۳-۱ PLC های مختلف زیمنس

در طبقه بندی محصولات زیمنس PLC ها در زیر مجموعه محصولات SIMATIC قرار میگیرند. برخی ازآنها بصورت Compact طراحی و ساخته شده اند به این معنا که منبع تغذیه و CPU و مدولهای ورودی و خروجی بصورت یکپارچه در کنار هم بیکدیگر متصل هستند و یک واحد تلقی میشوند. برخی دیگر بصورت مدولار (Modular) هستند که بر خلاف نوع Compact کاربر میتواند مدولهای دلخواه از آن خانواده را بسته به نیاز خود انتخاب و در کنار هم قرار دهد.

PLC های زیمنس را میتوان به پنج خانواده زیر تقسیم نمود:

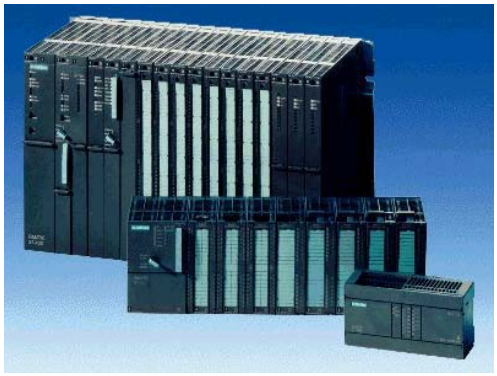
Simatic S5



این PLC ها که نسبتاً قدیمی هستند انواع مختلف دارند برخی مانند S5-90U یا S5-95U بصورت Compact بوده و حوزه عملکرد محدود دارند . برخی دیگر مانند S5-100U و S5-115U مدولار بوده و برای سیستمهای کنترلی با ابعاد متوسط بکار میروند . برای حوزه های عملکرد وسیع PLC های دیگری با نامهای S5-135U و S5-155U از این خانواده عرضه شده اند.

برنامه نویسی PLC های فوق با نرم افزار STEP 5 انجام میگیرد.

Simatic S7



این PLC ها بعد از S5 عرضه شده اند و خود به سه خانواده مختلف تقسیم میشوند S7-200 بصورت Compact بوده و برای سیستمهای کنترلی کوچک بکار میرود . S7-300 مدولار است و عملکرد متوسط دارد S7-400 نیز مدولار است ولی میتواند حوزه عملکرد وسیع داشته باشد.

این PLC ها با نرم افزار STEP7 برنامه نویسی و پیکر بندی میشوند.

LOGO! Logic Modules

LOGO کنترل کننده ساده و ارزان قیمتی است که برای کارهای کنترلی کوچک (مانند ساختمانها یا ماشینهای کوچک) کاربرد دارد . این PLC بصورت Compact است و برنامه ریزی آن توسط کلیدهای روی آن انجام میشود. برای برنامه ریزی از طریق کامپیوتر باید نرم افزار LOGO! Soft Comfort نصب گردد.

Simatic C7

C7 ترکیبی است از S7-300 و Operator Control . علاوه بر اینکه کار کنترلی را انجام میدهد بر روی نمایشگر آن میتوان پیغامها ، رخداد ها ، مقادیر مربوط به فرآیند را دید و فانکشن هایی را نیز توسط صفحه کلید روی آن اعمال نمود.

C7 کمپکت بوده و انواع مختلف دارد که توانایی آنها با هم متفاوت است.

برای برنامه نویسی این PLC ها باید علاوه بر STEP7 نرم افزار Protool نیز روی کامپیوتر نصب شود.

Simatic 505

سری 505 که خود انواع مختلف دارد برای کاربرد در حوزه های کوچک و متوسط طراحی شده است همه اعضای این خانواده بصورت Compact عرضه میشوند و برنامه نویسی آنها با نرم افزار TISOFT انجام میگیرد.سخت افزار و نرم افزار مربوط به Texas Instruments میباشد.

از موارد فوق آنچه در این کتاب مورد بحث قرار میگیرد خانواده S7 میباشد که اجزای آن در صفحه بعد معرفی شده اند.

۱-٤ خانواده S7

S7-200

- یک micro PLC ارزان قیمت است.
- میتواند برای مقاصد ساده تا نسبتاً پیچیده کنترلی بکار رود.
- نصب ، برنامه نویسی و کار با آن ساده است.
- بصورت Compact عرضه میشود و I/O های آن On-Board است.
- انواع مختلف دارد و در برخی از انواع آن میتوان مدول اضافی نیز در کنار CPU قرار داد .
- برنامه نویسی آن با نرم افزار Step7-Micro/Win انجام میشود.

S7-300

- یک mini PLc است
- حوزه عملکرد آن متوسط است.
- مدولار است
- مدولهای آن تنوع زیاد دارد.
- بسهولت قابل توسعه است
- برنامه نویسی آن با STEP7انجام میشود.

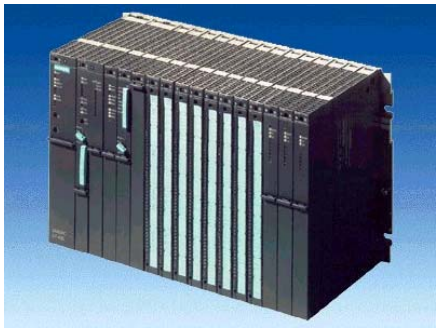
S7-300F

- برای سیستمهایی که نیاز به ایمنی زیاد دارند یا اصطلاحاً Fail-Safe هستند طراحی شده است
- پایه آن S7-300 است.
- در انتهای کد CPU حرف F معرف این نوع است مانند CPU 315F

S7-300C

- شبیه S7-300 است با این تفاوت که CPU همراه با مدول دیگری مانند ورودی / خروجی بصورت Compact عرضه شده است.
- در انتهای کد CPU حرف C معرف این نوع است مانند CPU 314C

S7-400



- حوزه عملکرد وسیع دارد.
- مدولار است
- حجم زیادی از سیگنالها را میتواندپوشش دهد.
- براحتی قابل توسعه است
- در مقایسه با S7-300 سرعت پردازش بالاتر ، حافظه بیشتر و امکانات وسیعتری را داراست.
- برنامه نویسی آن با STEP7 انجام میشود

S7-400H

پایه آن همان S7-400 است ولی در جایی که High Availability مورد نیاز است بکار میرود مانند :

- پروسه ای که اگر متوقف شود منجر به خسارت زیاد میشود مثلا محصول گرانقیمتی از بین میرود.
- جایی که هزینه راه اندازی مجدد سیستم پس از رفع عیب بالاست.
- جایی که بهره برداری از پروسه بدون مانیتورینگ و با حداقل پرسنل تعمیراتی انجام میشود.

به این سیستم Redundant نیز گفته میشود و در آن دو عدد CPU بعنوان رزرو گرم یکدیگر (Hot-Standby) قرار میگیرند . در صورت بروز خطا روی یکی از CPUها یا مدولهای مربوط به آن ، سیستم بطور اتوماتیک در زمان بسیار کوتاهی به CPU دیگر سوئیچ میشود. در طول مدت سوئیچ شدن خروجیها ثابت میمانند تا مشکلی در فرآیند پیش نیاید. پس از عملیات سوئیچ میتوان مدول معیوب را تعویض یا رفع عیب کرد.
برای برنامه ریزی و پیکر بندی این سیستم علاوه بر Step7 باید پکیج H-System نیز نصب گردد.

S7-400FH

- پایه آن S7-400 است
- توانایی های S7-400H را داراست.
- توانایی های F-system را نیز دارد یعنی برای کاربردهایی که درجه ایمنی بالا نیاز دارند نیز مناسب است.

از خانواده S7 آنچه در این کتاب مورد بحث قرار میگیرد صرفا مواردی است که توسط نرم افزار Step7 برنامه نویسی و پیکر بندی میشود. بنابراین S7-200 خارج از این چارچوب بوده و به آن پرداخته نمیشود.

۱-۵ Step7 و نسخه های مختلف آن

در نگاه اول نرم افزار Step7 را باید به دو نوع زیر تقسیم نمود:

۱. **Step7-MicroWin** که برای PLC های S7-200 بکار میرود.

۲. **Step7** که برای S7-400 , S7-300 و همچنین C7 بکار میرود.

مورد دوم یعنی Step7 نسخه های مختلفی داردکه آخرین آنها نسخه Step7 V5.3 میباشد از مارس 2004 عرضه شده است وتفاوتهای مختصری با نسخه قبلی آن یعنی نسخه 5.2 دارد

Step7 (V5.2)

Step7 V5.2 از دسامبر 2002 به بازار آمد و جایگزین نسخه قبلی یعنی Step7V5.1 گردید. بطور کلی این نرم افزار قادر به انجام امور زیر روی کنترل کننده ها و متعلقات آنها میباشد:

- پیکر بندی سخت افزار و تنظیم پارامترهای آن
- پیکر بندی و تنظیم ارتباطات (شبکه)
- برنامه نویسی
- تست ، راه اندازی و عیب یابی
- آرشیو سازی

در V5.2 نسبت به نسخه قبلی امکانات جدیدی اضافه شده است که ازمهمترین آنها میتوان امکان پیکربندی سخت افزار در مد کاری RUN یا اصطلاحاً قابلیت CiR (Configuration in Run) را نام برد.در فرآیندهای پیوسته که هیچ توقفی نباید ایجاد شود توسط این قابلیت میتوان در مد RUN پیکر بندی سخت افزار را تغییر داد مثلاً یک مدول جدید اضافه کرد. در اینحال وقفه ای که به پروسه داده میشود کمتر از یک ثانیه خواهد بود و در طول اینمدت ورودیها و خروجیها آخرین حالت خود را حفظ میکنند.

CiR برای CPU های S7-400 از FirmWare 3.1 به بعد امکان پذیر است.

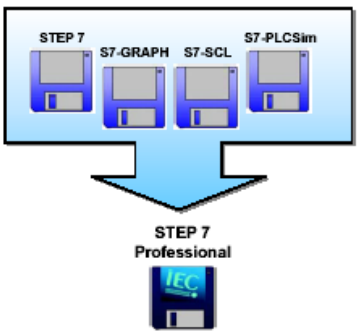
Step7 Mini , Step7 lite

ایندو نسخه هایی از Step7 هستند که نسبت به Step7 پایه (یعنی V5.1 یا V5.2) امکانات کمتری درآنها وجود دارد و برای کارهای نسبتاً ساده تر طراحی شده اند. بعنوان مثال نسخه Lite :

- فقط برای S7-300 و C7 قابل استفاده است.
- برنامه نویسی فقط به سه زبان LAD، STL و FBD امکان پذیر است.
- ارتباط با شبکه را ساپورت نمیکند.

Step7 Proffesional

در این نسخه علاوه بر Step7 V5.2 پکیج های دیگری که قبلاً بصورت Optional عرضه میشدند یکجا ارئه شده اند که عبارتند از :



S7-PLCSIM سیمولاتور نرم افزاری است

S7-PDIAG برای تشخیص عیب بکار میرود

S7-Graph V5.2 برای برنامه نویسی بصورت SFC بکار میرود

S7-SCL V5.2 برای برنامه نویسی بصورت ST بکار میرود

تذکر :
آنچه در این کتاب تحت عنوان برنامه نویسی و کار با Step 7 بیان میشود مربوط به نسخه Proffesional است .

۱-٦ مزیتهای Step7 نسبت به Step5

Step7 نسبت به Step5 نقاط قوت و مزیتهای متعددی دارد اما از مهمترین ویژگیهای آن میتوان به دو مورد زیر اشاره کرد:

۱- تطابق با استاندارد IEC1131 :

زیمنس مدعی است که این استاندارد بویژه بخش سوم آنرا که مربوط به برنامه نویسی است در Step7 تا حد زیاد رعایت کرده است. در حالیکه Step5 فاقد این تطابق میباشد.

۲- قابلیت پیکر بندی سخت افزار و شبکه از طریق نرم افزار :

در STEP 5 صرفاً امکانات تهیه برنامه جهت PLC وجود داشت ولی در STEP 7 علاوه بر برنامه نویسی میتوان سخت افزار سیستم و شبکه را از طریق آن پیکر بندی نمود.

۱-۷ ملزومات نصب و استفاده از Step7 Proffesional

نرم افزار جهت اجرا موارد زیر را بعنوان حداقل لازم دارد:

۱- سیستم عامل

سیستم عامل کامپیوتر میتواند هر کدام از موارد زیر باشد:

Windows 95
Windows 98
Windows Me
Windows NT4 workstation (SP6a)
Windows 2000 Proffesional (SP2)
Windows XP Proffesional

۲- مشخصات سخت افزاری کامپیوتر

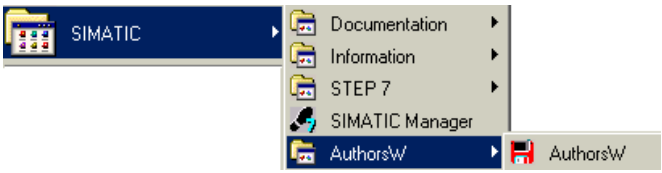
مشخصات CPU و RAM و هارد دیسک کامپیوتر با توجه به سیستم عامل آن مطابق با جدول زیر باشد. بعضاً ممکن است بدلیل عدم وجود فضای کافی روی هارد دیسک ، نصب Step7 بدون هیچ پیغامی قطع شود.

		Win95	Win98	Win Me	Win NT	Win 2000	Win XP
Processor	حداقل	P133				P166	P300
	پیشنهادی	P III به بالا					
RAM	حداقل	16	24	32	64	64	64
	پیشنهادی	32	32	64	64	128	128
فضای خالی Hard Disk	حداقل 450 MB						

۳- مجوز استفاده از نرم افزار (Autoroziation)

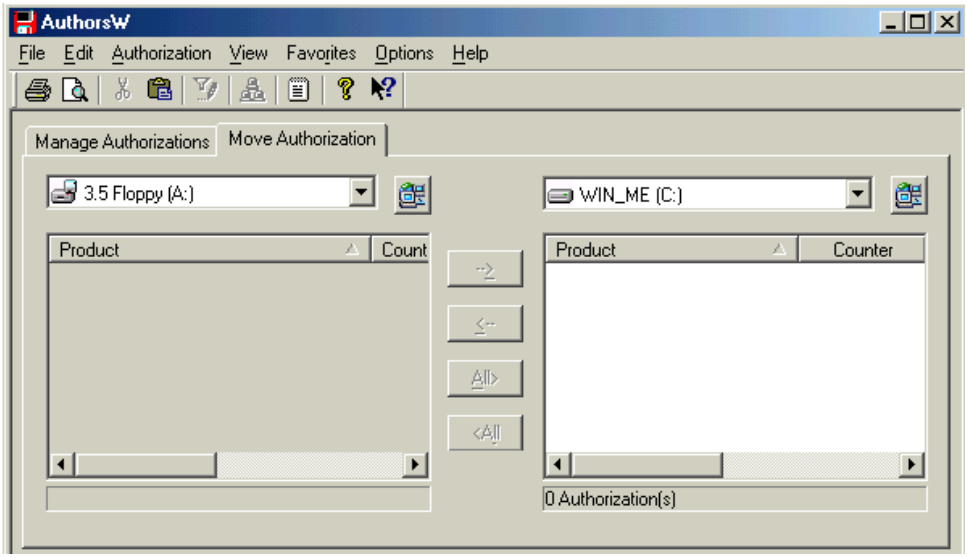
برای اسفاده از STEP7 مجوز خاص آن برنامه لازم میباشد و برنامه از این نظر داری قفل نرم افزاری است که روی دیسکت جداگانه قرار دارد . با استفاده از این دیسکت که معمولاً Single License است میتوان یکبار برنامه را روی کامپیوتر نصب نمود و پس از آن شمارنده داخلی قفل روی دیسکت صفر میگردد.اگر نیاز باشد که Authorization به کامپیوتر دیگری انتقال یابد ابتدا دیسکت را داخل درایو کامپیوتر قبلی قرار داده و Authorization را از روی آن برمیداریم مشاهده میشود که شمارنده مربوطه یکی افزایش می یابد سپس دیسکت را در درایو کامپیوتر جدید قرار داده و Authorization را به هارد آن منتقل مینماییم.

گذاشتن و برداشتن Authorization توسط زیر برنامه AuthorzW انجام میشود که با کلیک روی Start در صفحه Desktop ویندوز و از مسیر زیر قابل اجراست:



دیسکت را داخل درایو قرار داده و برنامه فوق را اجرا میکنیم با کلیک رو Tab با عنوان Move Authorization پنجره ای مانند شکل زیر خواهیم داشت که در سمت چپ آن عنوان برنامه های مختلف در ستون Product و شمارنده مربوطه در ستون Counter مشاهده میشود یکی از این موارد Step7 V5.2 است . با انتخاب آن و کلیک کردن روی علامت **≥--** میتوان آنرا به آدرس دلخواه روی هارد کامپیوتر انتقال داد.

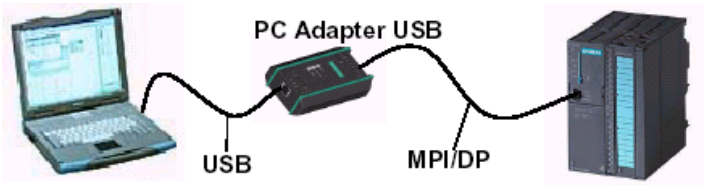
بدیهی است برداشتن Authorization با استفاده از علامت **--≤** انجام میشود.



٤- کارت یا مبدل ارتباطی بین کامپیوتر و PLC که میتواند یکی از انواع زیر باشد:

- PC Adaptor

این آداپتور از یکطرف به پورت MPI کنترل کننده وصل میشود و از سمت دیگر به کامپیوتر. دو نوع آداپتور وجود دارد که یکنوع به پورت USB و نوع دیگر به RS232 متصل میگردد.شکل زیر آداپتور قابل اتصال به پورت USB را نشان میدهد.



- **کارت برای نصب در اسلات ISA یا PCI کامپیوتر**

با نصب این کارت خروجی مستقیماً توسط کابل و کانکتور به PLCمتصل میگردد و نیاز به آداپتور بیرونی نمی باشد (مانند کارت CP5611 شکل زیر)

- **کارت PCMCIA**

این کارت در اسلات Notebook نصب میگردد مانند کارت CP5511

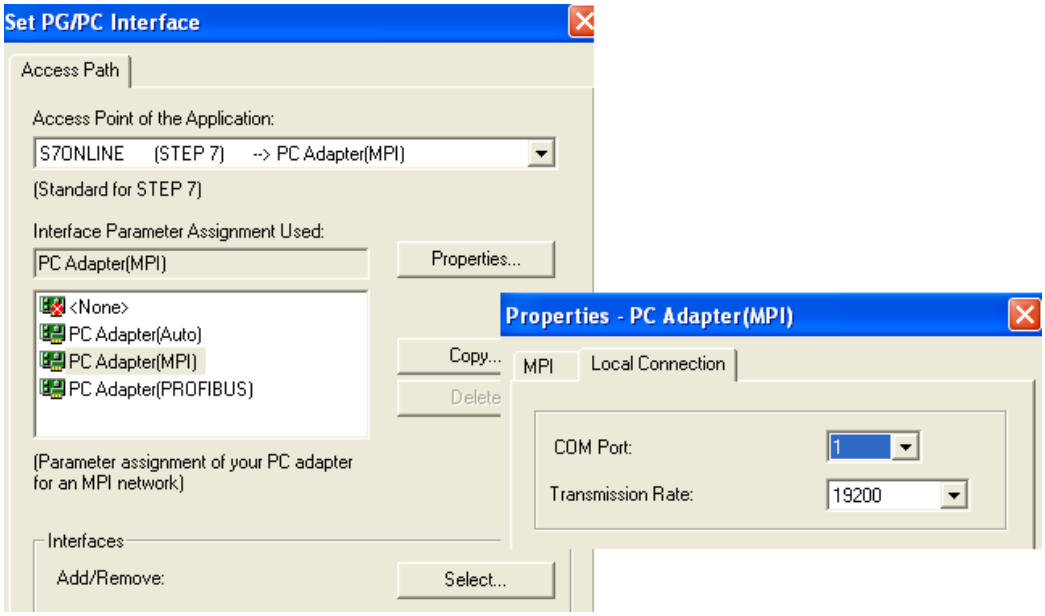
تذکر:

اگر بجای کامپیوتر از PG استفاده شود نیازی به استفاده از مبدلهای فوق نیست . PG های زیمنس دارای پورت خروجی که مستقیماً به PLC وصل میگردد هستند.

پس از اینکه کارت ارتباطی در اسلات کامپیوتر قرار گرفت و توسط کابل ارتباطی به پورت PLC متصل گردید باید تنظیمهای لازم انجام پذیرد. برای آداپتور نیز ابتدا آنرا به پورت PLC وصل کرده سپس ارتباطش را با کامپیوتر توسط کابل ارتباطی برقرار می کنیم.


Set PG/PC Interface

تنظیمات لازم توسط برنامه Set PG/PC Interface که آیکون آنرا بعد از نصب Step7 میتوان در Control Panel مشاهده کرد امکان پذیر است. فرض کنید تنظیمات آداپتور را میخواهیم انجام دهیم با کلیک کردن روی آیکون فوق پنجره ای مانند شکل زیر باز میشود.



MPI در حالتی انتخاب می شود که آداپتور به پورت MPI مربوط به PLC متصل باشد.

Profibus در حالتی انتخاب می شود که آداپتور به پورت DP مربوط به PLC متصل باشد.

Auto هر دو حالت فوق را پوشش میدهد.

با کلیک روی Properties میتوان مشخص کرد که آداپتور به کدام پورت سریال (Com1 یا Com2) متصل شده است. سایر پارامترها را معمولاً برای آداپتور لازم نیست تغییر بدهیم.سرعت پیش فرض 19200 میباشد اگر 38400 انتخاب شود بشرط اینکه کابل ارتباطی آنرا ساپورت کند(مثلاً کابل 6ES7 972-0CA20-0XA0 نباشد) باید این تنظیم توسط Dip سوئیچ روی آداپتور در حالتی که اکتیو نیست نیز انجام شود.

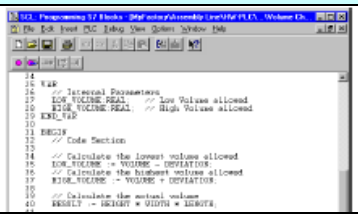
نکته دیگری که باید خاطر نشان شود اینست که سیستم عامل های Windows 2000 , XP , 95,98 , Me بطور اتوماتیک کارت یا آداپتور را میشناسند ولی در Windows NT باید بصورت دستی اختصاص داده شود چون NT قابلیت Plug and Play را ندارد.

۱-۸ نرم افزارهای جنبی و مرتبط با Step7

برخی نرم افزار های دیگر که توسط زیمنس در خانواده Simatic عرضه شده اند و بعضاً مکمل Step7 هستند با تقسیم بندی به سه دسته Engineering، Runtime و HMI در جدول زیر نشان داده شده اند.

Engineering Tools

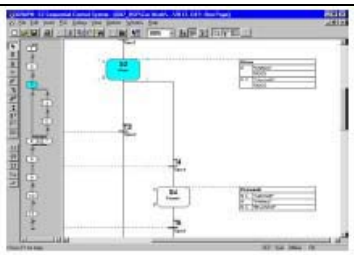
S7 SCL



زبان برنامه نویسی سطح بالا میباشد که با زبان ST ذکر شده در استاندارد IEC1131-3 تطبیق دارد و برای PLC های S7-300 (CPU314 و بالاتر) و S7-400 و C7 بکار میرود. همانطور که قبلاً اشاره شد این نرم افزار در نسخه Step7 Professional موجود است.

S7 HiGraph: برای کنترل ترتیبی بصورت گرافیکی ، با ابزارهای پیشرفته و در PLC های S7-300، S7-400 و C7 بکارمیرود.

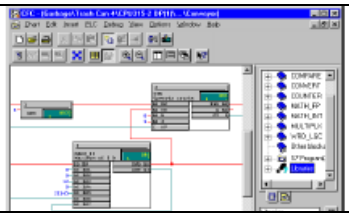
S7 GRAPH



برنامه نویسی بصورت گرافیکی است که برای کنترل ترتیبی بکار میرود وبا زبان SFC مندرج در استاندارد IEC1131-3 تطبیق داردو برای PLC های S7-300 (CPU315 و بالاتر) و S7-400 و C7 بکار میرود این نرم افزار نیز در نسخه Step7 Professional موجود است

S7 PLCSIM :سیمولاتور نرم افزاری است که برای تست برنامه وقتی PLC در دسترس نیست بکار میرود. این نرم افزارنیز در نسخه Step7 Professional موجود است.

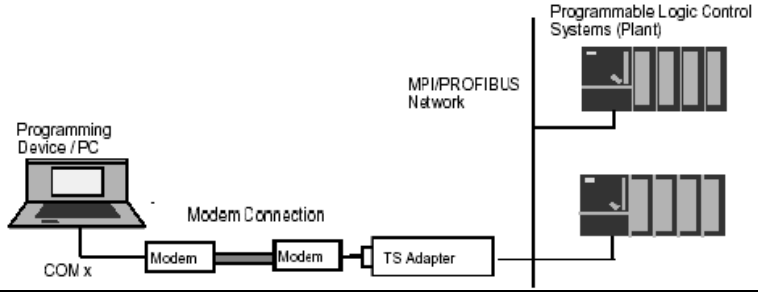
CFC



توسط این نرم افزار برنامه نویسی بصورت گرافیکی توسط یکسری بلوکهای از پیش تعیین شده طراحی و انجام میشود. این نرم افزار را باید جداگانه تهیه کرد و برای S7-300 و S7-400 و F/H Systems کاربرد دارد

S7 PDIAG: ابزار عیب یابی (Diagnostic) است که برای PLC های S7-300 (CPU314 و بالاتر) و S7-400 بکار میرود. در نسخه Step7 Professional موجود است.

TeleService:برای ارتباط با PLC از طریق خط تلفن بکار میرود. وقتی PLC توسط آداپتور خاص (TS) به مودم متصل باشد. با استفاده از کامپیوتر بصورت Remote میتوان آنرا از هر نقطه ای برنامه نویسی و رفع عیب کرد

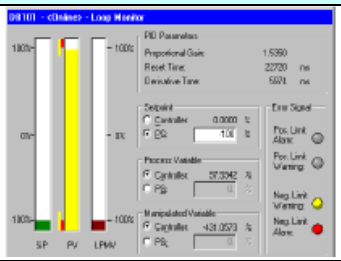


DOCPRO : برای مستند سازی بکار میرود با استفاده از آن میتوان پس از اتمام پیکر بندی و برنامه نویسی نقشه های Wiring و متن برنامه را با فرمت مناسب تهیه و چاپ کرد.

Runtime Software

Standard PID Control

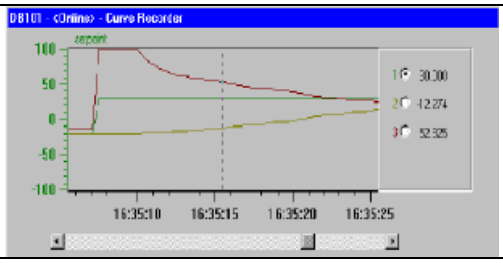
ابزار کمکی برای طراحی کنترل کننده های PID است که برای PLC های S7-300 (CPU31C و بالاتر) و S7-400 و C7 بکار میرود.



Fuzzy Control : برای کنترل فازی است و در مواردی بکار میرود که توصیف ریاضی پروسه مشکل یا ناممکن باشد. در برخی موارد ترکیب این روش با لوپ های PID نتیجه بهینه را برای کنترل سیستم بهمراه دارد.

Modular PID Control

ابزار ی است که برای طراحی لوپ های کنترلی پیچیده بکار میرود و دارای فانکشن ها و بلوکهای از قبل طراحی شده میباشد.

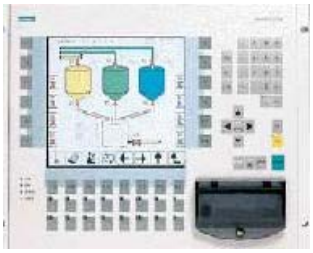


Neurosystem :شبکه های عصبی مورد استفاده در سیستم کنترل را میتوان با این ابزار طراحی کرد و آموزش داد.

PRODAVE MPI : برای پردازش ترافیک دیتا در شبکه MPI بین سیستمهای S7 ، M7 و C7 بکار میرود.

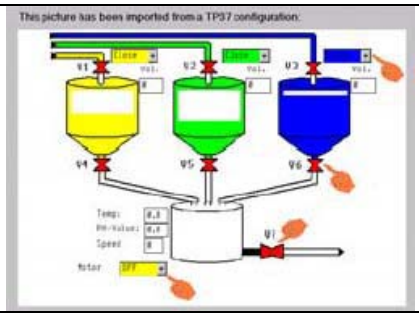
HMI Software

SIMATIC ProTool ابزار پیکر بندی است که برای سیستم های کنترل اپراتوری و بخش مانیتورینگ مربوط به C7 بکار میرود.



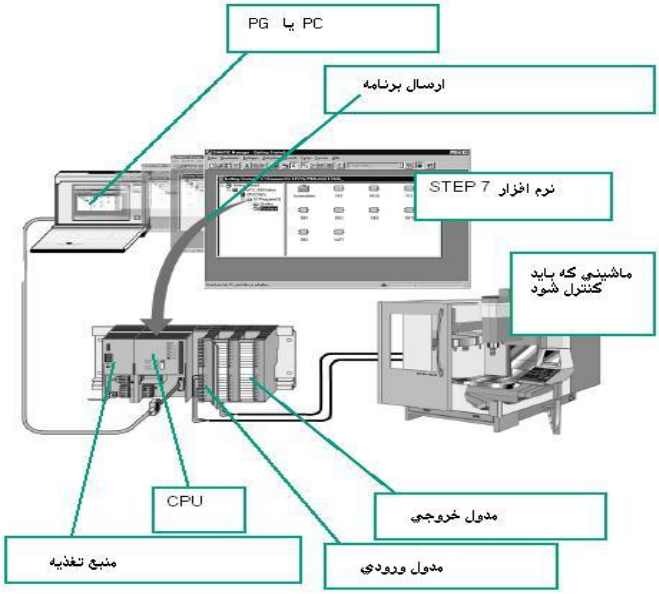
SIMATIC WinCC

نرم افزاری است که برای طراحی سیستم مانیتورینگ بکار میرود .



۱-۹ جایگاه نرم افزار Step7 در سیستم کنترل

در یک سیستم کنترل مبتنی بر PLC های S7 زیمنس اجزایی مانند شکل زیر را میتوان مشاهده کرد:



اکنون به شکل فوق بعنوان پروژه ای نگاه کنید که قرار است ابتدا طراحی سپس راه اندازی و بهره برداری شود با این فرض جایگاه Step7 در فازهای مختلف بشرح زیر است.

طراحی

در هنگام طراحی معمولاً نیازی به اینکه PLC یا ماشین در کنار PC یا PG موجود باشند نیست. فقط لازم است که قبل از شروع کار، فرآیند بخوبی مطالعه شده ، ورودی و خروجیها مشخص باشند و منطق سیستم کنترل معلوم شده باشد. بهتر است سخت افزار PLC نیز انتخاب شده باشد . با چنین معلوماتی میتوان کار طراحی را با استفاده از Step7 بصورت Offline یعنی بدون اتصال به PLC انجام داد.

تست On-Line

پس از تکمیل برنامه لازم است آنرا به PLC دانلود کنیم پس در اینحالت PC یا PG و نرم افزار و PLC ابزار کار هستند. اگر سیمولاتور نرم افزاری در دسترس باشد بسیاری از نیازهای این مرحله را مرتفع میکند و نیاز چندانی به PLC نیست.

Commisioning یا تست و راه اندازی

در این مرحله ماشین یا تجهیز نیز به جمع قبلی می پیوندد و برنامه بصورت عملی وابتدا در حالتی که ماشین بدون بار است یا از تجهیز هنوز بهره برداری نمیشود تست میگردد که به این مرحله تست سرد (Cold Test) نیز میگویند. سیگنالها بتدریج و نه یکدفعه وارد مدار میشوند و بخشهای برنامه قدم به قدم تست میگردد. پس از آن تست گرم ماشین شروع میشود یعنی ماشین زیر بار میرود واز تجهیز بصورت آزمایشی بهره برداری میشود تا سایر ورودی و خروجیهایی که در تست سرد فعال نبودند تست گردند. برای انجام تست های فوق وجود Step7 روی PC یا PG و ارتباط Online با PLC ضروری است.

Operation یا بهره برداری

پس از تکمیل مراحل تست و اعمال تغییرات لازم در برنامه PLC ، کار عادی فرآیند شروع میشود . در اینجا نیازی به PG یا PC و نرم افزار Step7 نیست. اگرچه باید برای نیازهای احتمالی در دسترس باشند.

Troubleshooting یا عیب یابی

در صورتی که مشکلی در کار بهره برداری از فرآیند پیش بیاید که ناشی از اجزای سیستم کنترلی باشد . مجددا بهPG یا PC و نرم افزار Step7 نیاز پیدا میشود. این برنامه با امکانات مختلفی که در آن تعبیه شده میتواند به شناخت عیب و رفع آن کمک زیادی بنماید.

۲– آشنایی با محیط Step 7

مشتمل بر :

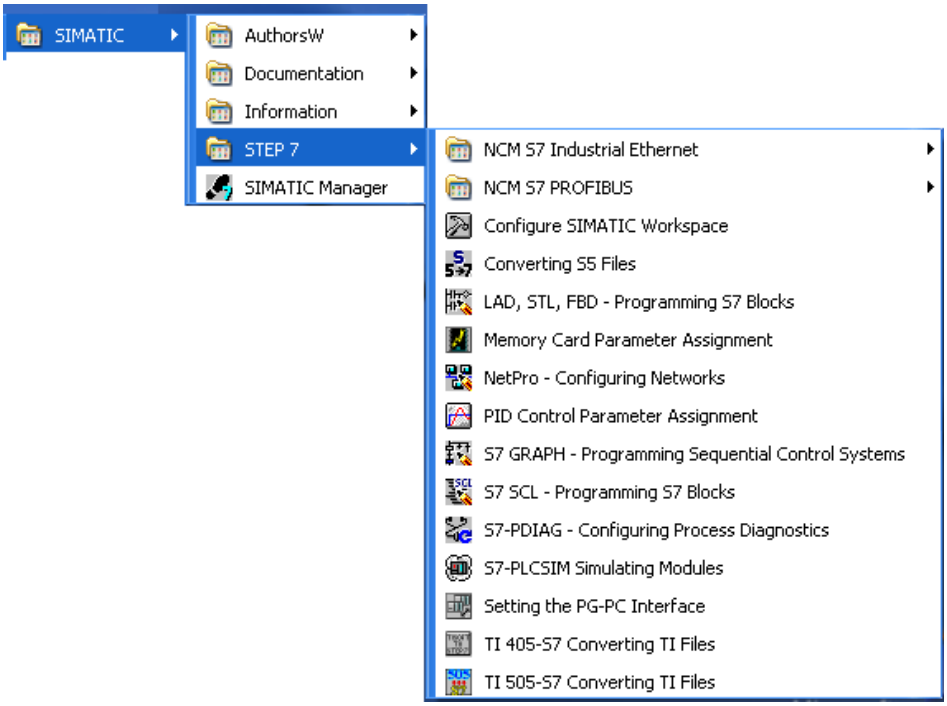
۲–۱ نگاهی به برنامه نصب شده

۲–۲ شروع کار با Step7

۲–۳ منوهای Simatic Manager

۲-۱ نگاهی به برنامه نصب شده

پس از نصب Step7 با کلیک روی Start در Desktop ویندوز ، برنامه های نصب شده را میتوان در مسیری مانند شکل زیر مشاهده کرد.

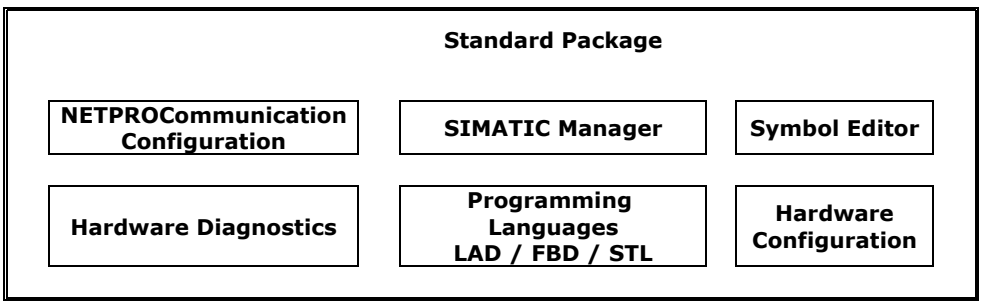


بطور معمول کاربر فقط نیاز دارد که Simatic Manager را اجرا کند و سایر برنامه ها و زیر برنامه ها در صورت لزوم توسط آن فرا خوانده میشوند. با این وجود میتوان زیر برنامه ها را بصورت مجزا نیز اجرا کرد. البته برخی زیر برنامه ها در لیست فوق ظاهر نمیشوند ولی موقع فراخوانی توسط Simatic Manager اجرا میشوند (مانند زیر برنامه ای که برای پیکر بندی سخت افزار بکار میرود)

در هنگام نصب Step7 Professional برنامه های زیر اختیاری(Optional) هستند که در صورت عدم انتخاب، نصب نشده و در لیست فوق ظاهر نمی شوند:

- S7 SCL
- S7-PDIAG
- S7-PLCSIM
- S7 Graph

بجز این موارد سایر برنامه ها بعنوان پکیج استاندارد Step7 محسوب میشوندواین پکیج دارای زیر برنامه هایی است که در شکل زیر نمایش داده شده است.



Symbol Editor

این برنامه برای ایجاد و اصلاح سمبل هایی بکار میرود که در برنامه نویسی PLC استفاده شده اند.

Diagnostics

با این ابزار می توان وضعیت تک تک مدولهای PLC را از نظر وجود یا عدم وجود خطا چک کرد.

Programming Language

برای برنامه نویسی با یکی از زبانهای STL , FBD , LAD بکار میرود.

Hardware Configuration

برای پیکر بندی و اختصاص پارامتر به سخت افزار بکار میرود .

Network Configuration

ابزار ساختار بندی شبکه است با استفاده از آن میتوان Node های ارتباطی و اتصالات را تنظیم نمود و به آنها پارامتر اختصاص داد.

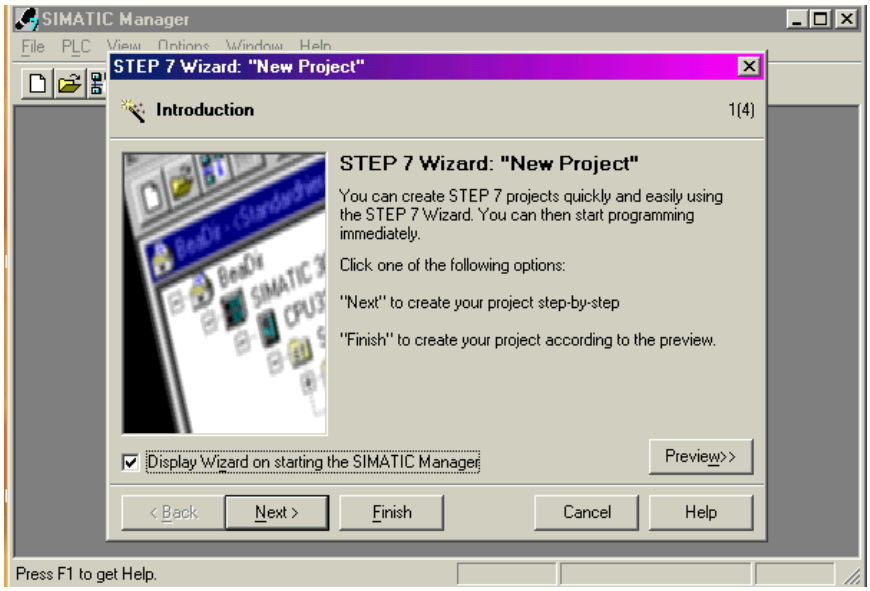
تذکر :

در پوشه Documentation که در شکل صفحه قبل نیز مشخص است میتوان لیست یکسری فایلهای PDF را مشاهده کرد . این فایلها مکمل Help برنامه هستند و کاربر با مطالعه آنها میتواند اطلاعات مفیدی از Step7 بدست آورد.

۲-۲ شروع کار با Simatic Manager

همانطور که اشاره شد Simatic Manager برنامه اصلی است که کاربر نیاز به اجرای آن دارد. پس از نصب Step7 آیکون این برنامه روی صفحه Desktop ظاهر میشود میتوان آنرا با استفاده از آیکون مزبور یا از مسیری که در شکل صفحه قبل آمده اجرا نمود.

با اجرای Simatic Manager معمولاً پنجره Wizard ظاهر میشود که توسط آن میتوان بخشی از امور مورد نیاز را انجام داد. انتخاب CPU و بلاکهای برنامه نویسی میتواند با Wizard دنبال شود. ولی بدلیل جامع نبودن آن توصیه میشود که کاربران ضمن غیرفعال کردن Wizard (توسط چک باکس پایین آن) کار را بصورت دستی ادامه دهند.

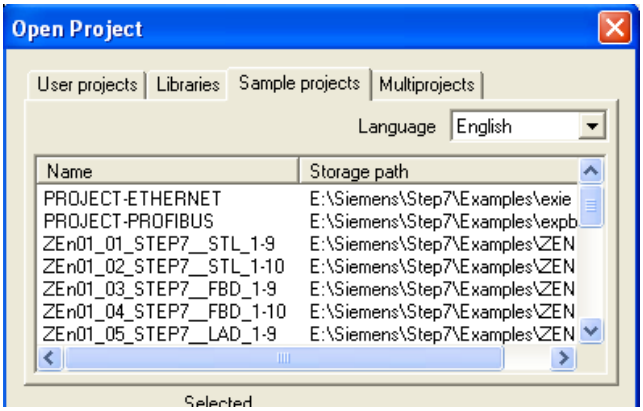


پس از Cancel کردن Wizard به محیط Simatic Manager وارد میشویم . قدمهای زیر بترتیب باید برداشته شوند.

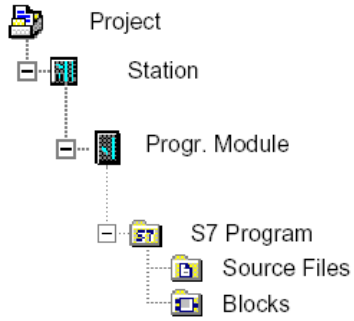
قدم اول : ایجاد پروژه (Project)

پروژه چیست؟ پروژه مجموعه ای است که اطلاعات پیکر بندی سخت افزار ، شبکه و برنامه نویسی PLC را در بر میگیرد.

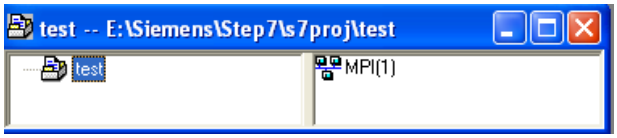
پروژه هایی که از قبل توسط کاربر تهیه وذخیره شده اند را میتوان توسط منوی File > Open باز کرد . زیمنس نیز برخی مثالهای نمونه را همراه با نرم افزار ارائه داده است که لیست آنها را پس از اجرای File > Open در بخش Sample Project شکل زیر می بینیم.



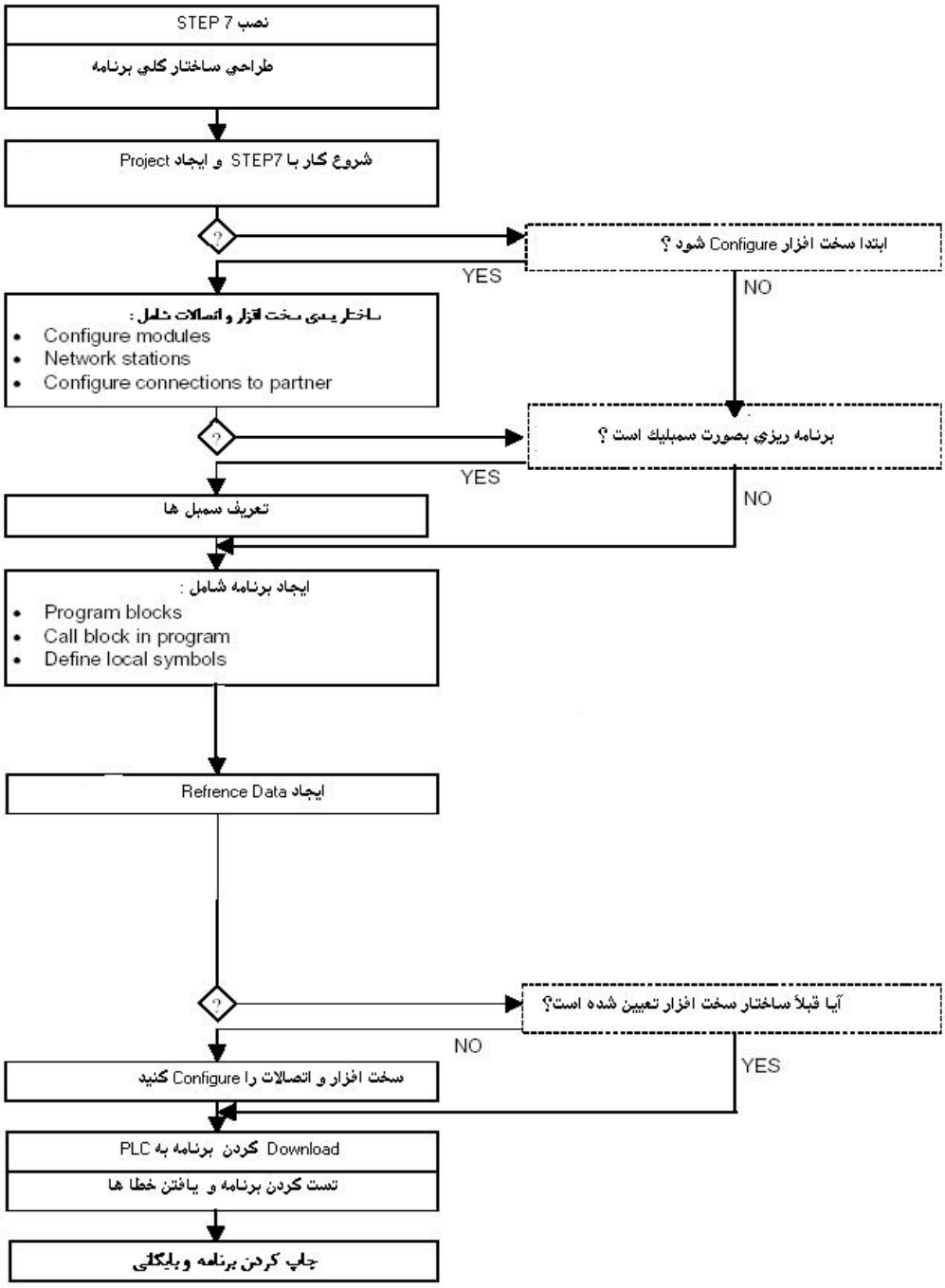
اگر یکی از این پروژه ها را باز کنیم مشاهده میکنیم که در پنجره سمت چپ برنامه سلسه مراتبی مانند شکل زیر ظاهر میشود:

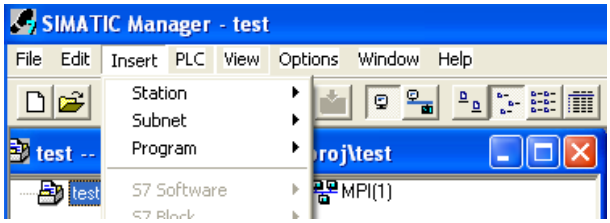


فرض کنیم از ابتدا میخواهیم پروژه جدیدی تعریف کنیم . توسط مسیر File>New>New Project> و وارد کردن اسم دلخواه (که در اینجا Test فرض شده) این کار را انجام داده و میبینیم که پنجره جدیدی مانند شکل زیر باز میگردد:



در این مرحله برای برداشتن قدم بعدی دو انتخاب وجود دارد. انتخاب اول اینکه ابتدا سخت افزار را پیکر بندی کنیم سپس به آن برنامه اختصاص دهیم و انتخاب دوم اینکه ابتدا برنامه نویسی را انجام داده و در آخر سخت افزار را معرفی کنیم. هر دو روش فوق امکان پذیر است و فلوچارت صفحه بعد نیز این امکان را نشان میدهد.


نصب STEP 7
طراحي ساختار كلي برنامه
شروع كار با STEP7 و ايجاد Project
ابتدا سخت افزار Configure شود ؟
YES
NO
ساختار يعني سخت افزار و اتصالات شامل :
Configure modules
Network stations
Configure connections to partner
برنامه ريزي بصورت سمبليك است ؟
YES
NO
تعريف سمبل ها
ايجاد برنامه شامل :
Program blocks
Call block in program
Define local symbols
Refrence Data ايجاد
آيا قبلاً ساختار سخت افزار تعيين شده است؟
NO
YES
سخت افزار و اتصالات را Configure كنيد
Download كردن برنامه به PLC
تست كردن برنامه و يافتن خطا ها
چاپ كردن برنامه و بايگاني

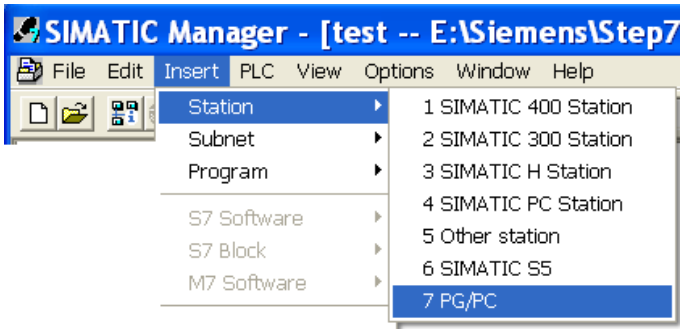


در روش اول بـا مـنوی Insert > Station سـخت افـزار را انـتخاب مـی کـنیم و در روش دوم بـا منوی Insert > Program بلاکهای برنا مه نویسی را وارد پروژه مینماییم. اما کدام روش بهتر است؟

در کاربـرد هایی که بر اساس مطالعات فاز طراحی ، سخت افزار سیستم مشخص و انتخاب شده است بهتر است در هنگام ایجاد پروژه نیز ابتدا سـخت افـزار انـتخاب گـردد و پـس از تنظیم پارامترها و تعیین آدرسها برنامه نویسی انجام شود.اگر در این موارد ابتدا برنامه را بنویسیم ممکن اسـت بعـداز انـتخاب سـخت افزار تغییردر آدرسهای برنامه ضرورت پیدا کند. در کاربردهایی که یک برنامه ممکن است برای سیستمهای مختـلف بکـار رود میـتوان بـدون انـتخاب سخت افزار برنامه نویسی را یکبار انجام داد سپس هر بار آن را به سخت افزار مورد نظر کپی نمود. پس در مجموع برای کاربردهای معمول ، روش اول بهتر است.

قدم دوم : ایجاد Station یا ایجاد Program

اگر روش اول مد نظر باشد. ا زمنوی Insert > Station مانند شکل زیر میتوان سخت افزار مورد نظر را انتخاب نمود:



بـا توضیحاتی که در بخش اول کتاب داده شد خواننده محترم با برخی از Station ها نظیر 300 و 400 و نوع H آشنا گردید.و احتمالاً بخاطر دارد که برای پیکر بندی H Station برنامه Step7 به تنهایی کافی نیست و لازم است پکیج H-System نیز نصب گردد.

در مورد سایر Station های موجود در پنجره فوق دانستن نکات زیر خالی از فایده نیست:

Simatic PC Station : برای مشخص کردن کارتهای ارتباطی (شبکه) که روی کامپیوتر نصب میشود بکار میرود.

Other Station: برای محصولاتی که ساخت زیمنس هستند بکار میرود.

Simatic S5: ارتباط PLC از نوع S5 با سیستم موجود (S7) را تعریف میکند. توجه شود که مدولهای S5 با برنامه Step7 قابل پیکر بندی نیستند و سیستم S5 فقط بصورت یک باکس ظاهر میشود.

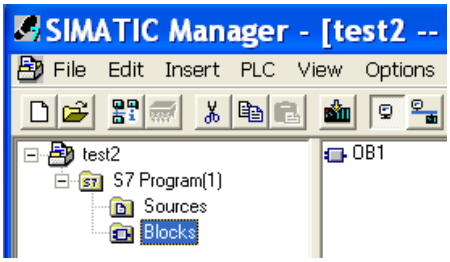
PG/PC: برای پیکر بندی ارتباط کامپیوتر یا PG با PLC بکار میرود.

با کلیک کردن روی Station موردنظر آیکون آن به پنجره اضافه میشود.

بـاید خاطـر نشـان کـرد کـه یک پروژه میتواند چندین Station را در بر داشته باشد . مثلاً چند PLC از نوع 300 یا 400 ویا ترکیبی از ایندو بگونه ای که هرکدام بخش جداگانه ای از پروژه را کنترل نمایند . این Station ها میتوانند از طریق شبکه به یکدیگر متصل باشند .

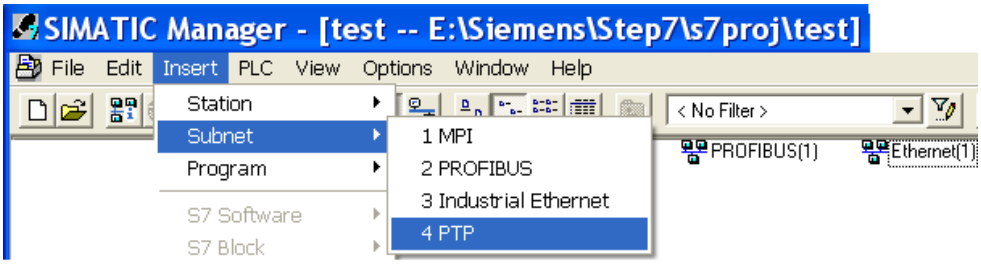
بـرای وارد کـردن هر Station جدید ابتدا لازم است با ماوس روی اسم پروژه در پنجره کلیک کرده سپس از منوی Insert>Station آنرا وارد نماییم.

حال اگر روش دوم مدنظر باشد یعنی وارد کردن برنامه بدون انتخاب سخت افزار ، در اینصورت از منوی Inser>Program>s7 Program استفاده میکنیم.بدیهی است در اینحالت فقط پوشه های مربوط به برنامه نویسی به پنجره اضافه میشوند.



قدم سوم : وارد کردن شبکه

برای وارد کردن شبکه ابتدا روی اسم پروژه در پنجره کلیک کرده سپس از منوی Insert>Subnet مانند شکل زیر آنرا به پروژه وارد میکنیم . البته این کار در ابتدا ضرورتی ندارد و بعداً میتواند با روش فوق یا روشهای دیگری که در بحث شبکه شرح داده خواهند شد نیز انجام گیرد.



۲-۳ منوهای Simatic Manager

همانطور که در شکل فوق نیز مشاهده میشود در بالای برنامه Simatic Manager منوهای مختلفی وجود دارد که ذیلا به برخی کاربردهای آنها اشاره میشود:

File برای ایجاد، باز کردن ،ذخیره سازی ، حذف،سازماندهی ،آرشیو کردن پروژه استفاده میشود.

Edit امکاناتی مانند Copy و Paste و Cut و Rename کردن و همچنین مشاهده Properties المانها را دارد

Insert برای وارد کردن المانهای سخت افزاری ، شبکه و بلاک های برنامه نویسی به پروژه بکار میرود.

PLC برای ارسال اطلاعات به PLC و یا گرفتن اطلاعات از PLC و مواردی ازاین قبیل استفاده میشود.

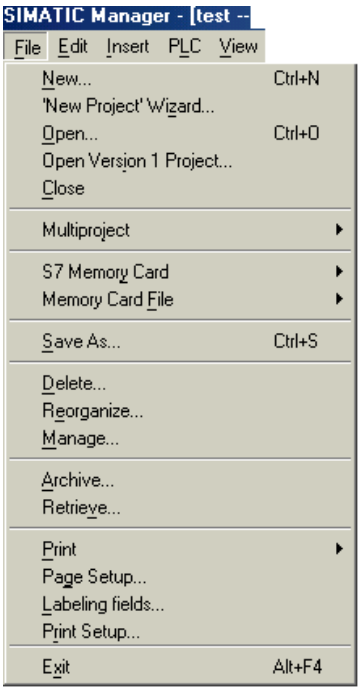
View نحوه نمایش Object ها و فیلتر گذاری روی آنها از این منو امکان پذیر است.

Options کاربر توسط امکانات این منو میتواند تنظیمات پیش فرض برنامه راتغییر دهد.

Window در این قسمت نحوه نمایش پنجره Simatic Manager قابل انتخاب است.

Help راهنمای استفاده و بکار گیری Simatic Manager میباشد.

با کلیک کردن روی هر کدام از منوهای فوق لیستی باز میشود که امکانات مختلف در آن نمایش داده میشود. توضیح تمامی این امکانات در این مقطع که هنوز کاربر با برخی مفاهیم ممکن است آشنا نباشد امکان پذیر نیست. ازاینرو صرفا به برخی از آنها اشاره میشود و سایر موارد در خلال بحثهای آینده بیان خواهند شد.



منوی File

امکانات مختلف این منو در شکل روبرو نشان داده شده است که به برخی اشاره میشود:

New : ایجاد پروژه جدید

New Project Wizard: ایجاد پروژه جدید با استفاده از Wizard

Open : باز کردن پروژه از قبل ذخیره شده

Open Version 1 Project : ایجاد پروژه جدید تحت نسخه ۲ از پروژه ای که قبلا تحت نسخه ۱ ایجاد و ذخیره شده است

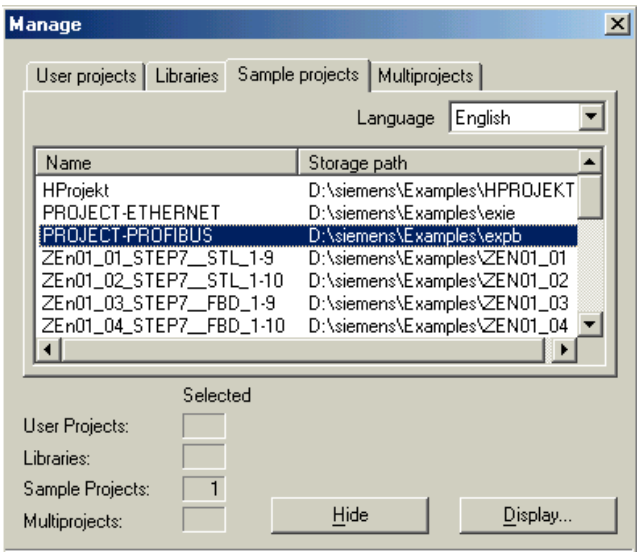
Close : بستن پروژه ای که باز است

Save As: ذخیره سازی پروژه به نام دیگر

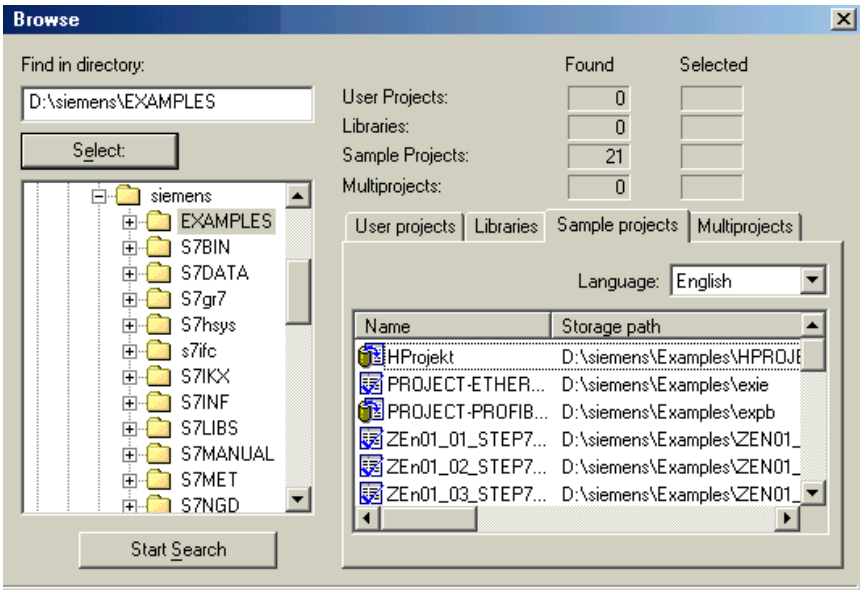
Delete: حذف کردن کامل یک پروژه

Reorganize : دیتابیس و فایلهای مربوط به پروژه را سازماندهی مجدد میکند و Gap های ناشی از پاک کردن Object ها را حذف میکند در نتیجه حافظه مورد نیاز برای پروژه کاهش می یابد. با اینکار برخی خطاهای غیر منتظره برنامه نیز رفع میشوند

Manage : توسط آن میتوان یک پروژه را مخفی ساخت تا در لیست پروژه ها ظاهر نشود. با کلیک کردن روی آن پنجره جدیدی مانند شکل زیر باز میشود ابتدا از بالای پنجره انتخاب میکنیم که چه نوع پروژه ای مخفی شود مثلا اگر پروژه های تهیه شده توسط کاربر مد نظر است باید User Project را انتخاب کنیم.سپس از لیستی که در زیر آن ظاهر شده نام پروژه را انتخاب کرده و کلید Hide را می فشاریم .مشاهده میکنیم که پروژه از لیست حذف میگردد.

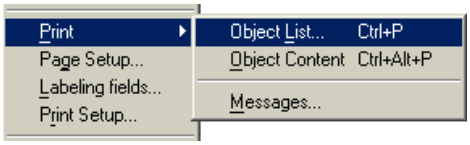


اگر پروژه ای قبلا مخفی شده باشد از طریق همین پنجره و با انتخاب Display میتوان آنرا غیر مخفی کرد. با کلیک روی Display پنجره زیر باز میشود ابتدا باید محل ذخیره سازی پروژه را مشخص کرده سپس روی Start Search کلیک کنیم. لیست پروژه های موجود در آن دایرکتوری در سمت راست ظاهر میشود. پروژه هایی که قبلا مخفی شده اند روی آیکون آنها علامت زرد رنگی وجود داردو با کلیک کردن روی آنها به حالت عادی برمیگردند.



Archive: میتوان پروژه ای که کل اطلاعات سخت افزار ، شبکه و برنامه را در بر دارد را بصورت فشرده آرشیو سازی کرد.فایل فشرده زیپ شده را میتوان روی هارد دیسک کامپیوتر یا روی فلاپی دیسک ذخیره نمود تا بصورت Backup از برنامه در موقع ضرورت استفاده گردد .برای آرشیو سازی از منوی File > Archive استفاده میکنیم. بصورت پیش فرض فایل فشرده از نوع ZIP است با این وجود میتوان نوع آن را تغییر داد برای اینکار همانطور که بعدا بیان خواهد شد از منوی Option>Customize استفاده مینماییم.

Retrieve: عکس عمل آرشیو سازی است یعنی پروژه ای که قبلا بصورت فایل فشرده ذخیره شده را توسط Retrieve میتوان باز کرد. و محتویات آنرا در پنجره Simatic Manager مشاهده نمود.

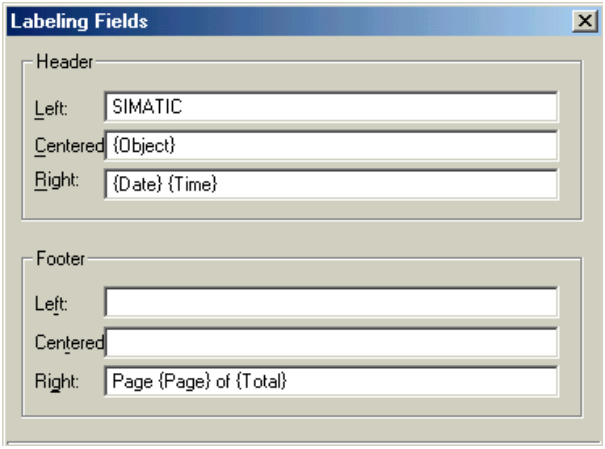


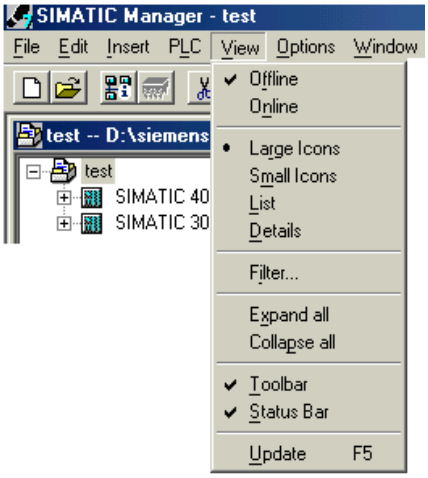
Print : برای چاپ کردن بکار میرود اگر Object List انتخاب شود محتویات پنجره Simatic Manager و اگر Object Contenet انتخاب شود جزئیات سخت افزار یا برنامه را چاپ میکند.

Page Setup : برای تنظیمات مربوط به کاغذ چاپ بکار میرود

Labeling fields : برای تنظیم اینکه در بالا یا پایین پرینت چه مواردی (مانند تاریخ ، زمان ، شماره صفحه و ...) ظاهر شود بکار میرود بصورت پیش فرض تنظیماتی مانند شکل صفحه بعد وجود دارد.

Print Setup: مربوط به تنظیمات پرینتر است.





منوی View

امکانات مختلف این منو در شکل روبرو نشان داده شده است

Offline : دیدن پروژه بدون اتصال به PLC و فقط از روی کامپیوتر

Online: دیدن پروژه در حالت اتصال به PLC

LargeIcons
Small Icons
List
Details

نحوه ظاهر شدن شکل آیکون المانها بصورت کوچک یا بزرگ و ..

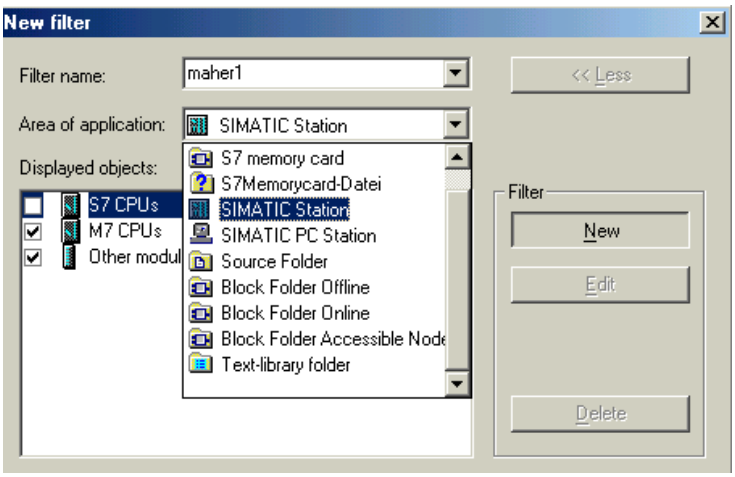
Expand all

ساختار درختی موجود در پنجره سمت چپ را بطور کامل باز میکند

Collapse all

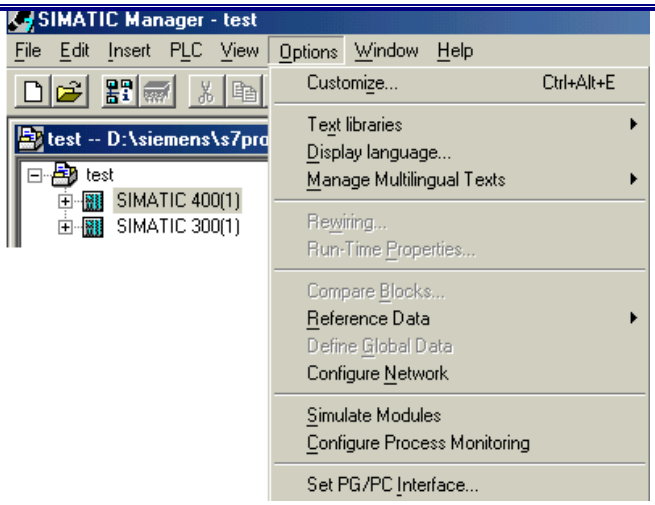
ساختار درختی موجود در پنجره سمت چپ را بطور کامل میبندد.

Filter : با انتخاب فیلتر پنجره ای مانند شکل زیر ظاهر میشود که میتوان روی Object های مختلف سخت افزاری و نر م افزاری موجود در پروژه فیلتر گذاشت تا برخی نمایش داده نشوند.مثلا با فیلتر شکل زیرآیکون CPU های S7 نشان داده نمیشود.

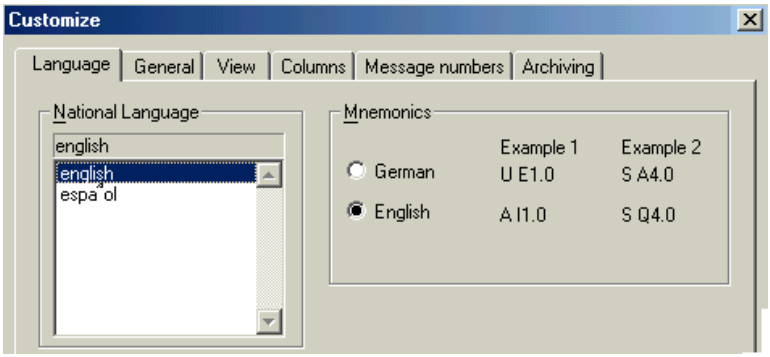


منوی Option

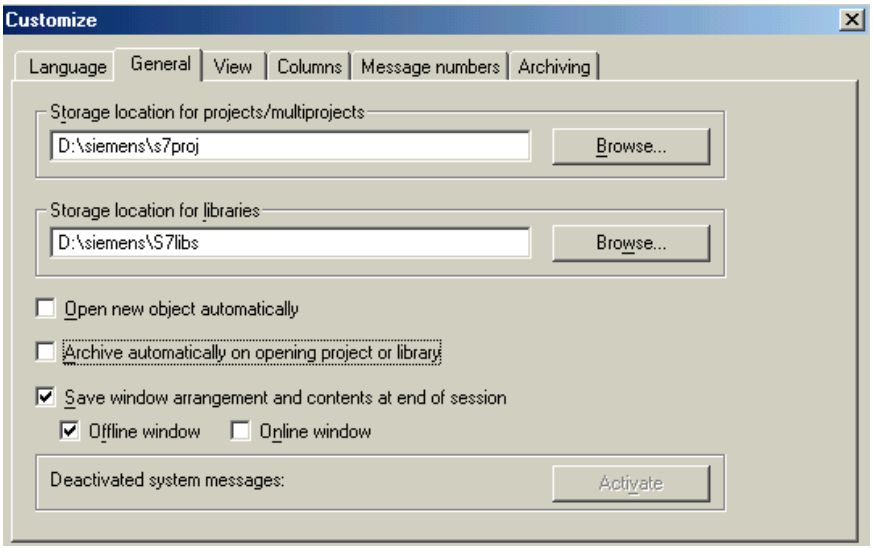
امکانات مختلف این منو در شکل زیر نشان داده شده است



یکی از کاربردهای این منو برای ایجاد تنظیمات دلخواه و تغییر پیش فرض های Simatic Manager است اینکار با استفاده از قسمت **Customize** و در پنجره ای مانند شکل زیر انجام میشود که Tab های مختلف دارد. **Language** برای انتخاب زبان است

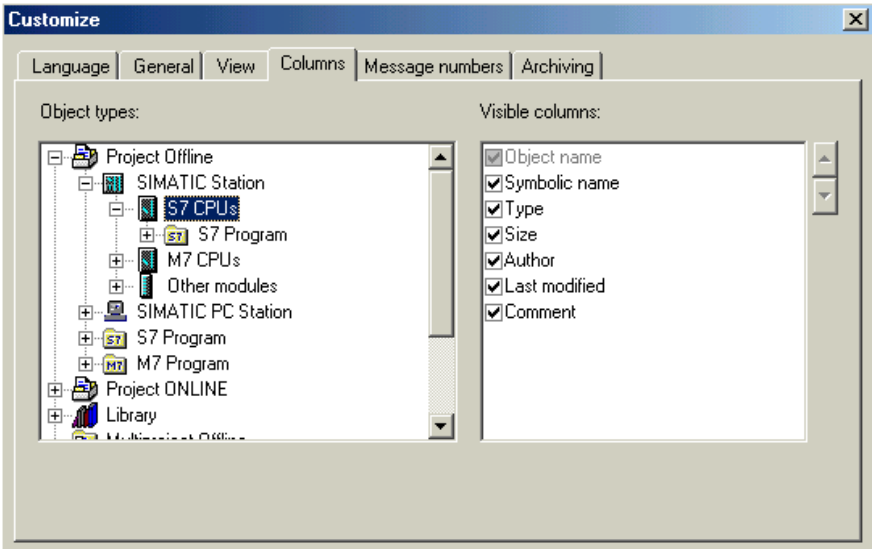


General برای انتخاب محل ذخیره سازی پروژه و موارد دیگر مثلا باز کردن اتوماتیک پروژه جدید یا آرشیو سازی اتوماتیک آنست

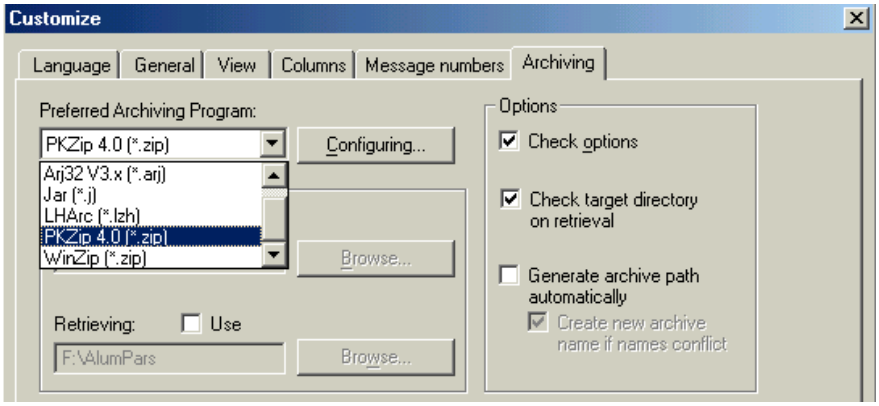


View برای تنظیمات حالت Online مانند رنگ زمینه بکار میرود

Column برای تنظیم اینکه برای هر کدام از Object ها چه اطلاعاتی نمایش داده شود مانند شکل زیر :



Archiving انتخاب نوع فشرده سازی پروژه در این قسمت مانند شکل زیر امکان پذیر است:



در پایان این قسمت لازم است متذکر شویم که بجای استفاده از منوهای Simatic Manager میتوان از Toolbarبالای پنجره آن که به شکل زیر است و ازمنوی View > Toolbar غیر فعال و فعال میشود استفاده نمود.

۳- پیکر بندی سخت افزار

مشتمل بر :

۳-۱ Hwconfig ابزار پیکر بندی سخت افزار

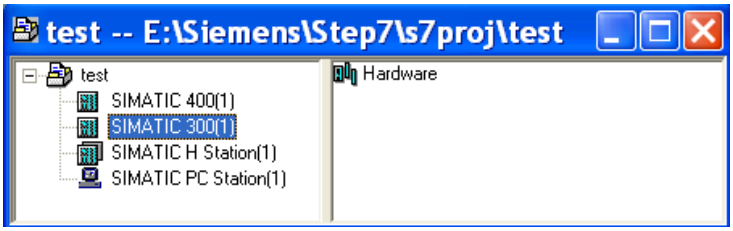
۳-۲ پیش نیازهای پیکر بندی سخت افزار

۳-۳ پیکر بندی S7-300

۳-٤ پیکر بندی S7-400

۳-۱ Hwconfig ابزار پیکربندی سخت افزار

با وارد کردن Station در پروژه ایجاد شده توسط Simatic Manager و کلیک کردن روی آن آیکون Hardware در پنجره سمت راست ظاهر میشود. فرض کنید Station های مختلفی از جمله یک 300 Station وارد پروژه کرده ایم در اینصورت پنجره پروژه مانند شکل زیر است .

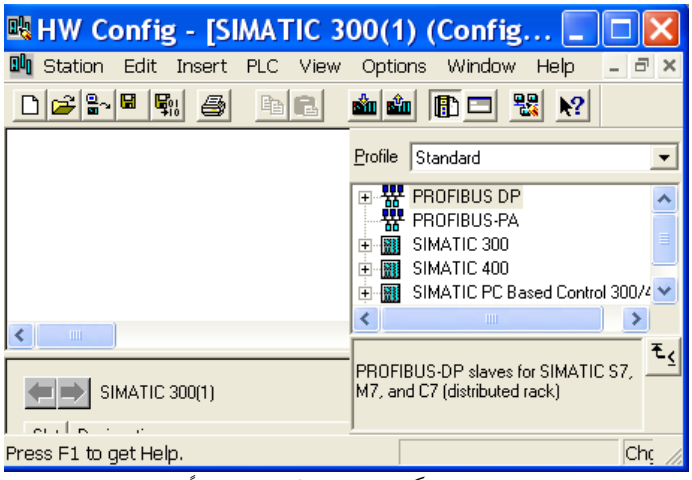


با کلیک کردن روی آیکون Hardware در پنجره سمت راست برنامه Hwconfig اجرا میشود . کلیه کارهای پیکر بندی سخت افزار و تنظیم پارامترهای آن توسط این برنامه انجام میشود. باید توجه داشت که برای Station از نوع 300 و 400 و H-Station آیکون Hardware ظاهر میگردد همینطور برای Pc Station آیکون پیکربندی آنرا خواهیم دید ولی سایر Station ها مانند S5 وPG و Others چنین شکلی ندارند نه در زیر نام پروژه ظاهر میشوند و نه آیکونهای فوق را میتوان برای آنها مشاهده کرد. بعبارت دیگر این Station ها توسط Step7 قابل پیکر بندی از نظر سخت افزار داخلی نیستند و صرفاً برای مقاصد دیگر مثلاً اتصال به شبکه وارد پروژه میشوند.

بطور کلی برای پیکر بندی سخت افزار باید قدمهایی بترتیب برداشته شوند . فلوچارت زیر مراحل این کار را نشان میدهد:

برای فهم بهتر مطلب ، ابتدا به محیط Hwconfig نگاهی می اندازیم سپس یک 300 Station را پیکر بندی کرده تا خواننده با برخی جزئیات آشنا گردد پس از آن به سراغ پیکر بندی 400 Station میرویم و تفاوتهای آن با نوع 300 و نکات خاص مربوط به آن را توضیح میدهیم.

همانطور که اشاره شد برای اجرای برنامه Hwconfig کافی است روی آیکون Hardware در پنجره پروژه کلیک کنیم. با اجرای این برنامه شکلی مانند زیر خواهیم دید. پنجره سمت راست Catalog نامیده میشود و وجود آن برای انتخاب سخت افزار لازم است. اگر فعال نبود با منوی View >catalog میتوان آنرا فعال کرد. اجزای سخت افزاری مطابق با توضیحی که خواهد آمد از این پنجره انتخاب شده و در پنجره سمت چپ ظاهر میگردند.



باید توجه داشت که اجزای سخت افزار باید از همان گروه Station باشند که قبلاً در پروژه وارد کرده ایم. بعنوان مثال نمیتوان یک Station 300 وارد پروژه کرد و بعد از آن در کاتالوگ اجزای مربوط به 400 را انتخاب نمود . برنامه این عدم تطابق را با پیغام Not The same System Family اعلام مینماید.

در کاتالوگ فوق در زیر هر Station کلماتی را می بینیم که هر کدام نشان دهنده سخت افزار خاصی هستند . این کلمات در جدول زیر بیان شده اند.

کد	شرح	عملکرد
Rack	Rack	نگهدارنده مدولها ، تغذیه کننده مدولها و ایجاد ارتباط بین آنها
PS	Power Supply	منبع تغذیه
CPU	Central Processing Unit	پردازشگر مرکزی
IM	Interface Module	ایجاد ارتباط بین چند رک
SM	Signal Module	اتصال با سیگنالهای ورودی و خروجی (I/O)
CP	Communication Processor	ایجاد ارتباط با شبکه
FM	Function Module	اجرای فانکشن خاصی مستقل از CPU

هر کدام از کدهای فوق بصورت یک پوشه (Folder) ظاهر میشوند که با باز کردن آنها میتوان انواع سخت افزار های مربوط به آن گروه را مشاهده کرد. نکته قابل توجه آنست که برخی از اجزای موجود در پنجره کاتالوگ ممکن است جزء محصولات روز زیمنس نباشند همچنین ممکن است پس از ارائه نرم افزار Step7 محصولات جدیدی از خانواده Simaticعرضه شوند لذا بهتر است همواره با استفاده از Service Pack های جدیدی که توسط زیمنس عرضه میشود و ومعمولاً بصورت آزاد (Free download) در سایت زیمنس (www4.ad.siemens.de) وجود دارد کاتالوگ موجود در نرم افزار Update گردد.

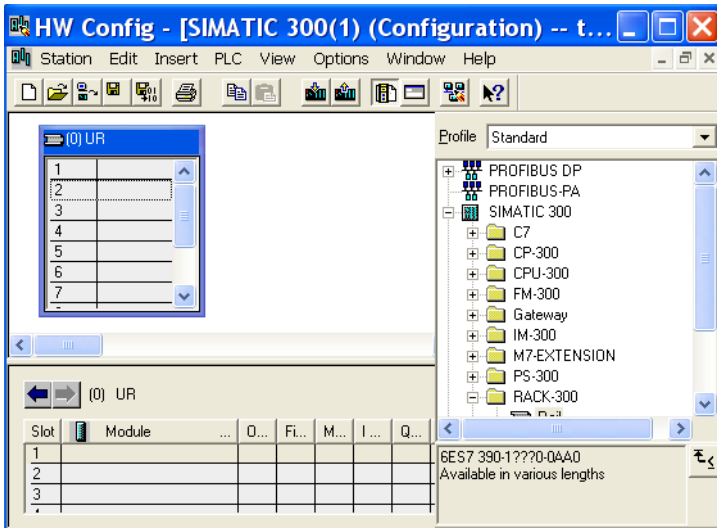
۳-۲ پیش نیازهای پیکربندی سخت افزار

قبل از شروع پیکر بندی لازم است در مرحله طراحی نوع و مشخصات سخت افزار مورد نیاز بررسی و تعیین شده باشد. اینکار معمولاً توسط جمع بندی سیگنالهای I/O و با پاسخ به سوالاتی مانند زیر انجام میشود. این پرسشها بعنوان نمونه است و جوانب دیگر نیز باید در طراحی دیده شود .در این مرحله استفاده از آخرین کاتالوگ زیمنس که بصورت CD عرضه میشود برای یافتن مشخصات فنی مدولهای PLC مفید است.

بررسی	نتیجه گیری
• چه تعداد I/O داریم ؟ • توزیع فیزیکی سیگنالها چگونه است ؟ • آیا برای I/O ها به شبکه نیاز داریم ؟ • آیا قابلیت های خاص برای CPU مد نظر است ؟ • شرایط محیط نصب چگونه است؟ • و ...	CPU مشخص میگردد.
• چند نوع سیگنال آنالوگ وجود دارد؟ • آیا برخی سیگنالها اهمیت بیشتری دارند؟ • آیا کارتهای I/O باید قابلیت خاصی داشته باشند؟ • هر کارت چه تعداد ورودی یا خروجی داشته باشد؟ • و	کارتهای I/O و تعداد آنها مشخص میگردد
• آیا کارت FM نیاز است ؟ • آیا کارت CP لازم است؟ • و	سایر مدولهای مورد نیاز مشخص میشوند.
• تعداد کل مدولها چقدر است ؟ • با احتساب اسلات رزرو آیا یک رک برای آنها کافیست؟ • آیا لازم است نوع کارتهای I/O را تغییر دهیم تا با گرفتن سیگنال بیشتر تعداد آنها کم شده و بتوان همه مدولها را روی یک رک جای داد؟ • در صورت نیاز به رک اضافی چه تعداد رک و IM مورد نیاز است؟ • و	رک اضافی مشخص میگردد.
• جریان مصرفی مدولها بر اساس مشخصات فنی هر کدام ازآنها چقدر است؟ • چه منبع تغذیه ای برای تامین جریان کل مدولها مناسب است؟	منبع تغذیه مشخص میشود.

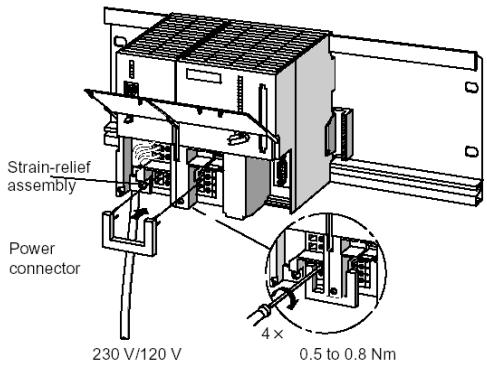
۳-۳ پیکربندی PLC از نوع S7-300

مطابق آنچه در بخش ۳-۱ گفته شد Station 300 را در simatic Manager وارد پروژه کرده سپس با کلیک روی آیکون Hardware آنرا با برنامه Hwconfig باز میکنیم. از پنجره کاتالوگ این برنامه در زیر گروه SIMATIC 300 اجزای سخت افزاری را وارد پروژه مینماییم . اولین قدم وارد کردن رک است. با دوبار کلیک روی آیکون رک اسلاتهای آن درپنجره سمت چپ مانند شکل زیر ظاهر میشوند.

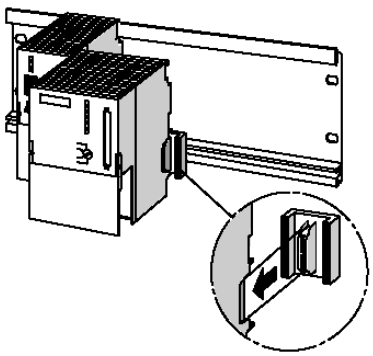


ویژگیهای رک 300 :

- یازده اسلات دارد.
- بصورت ریل است.
- فقط یکنوع دارد که هم بعنوان رک اصلی و هم بعنوان رک اضافی استفاده میگردد.
- فقط نقش نگهدارندگی برای مدولها دارد.
- ارتباط بین PS و CPU با کانکتور خاص مطابق شکل الف برقرار میشود و خود رک (ریل) در ایجاد این ارتباط نقشی ندارد.
- ارتباط بین مدولها با کانکتور خود آنها مطابق شکل ب برقرار میشود.
- مدولها باید روی آن کنار یکدیگر و بدون فاصله قرار گیرند.



الف) نحوه ارتباط منبع تغذیه با CPU



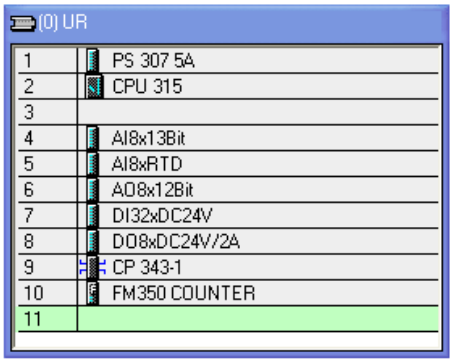
ب) نحوه ارتباط CPU با مدول بعدی

ترتیب مدولها در رک 300

روی رک نمیتوان هر مدول را در اسلات دلخواهی قرار داد. در هنگام پیکر بندی با Hwconfig باید ترتیب ذکر شده در جدول زیر رعایت شود.

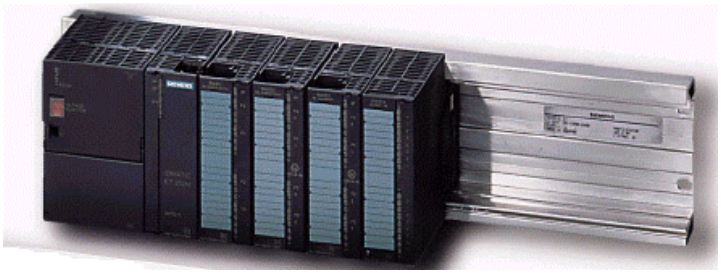
شماره اسلات	مدولهای مجاز
1	PS
2	CPU
3	IM یا خالی
4-11	SM و CP و FM

ابتدا روی اسلات خالی با ماوس کلیک کرده سپس مدول را از پوشه مربوطه از پنجره کاتالوگ انتخاب مینماییم با دوبار کلیک روی آن میبینیم که مدول در اسلات ظاهر میگرددیا آنکه با کشیدن آیکون مدول توسط ماوس (Drag) و قرار دادن آن در اسلات مورد نظر اینکار را انجام میدهیم. بدیهی است اگر اشتباهاً کاربر مدولی را در اسلات غیر مربوط قرار دهد با پیغام خطا توسط برنامه مواجه میشود. شکل زیر نمونه ای از اسلاتهای پرشده توسط برنامه را نشان میدهد.



وجود اسلات خالی در این مرحله توسط برنامه کنترل نمیشود بلکه در انتها وقتی کاربر چک سازگاری (Consistency Check) را از طریق منوی Station انجام میدهد پیغام خطای مربوط به اسلات خالی نشان داده میشود.

نکته دیگری که باید مد نظر باشد اینست که در عمل نباید هیچ فاصله خالی بین مدولها وجود داشته باشد . بنابر این اگرمثلا در پیکر بندی توسط برنامه در اسلات سوم IM قرار ندهیم (یعنی رک اضافی نداشته باشیم) چک سازگاری نیز خطایی را اعلام نمیکند . ولی در عمل باید مدول بعد از CPU را بدون فاصله کنار آن مانند شکل ؤیر قرار داد.



استفاده از رک اضافی (Expansion)

رک اصلی را Central و رک اضافی را Expansion میگویند. اگر تعداد I/O ها زیاد باشد و رک اصلی پر شود از رک اضافی استفاده میگردد. کابلی که رک اضافی را به رک اصلی وصل میکند طول محدودی دارد. بنابراین رک اضافی نمیتواند زیاد دور از رک اصلی قرار گیرد. بعبارت دیگر استفاده از رک اضافی وقتی مناسب است که سیگنالهای فیلد (Field) بصورت متمرکز و با فاصله کم نسبت به PLC باشند . در غیر اینصورت یعنی اگر سیگنالها بصورت پراکنده و توزیع شده باشند رک اضافی راه حل مناسبی نیست و استفاده از شبکه روش بهینه میباشد.

برای استفاده از رک اضافی در S7-300 نکات زیر مد نظر قرار گیرد:

- ماکزیمم ۳ رک اضافی به رک اصلی میتوان متصل کرد.
- ارتباط بین رک اصلی و رک اضافی توسط IM انجام میشود.
- IM ها چه در رک اصلی و چه در رک اضافی همیشه در اسلات شماره ۳ قرار میگیرند.
- IM ها بصورت جفتی بکار میروندیعنی یکی از دو حالت زیر :
  ۱. IM 360S در رک اصلی و IM 361R در رک های اضافی (ماکزیمم ۳ رک اضافی)
  ۲. IM 365 در رک اصلی و IM 365 در رک های اضافی (فقط ۱ رک اضافی)
- کابل هایی که IM را بهم متصل میکند خاص بوده و طول معینی دارند.در حالت ۱ طول کابل میتواند ۱ متر یا ۲/۵ متر یا ۵ متر یا ۱۰ متر باشد. ولی در حالت ۲ کابل فقط ۱ متری است.
- در حالت ۲ تغذیه نیز با کابل منتقل میشود ولی در حالت ۱ رک های اضافی نیاز به منبع تغذیه جداگانه دارند.
- ارتباط رک ها از طریق IM بصورت Daisy Chain است هر IM دارای دو پورت است . در رک اضافی یک پورت به رک ماقبل و پورت دوم به رک مابعد مانند شکل زیر متصل میگردد.

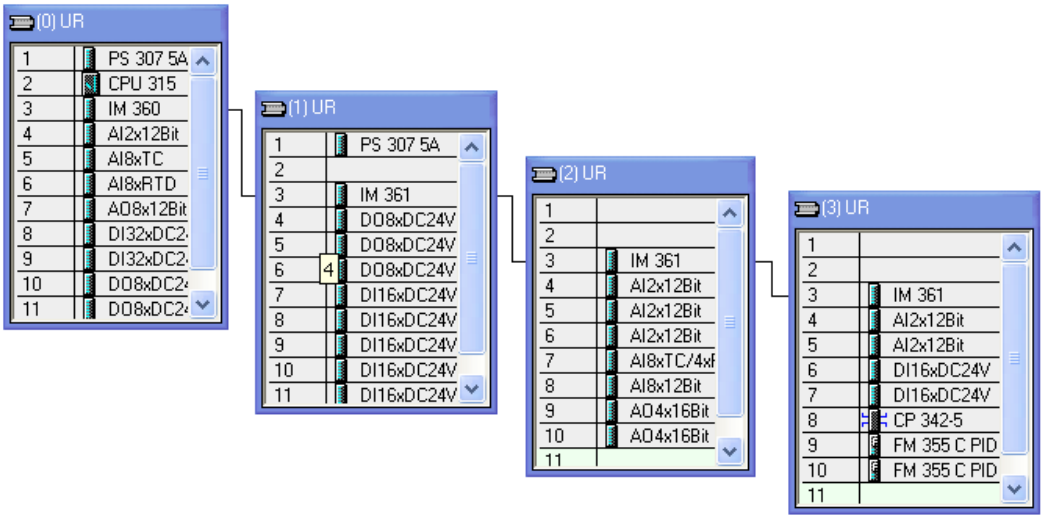
ترتیب مدولها در رک اضافی مانند رک اصلی است با این تفاوت که رک اضافی CPU ندارد.یعنی در پیکر بندی اسلات دوم خالیست.

ترتیب مدولها در رک اضافی	
شماره اسلات	مدولهای مجاز
1	PS
2	خالی
3	IM
4-11	SM و CP و FM

برای وارد کردن رک اضافی در Hwconfig آیکون Rail را با ماوس به پنجره سمت چپ Drag میکنیم یا یکبار با ماوس در پنجره سمت چپ در فضای خالی کلیک میکنیم به گونه ای که دیگر رک اصلی که UR (0) نام گذاری شده فعال نباشد در اینحال با هر بار کلیک روی آیکون ریل در پنجره کاتالوگ یک رک جدید وارد پنجره سمت چپ میشود.

در رک اصلی و رک های اضافی IM مناسب را (با توجه به توضیحات صفحه قبل) وارد میکنیم . می بینیم که اتصال بین رک ها اتوماتیک مانند شکل زیر برقرار میشود. پس از آن در رکهای اضافی مدولهای مورد نظر را قرار میدهیم.

تذکر : برخی CPU های 300 رک اضافی را ساپورت نمیکنند (مانند CPU 312) . در اینحالت کاربر نمیتواند رک اضافی به پنجره وارد نماید و با پیغام خطا مواجه میشود.

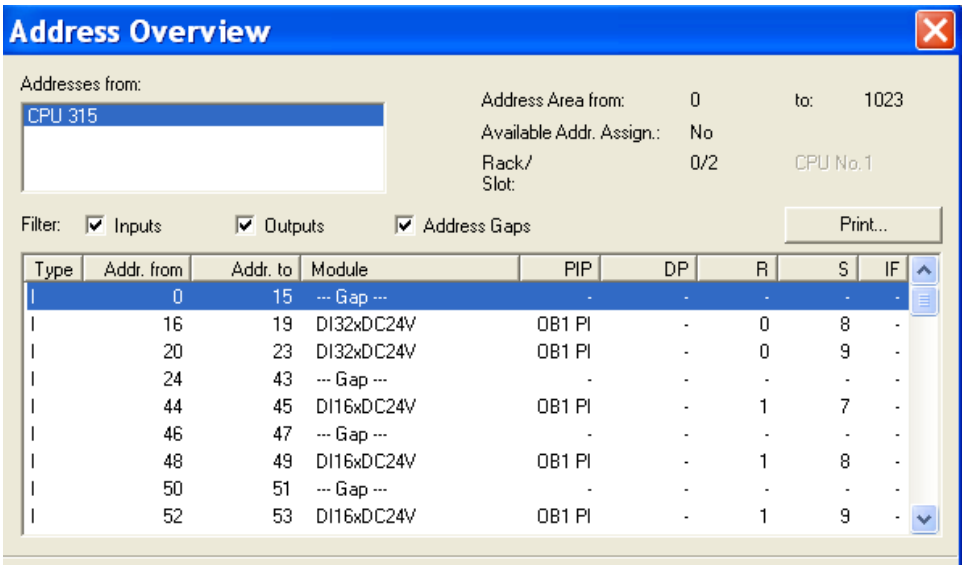


تنظیم پارامترهای مدولهای S7-300

هر مدول که به اسلات مورد نظر در Hwconfig وارد میشود علاوه بر پنجره بالا در پنجره دیگری زیر آن نیز مانند شکل زیر ظاهر میگردد. پنجره پایین توضیحات بیشتری راجع به مدول مانند کد سفارش و آدرس ارائه میدهد. همانطور که قبلا نیز اشاره شد ممکن است این کدسفارش ارائه شده توسط Step7 برای سخت افزارهای مختلف 300 و 400 و ... بروز نباشد و کاربر برای سفارش نباید از این کدها استفاده کند مگر اینکه Service Pack جدید نصب شده باشد ولی کلا بهتر است کار سفارش با مراجعه به آخرین کاتالوگ زیمنس انجام شود.

Slot	Module ...	Order number	Firmware	MPI ad...	I addr...	Q add...	Comment
1	PS 307 5A	6ES7 307-1EA00-0AA0					
2	CPU 315	6ES7 315-1AF00-0AB0		2			
3	IM 360	6ES7 360-3AA00-0AA0			2000		
4	AI2x12Bit	6ES7 331-7KB82-0AB0			256...259		
5	AI8xTC	6ES7 331-7PF10-0AB0			272...287		
6	AI8xRTD	6ES7 331-7PF00-0AB0			288...303		
7	AO8x12Bit	6ES7 332-5HF00-0AB0				304...319	
8	DI32xDC24V	6ES7 321-1BL80-0AA0			16...19		
9	DI32xDC24V	6ES7 321-1BL80-0AA0			20...23		
10	DO8xDC24V/2A	6ES7 322-1BF00-0AA0				24	
11	DO8xDC24V/2A	6ES7 322-1BF00-0AA0				28	

آدرسهای استفاده شده برای مدولها توسط برنامه بطور اتوماتیک اختصاص داده میشوند. همانطور که بعداً خواهد آمد میتوان بعضاً آنها را تغییر داد. ولی معمولاً الزامی برای این کار وجود ندارد. با استفاده از منوی View > Address Overview میتوان وضعیت آدرسهای ورودی(input) و خروجی(output) و فضای خالی بین آنها (Gap) را مشاهده کرد. بعلاوه میتوان کل فضای آدرسی که CPU انتخاب شده ساپورت میکند را در بالای پنجره مشاهده نمود.



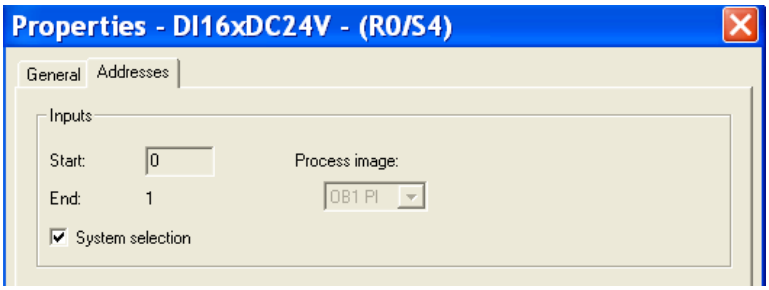
برای مشاهده جزئیات بیشتر در مورد پارامترهای مدولها کافیست روی آنها کلیک کنیم اینکار با کلیک راست و سپس انتخاب Object Properties نیز امکان پذیر است. بخشهای بعدی پارامترهای مربوط به مدولهای مختلف را به تفصیل بیان می نماید.

تنظیم پارامترهای کارتهای DI

در پنجره کاتالوگ در زیر مجموعه SM-300 کارتهای Digital Input متنوعی را مشاهده میکنیم که با کلیک روی آنها توضیحات مختصری راجع به کارت در پایین پنجره کاتالوگ ظاهر میشود. بطور کلی این کارتها را میتوان به شکل زیر دسته بندی کرد:

تقسیم بندی کارتهای Digital Input		
از نظر تعداد ورودی	از نظر ولتاژ	ارنظر قابلیت های خاص
• ٤ ورودی	• 24 VDC	• بدون ویژگی خاص
• ٨ ورودی	• 48 VDC	• تشخیص قطع شدن تغذیه
• ١٦ ورودی	• 120 VAC	• ایجاد وقفه بر اساس لبه ورودی
• ٣٢ ورودی	• 230 VAC	• تاخیر در گرفتن ورودی

برای کارتهایی که قابلیت خاص ندارند وقتی روی آنها کلیک کنیم پنجره ای مانند شکل زیر باز میشود که دو بخش دارد.



General: در این بخش توضیحاتی راجع به کارت ، ویژگیها و کد سفارش آن همراه با نام آن آمده است که کاربر در صورت تمایل میتواند نام را بدلخواه تغییر دهد.

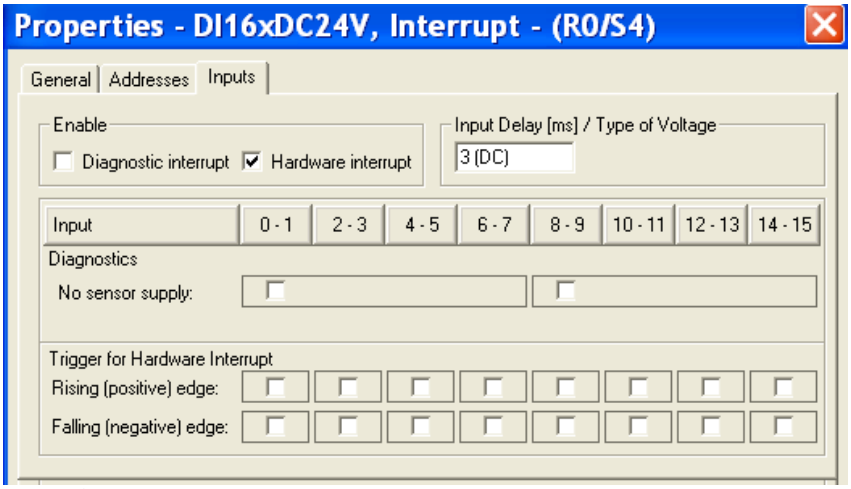
Address: در این بخش آدرسهایی که توسط سیستم به کارت اختصاص داده شده آمده است. Start آدرس شروع و End آدرس نهایی را نشان میدهد. بعنوان مثال در شکل برای کارت DI با ١٦ ورودی مشاهده مکنیم که آدرس شروع 0 و آدرس انتها 1 است بنابراین لیست آدرسهای ١٦ کانال که هرکدام یک Bit (0 یا 1) هستند مانند جدول زیر خواهد بود. بعبارت دیگر این مدول دارای ٢ بایت آدرس است و میدانیم که 2 Byte= 16 Bit

کانال	آدرس
0	0.0
1	0.1
2	0.2
3	0.3
4	0.4
5	0.5
6	0.6
7	0.7
8	1.0
9	1.1
10	1.2
11	1.3
12	1.4
13	1.5
14	1.6
15	1.7

اگر چند مدول DI مشابه یا متفاوت داشته باشیم نیز مشاهده میکنیم که آدرسهای تولید شده توسط سیستم با یکدیگر هیچ تداخلی ندارند.

در S7-300 تغییر آدرس توسط کاربر بعضاً امکان پذیر است. برخی از CPU های 300 این امکان را ساپورت میکنند(از CPU315 به بالا) . در اینحالت مانند شکل بالا گزینه System Selection قابل انتخاب است میتوان آنرا غیر فعال نمود و آدرس جدید را وارد کرد. شماره آدرس نمیتواند از Address Area مربوط به CPU بزرگتر باشد بعلاوه اگر آدرس جدید تداخلی با آدرسهای دیگر داشته باشد سیستم پیغام میدهد و در عین حال آدرس دیگری را پیشنهاد مینماید. در مجموع پیشنهاد میشود که حتی المقدور کاربر آدرسهای پیش فرض سیستم را تغییر ندهد.

در بین کارتهای DI موجود در کاتالوگ برخی از کارتها قابلیت های خاص دارند. توانایی اعمال وقفه (Interrupt) مهمترین قابلیت آنهاست که این ویژگی در توضیحات زیر پنجره کاتالوگ دیده میشود. بخش Properties این کارتها نسبت به کارتهای معمولی یک بخش اضافه بنام Inputs مانند شکل داردکه در آن میتوان قسمتهای زیر را مشاهده نمود:



Diagnostic Interrupt : در حالت عادی غیر فعال است اگر فعال شود در صورت قطع تغذیه سنسور (مثلاً بعلت قطع فیوز) شماره کانال مربوطه در بافر تشخیص عیب CPU ثبت میشود. همانطور که در شکل ملاحظه میشود در جلوی No Sensor Supply یک گزینه برای ورودیهای 0 تا 7 و یک گزینه نیز برای ورودی های 8 تا 15 وجود دارد. میتوان هردو یا یکی را بدلخواه فعال نمود. بدیهی است در صورت قطع تغذیه آنچه در بافر ثبت میشود آدرس گروه کانال است نه آدرس خود کانال.

Hardware Interrupt : در حالت عادی غیر فعال است اگر فعال شود جدول پایین که مربوط به تریگر کردن این وقفه است نیز فعال میگردد. در این جدول برای هر دو کانال وروردی یک گزینه وجود دارد . با انتخاب این گزینه میتوان تعیین کرد که وقتی ورودی کانال تغییر میکند (لبه مثبت یا لبه منفی) وقفه اعمال نماید.

تذکر : بحث وقفه ها بطور مفصل در جلد دوم کتاب خواهد آمد. در اینجا همینقدر باید اشاره کرد که فعال کردن وقفه در کارت به تنهایی کافی نیست و علاوه بر آن باید یک بلاک برنامه نویسی از نوع Organization Block نیز در پروژه موجود باشد. در غیر اینصورت با وقوع وقفه CPU به مد Stop میرود.

Input Delay : در این قسمت میتوان تعیین کرد که ورودی را با چند میلی ثانیه تاخیر بگیرد.توصیه میشود در دوحالت زیر این عدد را روی ماکزیمم بگذارید:

۱. برای سوئیچهای ساده و بدون حفاظت . تاخیر فوق باعث میشود که پرش لحظه ای ولتاژ مشکلی ایجاد نکند.
۲. اگر طول کابل تا سنسور زیاد و کابل بدون شیلد است. تاخیر فوق باعث میشود که ورودی را در زمان مناسب بگیرد.

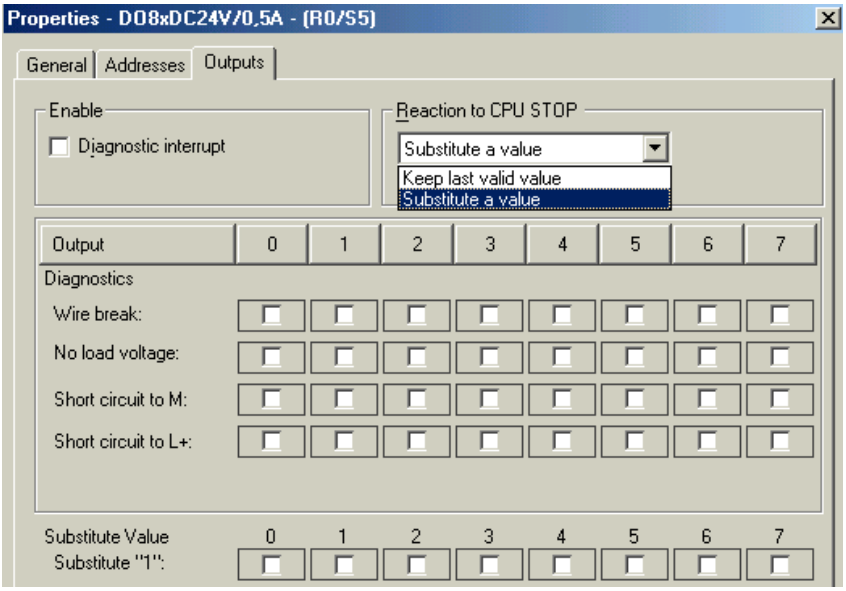
تنظیم پارامترهای کارتهای DO

کارتهای Digital Output موجود در کاتالوگ را نیز میتوان بصورت زیر دسته بندی نمود:

تقسیم بندی کارتهای Digital Output			
از نظر تعداد خروجی	از نظر جریان خروجی	از نظر ولتاژ	ارنظر قابلیت های خاص
• ٤ خروجی • ٨ خروجی • ١٦ خروجی • ٣٢ خروجی	• بدون رله با جریانهای 0.5 , 1 , 1.5 , 2 A با رله و جریانهای 5 , 8 A	• 24 VDC • 48 VDC • 120 VAC • 230 VAC	• بدون ویژگی خاص • تشخیص قطعی • تشخیص اتصال کوتاه • واکنش در موقع توقف CPU

تذکر : کارتهایی که برای F-System استفاده میشوند ممکن است ویژگی های خاص دیگری نیز داشته باشند.

در پنجره ای که برای نمایش ویژگیهای کارت DO ظاهر میشود بخشهای General و Address را میبینیم که توضیحات آن مشابه کارت DI میباشد. بخش Output فقط برای برخی از کارتها که قابلیت خاص دارند مانند شکل زیر ظاهر میشود .



Wire Break : برای تشخیص قطعی سیم قبل از ارسال خروجی، جریان ثابتی در زمان کوتاهی به آن تزریق میشود و براساس ولتاژ نتیجه گیری بعمل می آید

No Load Voltage: برای تشخیص عدم وجود ولتاژ

Short Circuit to M : برای تشخیص اتصال کوتاه به زمین

Short Circuit to L+: برای تشخیص اتصال کوتاه در ٢٤ ولت

برای هر یک از موارد فوق میتوان کانال مورد نظر را علامت زد و فعال نمود.

Reaction To CPU Stop : توسط این قسمت میتوان تعیین کرد که در صورت توقف CPU خروجی کانال مربوطه از DO چگونه باشد . ممکن است بخاطر مسایل ایمنی کاربر بخواهد این خروجی ها را 1 نگهدارد. با انتخاب Substitute Value و علامت زدن کانالها در زیر آن این امر میسر میشود. بدیهی است خروجی کانالی که علامت نخورده 0 خواهد بود.

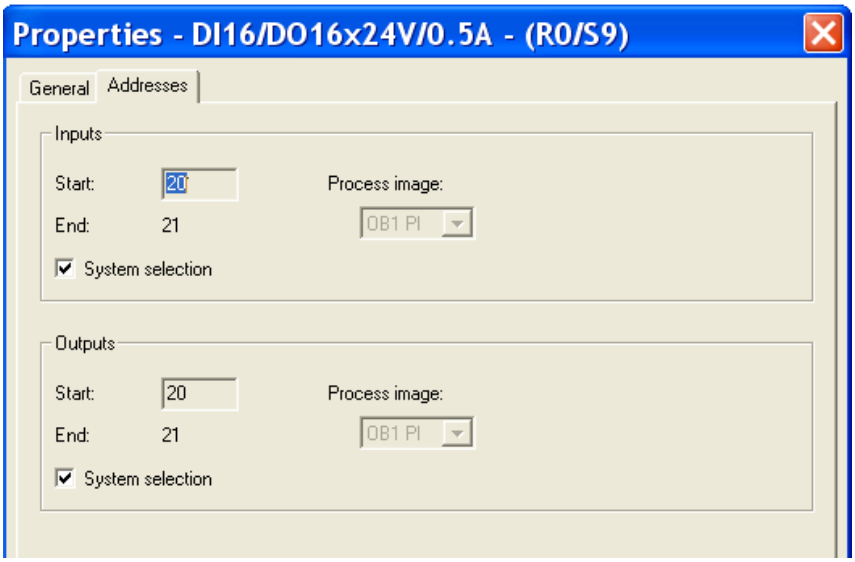
اگر Keep Last Value انتخاب گردد در هنگام توقف CPU آخرین مقدار قبلی سیگنال حفظ میگردد.

تنظیم پارامترهای کارتهای DI/DO

ایـن کارتهـا همـانطور کـه از نامشان پیداست ورودی و خروجی دیجیتال را باهم دارند. تنوع آنها کم است و دسته بندی آنها مانند جدول زیر میباشد.

تقسیم بندی کارتهای DI/DO			
از نظر تعداد ورودی / خروجی	از نظر جریان خروجی	از نظر ولتاژ	ارنظر قابلیت های خاص
• ۱۶ ورودی + ۱۶ خروجی • ۸ ورودی + ۸ خروجی	0.5 A	24 VDC	بدون ویژگی خاص

پارامـترهای ایـن کارتهـا صـرفاً شـامل دو بخـش General و Address میشود که مشابه DI و DO میباشد. تنها تفاوتی که وجود دارد اینست که بخش آدرس به دو قسمت یکی برای ورودی و دیگری برای خروجی تفکیک شده است.
لازم بذکر است که کارت DI/DO مختص S7-300 است و در نوع S7-400 وجود ندارد.



سیگنالهای آنالوگ

سیگنالهای Analoge Input استاندارد انواع مختلفی دارند ولتاژ، جریان و مقاومت از جمله این سیگنالها هستند . کارتهای ورودی آنالوگ ممکن است یکی یا ترکیبی از این سیگنالها را قبول کنند. قبل از پرداختن به تنظیمات کارتهای مزبور توضیحاتی را در مورد سیگنالها ارائه میدهیم.

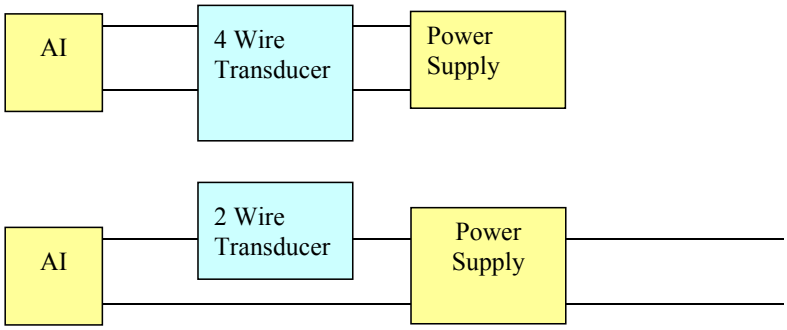
سیگنال آنالوگ از جنس ولتاژ

این سیگنالها انواع مختلفی بشرح زیر دارند.

- +/-25 mV
- +/-50 mV
- +/-80 mV
- +/-250 mV
- +/-500 mV
- +/-1 V
- +/-2.5 V
- +/-5 V
- 1.. 5 V
- +/-10 V

سیگنال آنالوگ از جنس جریان

جریان میتواند از ترانسدیوسر (مبدل) ۲ سیمه باشد یا از ترانسدیوسر ٤ سیمه . با توجه به شکل زیر میتوان دریافت که در نوع ٤ سیمه تغذیه سنسور از سیگنال آن جدا ست (۲ سیم برای تغذیه و ۲ سیم برای سیگنال) ولی در نوع ۲ سیمه تغذیه و سیگنال مشترک است.



سیگنال جریان از ترانسدیوسر های ۲ سیمه 4-20mA و در ترانسدیوسرهای ٤ سیمه بشرح زیر است:

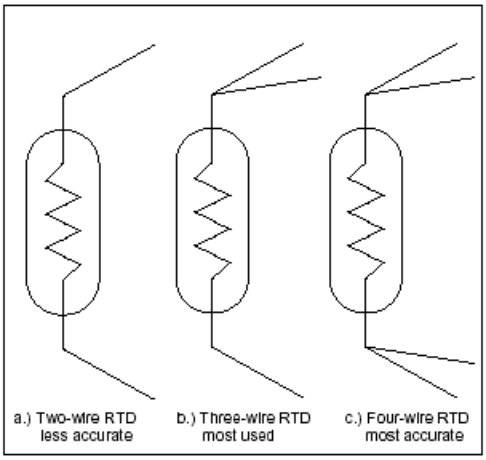
- -5 mA to +5 mA
- -10 mA to +10 mA
- -20 mA to +20 mA
- 0 mA to 20 mA
- 4 mA to 20 mA

سیگنال از نوع مقاومت

مقاومت میتواند یک مقاومت معمولی (Resistor) با اهم مشخص مانند انواع زیر باشد

48 Ω
150 Ω
300 Ω
600 Ω
6000 Ω

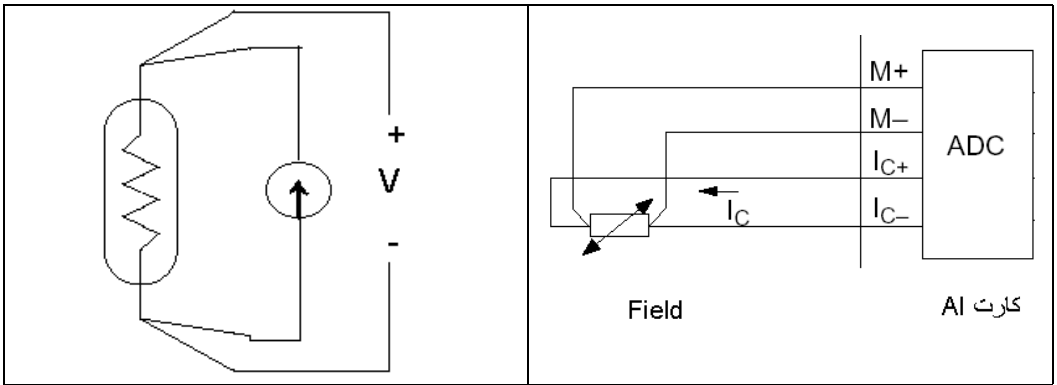
همینطور میتواند یک ترمومتر (RTD) از نوع ۲ سیمه یا ۳ سیمه یا ٤ سیمه باشد.



نوع ۲ سیمه کمترین دقت و نوع ٤ سیمه بیشترین دقت را دارد و کاربرد نوع سه سیمه مرسوم تر است. در نوع ۲ سیمه مقاومت کابل و دمای محیطی که کابل از آن میگذرد ایجاد خطا میکند ولی در نوع ٤ سیمه به ۲ سر ترمومتر منبع جریان ثابتی اعمال میگردد و ولتاژ در ۲ سر دیگر اندازه گیری میشود (شکل زیر). پس اولاً جریان عبوری از ترمومتر ثابت است ثانیاً از مسیری که در دوسری آن ولتاژ اندازه گیری میشود جریان نمیگذرد. بنابراین آنچه اندازه گیری میشود دقیقاً نسبت ولتاژ دو سر ترمومتر به جریان عبوری از آن یعنی در واقع مقاومت ترمومتر است. در نوع ۳ سیمه چون مسیر عبور جریان در یک طرف با نقطه ای که ولتاژ آن اندازه گیری میشود مشترک است اندک خطایی وجود دارد

ترمومترها انواع مختلف دارند که هر کدام برای رنج دمایی و شرایط خاصی مناسب هستند . معروفترین آنها Pt100 است . لیست انواع ترمومترها در زیر آورده شده است.

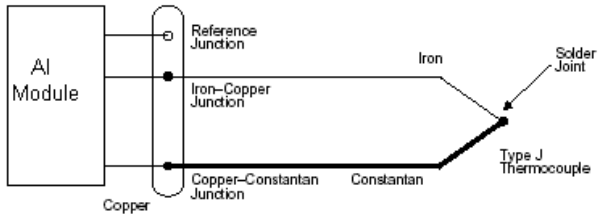
- **Pt100**
- **Pt200**
- **Pt500**
- **Pt1000**
- **Ni100**
- **Ni1000**
- .



ارتباط ترمومتر ٤ سیمه به کارت AI	ترمومتر ٤ سیمه

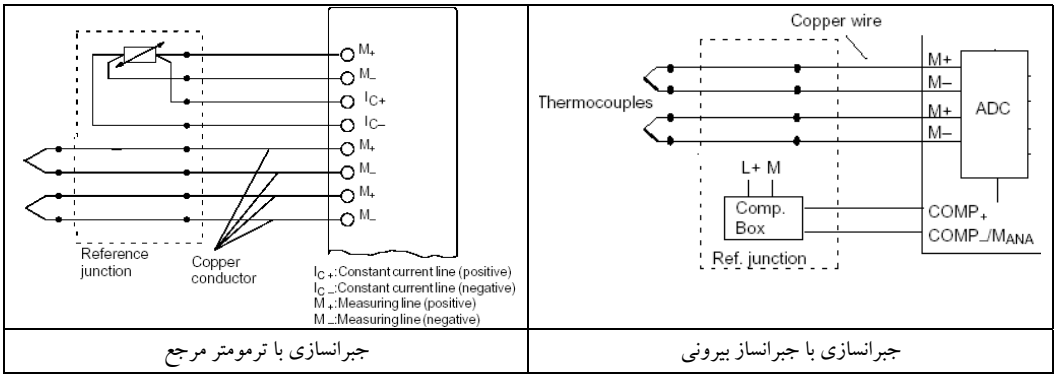
ترموکوپل ٤٧

همانطور که میدانیم ترمومتر از دو فلزغیر همجنس که از یک سمت به یکدیگر متصل هستند تشکیل شده است. با قرار گرفتن محل اتصال در محیط گرم سیگنالی بصورت mV در سمت آزاد ظاهر میشود که میتوان از آن استفاده کرد.



اگر کابلی که به ترموکوپل وصل میشود هر رشته آن همجنس فلز مربوطه از ترموکوپل یا به آن نزدیک باشد بدون هیچ مشکلی میتوان از سیگنال استفاده کرد. ولی اینکار به صرفه نیست زیرا کابل از جنس فلز ترموکوپل گرانقیمت خواهد بود. حال اگر از کابل مسی استفاده کنیم مشکلی که وجود دارد اینست که مانند شکل بالا در نقطه اتصال کابل به ترموکوپل دو ترموکوپل دیگر (بدلیل همجنس نبودن دو فلز تشکیل میشود که نهایتاً منجر به خطا در اندازه گیری میگردند. بنابراین ناچار هستیم از جبران کننده(Compensator) استفاده کنیم. یک روش اینست که دما را در نقطه اتصال کابل به ترموکوپل (Reference Junction) اندازه گیری کرده و از دمای ارسال شده به کارت کم کنیم.به این کار جبرانسازی بیرونی (External Compensating) گفته میشود.این امکان در کارتهای آنالوگی که ورودی ترموکوپل قبول میکنند پیش بینی شده و میتوان آنرا به جبران کننده بیرونی یا یک ترمومتر مرجع مطابق شکل زیر متصل نمود.

روش اول که از کابل همجنس ترموکوپل استفاده میشود ، جبرانسازی داخلی (Internal Compensating) نامیده میشود.

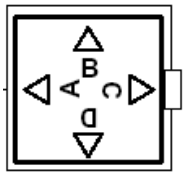


جبرانسازی با جبرانساز بیرونی | جبرانسازی با ترمومتر مرجع

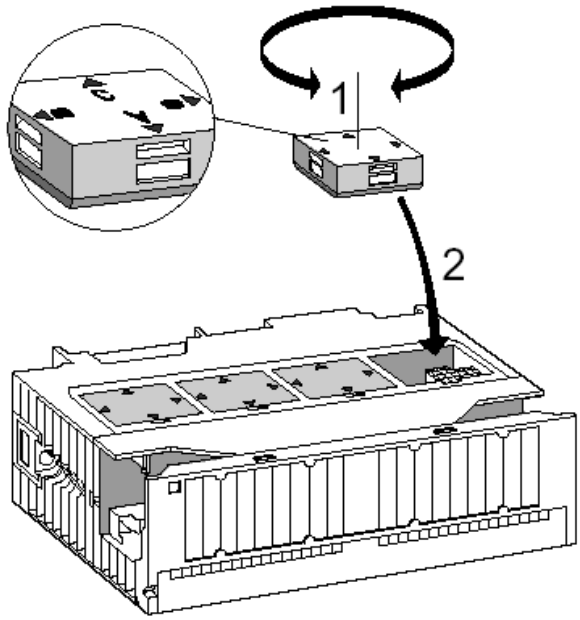
ترموکوپل ها بسته به جنس دو فلز آنها انواع مختلف دارند که هر کدام رنج دمایی خاصی را پوشش میدهند .(جدول زیر)

Thermocouple	Conductor		Temperature	Voltage Range
Type	Positive	Negative	Range (°C)	(mV)
E	Chromel	Constantan	-270° to 1,000°	-9.835 to 76.358
J	Iron	Constantan	-210° to 1,200°	-8.096 to 69.536
K	Chromel	Alumel	-270° to 1,372°	-6.548 to 54.874
T	Copper	Constantan	-270° to 400°	-6.258 to 20.869
S	Platinum-10% Rhodium	Platinum	-50° to 1,768°	-0.236 to 18.698
R	Platinum-13% Rhodium	Platinum	-50° to 1,768°	-0.226 to 21.108

تنظیم سخت افزاری لازم برای کارتهای AI



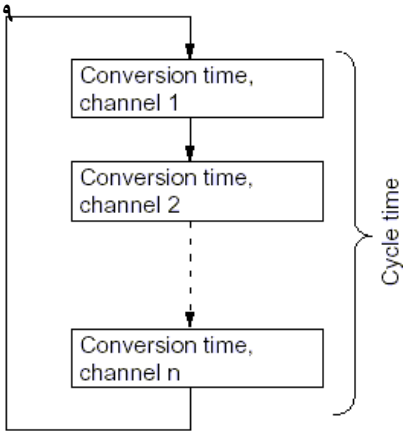
در کارتهای AI علاوه بر تنظیماتی که در نرم افزار Hwconfig اعمال میشود یک تنظیم سخت افزاری نیز لازم است. مدول کوچکی به نام Measuring Range Module در کنار کارت مطابق شکل وجود دارد که روی آن حروف D , C , B , A نوشته شده است بسته به نوع کارت و مطابق با دستورالعمل موجود در کاتالوگ آن این مدول باید تنظیم شود بنحویکه یکی از حروف فوق به سمت جلوی کارت قرار گیرد. در پارامترهای کارت که توسط Hwconfig تنظیم میشود وضعیت مدول فوق بعنوان راهنمایی داده میشود.



آنچه در شکل نشان داده شده است یک کارت AI 8x12 Bit (با کد سفارش 6ES7 331-7KF01-0AB0) است که انواع ورودی را قبول میکند هر دو کانال یک Measuring Range Module دارد که تنظیم آن باید مطابق جدول زیر باشد

.

وضعیت مدول	نوع سیگنال
A	ولتاژ mV
B	ولتاژ V
A	ترموکوپل
D	جریان از سنسور ۲ سیمه
C	جریان از سنسور ٤ سیمه
A	ترمومتر



نحوه خواندن سیگنالهای آنالوگ ورودی توسط PLC

بطور کلی همه مقادیر آنالوگ اعم از ولتاژ ، جریان و مقاومت ابتدا در کارت AI از آنالوگ به دیجیتال تبدیل میشوند سپس توسط CPU برای پردازش مورد استفاده قرار میگیرند.بنابر این برای PLC مقدار 0 و 1 یا بعبارت دیگری فرمت باینری یا Hex مفهوم دارد.

برای هر مدول آنالوگ یک Cycle Time وجود دارد. ابتدا در کانال ۱ تبدیل صورت میگیرد سپس کانال ۲ و نهایتاً کانال n . کل زمانی که این کار طول میکشد را Cycle Time میگویند. اگر کانال یا کانالهایی از کارت در عمل استفاده نشده باشد باید در پارامترهای کارت آنرا غیر فعال کنیم. اینکار منجر به کاهش زمان سیکل فوق خواهد شد.

پس از تبدیل مقدار باینری بصورت ۱۶ بیتی و مانند شکل زیر است . بیت آخر سمت چپ علامت عدد را نشان میدهد 0 برای مثبت و 1 برای منفی است.

Resolution	Analog Value															
Number of bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit weighting	VZ	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

برای هر نوع سیگنال جدولی وجود دارد که معادل Hex و Decimal ورودی مربوط به کارت را نشان میدهد .این جدولها در ضمیمه ۲ کتاب آورده شده اند. در این جداول علاوه بر رنج نرمال ، مقادیر بالای رنج ، زیر رنج ، Overflow و Underflow نیز داده شده اند.بعنوان مثال فرض کنید به کارت ورودی ، سیگنال 4-20 mA متصل شده است. PLC مقدار میلی آمپر را نمیشناسد. و در برنامه نویسی نمیتوان این مقادیر را مستقیماً بکار برد . ابتدا باید با توجه به کاتالوگ معادل Hex یا دسیمال آنهارا پیدا کرد سپس در برنامه از آنها استفاده نمود. بعنوان مثال مطابق جدول روبرو میبینیم که معادل دسیمال ٤ میلی آمپر عدد صفر و معادل دسیمال ۲۰ میلی آمپر عدد ۲۷۶٤۸ میباشد.این مقادیر از نظر PLC مفهوم دارد.

اگر خواننده مبناهای Hex و دسیمال و نحوه تبدیل آنها به یکدیگر را بخاطر ندارد میتواند به بخش چهارم کتاب مراجعه نماید.

Measuring Range 4 to 20 mA	Units		Range
	Decimal	Hexa-decimal	
> 22.810	32767	$7FFF_H$	Overflow
22.810 : 20.0005	32511 : 27649	$7EFF_H$: $6C01_H$	Overrange
20.000 16.000 : 4.000	27648 20736 : 0	$6C00_H$ 5100_H : 0_H	Nominal range
3.9995 : 1.1852	–1 : –4864	$FFFF_H$: $ED00_H$	Underrange
< 1.1852	–32768	8000_H	Underflow

ر جدول Resolution ضمیمه ۲ کتاب مشاهده میشود که وقتی گفته میشود Resolution ۱۵ بیتی است یعنی مقادیر یکی یکی پرش میکند و دقت بیشتر است. اگر Resolution ١٤ بیتی باشد یعنی دوتا دوتا پرش میکند و اگر ۸ بیتی باشد پله ها ۱۲۸ تایی است.

مقایسه نحوه تبدیل سیگنال آنالوگ در S7 و S5

شاید برای کاربرانی که با PLC های سری S5 زیمنس آشنایی دارند مقایسه بین نحوه تبدیل آنالوگ در S7 و S5 خالی از فایده نباشد.

همانطور که در صفحه قبل عنوان گردید در S7 سرریزی توسط مقدار سیگنال که از محاسبه ارزش ۱۶ بیت بدست می آید چک میشود. ولی در S5 یکی از بیتها برای Overflow رزرو شده که اگر 1 شد نشان میدهد مقدار سیگنال سرریز شده است. بطور کلی در S5 سه بیت برای مقاصد زیر رزرو شده است.

بیت 0 یعنی O برای Overflow رزرو شده است.

بیت 1 یعنی E برای Error Bit رزرو شده . مثلاً برای کارت با قابلیت خاص اگر این بیت 1 شود نشان دهنده Wire Break است.

بیت 2 یعنی A برای Activity Bit رزرو شده که نشان دهنده Valid یا Invalid بودن مقدار است.

Resolution	Analog Value															
Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Value (S7)	VZ	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Bit Value (S5)	PS	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	A	E	O

نتیجه اینکه رزرو شدن سه بیت در S5 منجر شده که Resolution حداکثر ۱۲ بیتی (با علامت) باشد . در حالیکه در S7 این بیتها آزاد است و Resolution حداکثر ۱۵ بیتی (با علامت) میباشد.

مقایسه نحوه تنظیم پارامترهای کارتهای آنالوگ در S7 و S5

در S7 بجز تنظیم مدول ذکر شده در صفحه ۴۷ سایر پارامترها توسط نرم افزار تنظیم میشوند. ولی در S5 این تنظیمات توسط Dip سوئیچهای روی کارت انجام میشد. شکل زیر یک نمونه از کارتهای S5 و نحوه تنظیم پارامترهای آنرا با سوئیچ مربوطه نمایش میدهد.

Table 11-1. Operating Mode Switch Settings for Analog Input Modules 464-8 to 11

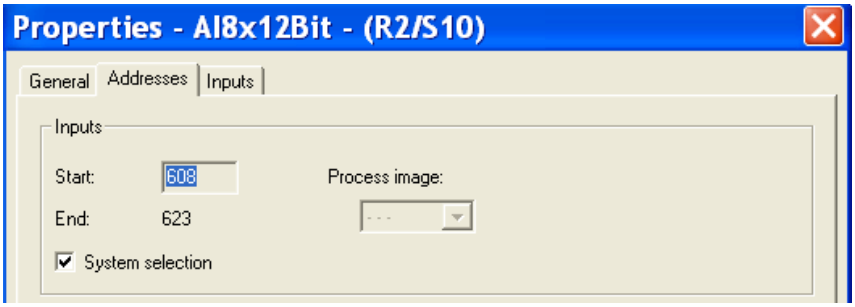
Function	Settings for Operating Mode Switch		
Power supply frequency	50 Hz 	60 Hz 	
Operation	1 channel (channel 0) 	2 channels (channel 0 and channel 1) 	4 channels (channel 0 - channel 3) 
Wire break	With wire break signal 	No wire break signal 	

تنظیم پارامترهای کارتهای AI

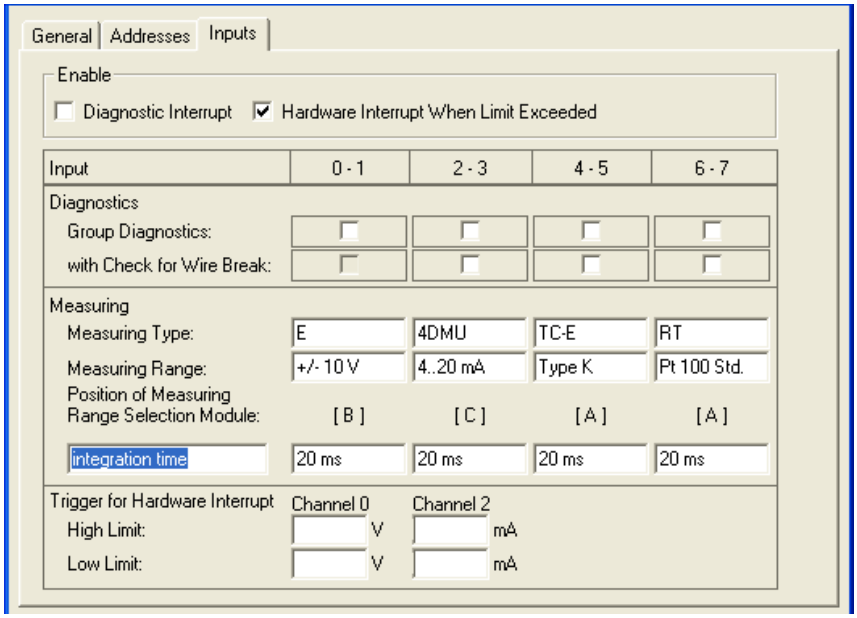
کارتهای Analog Input را میتوان بصورت زیر دسته بندی نمود.

از نظر تعداد ورودی	از نظر نوع سیگنال	ارنظر قابلیت های خاص
• ۲ ورودی • ٤ ورودی • ۸ ورودی	• ولتاژ • جریان • مقاومت • TC • RTD • ترکیبی از موارد فوق	• بدون ویژگی خاص • ایجاد وقفه • تشخیص قطعی

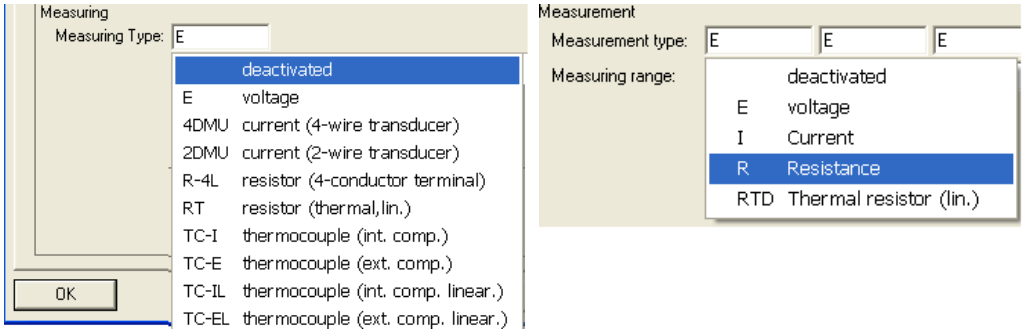
پس از وارد کردن هر کارت AI و کلیک روی آن پنجره ای که ویژگیهای آنرا نشان میدهد مانند شکل زیر باز میشود. با توضیحاتی که در مورد کارتهای دیجیتال داده شد بخش های General و Address روشن است و نیاز به شرح ندارد. صرفاً این نکته را باید افزود که برای هر ورودی آنالوگ ۲ بایت آدرس یا به عبارت دیگر یکWord اختصاص داده میشود. بنابراین برای کارت ۸ ورودی شکل زیر آدرس شروع 608 و آدرس انتها ۱٦ بایت بعد از آن یعنی 623 خواهد بود.



بخش Input در تمام کارتهای AI وجود دارد و برای انتخاب نوع سیگنال بکار میرود .بعنوان مثال حتی اگر کارت فقط ولتاژ را قبول کند، باز لازم است رنج آن را تعیین نمود.در عین حال اگر کارت ویژگی های خاص برای اعمال وقفه داشته باشد این پارامترها در بخش Input ظاهر میشوند. یکی از نکاتی که در بخش Input قابل توجه است وضعیت تنظیم سخت افزاری مدول Measuring Range میباشد که وضعیت A یا B یا C یا D آنرا مشخص میکند. ودر شکل زیر میتوان آنرا دید.

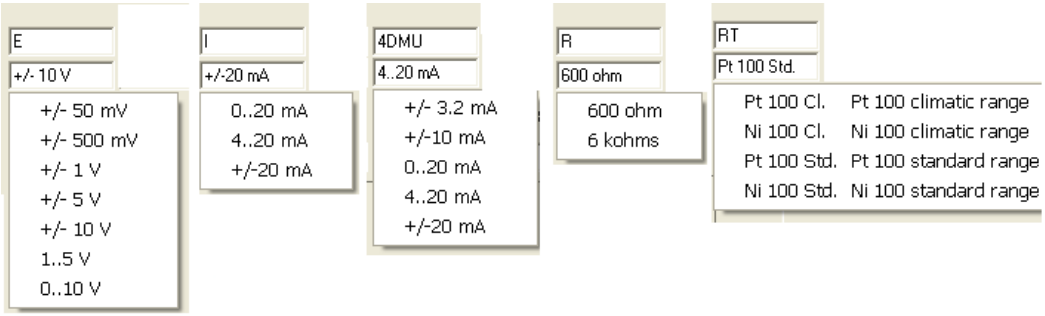


انتخاب نوع سیگنال برای هر کانال با کلیک کردن در جلوی سطر Measuring Type مانند شکل روبرو انجام میشود. بسته به نوع کارت ممکن است انتخابهای متفاوتی در این قسمت داشته باشیم. مثلاً در کارتی که خاص ترموکوپل است سایر سیگنالها ظاهر نمیشود. شکل زیر بخش Measurement type را برای دو نوع کارت مختلف نشان داده شده است.

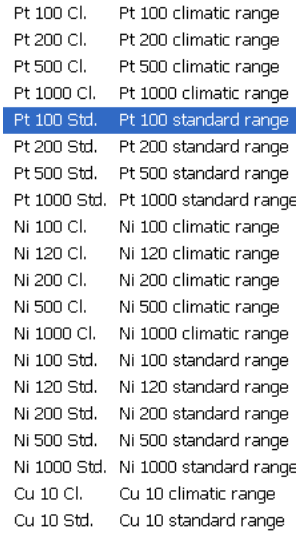


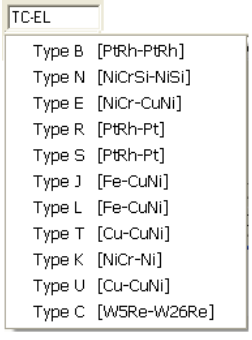
گزینه Deactivated که درپارامتر تمام کارتها ظاهر میشود برای غیرفعال کردن کانال است. طبق توضیحاتی که قبلاً داده شد اگر کانالی عملاً استفاده نشود برای اینکه زمان سیکل تبدیل آنالوگ به دیجیتال را کاهش دهیم این کانال را Deactivate می کنیم.

پس از انتخاب Measuring type در زیر آن اگر روی Measuring range کلیک کنیم لیستی که انواع رنج های قابل کاربرد برای آن سیگنال را در بر دارد نمایش داده میشود. شکل زیر نمونه ای از آنها را نشان میدهد.



برخی از کارتهای AI صرفاً برای ورودی خاصی مانند ترموکوپل یا ترمومتر طراحی شده اند. در این کارتها در بخش Measuring range تنوع بیشتری را مشاهده میکنیم شکل روبرو انواع ترمومتر در کارت مخصوص RTD و شکل زیر انواع ترموکوپل در کارت خاص TC را نشان میدهد. بعلاوه در این کارتها پارامترهای خاص دیگری مانند واحد اندازه گیری دما و ضرایب استاندارد برای RTD و امثال آن میتوان یافت.





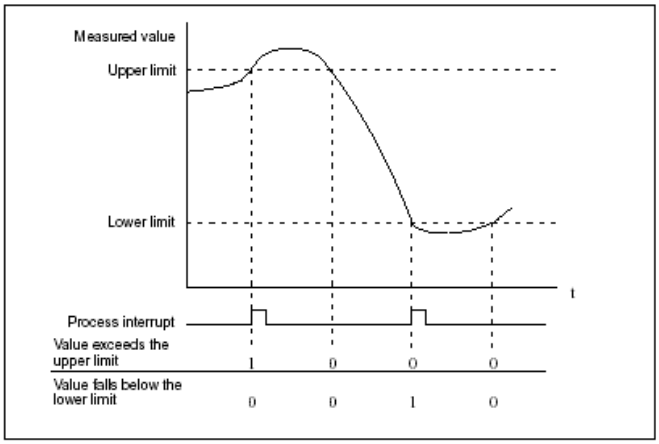
قابلیت های خاص کارتهای AI

مهمترین قابلیت های خاصی که در برخی کارتهای AI وجود دارد عبارتند از :

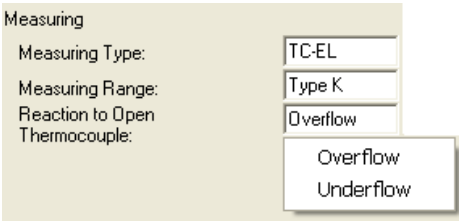
Diagnostics Interrupt : در صورت فعال شدن برای حالتهایی مانند Wire Break و قطع تغذیه شبیه کارتهای DI ایجاد وقفه میکند.

Hardware Interrupt : در صورت فعال شدن در صورتی که سیگنال از حدود تعیین شده تجاوز نماید ایجاد وقفه مینماید. این حدود را بعنوان High Limit و Low Limit میتوان در بخش Input مانند شکل صفحه ۵۰ دید.

در شکل زیر میتوان نحوه ایجاد وقفه وقتی سیگنال آنالوگ از حد بالا بیشتر یا از حد پایین کمتر میشود را ملاحظه کرد.

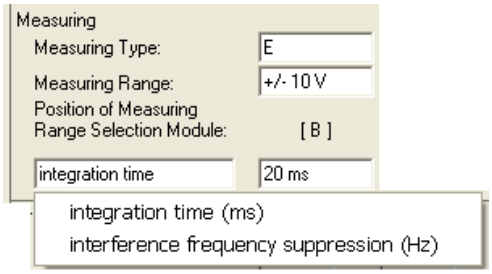


برای کارت AI 8xTC که خاص ترموکوپل است پارامتر دیگری با عنوان Reaction to Open Thermocouple وجود دارد. برای پروسه های گرمایشی باید Overflow در جلوی این پارامتر انتخاب شود. تا در صورت باز شدن ترموکوپل مقدار 7FFF توسط کارت برگردانده شود و لوپ کنترلی بطور اتوماتیک گرما را کاهش دهد. (وضعیت ایمن تر).



برای پروسه های سرمایشی باید Underflow در جلوی این پارامتر انتخاب شود. تا در صورت باز شدن ترموکوپل مقدار 8000 توسط کارت برگردانده شود و لوپ کنترلی بصورت اتوماتیک سرما را کاهش دهد. (وضعیت ایمن تر).

گزینه دیگری که در بسیاری از کارتهای AI وجود دارد Integration time است. این پارامتر در واقع Resolution را مطابق با جدول زیر تعیین میکند.



Resolution	Interference Frequency (HZ)	Integration Time (ms)
9+ Sign bit	400	2.5
12+ Sign bit	60	16.6
12+ Sign bit	50	20
14+ Sign bit	10	100

تنظیم پارامترهای کارتهای AO

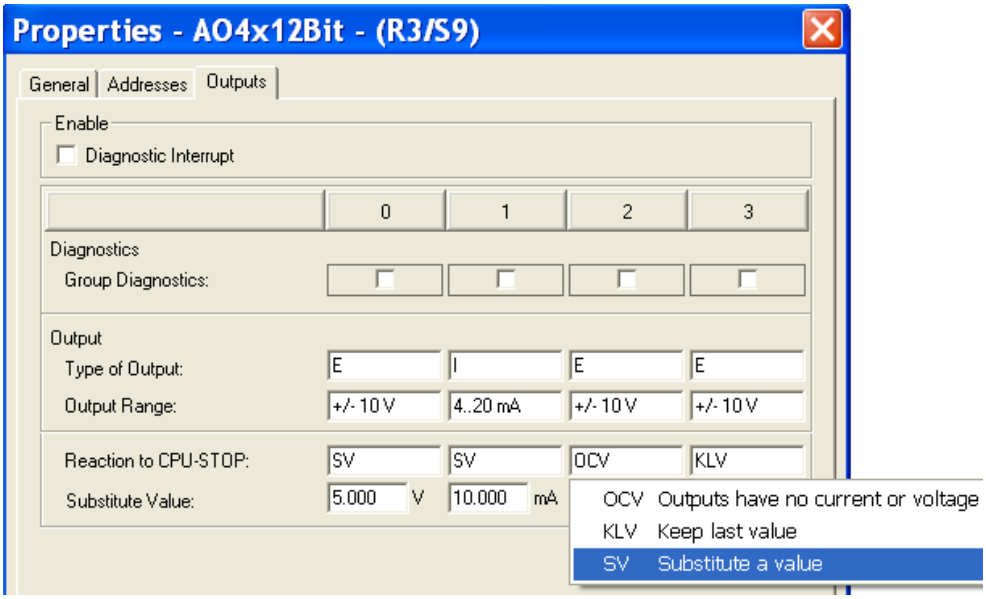
کارتهای Analog Output را میتوان بصورت زیر دسته بندی نمود.

از نظر تعداد خروجی	از نظر نوع سیگنال	ارنظر قابلیت های خاص
• ۲ ورودی	ولتاژ	بدون ویژگی خاص
• ٤ ورودی	جریان	تشخیص قطعی
• ۸ ورودی	ترکیب دو مورد فوق	تشخیص اتصال کوتاه
		واکنش به توقف CPU

ویژگی این کارتها در بخشهای General و Address نیاز به توضیح ندارد و آنچه برای کارتهای AI گفته شد در اینجا نیز صادق است از اینرو صرفاً در مورد بخش Output که در شکل زیر نیز نمایش داده شده مطالبی بیان میشود.

همانطور که در جدول بالا آورده شده خروجی این کارتها یا از جنس جریان است یا از جنس ولتاژ. برای هر یک از این دو حالت رنج های مختلفی در جلوی Output Range قابل انتخاب است.

بجز E و I گزینه سومی که برای انواع خروجی وجود دارد Deactivated است. همانطور که قبلاً برای ورودی آنالوگ توضیح داده شد برای خروجی های آنالوگ نیز یک سیکل تبدیل وجود دارد . در این سیکل مقادیر باینری تولید شده توسط CPU به مقادیر متناسب با جنس خروجی (ولتاژ یا جریان) تبدیل میگردند. جداول مربوط به این تبدیل نیزدر ضمیمه ۲ آورده شده است. حال اگر کانالی عملاً استفاده نشود با گزینه Deactivated آنرا غیر فعال میکنیم تا زمان سیکل تبدیل کاهش یابد.

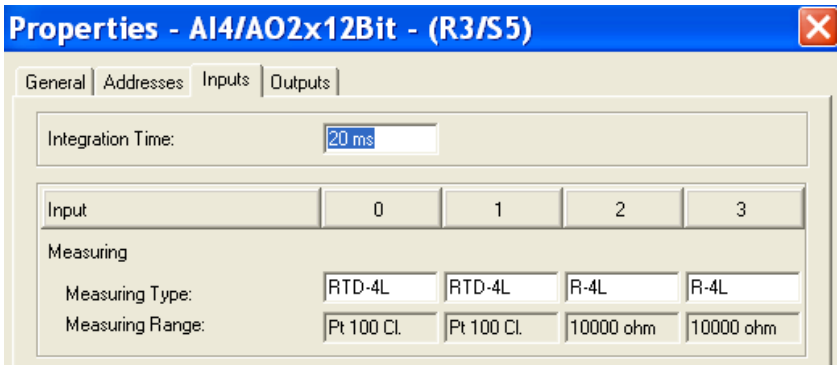


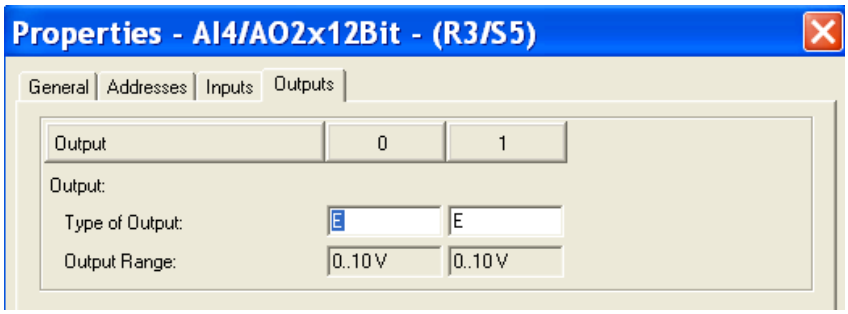
از ویژگی های خاص این کارت Diagnostic Interrupt است که اگر فعال شود و سپس Group Diagnostics برای کانال مورد نظر علامت بخورد در مواردی مانند اتصال زمین (برای خروجیهای ولتاژ) و اتصال کوتاه (برای خروجیهای جریان) و عدم وجود تغذیه وقفه اعمال میکند.

واکنش به قطع شدن CPU پارامتر دیگری است که قابل تنظیم است. OCV یعنی خروجی صفر شود، KLV یعنی آخرین مقدار قبلی حفظ گرددو SV یعنی خروجی برابر با مقدار مشخصی که در زیر آن تعیین میشود گردد. تنظیم این پارامتر با توجه به پروسه و موارد ایمنی باید انجام گیرد.

تنظیم پارامترهای کارتهای AI/AO

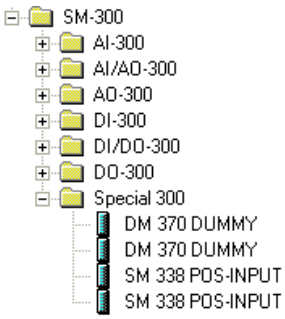
این کارتها ترکیبی از ورودی و خروجی را شامل میشوند.با توجه به توضیحاتی که تاکنون ارائه شده پارامترهای آنها نیاز به شرح ندارد. شکلهای زیر نشان میدهد که برای این کارتها دو بخش Input و Output وجود داردکه برای هرکدام میتوان نوع ورودی و خروجی را مشخص نمود و سایر پارامترها راتنظیم کرد.





تنظیم پارامترهای کارتهای Special 300

بجزکارتهای I/O در زیر مجموعه مدولهای SM از S7-300 کارتهای دیگری را تحت عنوان Special 300 میتوان یافت . مطابق شکل روبرو میبینیم که یکی از آنها کارت Dummy نام دارد. این کارت دارای هیچ ورودی یا خروجی خاصی نیست و همانطور که از اسمش پیداست یک مدول مجازی یا قلابی است. این مدول میتواند همانند سایر کارتهای I/O در اسلات ٤ تا ١١ قرار گیرد و کاربرد آن صرفاً برای رزرو کردن محل و آدرس آن اسلات برای استفاده در آینده است.



اگر کاربر بخواهد در بین کارتهای I/O اسلاتی را برای استفاده در آینده رزرو کند از آنجا که نباید بین کارتها در اسلات فاصله باشد این کارت را بکار می برد تا بدون اینکه کار خاصی انجام دهد ارتباط بین دوطرف را با کانکتور مربوطه برقرار سازد.

کارت خاص دیگری که در این گروه وجود دارد SM 338 است که برای ارتباط با انکودر در نظر گرفته شده است.

تنظیم پارامتر های CPU

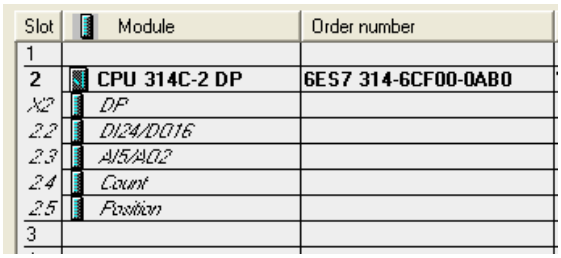
CPU های 300 انواع مختلفی دارند که توانایی آنها متفاوت است درجدول زیرمقایسه برخی پارامترهای اصلی CPU های غیرکمپکت این خانواده ارائه شده است. توصیه میشود در فاز طراحی قبل از انتخاب CPU مشخصات فنی آن از آخرین کاتالوگ سازنده استخراج و بدقت مطالعه گردد.برای انتخاب CPU از یکطرف نباید دست بالا یا باصطلاح Over Design عمل کرد و از طرف دیگر نباید آنرا دقیقاً مطابق با نیاز موجود انتخاب نمود بنحویکه امکان توسعه در آینده نداشته باشد.

CPU 318-2DP	CPU 315-2DP	CPU 314	CPU 312	
256 KB	128 KB	48 KB	16 KB	**RAM**
512	256	256	128	**Counter**
512	256	256	128	**Timer**
8192 byte	2048 byte	256 byte	128 byte	**Bit Memory**
65536 byte Input 65536 byte Output	128 byte	128 byte	128 byte	**Digital Channel**
4096 byte Input 4096 byte Output	16384 byte	1024 byte	256 byte	**Analog Channel**
max. 4	max. 4	max. 4	max. 1	**Rack**
Yes	Yes	No	No	**Profibus DP**

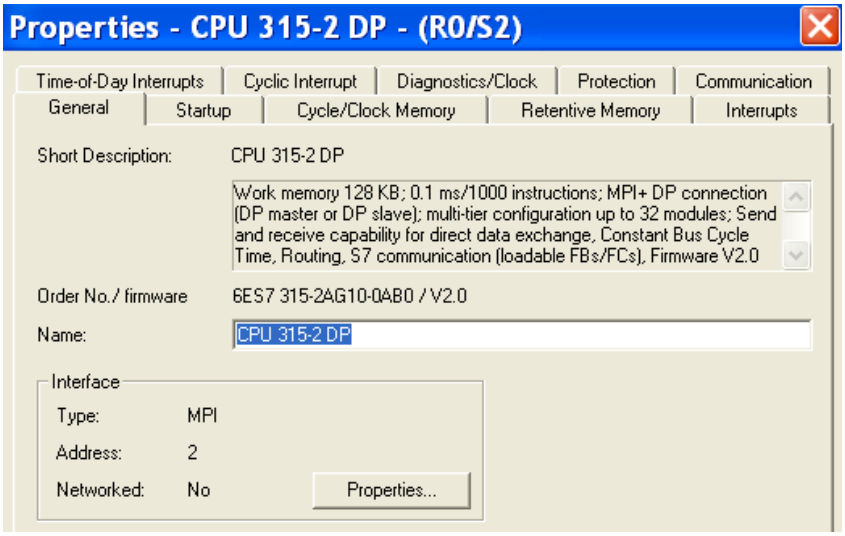
نکات زیر در موقع وارد کردن CPU به اسلات دوم در برنامه Hwconfig قابل توجه است:

۱. CPU هایی که در انتهای کد آنها کلمه 2DP نوشته شده مستقیماً دارای پورت ارتباط با شبکه پروفی باس میباشند. در موقع وارد کردن آنها در اسلات ، پنجره ای ظاهر میشود که میتوان از همینجا با کلیک کردن روی New شبکه ارتباطی را مشخص کرد. پس از وارد شدن CPU در اسلات میبینیم که یک سطر بنام DP در زیر آن ظاهر میگردد.

۲. CPU های کمپکت که در انتهای کد آنها حرف C نوشته شده وقتی وارد اسلات میشوند در زیر آنها مدولهای متصل به CPU نیز مانند شکل زیر ظاهر میگردند.در واقع مجموعه CPU و مدولهای متصل به آن همگی در اسلات دوم قرار میگیرند. تنظیم ورودی و خروجی های این بخش نیز شبیه ورودی و خروجی های کارتهای I/O انجام میشود ولی در اینجا مدول سخت افزاری شبیه آنچه در صفحات قبل ذکر شد وجود ندارد.



پس از وارد شدن CPU در اسلات با کلیک کردن روی آن پنجره ای که پارامتر های آنرا نمایش میدهد مانند شکل زیر ظاهر میگردد:



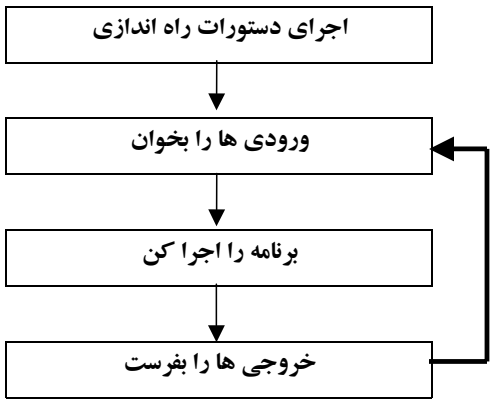
.بسته به نوع CPU ممکن است برخی بخشها موجود نباشند یا در برخی بخشها گزینه هایی فعال نباشند.

قبل از شروع به تنظیم پارامترهای CPU لازم است برخی مفاهیم برای کاربر روشن باشد این مفاهیم در زیر یادآوری میگردد.

سیکل اسکن CPU

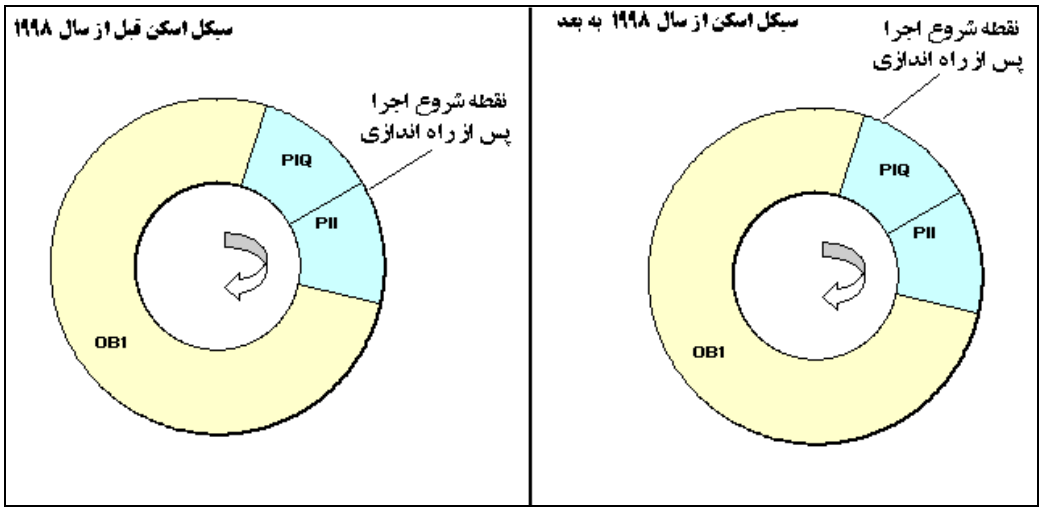
وقتی PLC روشن میگردد ابتدا مرحله راه اندازی (Startup) را طی میکند و اگر برنامه ای از قبل برای این مرحله نوشته شده باشد آنرا اجرا میکند پس از آن وارد مرحله Run میشود . مرحله RUN بصورت سیکلی دائماً اجرا میشود تا زمانیکه فرمان توقف(STOP) داده شود. بنابراین مد راه اندازی فقط یکبار و مد RUN مرتباً اجرا میگردد.

CPU در مد RUN ورودیها را (از کارتهای ورودی یا شبکه) میخواند سپس برنامه ای که از قبل در حافظه آن نوشته شده را اجرا میکند و بعد از آن خروجیهای تولید شده را (به کارتهای خروجی یا شبکه) میفرستد . این عملیات سه گانه مجدداً در سیکلهای بعدی تکرار میشود. اجرای مراحل فوق در واقع توسط سیستم عامل (Operating System) پشتیبانی میشود. اگر در بین اجرای سیکل وقفه ای اعمال شود سیستم عامل اجرای برنامه سیکلی را قطع کرده بسراغ وقفه میرود و پس از آن برنامه را ادامه میدهد.



در حافظه CPU بخشی برای ورودی بنام Process Image Input یا PII وجود دارد. در هر سیکل CPU تصویری از ورودیها را در این قسمت ذخیره میکند بعبارت دیگر مثل اینست که از مقادیر ورودی ها که در آن لحظه روی کارتهای ورودی موجود هستند در یک لحظه کوتاه عکسبرداری اانجام میشود. این مقادیر در برنامه سیکلی مورد استفاده قرار میگیرند. برنامه اصلی PLC که بصورت سیکلی اجرا میشود در بلاکی بنام OB1 یا Organization Block 1 نوشته میشود پس از اجرای برنامه خروجی های تولید شده دربخشی از حافظه CPU موسوم به PIQ یا Process Image Output ذخیره شده و از آنجا به کارتهای خروجی ارسال میگردند. اختصاص جداول PII و PIQ در حافظه سیستم برای سرعت بخشیدن به دسترسی CPU به مقادیر ورودی و خروجی نسبت به حالتی است که دیتاها مستقیماً از مدولها گرفته شده یا به آنها داده شوند.

سیکل اسکن در CPU هایی که توسط زیمنس تا سال 1998 ساخته شده از مدل بالا پیروی میکند. از 1998 در این سیکل یک جابجایی صورت گرفته است . ابتدا خروجی ها ارسال میگرددسپس ورودی ها خوانده میشود و بعد ازآن برنامه اجرا میشود. باکمی دقت در سیکل جدید میتوان دریافت که در حالت RUN عملاً تغییری در اجرای مراحل بوجود نیامده است و ترتیب مانند قبل است ولی در اولین مرتبه وقتی CPU از مد راه اندازی به مد RUN وارد میشود با مرحله " خروجی ها را بفرست" مواجه میگردد که منطقی بنظر میرسد یعنی اگر در دستورات راه اندازی خروجی هایی تولید شده باشند لازم است قبل از اینکه به مد RUN وارد شود و قبل از اینکه برنامه سیکلی بتواند آنها را تغییر دهد این خروجیها فرستاده شوند.



زمانیکه طول میکشد یک سیکل اسکن اجرا شود از جمع زمانهای زیر بدست می آید:

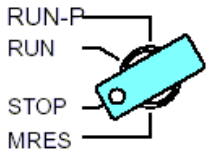
- زمان مربوط به خواندن ورودی و Update کردن PII
- زمان مربوط به ارسال خروجی ها از PIQ
- زمان پردازش برنامه اصلی
- زمان مربوط به سیستم عامل
- زمان مربوط به تبادل دیتا با شبکه

مدهای کاری PLC

مدهای کاری مهم PLC عبارتند از :

Stop: در این مد پردازش برنامه متوقف میشود ، دسترسی به I/O ها وجود ندارد. CPU بصورت Read و Write قابل دسترسی است. یعنی میتوان برنامه آنرا خواند و یا برنامه جدیدی را به آن انتقال داد.

Run: در این مد برنامه اجرا میشود. CPU به I/O ها دسترسی دارد.برنامه CPU بصورت Read Only است یعنی نمیتوان برنامه جدیدی را به آن Download کرد.



Run-P: در این مد برنامه اجرا میشود. CPU به I/O ها دسترسی دارد . در عین حال CPU بصورت Read و Write قابل دسترسی است.

تذکر : مدهای فوق توسط سوئیچ روی CPU (شکل روبرو) نیز قابل انتخاب است. در برخی از CPU ها این سوئیچ حالت Run-P ندارد.

MRES : این وضعیت برای ری ست کردن حافظه CPU بکار میرود یعنی هم مقادیر متغیرهای حافظه و هم برنامه ای که توسط کاربر به حافظه ارسال شده پاک میگردد. جزئیات بیشتر راجع به حافظه CPU در بخش بعد آمده است.

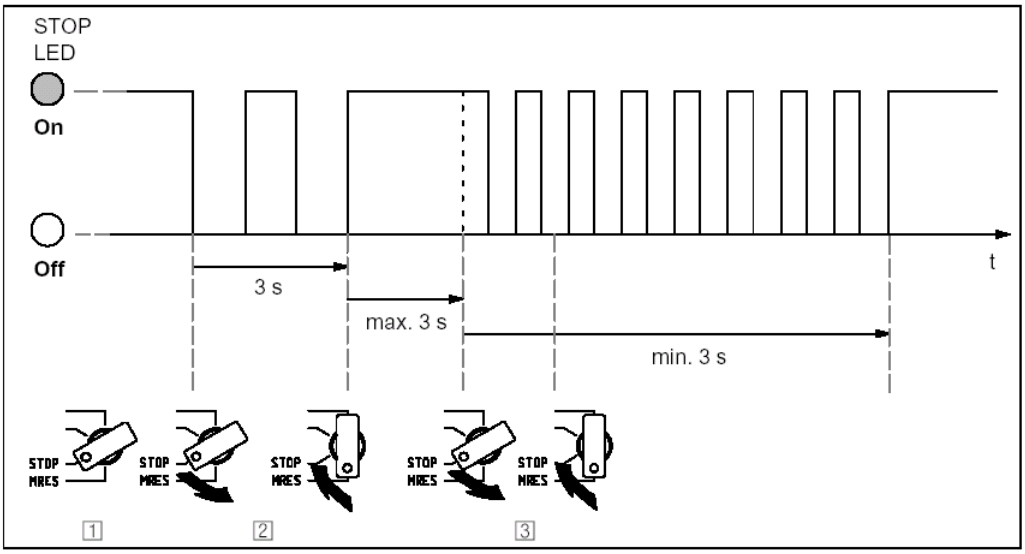
برای ری ست کردن CPU باید سوئیچ طبق سیکل زیر بین وضعیت MRES و STOP جابجا شود.

۱: سوئیچ در وضعیت Stop است و LED مربوط به Stop روشن است.

۲- سوئیچ را از STOP به MRES میبریم و ۳ ثانیه نگه میداریم و مجدداً آنرا به Stop برمیگردانیم.

۳. با کمی مکث (حداکثر ۳ ثانیه) سوئیچ را از Stop دوباره به MRES میبریم . LED فوق به حالت چشمک زن سریع در می آید.

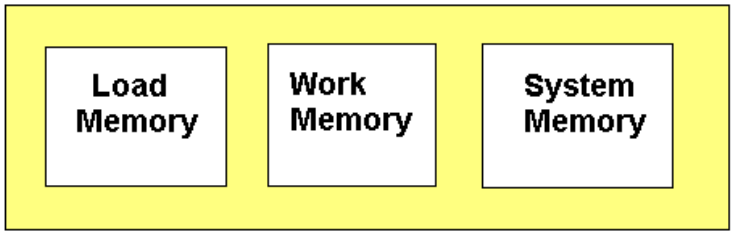
حالت چشمک زن فوق نشان دهنده اینست که حافظه CPU ری ست شده است. اگر در این مرحله LED بصورت چشمک زن در نیامد باید مراحل فوق از اول تکرار شود.



ری ست کردن از طریق Step7 : با استفاده از منوی PLC > Clear / Reset که در برنامه های مختلف Step7 از جمله در Hwconfig وجود دارد میتوان عمل ری ست را براحتی وبدون نیاز به سیکل عملیات سخت افزاری فوق انجام داد .

حافظه CPU های S7-300

بطور کلی حافظه CPU های S7 (اعم از 300 و 400) ساختاری شبیه شکل زیر دارد.



بخش های اصلی حافظه CPU های S7

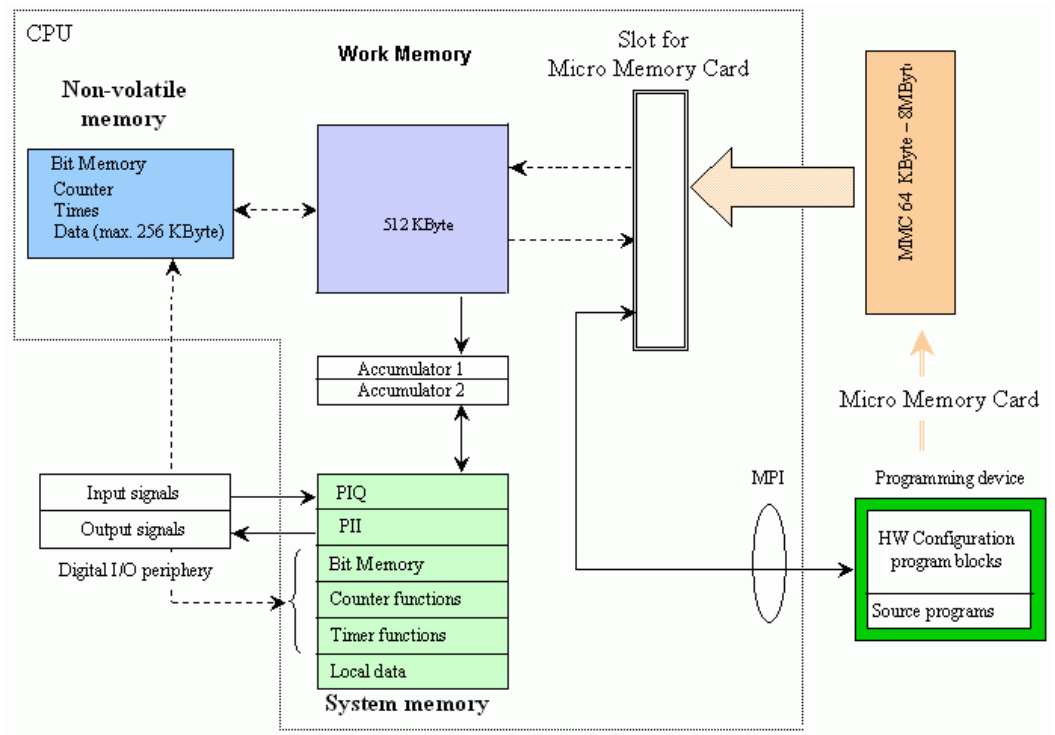
Load Memory: وقتی برنامه به CPU ارسال (Download) میشود در این قسمت وارد میگردد.

Work Memory: این حافظه بخشی از برنامه که اجرایی است را در بر میگیرد.بعنوان مثال یک فانکشن فقط در زمانی که صدا زده میشود

System Memory: این بخش عناصر حافظه مانند جداول PIQ , PII ، فلگ ها ، تایمرها ، کانترها و .... را در بر میگیرد.

در صورت ری ست شدن CPU کل محتویات بخش های Work Memory و System Memory پاک میشود. بخش Load Memory نیز بسته به نوع آن ممکن است پاک گردد.

طراحی بخشهای فوق در CPU های مختلف متفاوت است. در برخی CPU ها مانند 318-2DP حافظه Load Memory از نوع RAM یا EPROM است. اکثر CPU های جدید 300 فاقد Load Memoryداخلی هستند ویک کارت حافظه بیرونی بنام MMC (Micro Memory Card)که مانند شکل زیر در اسلات CPU قرار میگیرد بعنوان Load Memory آنها تلقی میگردد. بدون این کارت CPU راه اندازی نمیشود.



قبلاً طراحی بصورتی بود که حافظه CPU علاوه بر داشتن Load Memory داخلی میتوانست توسط یک کارت حافظه Memory Card از جنس RAM یا Flash EPROM افزایش پیدا کند . در طراحی جدید د ر برخی موارد MMC یا Master Memory Card که از نوع FEPROM است جایگزین کارتهای قبلی و Load Memory داخلی شده است. سایز MMC ها متفاوت بوده و کاربر میتواند بسته به نیاز آنها را سفارش دهد. بعنوان مثال در CPU 315-2DP سایز این کارت میتواند تا 8 MB باشد.

Non-Volatile Memory : همانطور که در شکل صفحه قبل ملاحظه میشود علاوه بر ۳ بخش ذکر شده در حافظه CPU بخش دیگری نیز موسوم به NV Memory وجود دارد. این بخش قابل برنامه ریزی است و میتوان تعیین کرد که چه دیتاهایی در آن ذخیره شوند. به دیتاهایی که در NVRAM تعریف میشوند Retentive گفته میشود. مقادیر این دیتاها در صورت قطع و وصل تغذیه میتواند حفظ شده و از بین نرود. حفظ شدن این اطلاعات بستگی به نوع راه اندازی CPU دارد که در قسمت بعد تشریح میگردد.

بعنوان مثال اگر یک کانتر در بخش NVRAM تعریف شود و در بین شمارش آن تغذیه قطع شود میتوان نحوه راه اندازی را طوری تعریف کرد که با وصل مجدد تغذیه مقدار قبلی حفظ شده و از آنجا به بعد را بشمارد.

انواع راه اندازی

سه نوع راه اندازی برای CPU های S7 وجود دارد:

۱. Cold Restart
۲. Warm Restart
۳. Hot restart (خاص S7-400)

Cold Restart

- تمامی تایمرها، کانترها و فلاگها ری ست میشوند چه از نوع قابل ذخیره (Retentive) باشند چه نباشند .
- برنامه از اولین دستور OB1 اجرا میگردد.

Warm Restart

- آنچه بعنوان Retentive تعریف شده پاک نمیشود.
- برنامه از اولین دستور OB1 اجرا میگردد.

Hot Restart

- هیچ دیتایی پاک نمیشود چه از نوع Retentive باشد چه نباشد.
- برنامه از جایی که قطع شده بود ادامه می یابد.
- خاص S7-400 است.
- CPU در اینحالت باید باتری Backup دارد.

شکل صفحه بعد توالی عملیات CPU را در هنگام گذر از مد Stop به RUN نشان میدهد. روشهای راه اندازی فوق با جزئیات مربوطه در این شکل ترسیم شده اند.

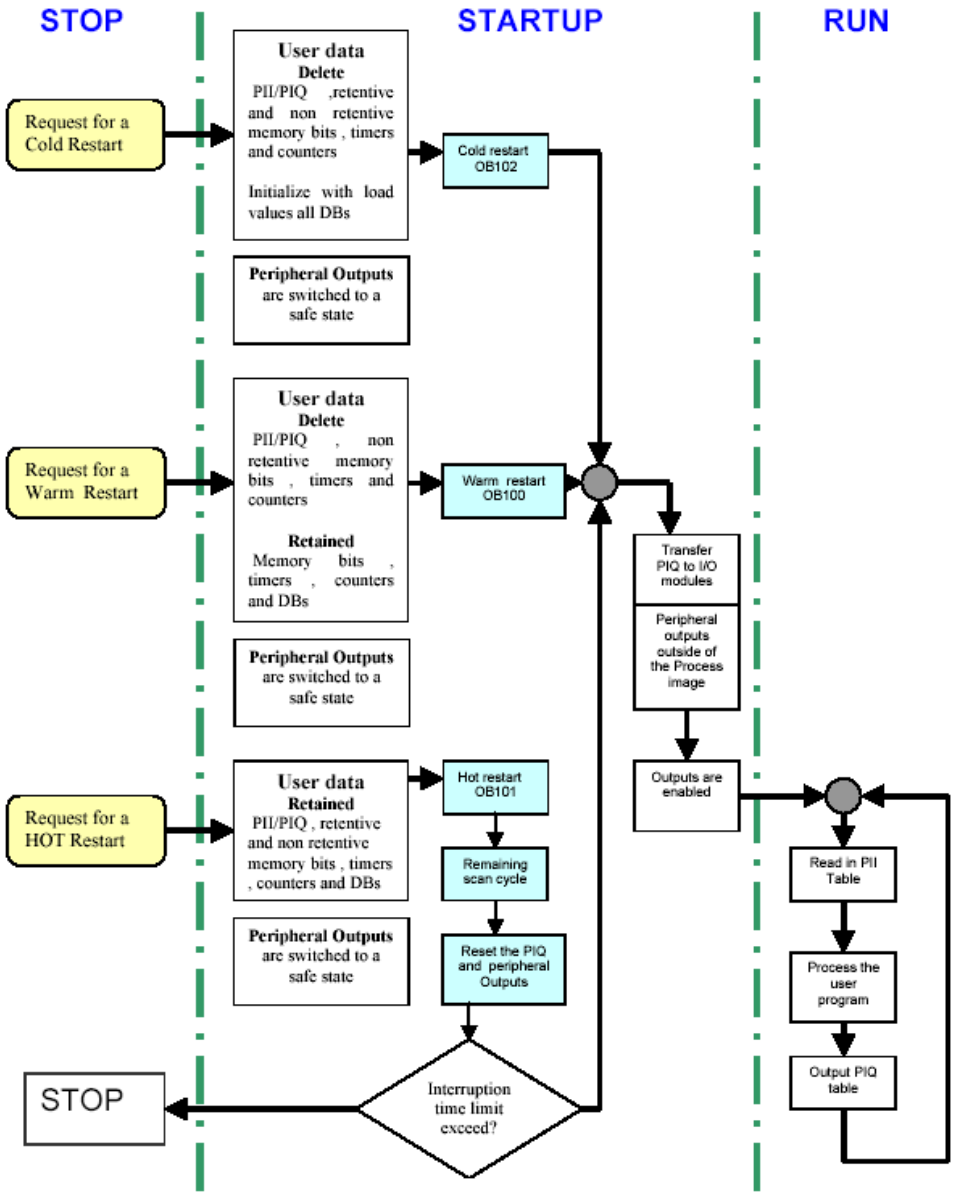
مد HOLD

در این مدپردازش برنامه متوقف شده و کاربر میتواند برنامه را قدم به قدم توسط PG یا PC تست کند. این مد کاری برای تست و عیب یابی برنامه بویژه در زمانی بکار میرود که برنامه نویسی از نظر دستورات صحیح است ولی بعلت وجود اشکال در منطق برنامه جواب مورد نظر بدست نمی آید. در این مد Debug کردن برنامه فعال میشود و میتوان نقاط قطع (breakpoint) تعریف نمود . نحوه Debug کردن برنامه جداگانه بحث خواهد شد.

اولویت مدهای کاری CPU

اگر چند مد بطور همزمان درخواست شود ، مدی که دارای اولویت بالاتری است انتخاب میشود. بعنوان مثال اگر سوئیچ CPU روی RUN قرار گیرد و همزمان از طریق PG مد STOP انتخاب شود ، CPU به مد STOPمیرود زیرا اولویت بالاتری دارد. این اولویتها در جدول زیر آمده است.

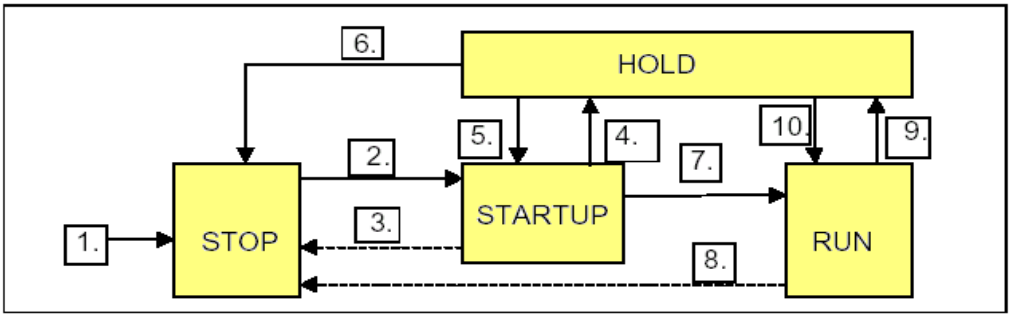
اولویت	مد کاری
بیشترین	STOP
	HOLD
	STARTUP
کمترین	RUN



توالی عملیات CPU

مراحل تغییر مدهای کاری CPU

شکل زیر مراحل تغییر مدهای CPU را نشان میدهد توضیحات مربوطه در جدول زیر آن آمده است.

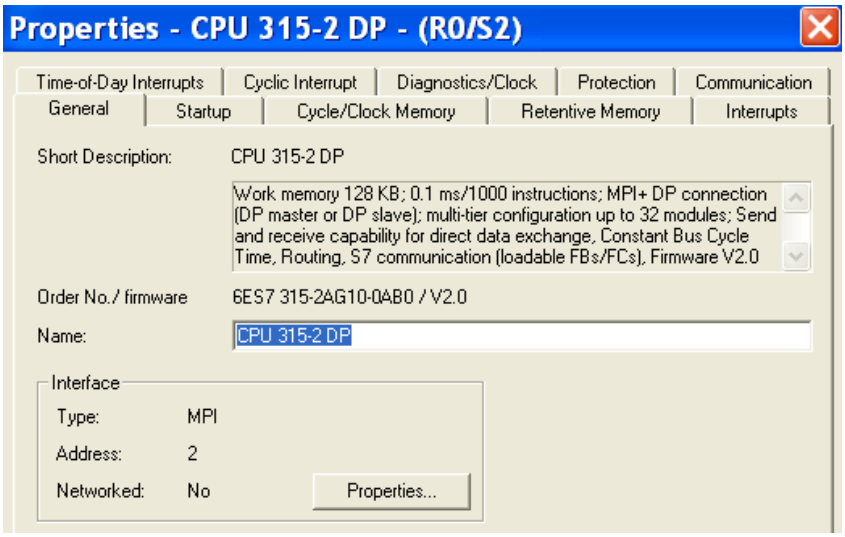


شرح	حالت تغییر
بعد از وصل تغذیه CPU در مد STOP قرار میگیرد.	1
CPU از مد STOP به مد STARTUP میرود وقتی که : • در حالت RUN یا RUN-P قرار گیرد. • راه اندازی بطور اتوماتیک با وصل تغذیه انجام شود.	2
CPU از مد STARTUP به مد STOP برمیگردد وقتی که: • اشکالی در طول راه اندازی بوجود آید. • توسط کاربر در مد STOP قرا گیرد (از طریق Step7 یا سوئیچ PLC) • دستور STOP در OB راه اندازی نوشته شود یا فانکشن STOP اجرا شود.	3
CPU از مد STARTUP به مد HOLD میرود وقتی که: • در برنامه راه اندازی به نقطه Breakpoint که کاربر تعیین کرده است برسد.	4
CPU از مد HOLD به مد STARTUP برمیگردد وقتی که: • بعد از نقطه Breakpoint دستور EXIT HOLD اجرا شود.	5
CPU از مد HOLD به مد STOP برمیگردد وقتی که: • توسط کاربر در مد STOP قرا گیرد(از طریق Step7 یا سوئیچ PLC) • دستور STOP در برنامه اجرا شود.	6
CPU از مد STARTUP به مد RUN میرود وقتی که: • نتیجه راه اندازی رضایتبخش باشد.	7
CPU از مد RUN به مد STOP برمیگردد وقتی که: • در مد RUN اشکالی آشکار شود و OB مربوط به خطاها برنامه ریزی نشده باشد. • توسط کاربر در مد STOP قرا گیرد(از طریق Step7 یا سوئیچ PLC) • دستور STOP در برنامه اجرا شود.	8
CPU از مد RUN به مد HOLD میرود وقتی که: • در برنامه کاربری به نقطه Breakpoint که کاربر تعیین کرده است برسد.	9
CPU از مد HOLD به مد RUN برمیگردد وقتی که: • بعد از نقطه Breakpoint دستور EXIT HOLD اجرا شود.	10

با توضیحاتی که ذکر شد اکنون میتوان برخی از پارامترهای CPU ها را معرفی نمود.

تنظیم پارامترهای CPU های S7-300

با کلیک کردن روی CPU در اسلات مربوطه پنجره ای باز میشود که پارامترهای مختلفی در آن قسمت بندی شده اند ممکن است بسته به نوع CPU برخی از این قسمتها یا برخی گزینه های داخل آن فعال نباشند.



بخش های مهم عبارتند از :

General : در این بخش همانطور که در شکل ملاحظه میشود یکسری اطلاعات کلی راجع به CPU آمده است. تنها تنظیمی که در این بخش میتوان انجام داد در قسمت Interface است . در این قسمت آدرس MPI مربوط به CPU آمده است.

MPI یا Multi Point Interface یکی از انواع شبکه های Simatic است که مفصلاً در مبحث مربوط به شبکه ها تشریح خواهد شد. در اینجا همینقدر ذکر میکنیم که پورت MPIروی CPU که عمدتاً برای ارتباط با PG یا PC استفاده میشود دارای یک آدرس است . این آدرس بصورت پیش فرض ۲ میباشد ونیازی به تغییر آن نیست با این وجود کاربر در صورت تمایل میتواند آنرا و همچنین سایر ویژگیهای مربوط به MPI را از همینجا تغییر دهد.

Cycle/ Clock Memory : در این قسمت دو گزینه برای تنظیم زمان سیکل اسکن به شرح زیروجود دارد

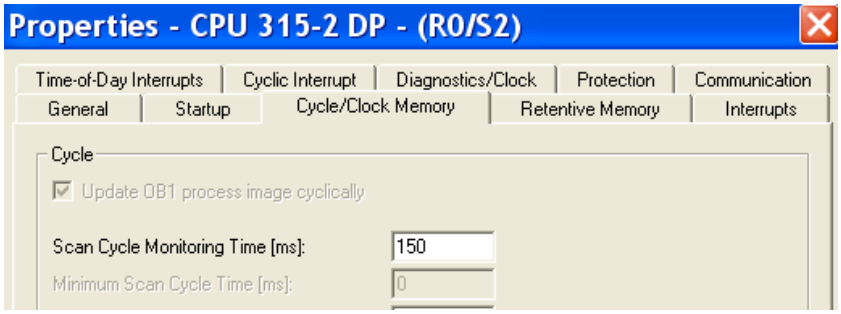
- **Scan Cycle Monitoring Time**

بطور پیش فرض زمان سیکل اسکن داده شده است. در حالت معمول نیازی به تغییر این عدد نیست مگر آنکه بدلیلی در حین پردازش سیکل اسکن از 150ms بیشتر گردد در اینصورت PLC به مد Stop میرود و برای جلوگیری از آن باید این زمان را بیشتر کرد. از علتهایی که میتواند منجر به افزایش زیاد زمان سیکل اسکن شود فراخوانی برنامه های وقفه بطور همزمان است که بعداً تشریح خواهند شد.

- **Minimum Scan Cycle Time**

این گزینه برای CPU های سری S7-400 و برخی CPU های 300 مانند 318 قابل انتخاب است بطور پیش فرض این زمان صفر است میتوان به آن مقدار داد که در اینصورت اگر در حین پردازش سیکل اسکن زودتر از زمان مینیمم تعیین شده تمام شود باندازه زمان باقی مانده صبر کرده و سیکل جدید را شروع میکند. در حالت معمول نیازی به تغییر این پارامتر نمیباشد.

شکل بالای صفحه بعد این دو گزینه را نشان میدهد.



Retentive Memory :

در این بخش میتوان حافظه NVRAM را سازماندهی کرد. تایمرها ، کانترها ، فلگها و سایر دیتاهایی که لازم است با قطع احتمالی تغذیه باقی پاک نشوند در این بخش مطابق شکل زیر تعریف میکنیم.

کاربرانی که با Step5 آشنایی دارند میدانند که تعریف دیتاهای Retentive در S5 انعطاف پذیری کمتری نسبت به S7 دارد. بعنوان مثال برای CPU های 115U میتوان یکی از سه حالت زیر را برای تایمرهای Retentive تعریف کرد:

همه تایمرها Retentive باشند

هیچکدام از تایمرها Retentive باشند.

نیمی از تایمرها Retentive باشند. (نیمه اول)

در حالیکه در S7 هر بازه دلخواهی را میتوان برای این تایمرها بعنوان Retentive تعریف کرد.

Startup:

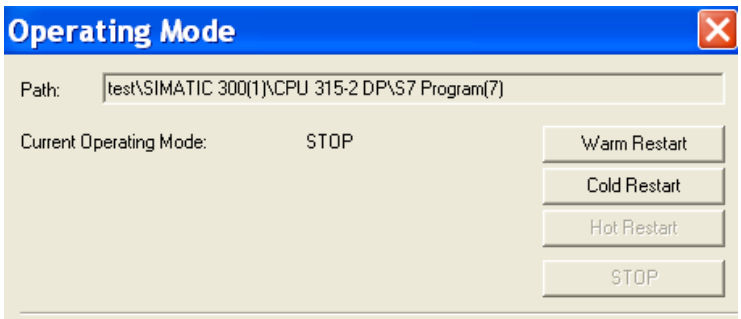
اگر CPU در مد RUN باشد و تغذیه قطع شود و مجدداً وصل گردد ، راه اندازی اتوماتیک مجدد بستگی به تنظیمی دارد که در این بخش مانند شکل زیر انجام میشود.

باید توجه داشت که این راه اندازی :

- در S7-400 بصورت Cold , Warm , Hot امکان پذیر است.
- در S7-300 با CPU 318 بصورت Cold , Warm امکان پذیر است.
- در S7-300 با سایر CPU ها فقط بصورت Warm امکان پذیر است.

تذکر :

توقف PLC و راه اندازی مجدد آن توسط نرم افزار Step7 نیز امکان پذیر است. برای اینکار از منوی PLC > Operating Mode در برنامه های مختلف از جمله Hwconfig و LAD/STL/FBD میتوان استفاده کرد. مانند شکل زیر



واضح است اگر راه اندازی بصورت Warm باشد دیتاهایی که در بخش قبلی یعنی Retentive Memory مشخص شدند باقی خواهند ماند.

نکته قابل ذکر دیگر آنست که برای هریک از راه اندازی های سه گانه فوق یک بلاک برنامه نویسی طراحی شده که کاربر در صورت لزوم میتواند در آنها دستورات مورد نظر رابنویسد. بعنوان مثال میتواند در آنها دستور چند ثانیه تاخیر را وارد کند تا پس از وصل تغذیه سیکل اسکن بلافاصله شروع نشود بلکه با کمی تاخیر و حصول اطمینان از رسیدن ولتاژ به حد نامی (بویژه در سطح Field) شروع گردد .

این بلاکها , OB102, OB101, OB100 نام دارند که توضیحات بیشتر راجع به آنها در جلد دوم کتاب خواهد آمد.

Protection

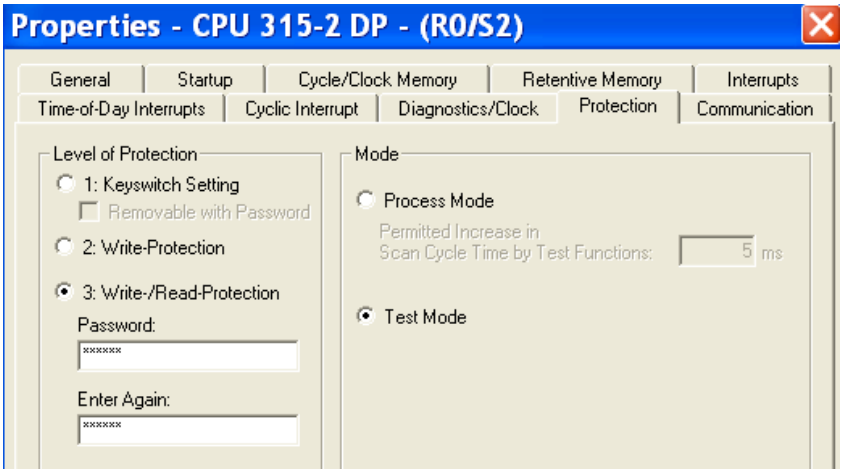
در این بخش که صرفاً برای CPU های سری S7-400 وجود دارد سطوح حفاظتی مختلف برای جلوگیری از دسترسی افراد غیر مجاز به برنامه CPU تعبیه شده است این حفاظت دارای ۳ سطح زیر میباشد:

- **سطح ۱** کاربر مجاز است به برنامه دسترسی داشته باشد (Read و Write) این سطح بعنوان پیش فرض است و در واقع حفاظتی محسوب نمیشود.
- **سطح ۲** کاربر صرفاً اجازه خواندن (Read) را دارد بنابر این Uploadامکان پذیر است ولی نمیتواند Download انجام دهد.
- **سطح ۳** کاربر اجازه Read , Write هیچکدام را ندارد . این سطح در مواقعی که برنامه بعنوان دانش فنی (Know-How) مطرح است مورد استفاده قرار میگیرد.

پس ازتعیین سطح حفاظتی لازم است تغییرات به CPU انتقال یابد . با فعال شدن Password وقتی در Simatic Manager مد Online را انتخاب کنیم قبل از باز شدن کلمه رمز را چک میکند.

از منوی **PLC>Access Right>Setup** در Simatic Manager میتوان فقط یکبار کلمه رمز را وارد کرد و از آن به بعد تا زمانی که Simatic Manager باز است دیگر کلمه رمز را نمی پرسد. از همین منو با **PLC>Access Right>Cancel** میتوان کلمه رمز را غیر فعال نمود.

لازم به ذکر است در صورت ری ست کردن PLC با سوئیچ MRES سطح حفاظتی که قبلاً تعریف شده برداشته شده و به حالت پیش فرض برمیگردد.

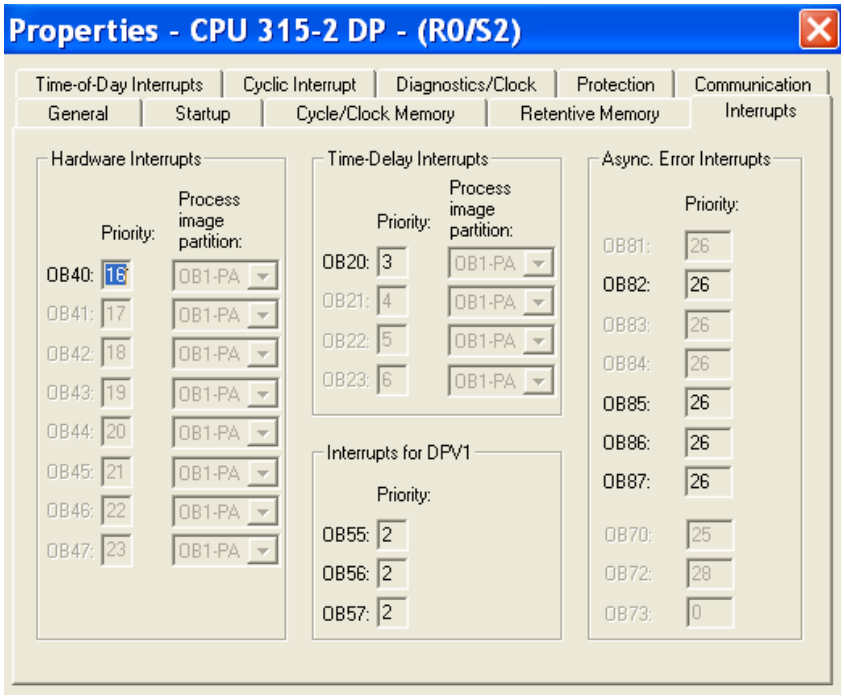


پارامتر های مربوط به وقفه

برای تنظیم پارامتر های وقفه سه بخش در پارامترهای CPU لحاظ شده است. بحث وقفه ها مفصلاً در جلد دوم کتاب ذکر خواهد شد در اینجا فقط اشاره میکنیم که:

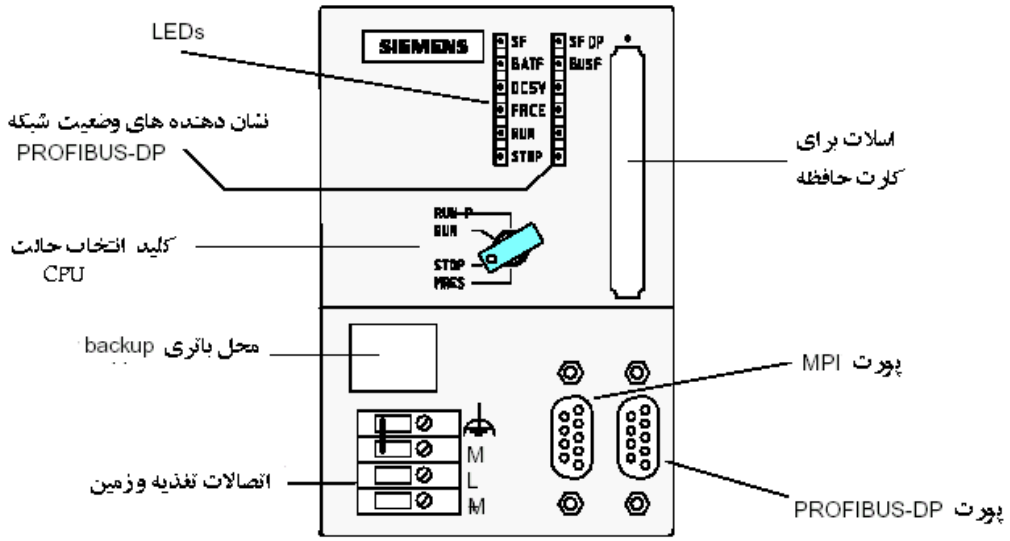
- هر نوع وقفه دارای بلاکهای برنامه نویسی (OB) خاص است.
- بسته به نوع CPU ممکن است برخی از این OB ها موجود نباشند.
- اولویت وقفه ها بالاتر از اولویت برنامه نرمال CPU (یعنی OB1) است .
- با فعال شدن وقفه برنامه عادی قطع شده و برنامه وقفه اجرا میشود.
- بخش **Interrupts** : برای وقفه های ناشی از خطاها و اشکالات بکار میرود.
- بخش **Time-of-Day Interrupts** برای وقفه هایی که باید در تاریخ و زمان مشخصی اتفاق بیفتند استفاده میشود.
- بخش **Cyclic Interrupts** برای وقفه های سیکلی بکار میرود و مهمترین کاربرد آن در لوپ های کنترلی است.

شکل زیر بخش Interrupt را از مجموعه پارامتر های CPU315-2DP نشان میدهد. این CPU فقط بلاکهای OB که در شکل فعال هستند را برای این نوع وقفه ساپورت میکند.



کلید و نشان دهنده های روی CPU

در انتهای بحث پارامترهای CPU و قبل از اینکه بسراغ سایر مدولها برویم بد نیست نگاهی به شکل روی CPU بیندازیم . شکل زیر CPU 318 را ازنمای روبرو نشان میدهد.

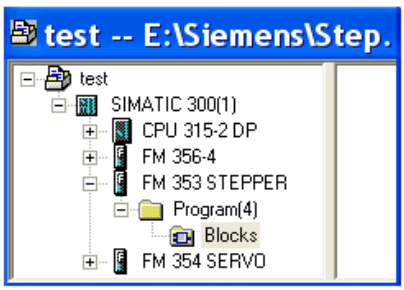


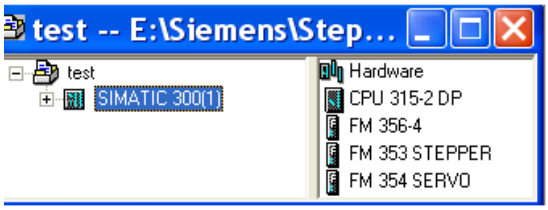
نمایش دهنده های روی CPU مطابق با توضیحات جدول زیر نمایشگر وضعیت PLC هستند.

LED	رنگ	شرح
SF	قرمز	اشکال سخت افزاری یا نرم افزاری
BATF	قرمز	اشکال در باتری
DC5V	سبز	تغذیه 5VDC مربوط به CPU و باس برقرار است.
FRCE	زرد	حالت Force فعال است (توضیح در مباحث آتی)
RUN	سبز	حالت اجرا
STOP	زرد	حالت توقف
SF DP	قرمز	اشکال سخت افزاری یا نرم افزاری روی شبکه DP
BUSF	قرمز	اشکال در باس شبکه DP

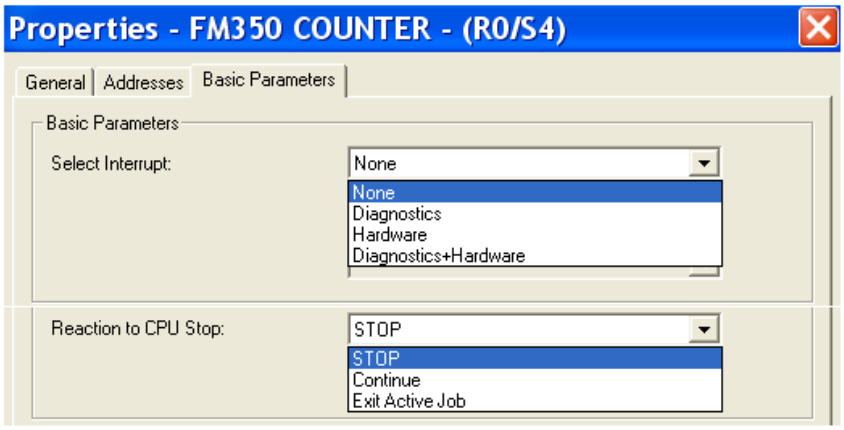
مدولهای FM

همـانطور کـه قـبلاً اشاره شد Function Modules مدولهایی هستند که فانکشن خاصی را مستقل از CPU اجرا میکنند و باصطلاح باری را از دوش CPU برمـیدارند. ورودی هـا مستقیماً به این مدولها داه میشوند و خروجی ها نیز مستقیماً از آنها ارسال میشوند در عین حال FM هـا میتوانـند بـا CPU تـبادل دیـتا داشـته باشـند.در برخـی کاربـردها مانند شمارش سریع که امکان آن توسط CPU وجود ندارد چاره ای جز اسـتفاده ار مدولهـای FM نیسـت. بـرای تنظیـمات مـربوط بـه فانکشن داخلی FM علاوه بر Step7 پکیج پیکر بندی جداگانه ای نیز لازم اسـت.در این پکیج ها علاوه بر ابزار ÷یکر بندی فانکشن بلاکهای خاص FM که توسط آنها پارامترها به FM اختصاص داده میشوند عرضه میشـود .اگـر پـس از وارد کردن FM در Hwconfig و ذخیره سازی به برنامه Simatic Manager باز گردیم میبینیم که آیکون FM مانند شکلهای زیر در پنجره ظاهر شد ودارای پوشه بلاک جداگانه ای است. بلاکهای خاص FM در این پوشه قرار میگیرند.





با وارد کردن FM مورد نظر در اسلات ٤ تا ١١ از رک 300 وسپس کلیک روی آن پنجره ای که باز میشود که دو بخش آن یعنی بخش Generel که توضیحات کلی راجع به مدول را در بر دارد و بخش Address در همه مشترک هستند .آدرسهایی که در بخش Address آمده بشرحی که برای SM ها ذکر شد قابل تغییر هستند. بخش دیگری که در بسیاری از FM ها وجود دارد به Parameters Basic موسوم است دراین بخش از جمله میتوان نوع وقفه که توسط FM به CPU اعمال میشود را تنظیم کرد . وقفه ازنوع Diagnostic در صورت وجود اشکال مانند قطعی و اتصال کوتاه عمل میکند. وقفه از نوع Hardware Interrupt وقتی مقدار Actual از مقدار رفرنس که در پارامترهای مدول تنظیم شده بیشتر شود عمل مینماید.واکنش به توقف CPU نیز در برخی قاب تنظیم است. پارامترهای دیگری نیز بطور خاص برای برخی FM ها وجود دارد.



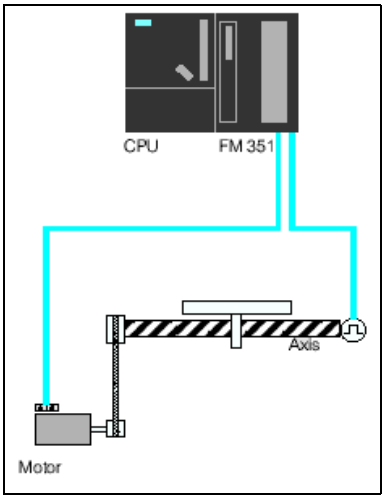
در S7-300 فانکشن مدولهای مختلفی وجود دارند که به برخی از آنها اشاره میگردد:

- **FM350-1 (Counter Module)**

این مدول کانتر یک کاناله ای است که برای شمارش های ساده بکار میرود. انکودرهای افزایشی (Incremental) را میتوان مستقیماً به این مدولها متصل کرداین مدول قادر است پالسهایی را از انکودر با فرکانس ماکزیمم 500KHZ دریافت کند. FM350-1 دارای مدهای کاری مختلف است مانند مد شمارش مداوم ، مد شمارش پریودیک و مد شمارش یک نوبته . نحوه پیکر بندی این مدول در صفحه بعد آمده است.

- **FM350-2 (Counter Module)**

این مدول کانتر ۸ کاناله ای است که برای شمارش های یونیورسال بکار میرود. میتواند به انکودرهای افزایشی (Incremental) متصل گرددو پالسهایی با فرکانس ماکزیمم 10 KHZ از آنها بگیرد . همچنین میتواند به برخی سنسورهای خاص با فرکانس سیگنال حداکثر 20KHZ وصل گردد.



- **FM351 (Position Module)**

این مدول برای کنترل موقعیت تراورس های سریع و درایو های لرزشی بکار میرود. دارای ٤ خروجی دیجیتال برای کنترل موتور است موتور میتواند توسط کنتاکتور یا درایو تغذیه شود .

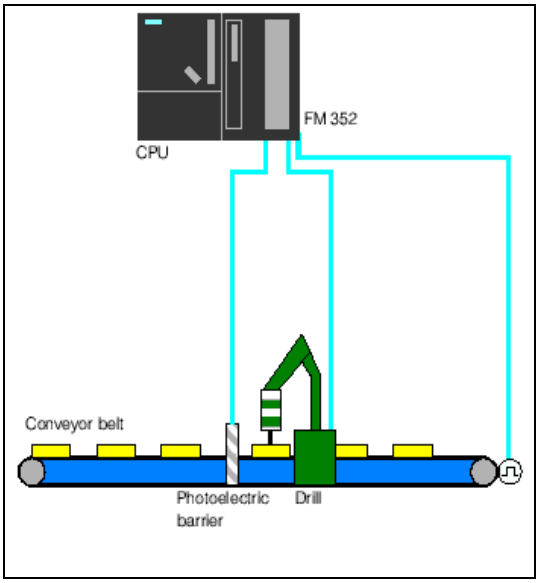
این مدول موقعیت را در دو محور کنترل میکند. شکل روبرو یک نمونه کاربرد آنرا نشان میدهد.

- **FM353 (Position Module)**

این مدول برای کنترل موقعیت موتورهای پله ای بکار میرود.

- **FM354 (Position Module)**

این مدول برای کنترل موقعیت سروموتورها بکار میرود.



- **FM352 (Electronic Cam Controller)**

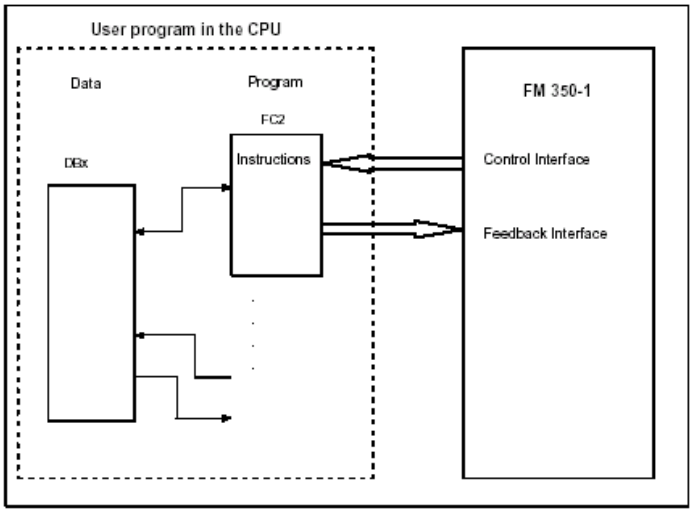
این مدول برای Cam Control بکار میرود . سرعتش خیلی بالاست. موقعیت را از طریق سنسورها که به ورودی آن متصل هستند دریافت میکند سپس فرمانهای لازم را برای کنترل ماشین میفرستد. از کاربردهای آن میتوان به نوار نقاله ای که قطعات روی آن حرکت کرده و در جلوی ماشین دریل یا سنگ قرار میگیرند مانند شکل روبرو اشاره کرد.

- **FM355 (Closed Loop Controller)**

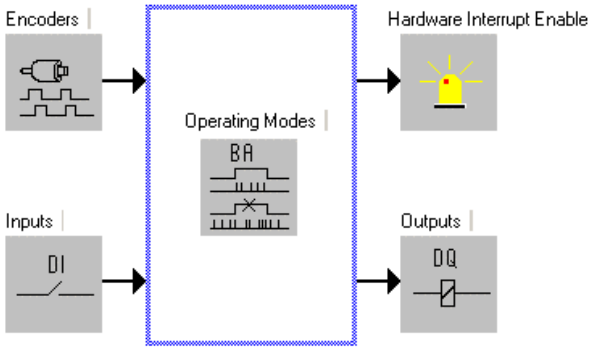
این مدول یک لوپ کنترلر ٤ کاناله است که میتواند برای کنترل فشار و دما بکار رود. اگر بصورت Continues Control تنظیم شود از ٤ خروجی آنالوگ آن استفاده میشود و اگر بصورت Step Control تنظیم شود ۸ خروجی دیجیتال آن بکار برده میشود.

پیکر بندی مدول FM350-1

همانطور که اشاره شد این مدول یک شمارنده سریع است که میتواند به انکودر وصل شده و عمل شمارش را مستقل از CPU انجام دهد.علاوه بر ورودی انکودر این مدول دارای ورودی دیجیتال دیگری است که با صفر و یک شدن آن میتوان عمل شمارش را کنترل نمود. این کنترل از طریق فانکشن های خاصی که در برنامه CPU صدا زده میشوند نیز امکان پذیر است شکل زیر نحوه ارتباط CPU با این مدول را نشان میدهد.

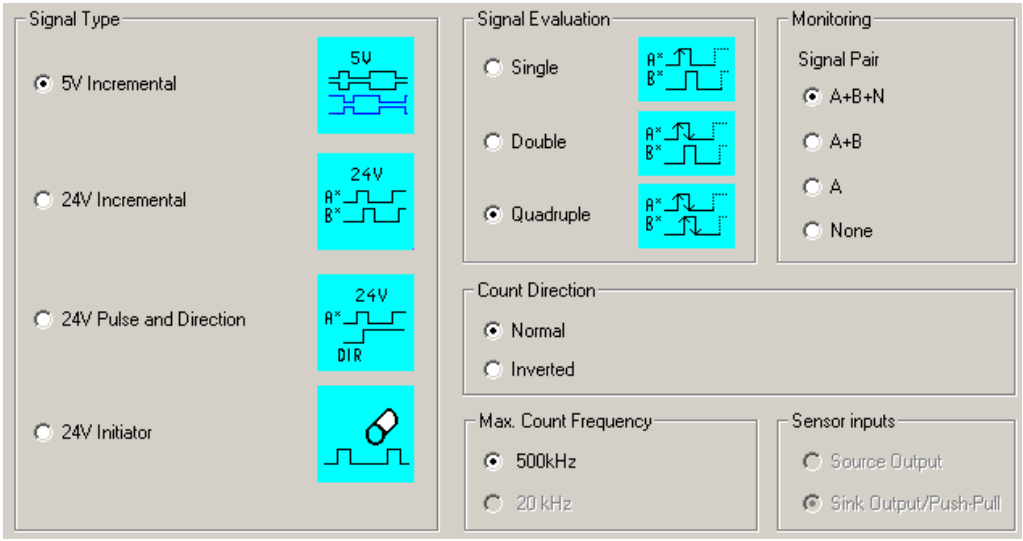


برای تنظیم پارامترهای FM350-1 نرم افزار استاندارد Step7 کافی نیست و لازم است پکیج پیکر بندی مربوطه نیز روی آن نصب گردد. این پکیج همراه با کارت FM توسط فروشنده عرضه میگردد در عین حال بصورت آزاد از سایت زیمنس قابل Download است. پس از نصب پکیج Step7 را اجرا کرده و در HWconfig مدول FM350-1 را در اسلات ۴ تا ۱۱ قرار میدهیم سپس با دوبار کلیک روی آن مشاهده میکنیم که برنامه جدیدی اجرا شده و شکلی مانند زیر نمایش داده میشود.

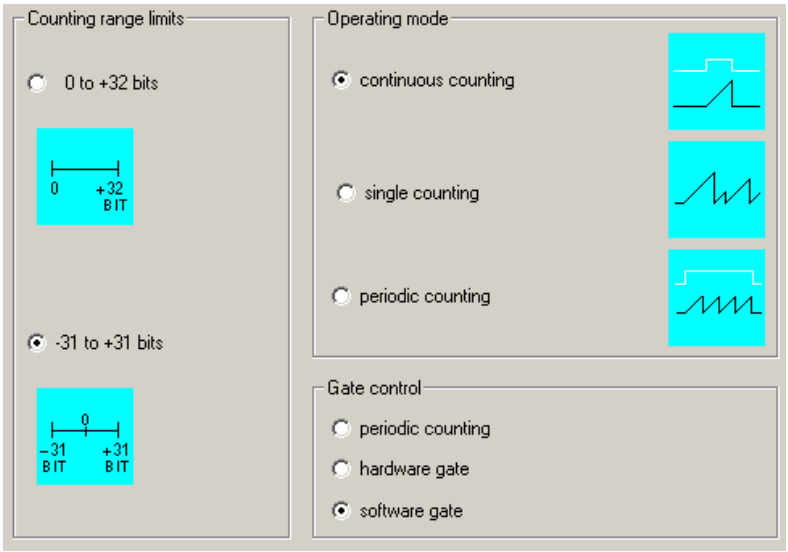


با کلیک کردن روی هر یک از باکسهای فوق پنجره جدیدی باز میشود که میتوان پارامترهای مربوطه را توسط آن مطابق توضیحات صفحه بعد تنظیم نمود تنظیم نمود.

با کلیک کردن روی باکس Encoders پنجره ای مانند شکل زیر باز میشود. بسته به نوع انکودری که در سمت چپ انتخاب میشود گزینه های سمت راست فعال یا غیر فعال خواهند شد.

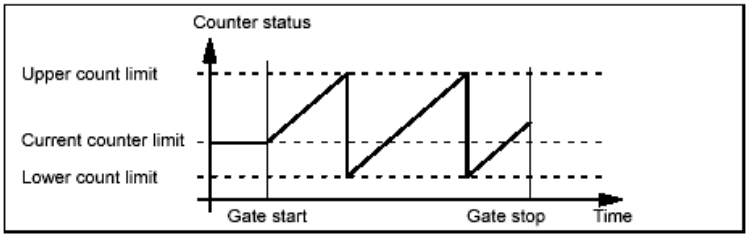


با کلیک کردن روی باکس Operating Mode نیز شکل دیگری مانند زیر باز میشود



در مد Continuous اگر کانتر افزایشی به حد ماکزیمم خود برسد و باز پالس شمارش دریافت شود کانتر به حد مینیمم خود پرش کرده و از آنجا شروع به افزایش میکند. در کانتر کاهشی عکس عمل فوق اتفاق می افتد یعنی وقتی به حد مینیمم رسید با پالس جدید به حد ماکزیمم پرش و از آنجا شروع به کاهش مینماید.در مد Single Counting اگر کانتر به حد ماکزیمم یا خود برسد و باز پالس شمارش دریافت شود کانترافزایشی به نقطه مینیمم و کانتر کاهشی به نقطه ماکزیمم زفته و آنجا باقی میماند. مد Peridic Counting شبیه Continuous است با این تفاوت که نقطه شروع از مقدار Load Value است.

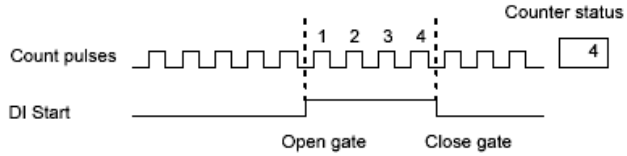
همانطور که در شکل پایین صفحه قبل ملاحظه میشود کنترل شمارش کانتر(Gate Control) میتواند بصورت سخت افزاری یا نرم افزاری باشد.در هر دو حالت شمارش فقط وقتی اتفاق می افتد که گیت باز شده باشد بدیهی است پس از بسته شدن گیت وجود پالسهای انکودر تاثیری روی شمارنده ندارد.



در کنترل سخت افزاری (Hardware Gate Control) ورودی فیزیکی مشخص شده در بخشInputs پنجره پیکر بندی (شکل دو صفحه قبل) عمل کنترل را بعهده دارد.براساس این ورودی یکی از دو حالت کنترلی زیر را میتوان انتخاب نمود:

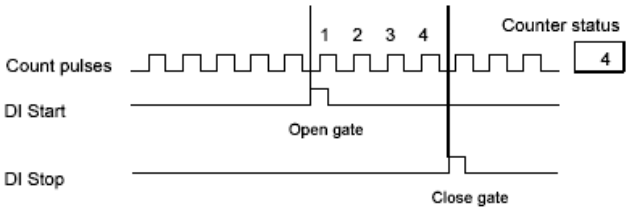
۱- Level Control

در این حالت کانتر بمحض یک شدن ورودی DI Start بکار می افتد و با صفر شدن ورودی غیر فعال میشود شکل زیر:

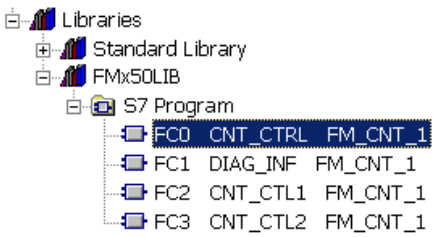


۲- Edge Control

در این حالت کانتر بمحض یک شدن ورودیDI Start بکار می افتد و با یک شدن ورودی DI Stop غیر فعال میشود شکل زیر:



کنترل گیت میتواند بصورت نرم افزاری (Software Gate Control) باشد یعنی شروع یا توقف شمارش توسط برنامه کنترل شود. در همینجا لازم است یادآوری شود که با نصب پکیج پیکر بندی FM فانکشن های نرم افزاری جدیدی به برنامه Step7 اضافه میشوند که میتوان آنها را در بخش کاتالوگ و در زیر مجموعه Library مشاهده نمود. فانکشن FC0 یا CNT_CTRL برای کنترل گیت شمارنده بکار میرود .

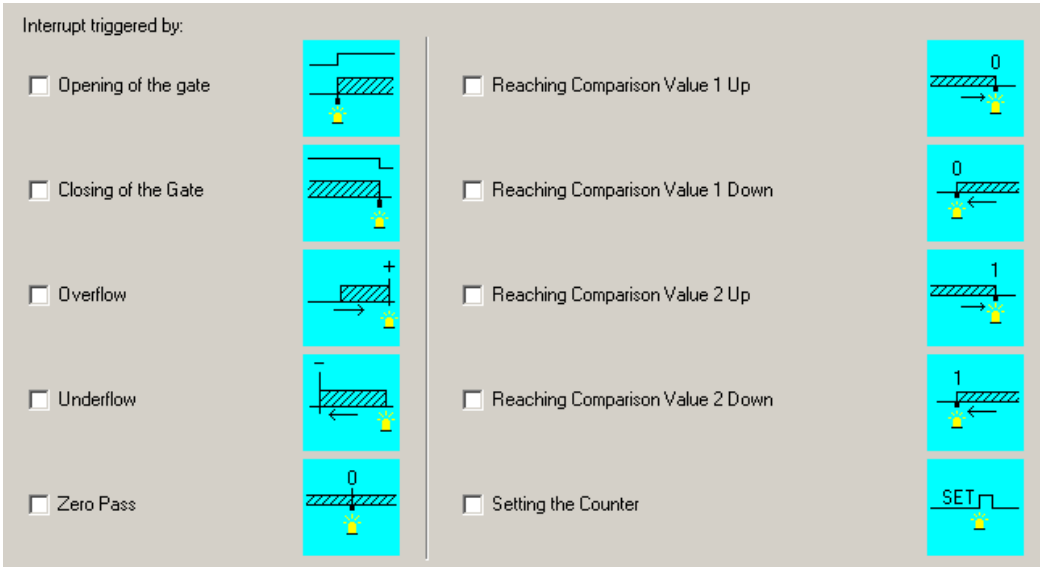


در پیکر بندی FM350-1 باتوجه به شکل صفحه ۷۱ دو بخش دیگر نیز مشاهده میشود که عبارتند از :

Hardware Interrupt Enables

با کلیک روی این باکس شکل زیر ظاهر میشود که توسط آن میتوان حالتهای مختلفی که کانتر بر اساس آنها میتواند وقفه ایجاد کند را مشاهده و در صورت نیاز فعال نمود. Overflow ، Underflow ، عبور از صفر و ست شدن کانتر از جمله این حالتها هستند.

لازم به ذکر است که تنظیمات این قسمت به تنهایی برای اعمال وقفه کافی نیستند و علاوه برآنها باید در HWConfig با کلیک راست روی مدول FM در بخش Properies و قسمت Basic Parameters وقفه را فعال نمود.



Outputs

با کلیک روی باکس Output شکلی مانند زیر ظاهر میشود و میتوان انتخاب کرد که از دو بایت خروجی های FM کدامیک و در چه شرایطی یک شود.

پیکر بندی سایر FM ها

برای پیکر بندی سایر FM ها باید مشابه FM350-1 عمل کرد . برای هر یک از آنها پکیج خاص باید نصب شود و پس از آن در باکسهای مختلف پارامترهای مربوطه تنظیم شود. شرح پارامترهای همه FM ها از حوصله کتاب حاضر خارج است و مبحث جداگانه ای را طلب میکند.

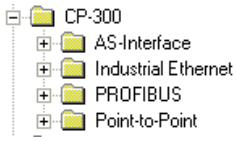
تذکر

قبل از نصب مدولهای FM روی ریل لازم است Submodule کنار آنها شبیه آنچه برای کارتهای AI ذکر شد در موقعیت مناسب تنظیم شود. بعنوان مثال برای FM350-1 وضعیت A یا D مطابق جدول زیر انتخاب گردد.

A	5 V Differential Signals
D	24V Signals

مدولهای CP

این مدولها همانطور که از نامشان (Communication Processor) پیداست برای ارتباط با شبکه بکار میروند در پنجره کاتالوگ انواع کارتهای CP در چند دسته مانند شکل زیر قرار گرفته اند.



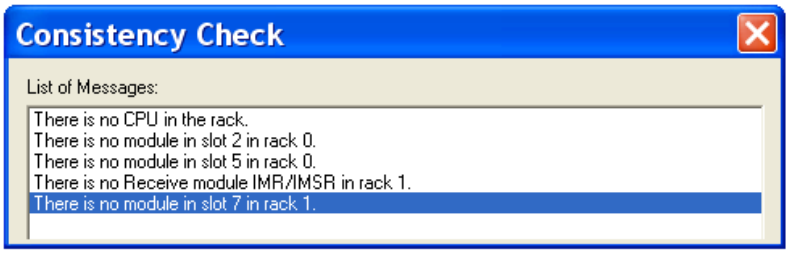
از آنجا که بحث شبکه و پیکر بندی آن بصورت جداگانه مطرح خواهد شد در اینجا از بحث در مورد این کارتها صرفنظر میکنیم.

منبع تغذیه PS

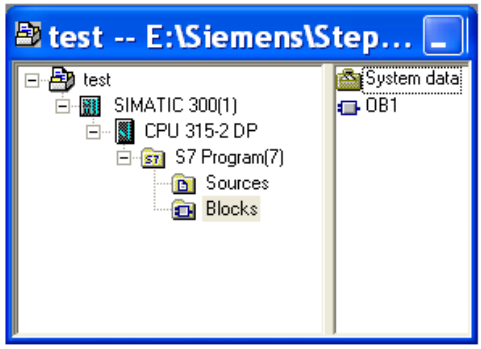
منبع تغذیه های 300 همگی 120/230 ولتی هستند که بسته به جریان آنها به سه دسته 2A و 5A و 10A تقسیم میشوند.این کارتهاهیچ تنظیم خاصی که لازم باشد توسط Hwconfig انجام شود نیاز ندارند.

پایان پیکر بندی و چک سازگاری اجزا

تا این مرحله با وارد کردن اجزای مختلف خانواده 300 به رک و تنظیم پارامتر های آنها آشنا شدیم. آخرین قدم در اینجا چک سازگاری اجزاست. این کار با منوی File > Consistency Check انجام میگردد.اگر اشکالی وجود داشته باشد(مثلاً اسلات خالی در بین مدولهایا عدم وجود CPU در رک اصلی) پنجره ای مانند شکل زیر نمایش داده میشود.



پس از رفع اشکال وانجام چک مجدد پیغام No Error ظاهر میشود . در این مرحله باید تغییرات را ذخیره کنیم منوی File > Save فقط ذخیره سازی را انجام میدهد. و منوی File> Save and Compile علاوه بر ذخیره کردن عمل کامپایل وچک سازگاری را نیز انجام میدهد. کامپایل کردن منجر به ایجاد آیکونی بنام System Data مانند شکل زیر در پوشه بلاک Simatic Manager میگردد که اطلاعات سخت افزار پیکر بندی شده را در بر دارد.



آخرین مرحله پس از ذخیره سازی دانلود کردن به PLC است

۳-٤ پیکربندی S7-400

با توضیحاتی که برای پیکربندی S7-300ارائه شد کاربر بسهولت میتواند پیکربندی S7-400را که اکثر جهات به آن شباهت دارد انجام دهد. قدمهایی که باید برداشت شبیه قبل است با این وجود یکبار دیگرآنها را مرورمیکنیم.

واردکردن Station400در Simatic Manager

- بازکردنStation توسط برنامه Hwconfig
- وارد کردن رک
- گذاشتن اجزای مورد نیاز در رک از کاتالوگ400
- تنظیم پارامترهای مدولها
- چک سازگاری وذخیره سازی

تفاوتهایی که پیکربندی S7-300 با S7-400دارد عبارتند از:

۱. رک های 400چندین نوع دارد وفانکشن آنها با 300 متفاوت است.
۲. تعداد رک اضافی که میتواند به رک اصلی متصل شود بیش از این تعداد در 300است.
۳. فضای خالی بین مدولها در رک 400اهمیت ندارد.
٤. کارت DI/DO و AI/AO در S7-400 وجود ندارد.
٥. تنوع کارتهای 400 نسبت به 300کمتر است.
٦. تمام CPUهای 400 تغییر آدرس مدولهای ورودی و خروجی را ساپورت میکنند.
۷. قابلیت Multicomputingدر CPUهای 400وجود دارد ومیتوان چند CPUاز این نوع را در یک رک وارد کرد.
۸. علاوه برراه اندازی های Coldو Warm مد Hot Startupنیز برای CPU های 400وجود دارد.
۹. توانایی ها، سرعت و حجم آدرسی که CPU های 400دارند بسیار بیشتر از 300است.

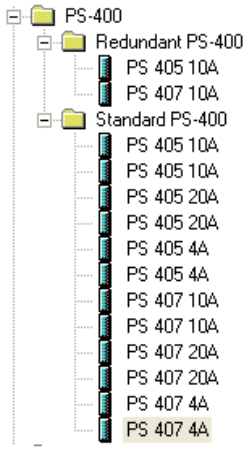
با وجود توضیحات فوق موارد خاصی که نیاز به توضیح بیشتر دارند بشرح زیر ارائه میشود:

منبع تغذیه PS

منابع تغذیه در S7-400را میتوان طبق جدول زیر دسته بندی کرد.

نوع	جریان	ولتاژ
Redundant	10 A	24 V
Standard	4 A 10 A 20 A	120/230 V

میتوان ۲ منبع تغذیه از نوع Redundant را در کنار هم در اسلات های اولیه قرارداد. اگر یکی از این منابع تغذیه قطع شود سیستم از دومی تغذیه میگردد. لازم بذکر است برخی از منابع تغذیه عملاً و در هنگام پیکر بندی ۲ اسلات را اشغال میکند.

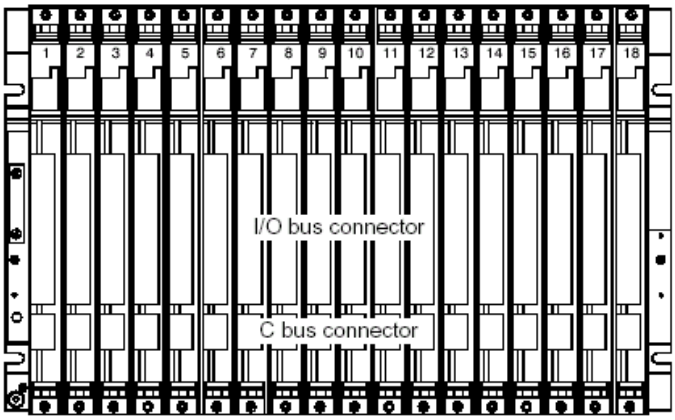


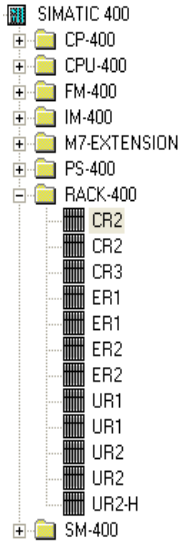
رک های S7-400

رک های S7-400 وظایفی بشرح زیر دارند:

- نگهدارنده مدولهاست
- تغذیه کننده مدولها از طریق Backplane Bus است
- دارای I/O BUS برای ارتباط سیگنالهاست
- دارای Communication Bus برای ارتباط شبکه است.

شکل زیر یک نمونه از رک های 400 راکه دارای ۱۸ اسلات است نشان میدهد.





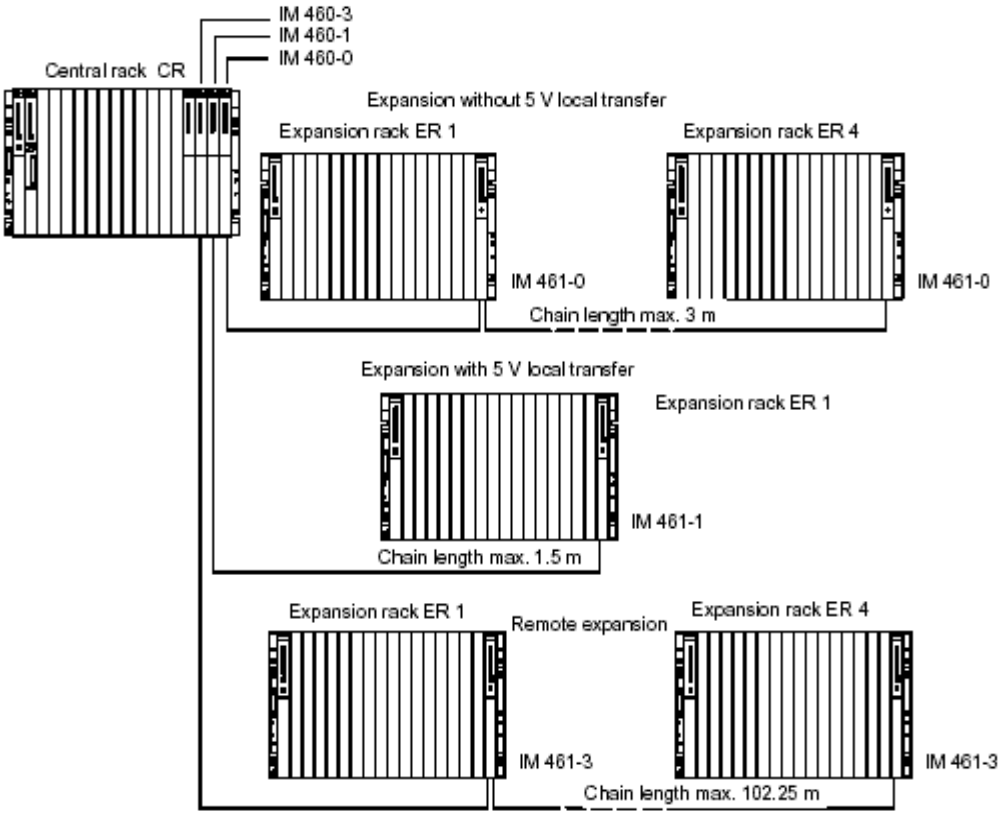
همانطور که در شکل روبرو نمایش داده شده است. رک های S7-400 انواع مختلف دارند.انواع این رک ها در جدول زیر معرفی شده اند.

نام	نوع	کاربرد	تعداد اسلات	ملاحظات
UR1	Universal	بعنوان رک اصلی بعنوان رک اضافی	18	دارای ۲ نوع است که در یکنوع آن میتوان ۲ منبع تغذیه از نوع Redundant قرار داد.
UR2	"	"	9	"
ER1	Expansion	بعنوان رک اضافی	18	"
ER2	"	"	9	"
CR2	Segmented	بعنوان رک اصلی	10 + 8	دارای ۲ بخش است که هر بخش آن میتواند یک رک اصلی مستقل باشد.
CR2-H	Universal	برای PLC های Redundant بکارمیرود	2*9	شبیه دو رک اصلی UR2 است.

رک اضافی در S7-400

قبل از پیکر بندی رک اضافی در S7-400 باید نکات زیر را مد نظر قرار داد:

- در S7-400 ماکزیمم ۲۱ رک اضافی میتوانیم داشته باشیم.
- در رک اصلی میتوان حداکثر ٦ عدد IM وارد کرد.
- IM ها در رک اصلی میتوانند از اسلات ۳ به بعد قرار گیرند.
- IM ها در رک اضافی فقط در آخرین اسلات قرار می گیرند.
- هر IM میتواند به شکل زنجیری (Chain)حداکثر به چهار IMمتصل شود.

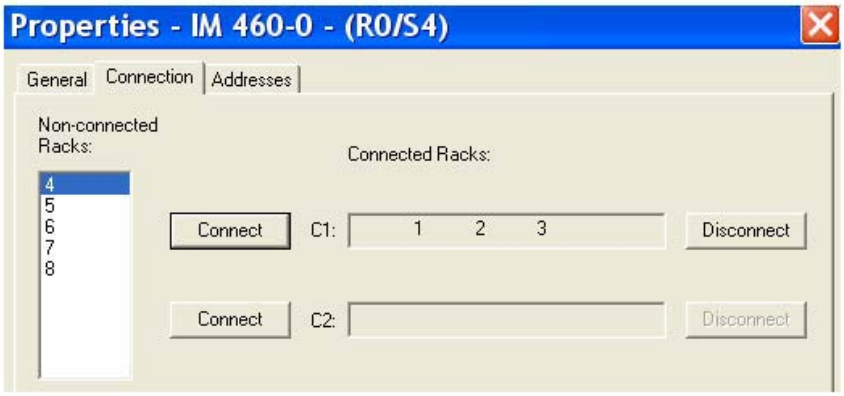


460-3	460-1	460-0	Send IM
461-3	461-1	461-0	Recieve IM
4	1	4	Max. ER /chain
102.25 m	1.5 m	5 m	Max.Distance
No	Yes	No	Power Transfer
6	2	6	Number of IM in Rack0

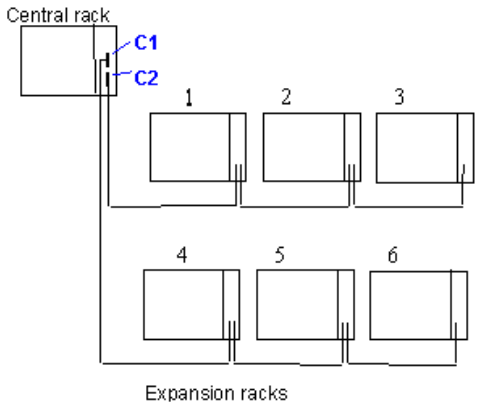
- ✓ IMها انواع مختلف دارند که در جدول روبرو آمده است .وبصورت جفتی (Send/Receive)بکار میروند.میتوان ترکیبی از آنها را استفاده کرد.
- ✓ اگر IMتغذیه رانیز منتقل کند دیگرنیازی به منبع تغذیه در رک اضافی نداریم.
- ✓ بجز IM های فوق سایر موارد که در پنجره کاتالوگ ظاهر میشوند برای ارتباط با رک اضافی نیستند بعنوان مثال IM467 برای ارتباط با شبکه پروفی باس بکار میرود.

پیکر بندی رک اضافی

روش وارد کردن رک اضافی شبیه آنچه برای S7-300ذکرشد میباشد. در s7-300تعداد رک اضافی محدود بود و با قرار دادن IM ارتباط توسط برنامه برقرار میگردید. ولی در S7-400 بدلیل زیاد بودن تعداد IMها واینکه مشخص نیست به چه صورت باید گروه بندی و اتصال داده شوند اینکار بطور دستی توسط کاربر باید انجام گیرد. برای این منظور با ماوس روی هرکدام از IM های رک اصلی کلیک کرده پنجره ای مانند شکل زیر باز میشود.



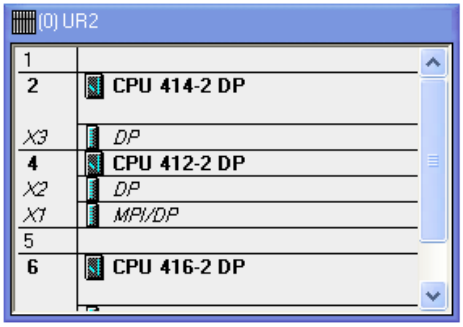
شماره رک اضافی موردنظر را انتخاب کرده و با کلید Connect آنرا به رک اصلی متصل میکنیم. از آنجا IM دارای ۲ پورت برای اتصال است که C1 و C2 نامیده میشود میتوان به هر یک از آنها ٤ رک اضافی را بصورت زنجیری مانند شکل زیر وصل نمود.



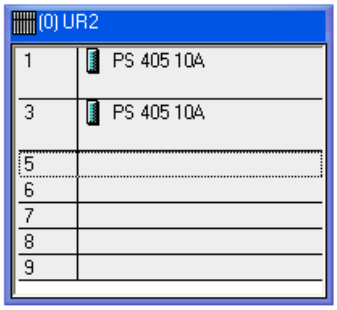
ترتیب مدولها در رک 400

برای چیدن مدولها در رک 400 باید توجه داشت که :

- CPUفقط در رک اصلی قرار می گیرد.
- IMاز نوع Receive فقط در رک اضافی قرار میگیرد.
- میتوان ۲ منبع تغذیه از نوع Redundant را در رکهایی که این آرایش را ساپورت میکنند از اسلات اول و پشت سر هم قرار داد.
- اگر CPUقابلیت multicomputing (که شرح داده خواهد شد) را داشته باشددر اینصورت میتوان چند CPUرا در اسلاتهای مختلف رک قرارداد.
- برخی منابع تغذیه و CPUها درعمل و نیز در هنگام پیکربندی دو اسلات را اشغال میکنند.



قراردادن چند CPUبا قابلیت Multicomputing در رک



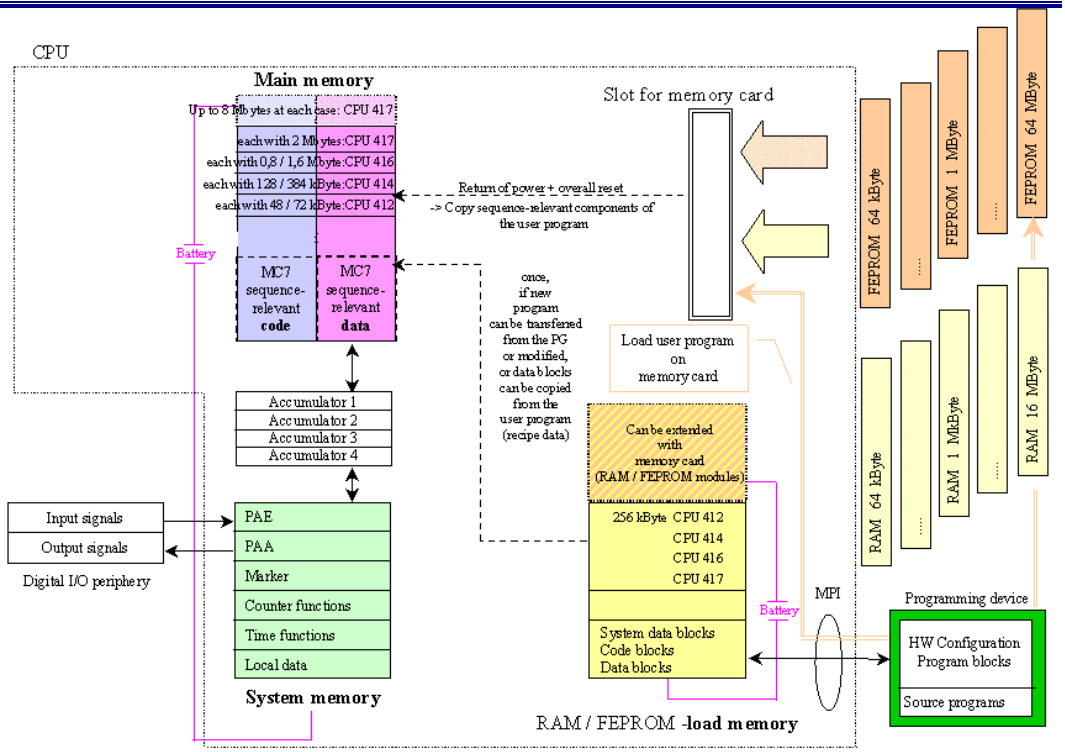
قراردادن دومنبع تغذیه Redundant در رک

با توجه به نکات فوق ترتیب مدولها در رک S7-400 را نمیتوان شبیه رک S7-300 در قالب فرمولی خاص ارائه کرد. همینقدر میتوان گفت که ابتدا PS سپس CPU و پس از آن سایر کارتها قرار میگیرند.

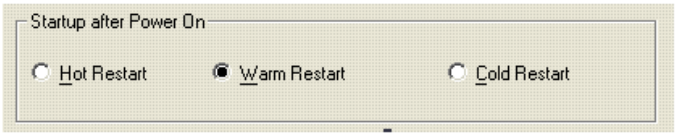
پارامتر های CPU های S7- 400

اکثر پارامترهای CPU در S7-400 شبیه S7-300 میباشد وتنظیم آنها به همان نحو است. نکات خاص واضافی که باید مد نظر داشت عبارتند از:

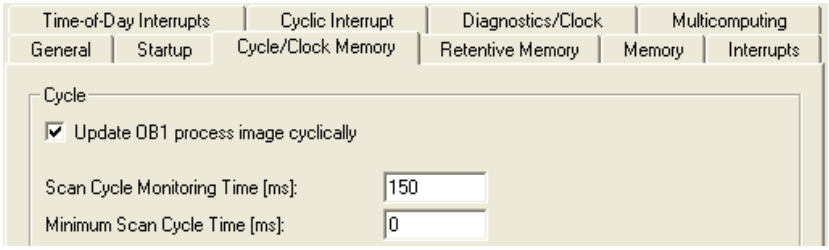
- حافظه CPU های 400 شکلی شبیه بالای صفحه بعد دارد. همانطور که مشاهده میشود علاوه بر وجود Load Memory بصورت داخلی که میتواند از نوع RAM یا FEPROM باشد میتوان کارت حافظه نیز از همین دو نوع را در اسلات مربوطه وارد کرد.ولی بدون وجود کارت حافظه نیز CPU راه اندازی میشود.
- برخی CPU های 400دارای ٤ آکومولاتور هستند ولی در نوع 300 فقط دوآکومولاتور وجود دارد.



- علاوه بر مدهای راه اندازی Cold, Warm مد Hot نیز وجود دارد که در بخش Startupمانند شکل زیر قابل تنظیم است.



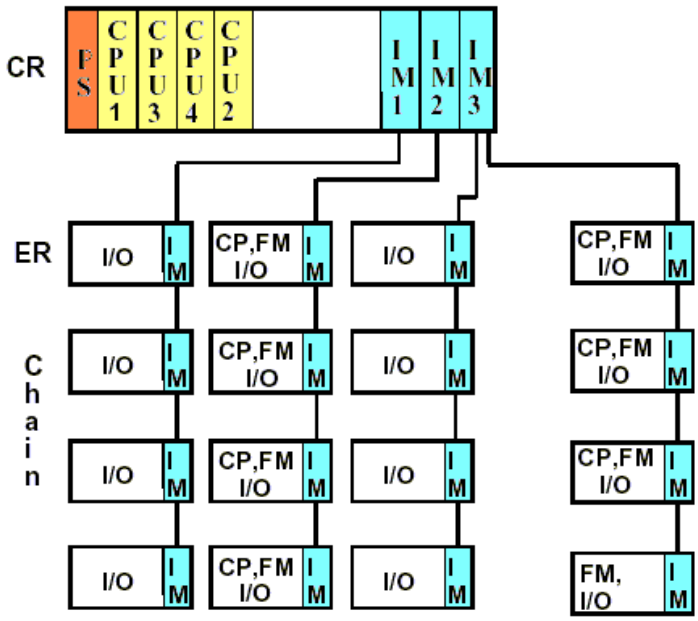
- برای سیکل اسکن میتوان حداقل زمان نیز در بخش Cycle/Clock Memoryتعریف کرد.



- بخش Protection برای همه CPU های نوع 400وجوددارد.
- در بخشهای مربوط به وقفه تعداد OBهایی که فعال هستند بیش از 300 است.
- در برخی CPUهای 400 قابلیت Multicomputing بشرح زیر وجود دارد که CPU های 300فاقد آن هستند.

عملکرد سنکرون چند CPU بصورت Multicomputing

عملکرد سنکرون چند CPU که صرفاً برای CPU های سری S7-400 امکان پذیر است بدین معنی است که چند CPU (ماکزیمم ٤ تا) در یک رک قرار گرفته وهر کدام مستقلاً برنامه ای را اجرا می کنند به گونه ای که کارها (Tasks) بصورت موازی انجام میشود. این CPU ها با هم استارت شده و با هم به مد STOP میروند هر CPU فقط به مدولهایی دسترسی دارد که در هنگام پیکر بندی به آن اختصاص داده شده است.



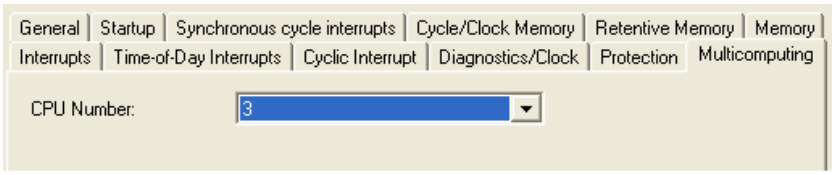
استفاده از سیستم Multicomputing از جمله در موارد زیر کاربرد دارد :

- ✓ وقتی برنامه کاربری بسیار بزرگ است و یک CPU نتواند برای آن بکار رود که در این حالت با آرایش فوق میتوان برنامه را بین چند CPU توزیع کرد.
- ✓ وقتی لازم باشدبخشی از برنامه سریعتر از سایر بخشها اجرا شودکه در این حالت آن بخش از برنامه را میتوان روی CPU سریع جداگانه ای پردازش نمود.

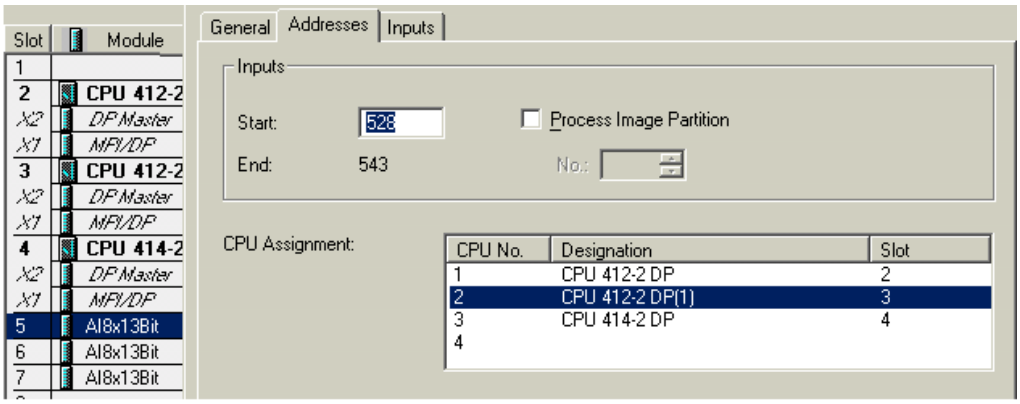
نکاتی که در استفاده از روش Multicomputing قابل توجه است عبارتند از :

۱. CPU قابلیت Multicomputing را داشته باشد. در توضیحات زیر پنجره کاتالوگ Hwconfig میتوان این موضوع را چک کرد.
۲. لزومی ندارد که همه CPU ها از یکنوع باشند مثلاً میتوان CPU412-2DP را با CPU414-2DP بکار برد.
۳. رک انتخاب شده باید از نوع UR باشد. بدیهی است رک تقسیم شده از نوع CR برای این منظور مناسب نیست زیرا در این نوع رک CPU ها بعنوان پردازشگرهای مستقل عمل میکنند.

٤. در این روش هر CPU یک شماره دارد این شماره را میتوان در بخش Multicomputing از پارامترهای آن مشاهده کرد



۵. مدولها پس از انتخاب باید به CPU مربوطه آدرس دهی شوند. بعنوان مثال اگرچند مدول ورودی Analog Input داشته باشیم با ماوس راست Properties را انتخاب کرده و در بخش Address مانند شکل زیر CPU مربوطه را انتخاب میکنیم.



برای اطمینان از اینکه مدولها بطرز صحیح اختصاص یافته اند در برنامه Hwconfig از منوی **View>Filter** استفاده کرده و شماره CPU را انتخاب میکنیم.پس از آن مشاهده میکنیم که رنگ مدولهایی که به این CPU مربوط نیست بصورت خاکستری در می آید.
نهایتاً پس از ذخیره سازی تغییرات در Hwconfig وبازگشتن به Simatic Manager میبینیم که در زیر Station 400 آیکون چندین CPU ظاهر شده است که هر کدام دارای پوشه بلاک جداگانه ای هستند. لازم بذکر است برنامه نویسی این CPUها تفاوت چندانی با برنامه نویسی CPUهای منفرد ندارد.

تذکر۱: قابلیت Multicomputing نباید با قرار دادن دو CPU در رک نوع CR اشتباه شود. برخی تفاوتها عبارتند از :

- در نوع CR رک دو بخش مجزا دارد که باس i/o و باس C bus آنها جداست ولی در نوع .Multicomp رک از نوع UR است و باسهای فوق در آن مشترک است.
- CPU های رک CR مستقل از هم بوده و با هم خاموش و روشن نمیشوند ولی در نوع Multicomp روشن و خاموش شدن همزمان است
- در نوع CR هر CPU به کارتهای SM همان بخش از رک دسترسی دارد ولی در نوع Multicomp محل کارت مهم نیست و توسط Hwconfig میتوان تعیین کرد که هر CPU به چه کارتهایی دسترسی داشته باشد.

تذکر۲: قابلیت Multicomputing نباید با H-system اشتباه شود زیرا در سیستم H یکی از دو CPU همیشه رزرو است و فقط وقتی دیگری از کار بیفتد وارد مدار میشود . برنامه دو CPU کاملا مشابه بوده و هر کدام از آنها به تمام کارتهای رک دسترسی دارند.

۴- شروع برنامه نویسی

مشتمل بر :

٤-١ سیستمهای عددی مورد استفاده در PLC

٤-٢ فرمت آدرس دهی در S7

٤-٣ فرمت دیتاها در S7

٤-٤ آکومولاتور ها ورجیسترهای CPU S7

٤-٥ بلاک های برنامه نویسی

٤-٦ نحوه ایجاد بلاک در Simatic Manager

٤-٧ آشنایی با محیط زیر برنامه LAD/STL/FBD

٤-٨ نحوه استفاده از بلاک ها

٤-٩ نحوه ایجاد و استفاده از جدول سمبل ها

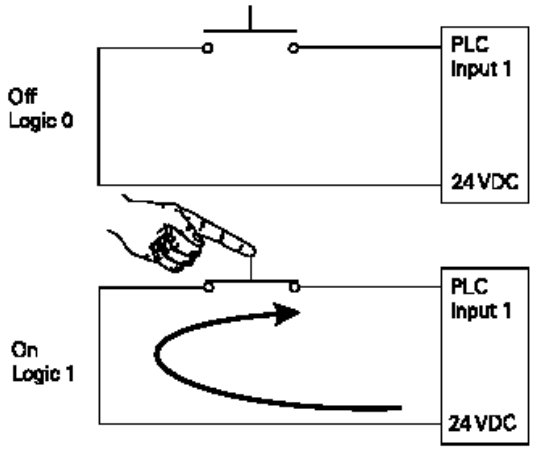
٤-١٠ نحوه استفاده از Reference Data

٤-١١ نحوه استفاده از Rewiring

٤-١٢ مقایسه بلاک ها Compare Blocks

٤-١ سیستمهای عددی مورد استفاده در PLC

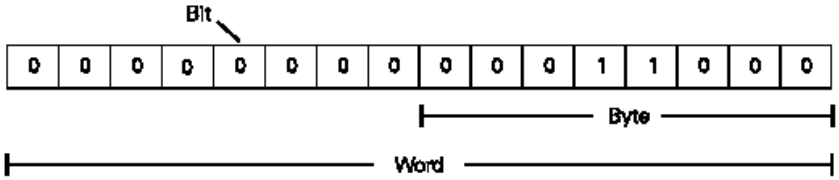
اعـدادی که ما معمولاً با آنها در محاسبات سروکار داریم اعداد مبنای ۱۰ هستند و Decimalخوانده میشوند. بسادگی میتوان اعداد مبنای ۱۰ را بـه هـر مبـنای دلخواهـی تـبدیل کرد. در کامپیوترها که PLC نیز عضوی از خانواده آنهاست اطلاعات در مبنای ۲ یعنی صفر و یک شناخته میشوند. از نظر الکتریکی وصل شدن مدار معادل یک منطقی و قطع شدن مدار معادل صفرمنطقی است.



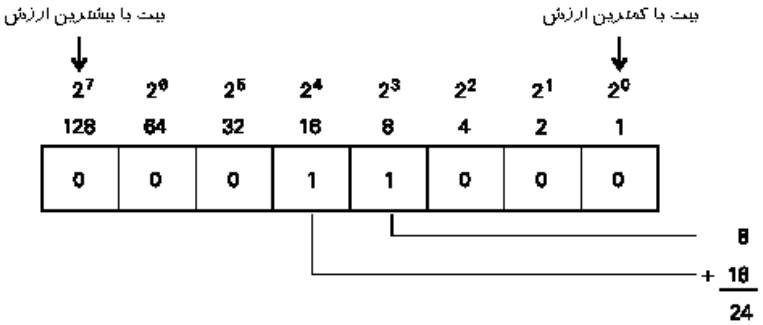
تـبدیل یکعـدد از مبـنای ۱۰ (Decimal) بـه مبـنای ۲ (Binary) سـاده اسـت کافیسـت با تقسیم متوالی آن بر ۲ باقیمانده های بدست آمده و آخرین خارج قسمت را از آخر به اول در کنار هم قرار دهیم.چند مثال در جدول زیر آمده است.

Decimal	Binary
4	100
9	1001
24	11000

اگرصـفر و یک های عدد باینری (مثلاً عدد ٢٤) را مانند شکل زیر در یک جدول بریزیم به هر خانه از این جدول یک Bit گفته میشود . هر ۸ بیت معادل یک Byte و هر دو بایت یعنی ١٦ بیت معادل یک Word است. هر دو Word نیز یک Double Word میباشد.



بیت سمت راست کمترین ارزش و بیت سمت چپ بیشترین ارزش را دارد بر اساس این ارزشها میتوان معادل Decimal عدد باینری را محاسبه کرد.



علاوه بر سیستم باینری دو سیستم عددی دیگری نیز در PLC مورد استفاده قرار میگیرد که عبارتند از Hexadecimal و BCD

Hexadecimal یـا مبـنای ۱٦ ارتـباط مستقیم با مبنای ۲ دارد. از آنجا که بیت های باینری در کنار هم بصورت طولانی ردیف میشوند فهم آنهـا مشکل است. با تبدیل باینری به Hex این بیتها در فضای کمتری جا میگیرند کاربر راحت تر میتواند مقدار باینری را تشخیص دهد. اعداد در مبنای ۱٦ مانند شکل زیر از صفر شروع شده و تا ۱۵ ادامه می یابند.

اعداد مبنای ۱٦

16 digits 0 , 1, 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E, F

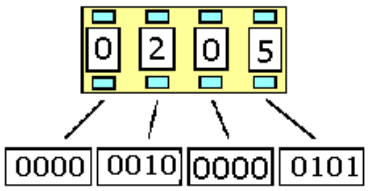
که در آن :

A=10 D=13
B=11 E=14
C=12 F=15

هر دیجیت هگز معادل ٤ بیت باینری است.بعنوان مثال معادل باینری و هگز عدد دسیمال ٤۳ بصورت زیر خواهد بود:

Decimal	**43**	
Binary	**1011**	**0010**
Hexadecimal	**B**	**2**
BCD	**0100**	**0011**

سیستم عددی **BCD** یا Binary Coded Decimal اعداد دسیمال هستند. این سیستم در برخی وسایل ورودی و خروجی PLC مانند شمارنده ها مورد استفاده قرار میگیرد. تفاوت BCD با Decimal در این است که اعداد وزن دهگان یا صدگان و .. ندارند. بعنوان مثال وقتی عدد ۲۰۵ را روی صفحه نمایش یک شمارنده بصورت BCD میبینیم برخلاف مبنای ۱۰ ، عدد صفر ارزش دهگان و عدد ۲ ارزش صدگان نخواهد داشت.در این روش عدد باینری به قسمتهای ٤ تایی شکسته میشود و معادل دهدهی هر ٤ بیت بدون توجه به وزن آن بصورت دسیمال بدست می آید. وقتی اعداد بدست آمده را در کنار هم قرار دهیم معادل BCD را خواهیم داشت مانندمثال بالا و شکل زیر:



عدد BCD	عدد دسیمال
0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**

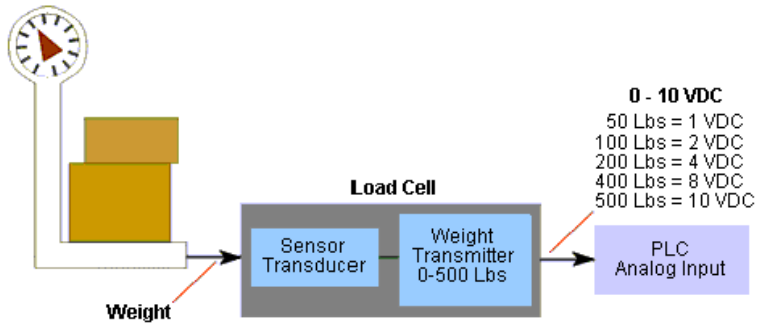
در جدول زیر مثالهایی برای مقایسه ٤ سیستم عددی که شرح داده شد آورده شده است.

Decimal	*Binary*	*BCD*	*Hexadecimal*
0	0	0000	0
1	1	0001	1
2	1	0010	2
3	11	0011	3
4	100	0100	4
5	101	0101	5
6	110	0110	6
7	111	0111	7
8	1000	1000	8
9	1001	1001	9
10	1010	0001 0000	A
11	1011	0001 0001	B
12	1100	0001 0010	C
13	1101	0001 0011	D
14	1110	0001 0100	E
15	1111	0001 0101	F
16	1 0000	0001 0110	10
17	1 0001	0001 0111	11
18	1 0010	0001 1000	12
19	1 0011	0001 1001	13
20	1 0100	0010 0000	14
.	.	.	.
.	.	.	.
126	111 1110	0001 0010 0110	7E
127	111 1111	0001 0010 0111	7F
128	1000 0000	0001 0010 1000	80
.	.	.	.
.	.	.	.
510	1 1111 1110	0101 0001 000	1FE
511	1 1111 1111	0101 0001 0001	1FF
512	10 0000 0000	0101 0001 0010	200

٤-۲ فرمت آدرس دهی در S7

آدرس دهی ورودی ها

ورودی PLC میتواند از جنس Bit یا Byte یا Word یا Dword باشد. بعنوان مثال برای وضعیت یک سوئیچ که به کارت DI متصل است و فقط حالت صفر یا یک دارد یک Bit کافی است وقتی ورودی یک عدد ۸ بیتی است یعنی عدد صحیح بین صفر تا ۲۵۵ در اینصورت یک Byte لازم است ولی برای اعداد بزرگتر یا به فرم اعشاری یک Word یا Dword مورد نیاز خواهد بود.بعنوان مثال وزن یک جسم که از طریق کارت AI دریافت میشود میتواند یک Word باشد.



برای آدرس دهی یک بیت باید ابتدا شماره بایت رابنویسیم سپس با گذاشتن نقطه آدرس بیت را در آن بایت مشخص کنیم مثال :

آدرس بیت	توضیح
0.0	بیت صفر از بایت صفر
4.7	بیت ۷ از بایت ٤

بدیهی است عدد سمت راست که بیت را مشخص میکند نمیتواند از ۷ بزرگتر باشد چون در یک بایت ۸ بیت داریم از صفر تا ۷ ازاینرو آدرسی مانند 0.8 نادرست خواهد بود.

کلیه آدرسهای ورودی در S7 با علامت I شروع میشوند. جدول زیر انواع آدرس دهی ورودی را نشان میدهد.

نوع ورودی	نحوه نمایش	مثال
Bit	I	I 0.1
Byte	IB	IB 1
Word	IW	IW 2
Dword	ID	ID 8

باید توجه داشت وقتی یک IW را در برنامه بکار میبریم آدرس IW بعدی باید حداقل ۲ بایت با آدرس قبلی فاصله داشته باشد. مثلاً IW4,IW2,IW0 و .....پس بکار بردن IW0 و IW1 اشتباه است زیرا ایندو با یکدیگر در بایت شماره ۱ مشترک میباشند.

IW 0 = BYTE0 + BYTE1

IW 1 = BYTE1 + BYTE2

نکته فوق را برای Double Word نیز باید رعایت کرد .یعنی هر آدرس با آدرس بعدی باید ٤ بایت فاصله داشته باشد. مثلاً ID0 و ID4

تذکر:آدرس دهی ورودیهای جنبی (peripheral) که از طریق شبکه دریافت میشوند با علامت PI میباشد

نوع ورودی	نحوه نمایش	مثال
Byte	PIB	PIB 1
Word	PIW	PIW 2
Dword	PID	PID 8

همانطور که دیده میشود در این حالت آدرس دهی برای Bit وجود ندارد.

آدرس دهی خروجی ها

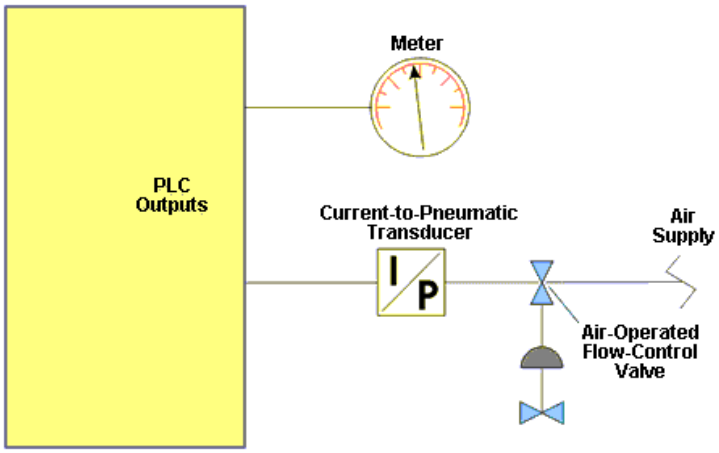
آنچه برای ورودیها شرح داده شد برای خروجی ها نیز صادق است.بجز اینکه برای خروجی ها بجای علامت I علامت Q بکار میرود.

نوع خروجی	نحوه نمایش	مثال
Bit	Q	Q 0.1
Byte	QB	QB 1
Word	QW	QW 2
Dword	QD	QD 8

برای خروجی های جنبی (peripheral) که از طریق شبکه ارسال میشوند نحوه آدرس دهی بصورت زیر خواهد بود..

نوع خروجی	نحوه نمایش	مثال
Byte	PQB	PQB 1
Word	PQW	PQW 2
Dword	PQD	PQD 8

شکل زیر نمونه ای از خروجی های آنالوگ PLC را نشان میدهد..



آدرس دهی متغیرهای حافظه

بجز ورودی و خروجی ها ، متغیرهای حافظه CPU که Bit Memory خوانده میشوند نیز میتوانند آدرس دهی شوند. این متغیرها معمولاً برای ذخیره نتایج میان برنامه بکار میروند. در S5 برای نمایش این متغیرها از علامت F که نشاندهنده Flag بود استفاده میشد.

نوع متغیر حافظه	نحوه نمایش	مثال
Bit	M	M 0.1
Byte	MB	MIB 1
Word	MW	MW 2
Dword	MD	MD 8

آدرس دهی تایمرها و کانترها

تایمرها با علامت T و کانترها با علامت C نمایش داده میشوند آدرس آنها با یک عدد صحیح که بعد از آنها بکار میرود مشخص میگردد مانند T1 یا C2

تذکر : در تمام موارد فوق شماره آدرس نباید از ماکزیمم آدرس تعیین شده در پارامترهای cpu تجاوز کند.

٤-۳ فرمت دیتاها در S7

دیتاهای S7 را میتوان به سه دسته Elementary ، Complex و Parameter تقسیم کرد . برخی از انواع دیتاها جدید است و در S5 موجود نبوده است.

- **دیتاهای Elementary**

نوع	فرمت	سایز (bit)	مثال	فرمت در S5
Bool	-	1	A I 0.0 =Q 0.1 A M1.1	A I 0.0 =Q 0.1 A F1.1
Byte	عدد #B16# که در آن عدد بین 00 تا FF است	8	L B#16#01 L IB0 T QB1 T MB3	عدد KB
Word	عدد #W16# که در آن عدد بین 0000 تا FFFF است	16	L W#16#6AC0 L IW0 T QW2 L MW4	عدد KH
DWord	عدد #DW16# که در آن عدد بین 0000_0000 تا FFFF_FFFF است	32	L ID4 T QD8	عدد DH
Integer (INT)	عدد صحیح با علامت از ۳۲۷٦۸- تا ۳۲۷٦۸+	16	L +35 L -415	عدد KF
Double Integer (DINT)	عدد صحیح با علامت L# که در آن عدد بین 2147483648- تا 2147483647 است	32	L -3200947891	
Real	عدد اعشاری (Floating Point)	32	L 1.23e+1	عدد KG
S5Time	زمان S5T# در پله های ۱۰ میلی ثانیه ای از صفر تا ۲ ساعت و ٤٦ دقیقه و ۳۰ ثانیه که بفرم 2H46M30S0MS نمایش داده میشود	16	L S5T#1M40S L S5Time#20MS	زمان KT
Time	زمان T# در پله های ۱ میلی ثانیه ای از صفر تا ۲٤ روز و ۲۰ ساعت و ۳۱ دقیقه و ۲۳ ثانیه و٦٤۷ میلی ثانیه که بفرم 24D20H31M23S647MS نمایش داده میشود	32	L T#10M20S L Time#1MS	وجود ندارد
Date	تاریخ D# تاریخ به شکل yyyy-mm-ddنوشته میشود	16		L D#2004-3-15 L Date#2004-3-15
Time_Of_Day	TOD#h:m:s.ms که در آن زمان بین 0:0:0.0 تا 23:59:59.999 در پله های ۱ میلی ثانیه	32	L TOD#1:10:3.3	**Time** وجود ندارد
CHAR	‘ یک کاراکتر ‘ کاراکتر میتواند حرف یا عدد باشد	8	L ‘a’ T IB0	ASCII Character

- **دیتاهای Complex**

نوع	فرمت	سایز	مثال	فرمت در S5
DATE_AND_TIME	DT#yy-mm-dd-h:m:s.ms سال میتواند تا ۲۰۸۹ بکار رود مقدار فوق در ۸ بایت بصورت BCD ذخیره میگردد.	64 Bit	DT#1993-12-25-8:01:1.23	وجود ندارد
STRING	STRING[n] n میتواند حداکثر ۲۵٤ باشد. بعنوان مثال اگر بخواهیم کلمه 'Test' را داخل متغیر String بریزیم باید برای آن حداقل String[4] را بکار ببریم	n+2 Byte	STRING[4]	وجود ندارد
ARRAY	فرمت آرایه یک بعدی بصورت زیر است ARRAY[x1..x2] که در آن x1 و x2 دو عدد صحیح هستند و میتوانند بین ۳۲۷٦۸- تا ۳۲۷٦۸+ باشند یعنی کلاً ٦٥٥۳٥عنصر میتوان تعریف کرد.		Test[1..3] که معادل با سه عنصر زیر است : Test[1], Test[2], Test[3]	وجود ندارد
	فرمت آرایه دو بعدی بصورت زیر است ARRAY[x1..x2 و y1..y2] که در آن x1 و x2 و y1 و y2 عدد صحیح هستند و میتوانند بین ۳۲۷٦۸- تا ۳۲۷٦۸+ باشند ولی کل عاصرنباید از ٦٥٥۳٥ بیشتر شود.		Test[1..3,1..2] که معادل با سه عنصر زیر است : Test[1,1] Test[1,2] Test[2,1] Test[2,2] Test[3,1] Test[3,2]	
	....آرایه میتواند حداکثر ٦ بعدی باشد ولی کل عناصرنباید از ٦٥٥۳٥ بیشتر شود. تذکر: عناصر آرایه باید از یک جنس باشند مثلاً همه میتوانند نوع Bool یا همگی بفرض از نوع INT باشند. نوع عناصر را در برنامه باید تعریف کرد.		یک عنصر از آرایه ٦ بعدی بصورت زیر است: Test[1,1,2,1,5,4]	
STRUCT	Struct نام در این قسمت نام و نوع متغیرها تعریف میشود End Struct		Test **Struct** a bool b int c word **End Struct**	وجود ندارد

- **دیتاهای Parameter Types**

نوع	فرمت	سایز	مثال	فرمت در S5
Timer	عدد صحیحT	2 bytes	T1	عدد صحیحT
Counter	عدد صحیحC	2 bytes	C1	عدد صحیحC
Pointer	آدرس P#	6 bytes	P#M50.0	وجود ندارد
Any	-	10 bytes	-	وجود ندارد

٤-٤ آکومولاتور ها و رجیسترهای حافظه CPU S7

آکومولاتورها

بیشتر CPU های S7 شبیه S5 دارای ۲ آکومولاتور هستند.که با ACCU1 و ACCU2 شناخته میشوند. برخی CPU های S7 دارای ٤ آکومولاتور میباشند یعنی علاوه بر ۲ مورد فوق ACCU3 و ACCU4 را نیز دارند.

مقادیری که به حافظه بار میشوند در ACCU1 قرار میگیرند در این شرایط وقتی مقدار جدیدی نیز قرار باشد که بلافاصله به حافظه بار شود (بعنوان مثال برای جمع یا مقایسه با مقدار قبلی) در اینصورت ابتدا محتویات ACCU1 به ACCU2 منتقل شده و مقدار جدید وارد ACCU1 میگردد. در بخشهای بعدی که دستورات برنامه نویسی تشریح میشوند در این مورد بیشتر بحث خواهد شد.

هر کدام از آکومولاتورهای فوق ۳۲ بیتی هستند بعنوان مثال آکومولاتور ۱ ساختاری مانند شکل زیر دارد:

31 24	23 16	15 8	7 0
ACCU1			
ACCU1-H		ACCU1-L	
ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
High Word - High Byte	High Word - Low Byte	Low Word - High Byte	Low Word - Low Byte

بدیهی است :

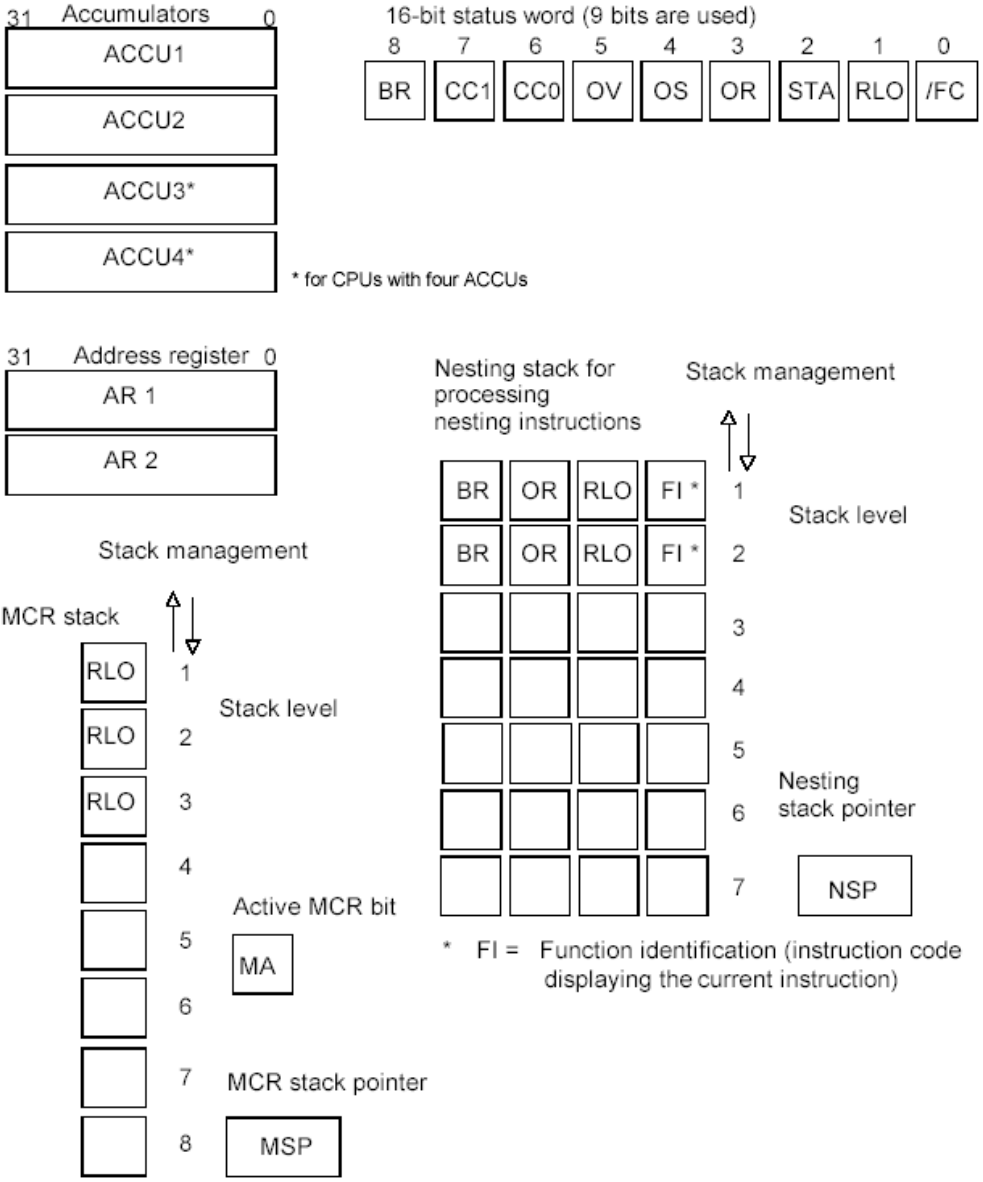
- وقتی یک بایت بار میشود وارد ACCU1-L-L میگردد. (۸ بیت)
- وقتی یک Word بار میشود وارد ACCU1-L میگردد. (١٦ بیت)
- وقتی یک Dword بار میشود وارد ACCU1 میگردد. (۳۲ بیت)

رجیسترها و Stack های حافظه CPU

CPU های S7 دارای رجیسترها و Stack های مختلفی هستند که شکل کلی آنها در صفحه بعد نشان داده شده است و عبارتند از :

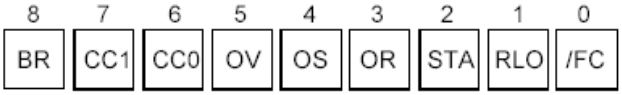
- **Status Word** که یک رجیستر ١٦ بیتی است و ۹ بیت آن همانطور که در صفحات بعد توضیح داده شده است در هنگام پردازش برنامه بسته به نتایج پردازش تغییر میکند.
- **Address Registers** دو رجیستر ۳۲ بیتی برای ذخیره آدرس است و از دستوراتی مانند LAR و TAR تاثیر می پذیرد.
- **Nesting Stack** پشته ای است که دارای ۷ سطح بوده و برای وقتیکه حالت Nesting در برنامه اتفاق می افتد مانند پرانتزهای تودرتو استفاده میشود.با باز شدن هر پرانتز مقدار RLO در این پشته ذخیره شده و با بسته شدن پرانتزبا RLO جدید ترکیب میشود.
- **MCR Stack** پشته ای است که مربوط به دستورات Master Control Relay میباشد و در بخش دستورات برنامه نویسی مفصلا توضیح داده شده اند.

CPU Registers



رجیستر Status Word

از آنجا که این رجیستر نقش مهمی در دستورات برنامه نویسی دارد و بسیاری از دستورات مانند دستورات پرش به آن وابسته هستند لازم است جداگانه تشریح گردد. این رجیستر دارای ۹ بیت به شکل زیر است.



در هر سیکل اسکن مقادیر بیتهای فوق تحت تاثیر نتایج برنامه 0 یا 1 میشوند یا تغییر نمیکنند(که با X نمایش داده میشود)
در S5 رجیستر فوق ۸ بیتی بود و بیت BR در آن وجود نداشت . بعلاوه بجای بیت FC بیت Show بکار میرفت.

بیتهای CC1 , CC0

این دو بیت نتیجه عملیات محاسباتی را طبق جدول زیر نشان میدهند:

نتیجه عملیات	CC0	CC1
مثبت	1	0
منفی	0	1
صفر	0	0
حالت تعریف نشده	1	1

بیت OV

این بیت اگر در نتیجه عملیات محاسباتی سرریزی (Overflow) اتفاق بیفتد 1 میشود. وقتی برنامه در همان سیکل اسکن به دستور محاسباتی جدیدی برسد این بیت ری ست شده و براساس نتایج جدید Update میگردد.

بیت OS

این بیت معرف OverFlow Stored است یعنی سرریزی قبلی که در همان سیکل اسکن اتفاق افتاده را ذخیره میکند. برخلاف بیت OV که با رسیدن به دستور محاسباتی جدید ری ست میشود این بیت مقدار سرریزی را تا پایان آن سیکل اسکن حفظ میکند.

بیت OR

این بیت در صورتی 1 میشود که در خلال دستورات منطقی (bit Logic) عمل AND قبل از OR وجود داشته باشد.

بیت STA

این بیت دقیقا وضعیت سیگنال منطقی را نشان میدهد که در آن لحظه 1 است یا 0.

بیت RLO

این بیت معرف نتیجه عملیات منطقی (Result of Logic Operation) است مثلا وقتی دو سیگنال که یکی 0 و دیگری 1 است با هم AND شوند این بیت نتیجه یعنی 0 را نشان میدهد.

بیت FC

این بیت معرف First Check است در واقع RLO را راهنمایی میکند که وارد Network جدید از برنامه شده یا از بلاکی که صدا زده شده برگشت به بلاک ماقبل اتفاق افتاده است. در این شرایط باید RLO نتایج قبلی را دور بریزد و متناسب با عملیات جدید Update شود.

بیت BR

در SAVE قبلی را هم داشته باشیم این نتیجه با دستور برنامه نویسی RLO ذکر شد نتیجه First Check اگر بخواهیم تحت شرایطی که در ذخیره میشود. BR

٤-٥ بلاکهای برنامه نویسی

بلاکهایی که در برنامه نویسی توسط Step7 بکار میروند به سه دسته زیر تقسیم میشوند.

۱- بلاکهای منطقی (logic Blocks)

این بلاکها حاوی دستورات برنامه نویسی هستند که به آنها Code Blocks نیز گفته میشود و عبارتند از :

- OB Organization Block
- FB Function Block
- FC Function
- SFB System Function Block
- SFC System Function

۲- دیتا بلاکها

این بلاک ها همانطور که از نامشان پیداست حاوی دیتا هستند . دیتاهایی که توسط بلاکهای منطقی خوانده یا نوشته میشوند انواع دیتا بلاکها عبارتند از :

- DB Data Block
- SDB System Data Block

۳- UDT یا User Defined Type

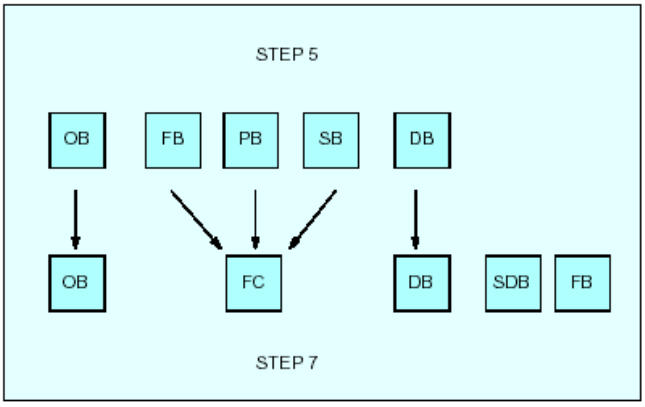
بلاکهایی هستند که حاوی دیتا بوده و بسته به نیاز همراه با دیتا بلاک برای پرهیز از نوشتن دیتاهای تکراری بکار میروند.

تعداد بلاک ها بسته به نوع CPU متفاوت است جدول زیر تعداد بلاکها را در چند CPU مختلف مقایسه کرده است:

CPU 416-2	CPU 412-1	CPU 314	CPU 312	
44	23	13	3	**OB**
2048	256	128	32	**FB**
2048	256	128	32	**FC**
4095	511	127	63	**DB**

مقایسه بلاک های برنامه نویسی Step7 با Step5

شکل زیر این مقایسه را نشان میدهد



بطور خلاصه میتوان گفت :

- OB در Step5 و Step7 با همان مفهوم و همان نام بکار میرود.
- DB در Step5 و Step7 با همان مفهوم و همان نام بکار میرود.
- FB در Step5 به FC در Step7 تبدیل شده است.
- PB یا Program Block در Step5 به FC در Step7 تبدیل شده است.
- SB یا Sequence Block در Step5 به FC در Step7 تبدیل شده است.

این تغییرات بیشتر بمنظور تطابق با استاندارد IEC1131 انجام شده است.

تذکر :
FB در Step7 فانکشن جدیدی است و نباید با FB مربوط به Step5 اشتباه گرفته شود.

Organization Blocks

OB ها در PLC رابط بین سیستم عامل (Operating System) و برنامه کاربری (user Program) هستند. این بلاکها توسط سیستم عامل فرا خوانده شده و برای مقاصدی چون کنترل سیکلی L،وقفه و راه اندازی بکار میروند. در بین OB ها نام OB1 برای اکثر کاربران آشناتر است. این بلاک هم در Step5 و هم در Step7 برای اجرای سیکلی برنامه کاربری استفاده میشود. در واقع برنامه اصلی کنترل در آن نوشته میشود و بلاک های غیر OBدیگر در صورت نیاز از داخل OB1 صدا زده میشوند.

OB ها انواع مختلف دارند . لیست و نام آنها را میتوان در جدول زیر مشاهده کرد:

نام **OB**	نوع	درجه اولویت
OB1	Main program scan	1
OB10 to OB17	Time-of-day interrupts	2
OB20 to OB23	Time-delay interrupts OB20	3 to 6
OB30 to OB38	Cyclic interrupts	7 to 15
OB40 to OB47	Hardware interrupts	16 to 23
OB60	Multicomputing interrupt	25
OB70, OB72	Redundancy errors	25 28
OB80, OB82 to OB87	Asynchronous errors	26
OB90	Background cycle	29
OB100 to OB102	Startup	27
OB121, OB122	Synchronous errors	

همانطور که مشاهده میشود هر گروه از OB ها یک درجه اولویت (priority) دارد. هر چه عدد اولویت بالاتر باشد اهمیت آن بلاک از نظر اجرا بیشتر است. بعبارت دیگر اجرای هر OB میتواند توسط OB دیگری که درجه اولویت بالاتر دارد قطع گردد. OB1 کمترین اولویت را دارد و هرکدام از OB های دیگر میتوانند آنرا قطع کنند. بسته به نوع CPU ممکن است برخی از این OB ها موجود نباشند.

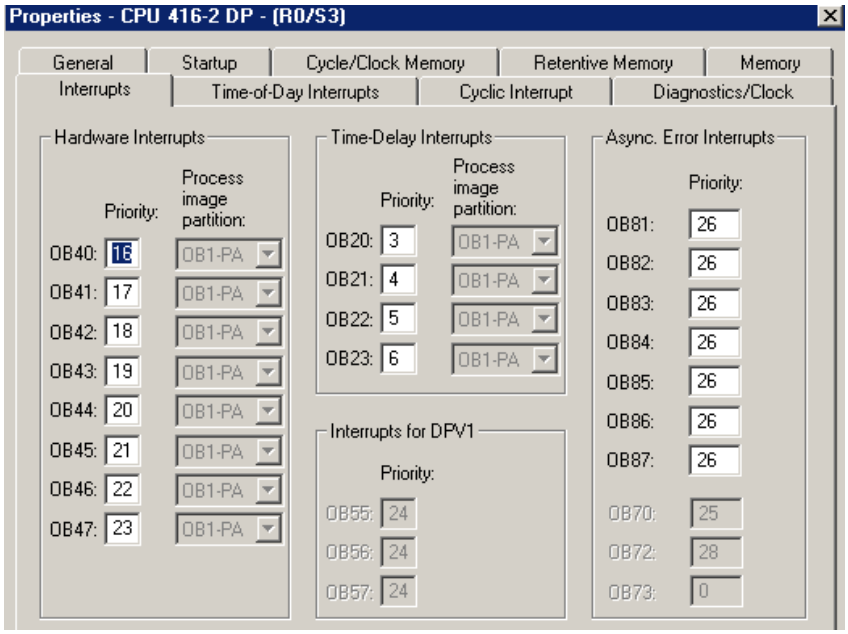
درجات اولویت OB ها که در جدول فوق آورده شده بعنوان پیش فرض سیستم میباشد در S7-300 اولویتها ثابت بوده ولی در S7-400 میتوان آنها را تغییر داد. در پارامترهای CPU در بخشهایی که مربوط به وقفه است مانند Time of Day Interrupt و Cyclic Interrupt و Interrupt اولویت ها لیست شده و قابل تغییر میباشند .

شکل زیر بخش وقفه های سخت افزاری CPU 416-2DP را نشان میدهد.باید توجه داشت که تغییر اولویتها در رنج خاصی بشرح زیر میتواند انجام شود:

- OB10 تا OB47 بین ۲ تا ۲۳
- OB70 تا OB72 بین ۲۵ تا ۲۸
- OB81 تا OB87 بین ۲٤ تا ۲٦

در محدوده فوق OB های مختلف میتوانند دارای درجه اولویت یکسان نیز باشند.

اگر به درجه اولویت عدد صفر داده شود آن OB اصطلاحاً Deselect شده و فراخوانده نمیشود.



بجز OB1 سایر OB ها در جلد دوم کتاب مورد بحث قرار میگیرند .

مقایسه OB های S5 و S7

۱) OB1 در S7 به همان نام و همان مفهوم OB1 در S5 بکار میرود.

۲) برخی OB ها در S7 همان فانکشن OB های S5 را دارند ولی شماره آنها عوض شده است. بعنوان مثال برای راه اندازی Cold بلاک OB100 در Step7 جایگزین بلاک OB21 شده است.جزئیات بیشتر در جدول صفحه بعد آمده است.

۳) برخی OB های S5 در S7 حذف شده اند و فانکشن های سیستم یعنی SFC جایگزین آنها گردیده است مثال :

	در S5	در S7
فعال و غیر فعال کردن وقفه	OB120	SFC40 و SFC41

٤) برخی OB های S5 در S7 با یک دستور برنامه نویسی جایگزین شده اند مثال :

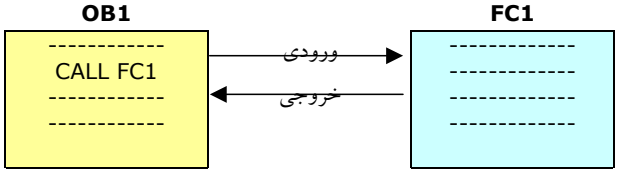
	در S5	در S7
ایجاد لوپ	OB160	LOOP
پاک کردن آکومولاتور	OB111	PUSH

جدول مقایسه OB های S5 و S7

Function		در S5	در S7
Main program	Free cycle	OB1	OB1
Interrupts	Time-delay (delayed) interrupt	OB6	OB20 to OB23
	Time-of-day (clock-controlled) interrupt	OB9	OB10 to OB17
	Hardware interrupts	OB2 to OB5	OB40 to OB47
	Process interrupts	OB2 to OB9	Replaced by hardware interrupts
	Cyclic (timed) interrupts	OB10 to OB18	OB30 to OB38
	Multicomputing interrupt	-	OB60
Startup	Manual complete (cold) restart OB100	OB21 (S5-115U) OB20 (S5-135U)	OB100
	Manual (warm) restart	OB21 (S5-135U)	OB101
	Automatic (warm) restart	OB22	OB101
Errors	Error	OB19 to OB35	OB121, OB122, OB80 to OB87
Other	Processing in STOP mode	OB39	Omitted
	Background processing	-	OB90

فانکشن FC (Function)

FC ها بلاک های منطقی و حاوی دستورات برنامه نویسی هستند که از داخل بلاکهای دیگر صدا زده میشوند. بطور معمول FC یکسری ورودی میگیرد و بر اساس برنامه ای که داخل آن نوشته شده خروجیهایی را ایجاد مینماید. ورودی ها در بلاک ماقبل که FC را صدا میزند به FC داده میشود و خروجی های FC نیز در همان بلاک مورد استفاده قرار میگیرد.



استفاده از FC بخصوص وقتی در برنامه اصلی نیاز به انجام فانکشنی تکراری باشد که فقط هر بار ورودی های آن تغییر کند یا آدرس خروجی هایش عوض شود بسیار مفید است و منجر به کاهش حجم دستورات برنامه نویسی میگردد.

در قسمتهای بعد تشریح خواهد شد که در هنگام ایجاد FC کاربر چگونه ورودی و خروجی های آن را مشخص کند اما در عین حال ممکن است یک فانکشن FC ورودی و خروجی نیاز نداشته باشد و صرفا دستور خاصی را اجرا کند.

فانکشن بلاک FB (Function Block)

FB از نظر عملکرد شبیه FC است یعنی ورودی میگیرد و خروجی ایجاد میکند ولی یک تفاوت مهم با FC دارد و آن اینکه دارای حافظه است. حافظه آن یک دیتا بلاک خاص است . وقتی FB صدا زده میشود باید همراه با آن نام دیتا بلاک که حافظه اش تلقی میشود را نیز بکار برد مثال:

CALL FB1 , DB1

تمام ورودی و خروجی های FB و سایر پارامترهایی که در موقع ایجاد FB تعریف میشوند در این دیتا بلاک ذخیره میگردند.

دیتا بلاک DB (Data Block)

دیتا بلاکها برخلاف OB ,FC,FB حاوی دستورات Step7 نیستند بلکه برای ذخیره سازی دیتاها بکار میروند.میتوان یک دیتا بلاک را Write Protect کرد تا PLC در حین اجرا ی برنامه نتواند دیتاهای آن را عوض کند این کار با کلیک راست روی DB و انتخاب Properties امکان پذیر است.

دیتا بلاکها بر ۲ نوع هستند:

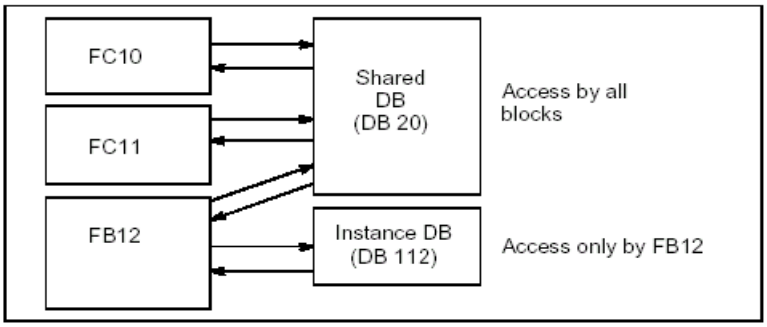
۱- Shared DB

این دیتا بلاکها همانطور که از نامشان پیداست بصورت اشتراکی بوده و برای تمام بلاکهای برنامه قابل دسترسی هستند. دیتاهای آنها با بسته شدن دیتا بلاک از بین نمیروند.سایز DBمتغیر است و بستگی به نوع CPU دارد.

۲- Instance DB

این دیتا بلاکها بعنوان حافظه ای برای FB ها منظور شده اند پارامترهایی که به FB ارسال شده و متغیرهای استاتیک در این DBها ذخیره میشوند و وقتی اجرای FB کامل میشود از بین نمیروند.

شکل زیر نمونه ای از کاربرد ۲ نوع دیتابلاک فوق را نشان میدهد.



بلاکهای سیستم System Blocks

بلاکهایی هستند که از قبل برای مقاصد خاصی نوشته شده اند و به دو دسته زیر تقسیم میشوند:

- **SFC** یا System Function
- **SFB** یا System Function Block

SFB ها همانند FB ها دارای حافظه هستند و باید با نام دیتا بلاک مربوط به آنها صدا زده شوند ولی SFC ها مانند FC ها حافظه ندارند.مثالی از این بلاک ها در زیر آمده است . لیست کامل بلاک های سیستم را میتوانید در ضمیمه ۵ ببینید.

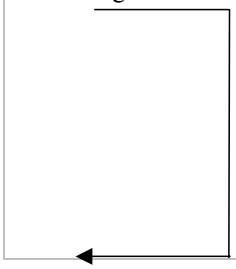
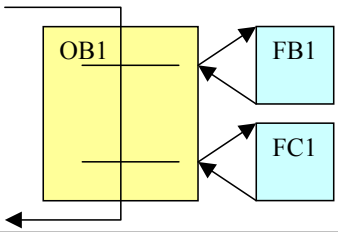
SFC 46	منجر میشود که cpu به مد stop برود
SFB 41	برای لوپ کنترل بکار میرود

فراخوانی بلاکها از داخل یکدیگر

همانطور که میدانیم برنامه میتواند به یکی از دو روش زیر نوشته شود که در شکل نیز نشان داده شده است :

۱- بطور کامل در بلاک اصلی OB1 که به این روش Linear Programming گویند.

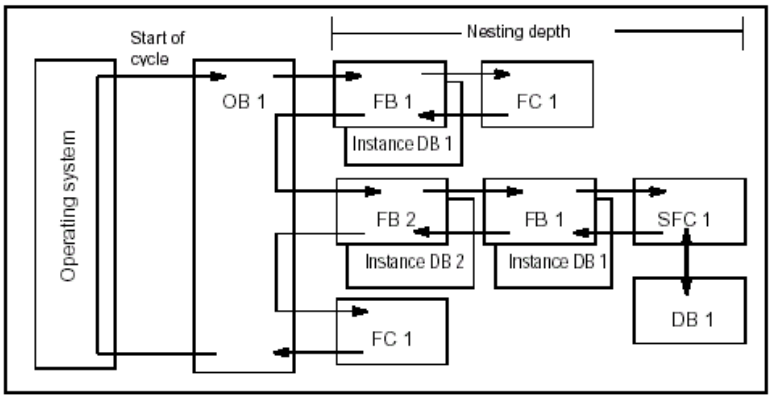
۲- بخشهای برنامه در بلاک های مختلف نوشته شده و از بلاک ماقبل صدا زده شوند که به این روش Structured Programming میگویند.

Linear Programming	Structured Programming
Main Program = OB1 	Main Program 

از OBمیتوان FC و FB را صدا زد و همینطور از FB و FC نیز میتوان سایر FB ها و FC ها را صدا زد. نکته ای که باید رعایت شود اینستکه توسط Step7 ابتدا باید بلاک نهایی و سپس بلاک های ماقبل ایجاد شود.یعنی مثلا در شکل فوق ابتدا FB1 و FC1 و سپس OB1 .

تعداد بلاکهایی که از داخل هم فرا خوان میشوند بستگی به Nesting Depth دارد که از مشخصات CPU است. مثلا برای CPU 315 این پارامتر ۸ برای CPU 414-3 این پارامتر ۲۴ میباشد یعنی حداکثر ۲۴ بلاک تودرتو میتوانیم داشته باشیم.

شکل زیر Nesting Depth را بصورت نمونه نشان میدهد.

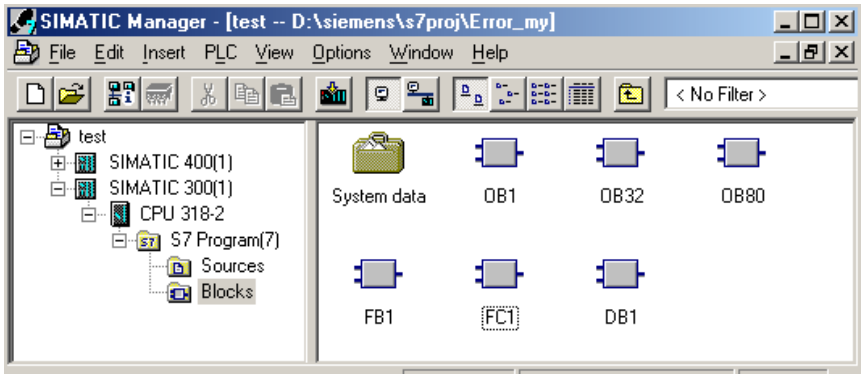


تذکر : در روش structured Programming لازم است تمام بلاکها به PLC دانلود شوند در غیر اینصورت اگر بلاکی صدا زده شود و در حافظه موجود نباشد PLC متوقف شده و چراغ SF روشن میشود.

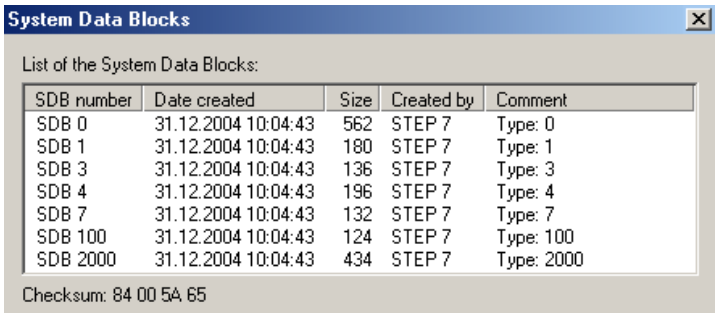
٦-٤ نحوه ایجاد بلاک در Simatic Manager

بلاکهای OB و FB و FC و DB که توضیح داده شد توسط Simatic Manager ایجاد میشوند. باید توجه داشت که بلاکهای سیستم یعنی SFB ها و SFC ها از قبل ایجاد شده اند و در نرم افزار موجود هستند از اینرو این بلاکها را نمیتوان ایجاد کرد.

قبلا توضیح داده شد که میتوان برنامه نویسی را قبل یا بعد از پیکر بندی سخت افزار انجام داد. اگر قبلا سخت افزار پیکر بندی شده باشد با باز کردن Station در پنجره سمت چپ پوشه S7 Program را مشاهده میکنیم که در زیر CPU قرار گرفته است. با باز کردن این پوشه دو پوشه دیگر مطابق شکل مشاهده میکنیم یکی به نام Blocks و دیگری بنام Sources

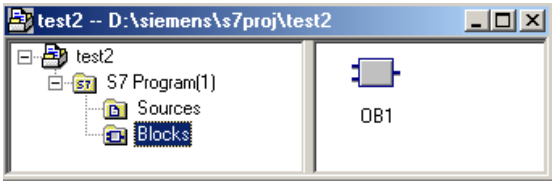


پوشه Source برای برنامه نویسی به صورت Source بکار میرود که در جلد دوم کتاب تشریح خواهد شد. بلاکهای برنامه در پوشه Blocks ایجاد میشوند. معمولا پس از پیکر بندی سخت افزار بطور پیش فرض یک بلاک OB1 در داخل پوشه Blocks ایجاد میشود که البته خالیست و برنامه ای داخل آن وجود ندارد.بعلاوه اگر بعد از اتمام کار پیکر بندی سخت افزار توسط Hwconfig عمل کامپایل را توسط Save and Compile انجام داده باشیم آیکون دیگری به نام System data در پوشه Blocks مانندشکل فوق ظاهر میشود که اطلاعات سخت افزار و شبکه را در خود دارد این اطلاعات در تعدادی SDB یا System Data Blocks ذخیره میشوند که با کلیک روی آیکون فوق میتوان آنها را مانند شکل زیر مشاهده نمود:



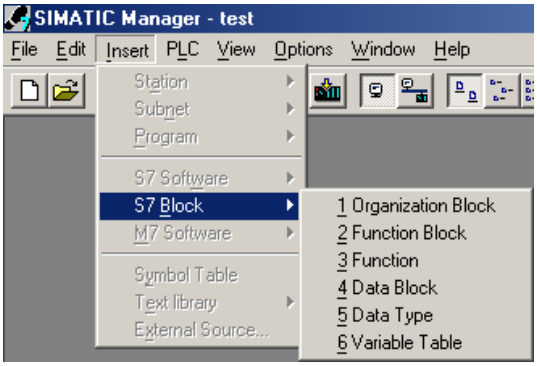
این اطلاعات به PLC دانلود شده و براساس آنها PLC میتواند سخت افزار پیکر بندی شده را تشخیص دهد.

اگر سخت افزار قبلا پیکر بندی نشود و کاربر بخواهد ابتدا برنامه نویسی را انجام دهد. دراینصورت مانند شکل زیر پوشه S7 Program و در زیر مجموعه آن پوشه Blocks را خواهیم داشت.



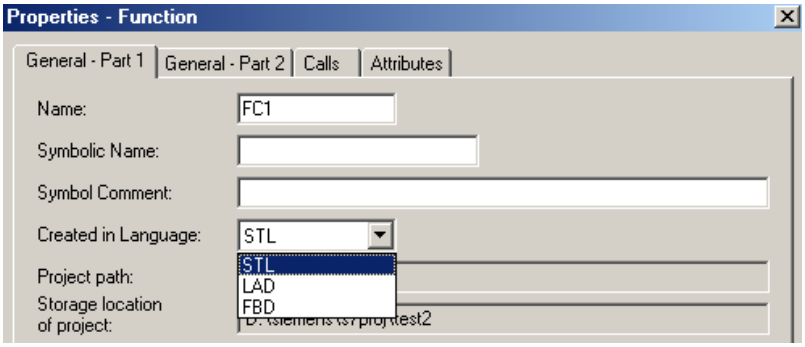
در هردوحالت یعنی چه با وجود سخت افزار و چه بدون آن برای ایجاد بلاک میتوان به یکی از دو روش زیر عمل کرد :

۱- کلیک روی پوشه Blocks و سپس با استفاده از منوی Insert > S7 Blocks و انتخاب Block مورد نظر از لیست مانند شکل:



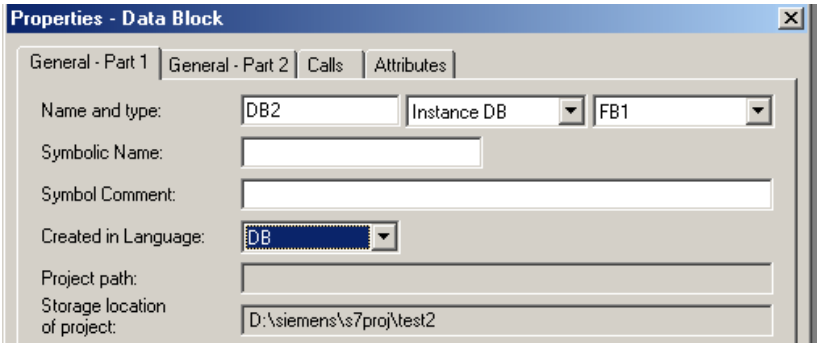
۲- کلیک راست روی پوشه Blocks یا در پنجره سمت راست و انتخاب Insert New Object که در اینصورت نیز لیست فوق را خواهیم دید.

پس از انتخاب بلاک مورد نظر پنجره جدیدی مانند شکل زیر باز میشود



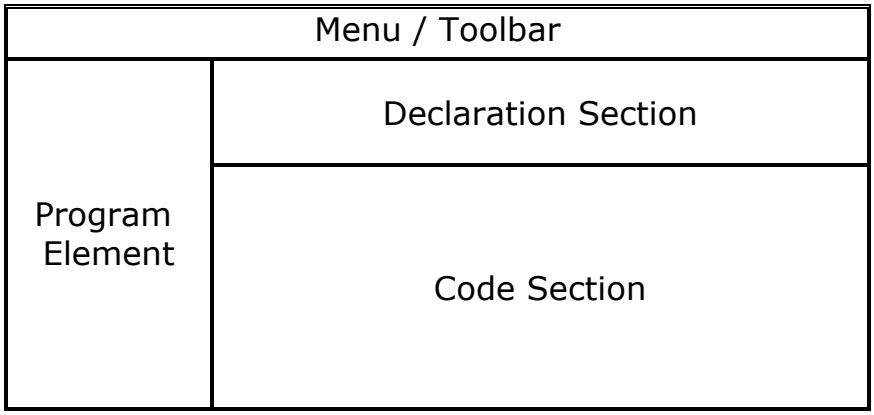
نکاتی که باید به آنها توجه کرد :

- .هر بلاک دارای یک شماره است و خود برنامه این شماره را بعد ا زاسم بلاک پیشنهاد میدهد که در صورت نیاز میتوان آنرا تغییر داد.
- زبان برنامه نویسی برای بلاکهای منطقی یعنی FC , FB , OB زبان STL است که در صورت دلخواه قابل تغییر است.
- وقتی دیتا بلاک ایجاد میشود نوع آن بطور پیش فرض Shared می باشد . میتوان آنرا از نوع Instance انتخاب کرد (شکل زیر) که در اینصورت شماره FB را باید وارد کنیم.یعنی FB باید قبلا در پوشه Blocks ایجاد شده باشد.
- اگر در Simatic Manager از منوی Option > Customize در بخش General گزینه Open New Object فعال شده باشد در اینصورت هر بلاک بمحض ایجاد شدن با برنامه مربوطه اش باز میشود.



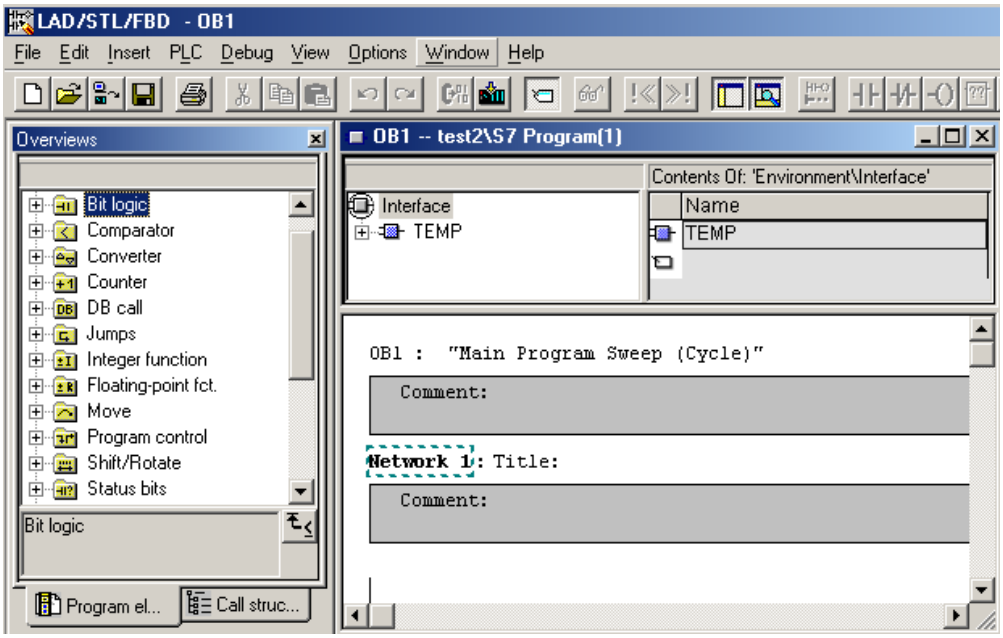
٤-۷ آشنایی با محیط زیر برنامه LAD/STL/FBD

اگر روی هر کدام از بلاکهای OB و FB و FC که در پوشه Blocks برنامه Simatic Manager ایجاد شده اند کلیک کنیم برنامه جدیدی به نام LAD/STL/FBD اجرا شده و بلاک مورد نظر در محیط آن باز میشود. شکل کلی محیط این برنامه بصورت زیر است:



Program Element:

این بخش المانهای برنامه نویسی را در بر دارد. بسته به اینکه چه نوع زبانی انتخاب شده باشد این المانها متفاوت خواند بود شکل زیر این المانها را برای زبان LAD نشان میدهد. کاربر میتواند با انتخاب هر یک از زبانهای LAD یا STL یا FBD از منوی **View** تفاوت محتویات این قسمت را برای زبانهای مختلف ملاحظه کند.



درصورت فعال نبودن بخش Program Element کاربر میتواند آنرا از طریق منوی Insert > Program Element فعال نماید.

Declaration Section:

این بخش جهت تعریف متغیرهای محلی و ورودی و خروجی بلاک بکار میرود.برای OB ها در این بخش صرفا متغیر Temp را میتوان دید ولی برای FB و FC میتوان علاوه بر Temp ورودی و خروجی و موارد دیگری را مشاهده کرد. جدول زیر موضوع را کاملتر میکند.

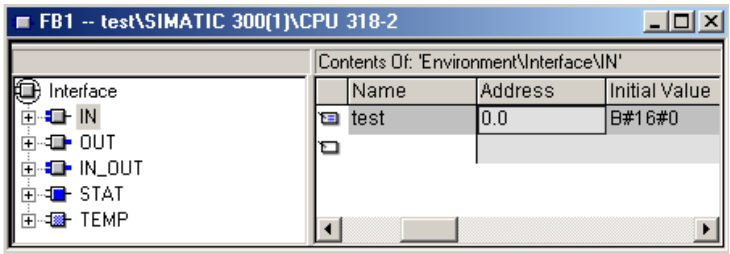
Declaration	FC	FB	OB
in	X	X	-
Out	X	X	-
In/out	X	X	-
temp	X	X	X
static	-	X	-

نکاتی که با توجه به جدول فوق باید به آن توجه داشت :

- متغیر Temp متغیر موقت محلی است که در تمام بلاکها قابل تعریف است. مقدار ذخیره شده در این متغیر با بسته شدن بلاک از بین میرود.
- از متغیرهای Temp که توسط کاربر تعریف میشوند میتوان شبیه فلاگ ها برای ذخیره سازی نتایج میان برنامه استفاده کرد.
- در OB ها متغیرهای Temp از قبل تعریف شده ای هستند که کاربرد مهمی در مدیریت خطاها دارند و در جلد دوم کتاب بحث میشوند.
- OB ورودی و خروجی ندارد بنا براین نمیتوان شبیه فانکشن آنرا صدا زد.
- متغیر in برای ورودیهای بلاک و متغیر out برای خروجیهای بلاک تعریف میشود.
- متغیر in/out برای یک خروجی که بعنوان ورودی نیز بکار میرود (مانند خروجی که در حالت خود نگهدار بعنوان ورودی استفاده میشود)
- متغیر Static خاص FB است . موارد‌مربوط به FB بجز نوع Temp همگی در DB مربوطه ذخیره میشوند. پس اگر نیاز به متغیری برای ذخیره سازی نتایج میان برنامه باشد و لازم باشد که این مقادیر در DB ذخیره شوند آنرا از نوع Static تعریف میکنیم.
- قبل از تمام متغیرهای فوق الذکر وقتی در برنامه نویسی استفاده میشوند باید علامت # قرار داد. مثال :

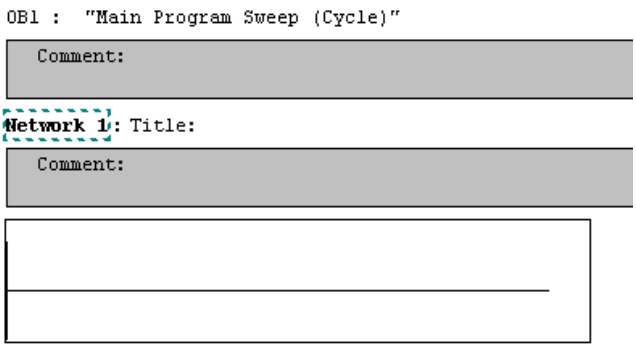
L #test

شکل زیر بخش Declaration را برای یک FB نشان میدهد.که ورودی test توسط کاربر در آن تعریف شده است.



Code Section:

این بخش جهت برنامه نویسی استفاده میشود و خاص بلاکهای منطقی است . شکل زیر این بخش را برای بلاک OB1 که زبان برنامه نویسی آن بصورت LAD انتخاب شده نمایش میدهد.همانطور که اشاره شد میتوان زبان برنامه نویسی را از طریق منوی View تغییر داد و STL یا FBD یا LAD را انتخاب کرد. در حالت STL و FBD بخش Code Section تنها تفاوتی که با شکل زیر دارد آنست که در انتها بجای دیاگرام T شکل یک باکس خالی نمایش داده میشود.



همانطور که در شکل ملاحظه میگردد در این قسمت موارد زیر وجود دارد:

Block Title

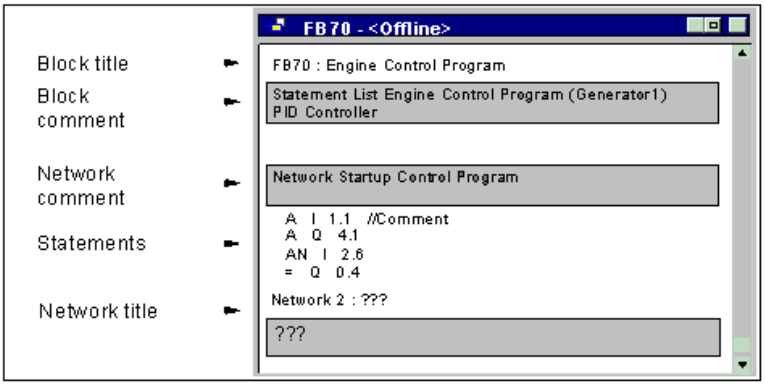
با کلیک کردن روی این قسمت نام بلاک و عنوان آن را میتوان نوشت حداکثر ٦٤ کاراکتر.

Block Comment

این باکس خاکستری رنگی برای Comment است در این باکس کاربر میتواند توضیحاتی در مورد بلاک و عملکرد آن بنویسد..

Network Comment

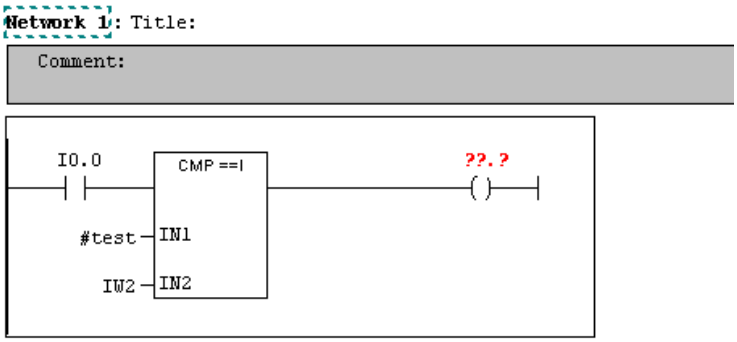
قبل از هرچیز باید توجه داشت که Network در اینجا معنای شبکه ندارد بلکه نشان دهنده یک قسمت از برنامه یا بعبارت دیگر یک Segment است. هر بلاک برنامه نویسی میتواندشامل Network های مختلف باشد. که بدنبال هم قرار گرفته اند و در هر کدام از آنها یک بخش از برنامه نوشته شده است. برای اضافه کردن Network میتوان از منوی Insert > Network یا از آیکون  که در Toolbars بالای برنامه وجود دارد استفاده کرد.هر Network یک Title دارد که میتوان با حداکثر 64 کاراکتر آنرا معرفی کرد . در زیر آن باکس خاکستری Network Comment است که میتوان در آن توضیحاتی را در مورد این بخش از برنامه نوشت. برنامه هر Network در قسمت Statement نوشته میشود . شکل زیر برنامه ای که به زبان STL در این بخش نوشته شده را نشان میدهد.



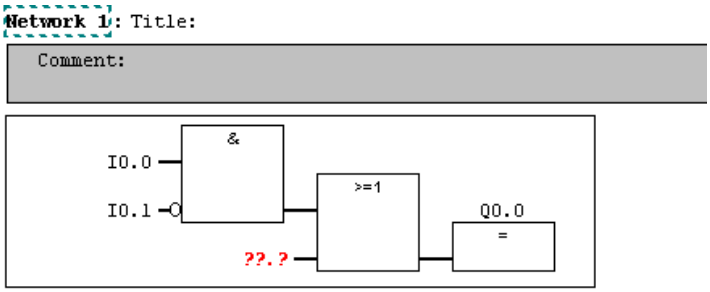
معمولا در برنامه یک بلاک ، بخشهای مختلف برنامه کنترل را از یکدیگر جدا کرده و هر بخش را در Network جداگانه مینویسند. بعنوان مثال کنترل راه اندازی یک موتور در Network 1 و برنامه توقف آن در Network 2 نوشته میشود. اگر فرض کنیم این Network ها در بلاک OB1 نوشته شده باشند در هر سیکل اسکن تمام آنها بدنبال هم اجرا میگردند.

برای وارد کردن دستورات برنامه نویسی ابتدا در قسمت Statement کلیک میکنیم . اگر زبان برنامه نویسی بصورت STL باشد لازم است برنامه در این قسمت تایپ شود ولی اگر به زبان LAD و FBD باشد از المانهای موجود در Toolbar یا المانهای بخش Program Element استفاده کرده و با Drag کردن توسط ماوس آنها را به Network وارد میکنیم.

شکل زیر مثالی از برنامه ای است که بصورت LAD نوشته شده است.

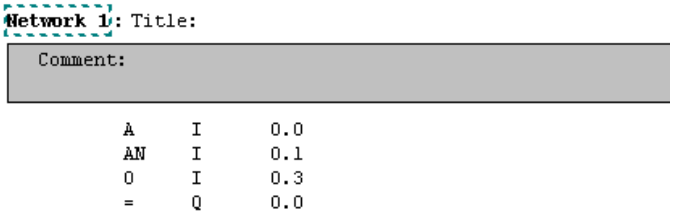


شکل زیر مثالی از برنامه ای است که بصورت FBD نوشته شده است.



آدرس ها و پارامترها باید با فرمت تعیین شده بکار رود در غیر اینصورت با رنگ قرمز ظاهر میشوند که نشان دهنده وجود اشکال است . همینطور اگر آدرسی تعیین نشود با علامت سوال مانند شکل های بالا ظاهر خواهد شد.

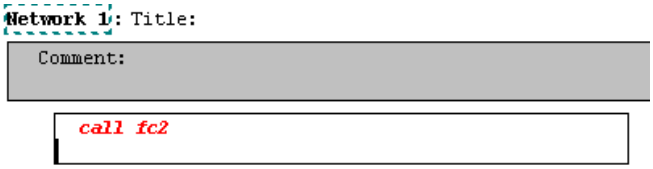
شکل زیر برنامه ای را نشان میدهد که بصورت FBD نوشته شده است.



٤-٨ نحوه استفاده از بلاک ها

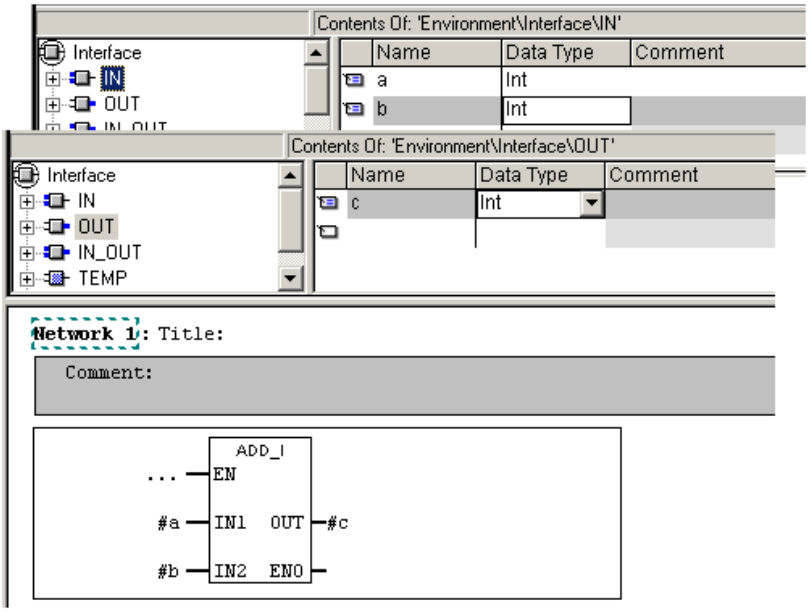
نحوه ایجاد و استفاده از فانکشن FC

همانطور که قبلا ذکر شد اگر از داخل یک بلاک مثلا OB1 بخواهیم فانکشنی را صدا بزنیم لازم است که قبلا آنرا ایجاد کرده و ورودی و خروجی های آن را تعیین کرده باشیم. اگر قبل از ایجاد فانکشن آنرا صدا بزنیم میبینیم که دستور با رنگ قرمز که نشان دهنده وجود اشکال است ظاهر میشود مانند شکل زیر:



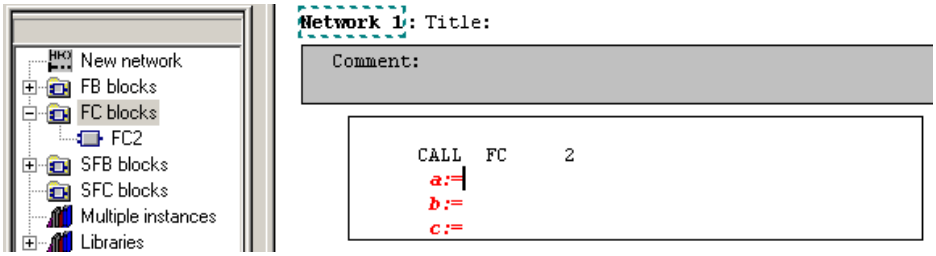
بنابراین لازم است ابتدا توسط Simatic Manager این فانکشن را ایجاد کرده و سپس با کلیک کردن روی آن آنرا توسط LAD/STL/FBD باز کنیم. پس از آن ابتدا باید ورودی و خروجی فانکشن را در قسمت Declaration تعریف نماییم.

فرض کنید که فانکشنی دو عدد صحیح a, b را میگیرد و جمع آنها یعنی C را برمیگرداند پس در قسمت Declaration باید a و b بعنوان ورودی و C بعنوان خروجی و همه اینها از نوع عدد صحیح یعنی Int تعریف شوند.سپس برنامه جمع در Network داخل FC نوشته شود مانند شکل زیر:



اکنون اگر FC را ذخیره کنیم و به OB1 برگردیم مانند شکل بالای صفحه بعد اولا در پنجره Program Element در زیر مجوعه **FC Blocks** آنرا مشاهده میکنیم ثانیا پس از صدا زدن فانکشن میبینیم که پارامترهای ورودی و خروجی آن در زیر دستور CALL ظاهر میشوند. با دادن مقادیر یا آدرس مناسب به ورودی ها و آدرس مورد نظر به خروجی باید آنرا تکمیل نمود. این فانکشن میتواند بارها و از هر بلاکی صدا زده شود.

مثالی که ذکر شد برنامه ساده ای بود که صرفا به جهت آشنایی کاربر ارائه گردید. در عمل برنامه های داخل فانکشن پیچیده تر از این هستند.



نحوه ایجاد و استفاده از فانکشن بلاک FB

روش ایجاد و استفاده از FB تا حدودی شبیه FC است ولی بدلیل نیاز به دیتا بلاک قدمهایی که باید برداشته شود بصورت زیرخواهد بود :

۱- ایجاد FB در Simatic Manager

۲- باز کردن FB توسط برنامه LAD/STL/FBD

۳- تعریف ورودی و خروجی و سایر متغیرها در قسمت Declaration

٤- نوشتن برنامه در Network های مربوطه

٥- ذخیره سازی FB

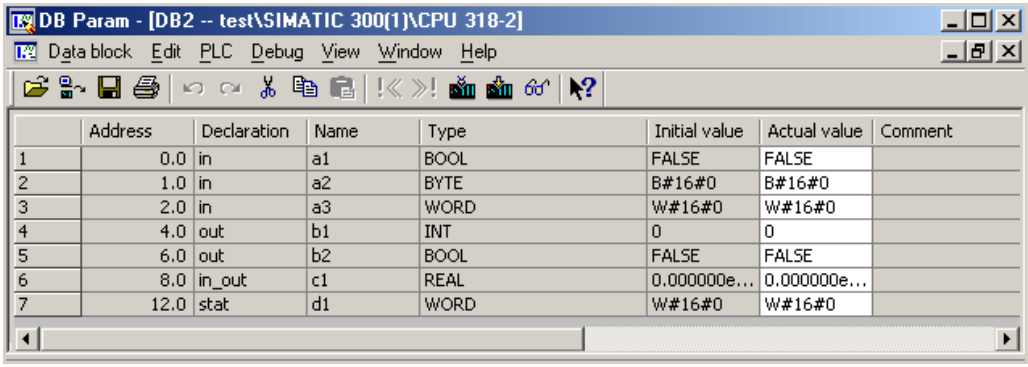
٦- بازگشت به Simatic Manager و ایجاد دیتابلاک

۷- انتخاب نوع Instance برای دیتابلاک و انتخاب FB مربوط به آن

پس از انجام مراحل فوق میتوان FB را در پنجره Program Element درزیر مجموعه **FB Blocks** مشاهده کرد و میتوان در هر بلاکی مثلا در OB1 آنرا صدا زد ولی در دستور CALL باید حتما نام DB مربوط به آن نیز آورده شود مثال :

CALL FB1,DB2

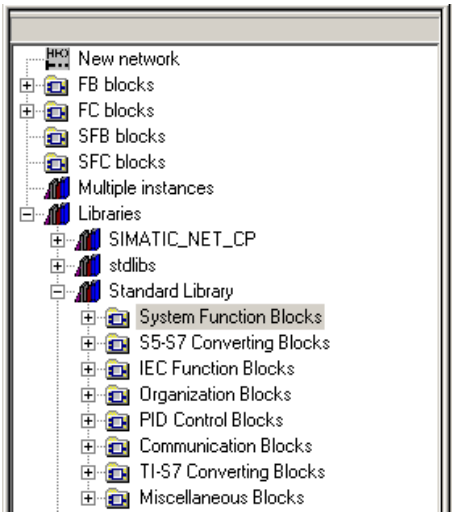
بدیهی است ورودیها و خروجی ها و سایر متغیرهای تعریف شده در Declaration فانکشن بلاک در زیر آن ظاهر خواهند شد. اگر DB قبلا موجود نباشد پس از دستور CALL برنامه سوال میکند که آنرا ایجاد کند یا نه و در اینصورت نیازی به مراحل ٦ و ۷ نمیباشد. در حین اجرای برنامه توسط PLC تمام پارامترهای فانکشن بلاک بجز نوع Temp در دیتا بلاک مربوط به آن بطور اتوماتیک ذخیره میشوند و نیازی به استفاده از دستور برنامه نویسی خاصی برای اینکار نیست.اگر کاربر بعد از مرحله ٦ فوق الذکر روی DB در Simatic Manager کلیک کند تا باز شود مشاهده خواهد کرد که محتویات DB دقیقا شبیه آنچه در بالای FB تعریف کرده میباشد (بجز متغیر Temp که در آن آورده نمیشود) نمونه در شکل زیر مشاهده میشود.



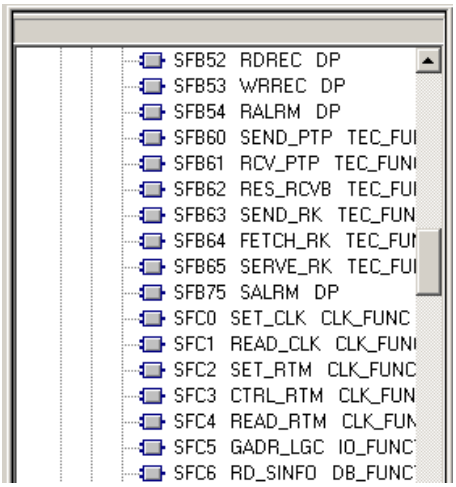
	Address	Declaration	Name	Type	Initial value	Actual value	Comment
1	0.0	in	a1	BOOL	FALSE	FALSE	
2	1.0	in	a2	BYTE	B#16#0	B#16#0	
3	2.0	in	a3	WORD	W#16#0	W#16#0	
4	4.0	out	b1	INT	0	0	
5	6.0	out	b2	BOOL	FALSE	FALSE	
6	8.0	in_out	c1	REAL	0.000000e...	0.000000e...	
7	12.0	stat	d1	WORD	W#16#0	W#16#0	

نحوه استفاده از فانکشنهای سیستم SFC و SFB

قبلا نیز اشاره شد که فانکشنهای سیستم (رجوع شود به ضمیمه ۵) از قبل نوشته شده و در برنامه موجود هستند. برای مشاهده آنها کافیست در پنجره Program Element مسیری مانند شکل **الف** زیر طی شود:



الف



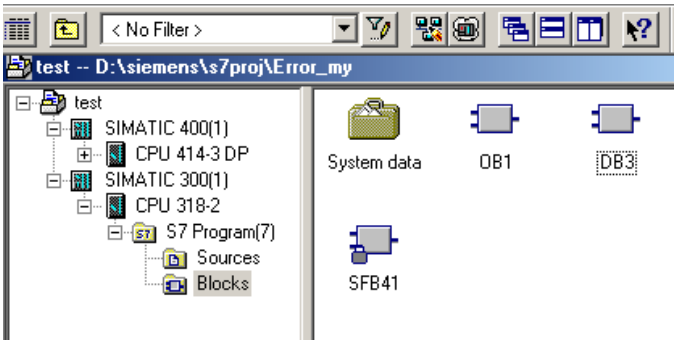
ب

اگر در این پنجره روی + کنار **System Function Blocks** کلیک کنیم لیستی از کل فانکشنهای سیستم مانند شکل **ب** نمایش داده میشود که در آنها نام فانکشن همراه با نام سمبلیک آن آمده است.

برای صدا زدن این فانکشنها کافیست با ماوس آنها را Drag کرده و به Network مورد نظر در برنامه منتقل نماییم. خواهیم دید که دستور Call ظاهر میشود. بدیهی است برای SFB ها باید حتما نام یک DB را نیز در جلوی دستور Call بنویسیم. اگر فانکشن دارای ورودی و خروجی باشد در زیر دستور Callآنها را مشاهده خواهیم کرد مانند شکل زیر :

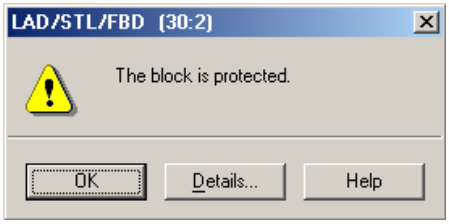
```
Network 1: Title:
  Comment:

     CALL  SFB   41 , DB3
      COM_RST :=
      MAN_ON  :=
      PVPER_ON:=
      P_SEL   :=
      I_SEL   :=
      INT_HOLD:=
      I_ITL_ON:=
      D_SEL   :=
      CYCLE   :=
      SP_INT  :=
      PV_IN   :=
      PV_PER  :=
```

پس از صدا زدن فانکشنهای سیستم اگر به پوشه Blocks در Simatic Manager برگردیم آیکون مربوط به آن بلاک را مشاهده خواهیم کرد .



اگر آیکون را از بالای پنجره Simatic Manager برداشته و روی آیکون بلاک مزبور قرار دهیم توضیحاتی در مورد عملکرد فانکشن و ورودی و خروجی های آن مشاهده خواهیم نمود. توضیح در مورد عملکرد برخی از فانکشنهای سیستم در جلد دوم کتاب و اگر مجالی باشد سایر آنها در آینده ارائه خواهند شد.

با مقایسه شکل آیکون بلاک سیستم با بلاکهای ایجاد شده توسط کاربر میبینیم که علامتی قفل مانند در کنار آن قرار گرفته است. این نشانه Protect بودن بلاک است . اگر روی آیکون بلاک سیستم کلیک کنیم مشاهده خواهیم کرد که پس از اجرای برنامه LAD/STL/FBD پیغام The Block Is Protected مانند شکل زیر ظاهر میشود.



Protect بودن بلاک به این معناست که کاربر نمیتواند برنامه داخل آنرا ببیند. ولی برنامه داخل آن موقع صدا زدن اجرا میشود. میتوان آنرا به برنامه ای کامپیوتری تشبیه کرد که بصورت EXE عرضه شده و Source آن در دسترس نیست.در جلد دوم کتاب با نحوه Protect سایر بلاک ها آشنا خواهیم شد.

همه بلاک های سیستم دارای ورودی و خروجی نیستند برخی از آنها مانند SFC 46 که بصورت سمبلیک STP وقتی صدا زده میشوند هیچ پارامتری در زیر آنها ظاهر نمیشود. بلاک فوق منجر به توقف پردازش CPU میگردد.

```
Network 1: Title:
  Comment:

      CALL  "STP"
```

نحوه ایجاد و استفاده از دیتا بلاک DB

دیتا بلاک نوع Instance که خاص FB است در بحث FB توضیح داده شد. نوع دیگر دیتا بلاک DB اشتراکی یا نوع Shared است که همه بلاک های منطقی به آن دسترسی دارند میتوانند در آن بنویسند یا از آن بخوانند. نحوه نوشتن و خواندن دیتا بلاک در بخش بعدی که دستورات برنامه نویسی آورده شده تشریح شده است . در اینجا صرفا به قدمهایی که برای ایجاد و استفاده از این نوع دیتا بلاک باید برداشت اشاره میشود:

۱- ایجاد دیتا بلاک در پوشه بلاک برنامه Simatic Manager

۲- باز کردن دیتا بلاک با دوبار کلیک کردن روی آن توسط برنامه LAD/STL/FBD

۳- تعریف سطرهای لازم با متغیرهای دلخواه و اسامی سمبلیک دلخواه

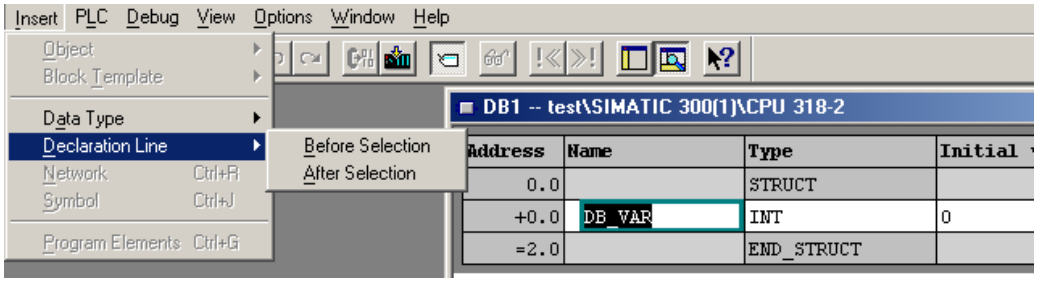
باید توجه داشت که در حالات زیر بدلیل اشکال مربوط به دیتا بلاک PLC در حین اجرا متوقف میشود و چراغ SF روی آن روشن میشود:

الف) دیتا بلاکی که در بلاک منطقی آدرس دهی شده توسط Simatic Manager ایجاد نشده باشد.

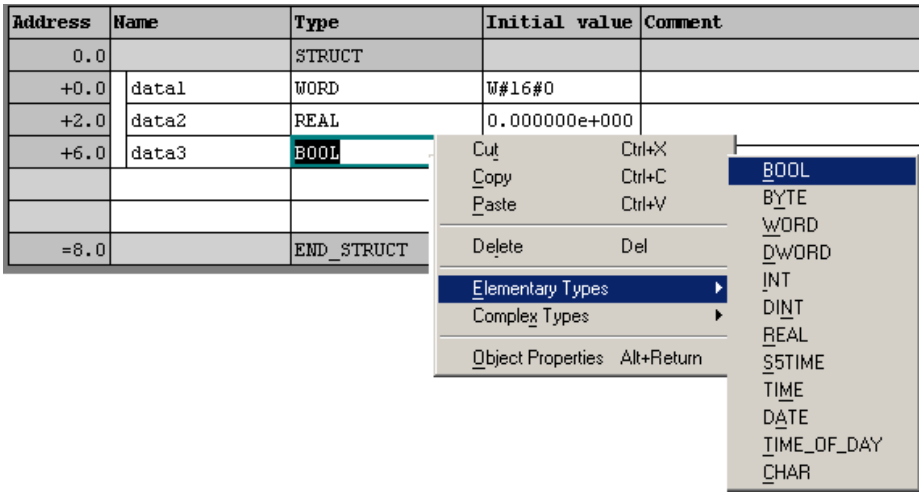
ب) دیتا بلاکی توسط Simatic Manager ایجاد شده ولی به PLC دانلود نشده باشد.

ج) در برنامه به آدرسی در دیتا بلاک اشاره شده که یا وجود ندارد یا نوع دیتای آن با آنچه در برنامه استفاده شده متفاوت باشد.

پس از باز شدن دیتا بلاک در محیط LAD/STL/FBD شکلی شبیه زیر خواهیم داشت که در اینحالت با استفاده ازمنوی **Insert > Declaration Line** میتوان یک سطر جدید بعد یا قبل از سطر جاری ایجاد نمود.



آدرس از صفر شروع میشود و بسته اینکه نوع دیتای که در آن سط تعریف میشود چند بایتی باشد اتوماتیک آدرس مربوطه ایجاد میشود . مثلا اگر در سطر 0 یک دیتا از نوع Word تعریف شود بایتهای 0 و 1 اشغال شده و آدرس بعدی 2 خواهد بود و اگر در آدرس 2 یک دیتا از نوع Real تعریف شود چون ٤ بایتی است آدرس بعدی 6 خواهد بود تعیین نوع دیتا با کلیک راست روی cell مربوطه از ستون Type مطابق شکل زیر است.



دیتا بلاک را میتوان Read Only کرد تا PLC فقط از آن بخواند و نتواند در آن چیزی بنویسد برای اینکار با کلیک راست روی DB در Simatic Manager و در قسمت Peroperties گزینه DB is write Protected را در Part2 فعال میکنیم مانند شکل زیر

در اینحالت اگر به PLC فرمان نوشتن در DB داده شود متوقف شده و چراغ SF روشن میشود.

نحوه ایجاد و استفاده از UDT

UDT یا User-defined Data Type نیز ازجمله بلاکهایی است که میتوان آنرا در پوشه Blocks برنامه Simatic Manager ایجاد کرد . UDT ها حاوی دیتاهایی هستند که یکبار تعریف شده ولی به مراتب میتوانند مورد استفاده قرار گیرند برای روشن شدن موضوع به ذکر مثالی میپردازیم. فرض کنید ۱۰ تجهیز مشابه داریم که سیگنالهای یکسانی بعنوان ورودی و خروجی دارند(مانند سیگنالهای Run, Stop,Error, Min,Max) برای معرفی این سیگنالها در دیتابلاک یک روش آنست که آنها را برای تک تک تجهیزات زیر هم لیست کنیم در اینصورت دیتابلاک دارای ۵۰ سطر خواهد بود و نوشتن آنها وقت گیر است. روش ساده تر استفاده از UDT است. کافی است در پوشه بلاکها یک UDT ایجاد کرده سپس۵ سیگنال فوق را در آن وارد کنیم . پس از آن دیتا بلاک را باز کرده و در آن صرفا ۱۰ تجهیز را معرفی میکنیم و آنها را به UDT لینک مینماییم. جدول UDT و دیتا بلاک مانند شکلهای زیر خواهند بود.

UDT2 -- PLc-9-81\SIMATIC 400(1)\CPU 414-2 DP

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Run	BOOL	FALSE	
+0.1	Error	BOOL	FALSE	
+0.2	Min	BOOL	FALSE	
+0.3	Max	BOOL	FALSE	
+2.0	Speed	INT	0	
=4.0		END_STRUCT		

همانطور که ملاحظه میشود دیتا بلاک بطور خلاصه نمایش داده میشود اگر بخواهیم کل دیتاها را ببینیم بدون اینکه نیازی به وارد کردن دیتای جدیدی باشد با استفاده از منوی **View> Data View** در برنامه LAD\STL\FBD جدول کامل پنجاه تایی را خواهیم دید.

DB15 -- PLc-9-81\SIMATIC 400(1)\CPU 414-2 DP

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Equipment1	UDT2		
+4.0	Equipment2	UDT2		
+8.0	Equipment3	UDT2		
+12.0	Equipment4	UDT2		
+16.0	Equipment5	UDT2		
+20.0	Equipment6	UDT2		
+24.0	Equipment7	UDT2		
+28.0	Equipment8	UDT2		
+32.0	Equipment9	UDT2		
+36.0	Equipment10	UDT2		
=40.0		END_STRUCT		

باید توجه داشت که آدرسهای دیتا بلاک متناسب با اطلاعات موجود در UDT پرش میکنند حال اگر دیتاهای UDT را کم یا زیاد کنیم این آدرسها عوض میشوند برای اینکار باید از منوی **File> Check and Update Access** آنرا باز سازی نماییم.

تذکر

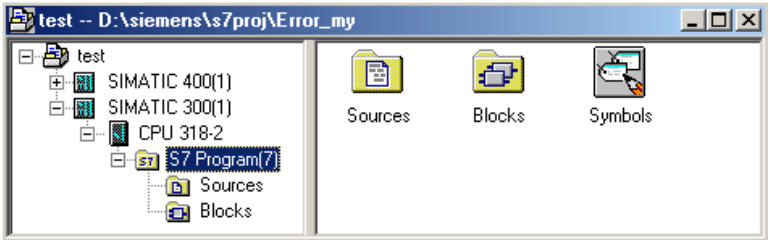
با بحثی که در مورد بلاکها انجام شد صرفا یک نوع بلاک دیگر باقی می ماند که VAT یا Varriable Table نام دارد . با توجه به کاربرد خاص این بلاک توضیحات مربوط به آنها در قسمتی جداگانه ذکر خواهد شد.

٤-۹ نحوه ایجاد و استفاده از سمبل ها

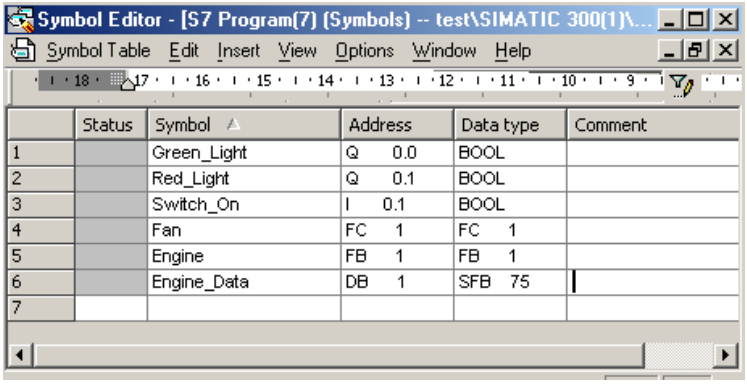
در برنامه STEP 7 با آدرسهای مربوط به سیگنالهای ورودی خروجی ، کانترها ، تایمرها ، دیتا بلاکها و .... سرو کار داریم که میتوانند آدرس مطلق یا سمبلیک داشته باشند. Q 4.0 , I 1.1 و C1 و T5 نمونه هایی از آدرس دهی مطلق هستند . اگر بجای آدرس های فوق از آدرس های سمبلیک معنی دار استفاده شود ، خواندن برنامه و عیب یابی آن ساده تر خواهد شد مثلاً بجای Q4.0 از سمبل Motor_ON استفاده کرده و بعد از آن این کلمه را در برنامه برای آدرس بکار ببریم.

سمبل ها میتوانند بصورت محلی یا اشتراکی تعریف شوند. سمبل های اشتراکی در کل برنامه و کل بلاکها قابل شناسایی و استفاده هستند ولی سمبل های محلی صرفاً در همان بلاکی که تعریف شده اند میتوانند شناسایی شوند اسامی که برای ورودی و خروجی و سایر متغیرها در بخش Declaration یک بلاک بکار میرود سمبل های محلی هستند. بدیهی است در بلاکهای مختلف میتوان سمبل های محلی متفاوت ولی با نام یکسان بکار برد ولی نام سمبل های اشتراکی در کل برنامه باید منحصر بفرد باشد.

برای ایجاد سمبل های اشتراکی لازم است از برنامه Symbol Editor استفاده کنیم که آیکون آن وقتی در Simatic Manager روی S7 Program کلیک کنیم آن مطابق شکل زیر در پنجره سمت راست ظاهر میشود.

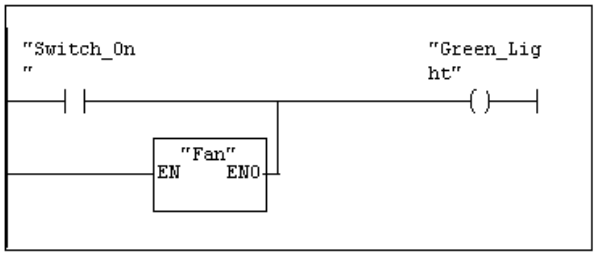


با کلیک کردن روی آن برنامه Symbol Editor اجرا شده و جدول خالی سمبلها نشان داده میشود . کاربر میتواند با وارد کردن سمبلها این جدول را مانند شکل زیر کامل نماید. سمبلهایی که در این جدول وارد میشوند بصورت اشتراکی بوده و در کل برنامه قابل استفاده هستند. همانطور که در شکل مشاهده میشود حتی برای نام بلاک ها نیز میتوان سمبل تعریف کرد



	Status	Symbol	Address	Data type	Comment
1		Green_Light	Q 0.0	BOOL	
2		Red_Light	Q 0.1	BOOL	
3		Switch_On	I 0.1	BOOL	
4		Fan	FC 1	FC 1	
5		Engine	FB 1	FB 1	
6		Engine_Data	DB 1	SFB 75	
7					

شکل زیر برنامه ای را در محیط LAD نشان میدهد که با توجه به سمبل های جدول صفحه قبل نوشته شده است. در هنگام نوشتن برنامه حتی اگر آدرسهای مطلق را بکار ببریم پس از Enter کردن خودبخود به شکل سمبلی که برای آن تعریف شده ظاهر میشود . در این حالت چنانچه بخواهیم برنامه را با آدرسهای مطلق مشاهده کنیم باید در LAD/STL/FBD از منوی **View > Display With** استفاده نماییم و **Symbolic Representation** را غیر فعال نماییم.



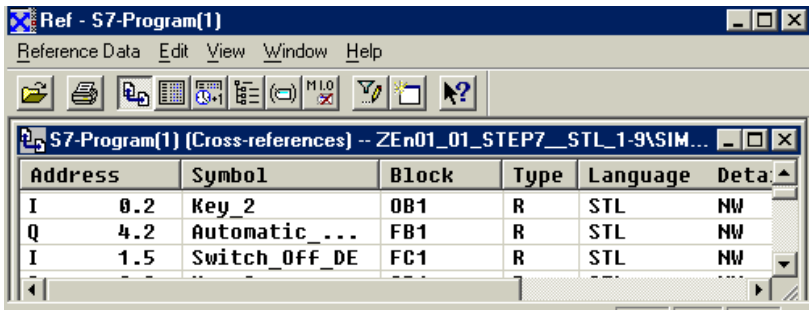
در انتهای بحث سمبل ها نکات زیر قابل ذکر است :

- نام سمبل حداکثر ۲٤ کاراکتر و تعداد سمبل ها حداکثر ۱٦۳۸۰ میباشد. حروف کوچک و بزرگ یکسان تلقی میشوند.
- در S7 میتوان جدول سمبل هاکه توسط یک ادیتور بیرونی ایجاد شده (از جمله توسط Step 5) را بداخل برنامه Import نمود. برای اینکار از منوی **Symbol > Import** در برنامه Symbol Editor استفاده میشود.
- در موقع دانلود کردن برنامه به PLC سمبلها دانلود نمیشوند و PLC فقط با آدرسهای مطلق کار میکند . بنابراین اگر برنامه ای را از PLC به کامپیوتر منتقل کنیم (Upload) در اینصورت برنامه بدون سمبل مشاهده خواهد شد مگر اینکه فایل سمبلها از قبل v روی کامپیوتر وجود داشته باشد.

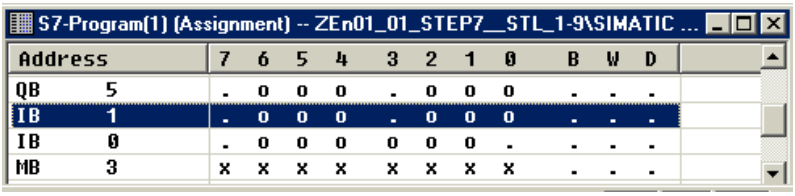
٤-۱۰ نحوه استفاده از Reference Data

Reference Data ابزاری است که علاوه بر ارائه دید کلی نسبت به برنامه ، جزئیات لازم را درمورد آدرسها ، سمبلها ، بلاکها و غیره مشخص مینماید. برای فعال کردن آن در Simatic Manager وقتی که پوشه Blocks باز است از منوی Option > Reference Data>Display استفاده میکنیم. پس از آن برنامه دیگری به نام Ref باز میشود که دارای امکانات زیر است :

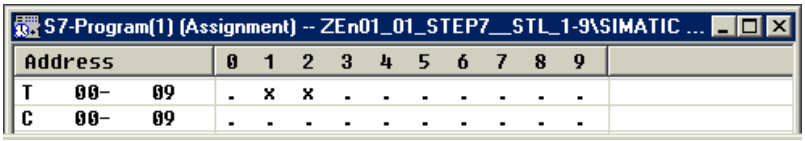
۱- نمایش لیست کلیه آدرسهای برنامه (I , Q , M ,T ,C) با نام سمبلیک ، نوع و نیز نام بلاکی که این آدرس در آن استفاده شده است. اگر روی آدرس مورد نظر کلیک راست ماوس را بزنیم با استفاده از Go to location بلاک مربوطه باز شده و محلی که این آدرس بکار رفته نمایش داده میشود.



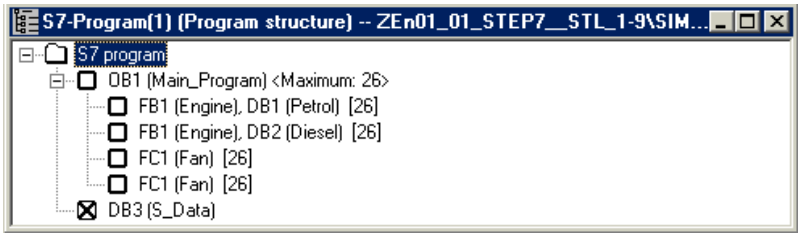
۲-نمایش آدرسهای ورودی ، خروجی بکاررفته و اینکه چه بیت هایی استفاده نشده اند علامت – برای آدرسهایی که استفاده نشده اند، O برای آدرسهایی که مستقیم و X برای آدرس هایی که غیر مستقیم بکار رفته اند(byte, word)



۳-نمایش تایمرها و کانترهای بکار رفته و اینکه بین آنها کدام استفاده نشده است.



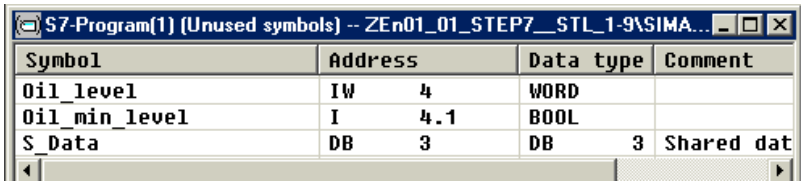
٤-نمایش ساختار برنامه و اینکه چه بلاکهایی از داخل بلاکهای دیگر Call شده اند.



در کنار نام بلاکها علائم مختلفی ممکن است استفاده شود راهنمای برخی از این علائم در جدول زیر آمده است:

Example	Meaning	Symbole
CALL FB10	Block called normally	
UC FB10	Block called unconditionally	
CC FB10	Block called conditionally	
-	Data block	
-	Block not called	

۵- نمایش سمبلهایی که تعریف شده ولی استفاده نشده اند .

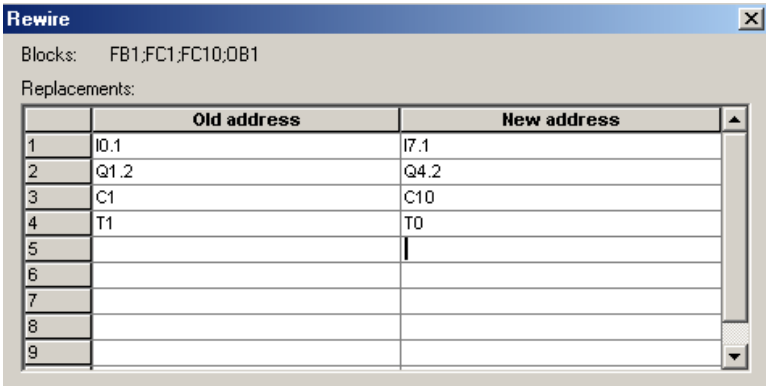


٦- نمایش آدرسهایی که فاقد سمبل هستند.

٤-١١ نحوه استفاده از Rewiring

فرض کنید برنامه بزرگی مشتمل بر چندیدن بلاک و هر بلاک چندین سطر نوشته شده و نیاز باشد که آدرس یا آدرسهایی را در همه بلاک ها عوض کنیم. قطعا انجام اینکار بصورت دستی دشوار خواهد بود. با امکان Rewiring که در برنامه Simatic Manager منوی Option وجود دارد این کار بسادگی انجام پذیر است.

پس از کلیک روی Rewiring پنجره ای مانند شکل زیر باز میشود که در آن آدرسهای قدیم و جدید را وارد کرده و OK را انتخاب میکنیم



عمل Rewiring بویژه در مواردی که کاربر ابتدا برنامه را نوشته باشد و سپس سخت افزار را پیکر بندی کرده باشد مورد نیاز خواهد بود . زیرا باید آدرس ها را مطابق آنچه در سخت افزار معرفی شده تغییر داد.

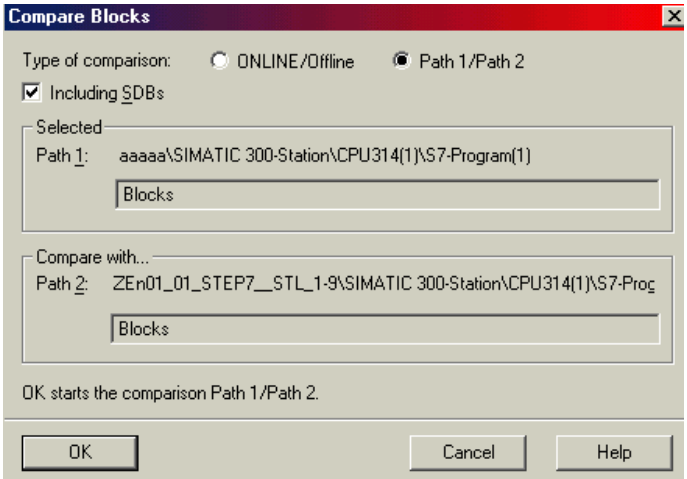
٤-١٢ مقایسه بلاکها Compare Blocks

میتوان یک یا چند بلاک را ازیک پروژه با پروژه دیگر مقایسه کرد بعنوان مثال میتوان برنامه PLC که در حال کار است را با برنامه ای که روی PC یا PG ذخیره شده باهم مقایسه و تفاوتهای آنها را شناسایی کرد.

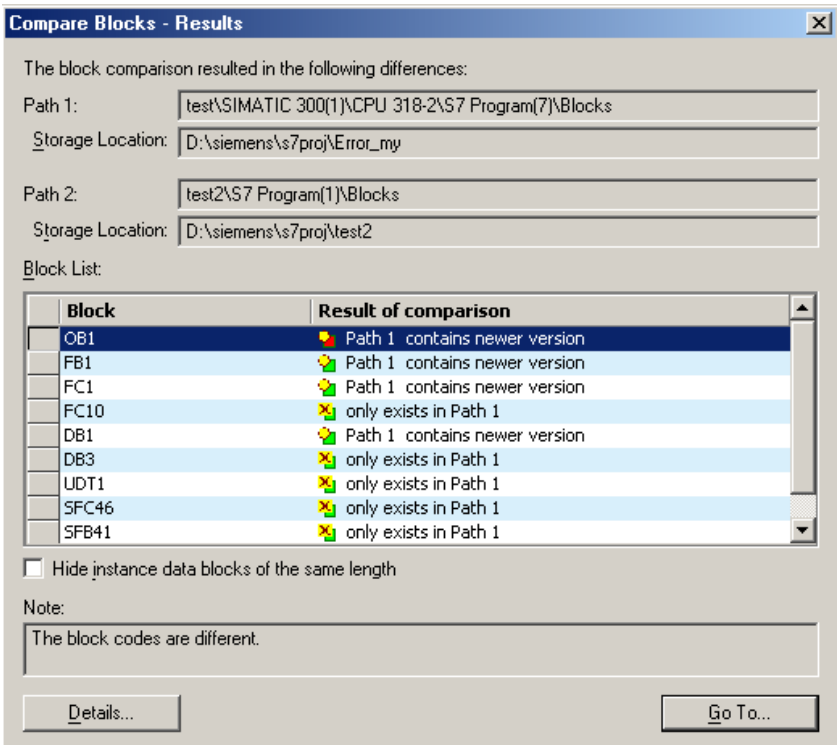
بلاکها در ۲ حالت زیر میتوانند مورد مقایسه قرار گیرند:

- یکی On Line و دیگری Off Line
- هردو Offline

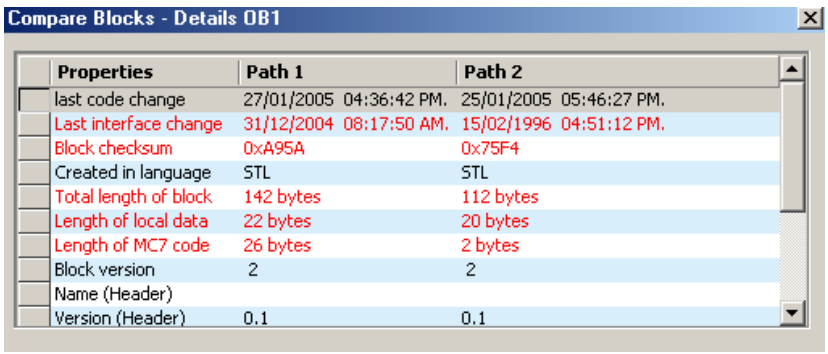
در Simatic Manager با استفاده از منوی Option > Compare Blocks پنجره زیر را خواهیم داشت:



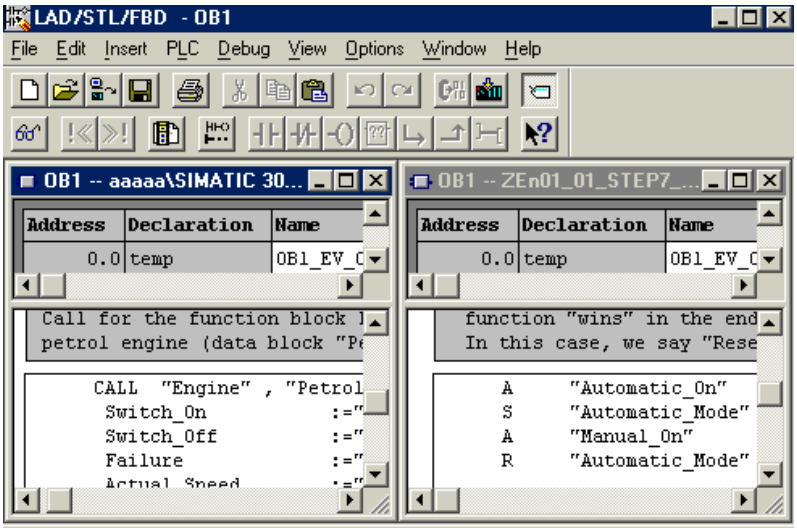
اگر بخواهیم کل بلاکهای دو پروژه را مقایسه کنیم ابتدا روی پوشه Blocks اولی کلیک کرده سپس روی Path2 رفته وهمزمان از پروژه دوم که آنرا در Simatic Manager باز کرده ایم پوشه Blocks را انتخاب مینماییم . این کار را میتوان برای دو بلاک همنوع از یک یا دو پروژه نیز انجام داد. پس از انتخاب روی OK کلیک کرده تا مقایسه صورت گیرد نتیجه مقایسه برای تک تک بلاکها لیست میشود



با کلیک کردن روی دکمه Detail باکسی مانند شکل زیر مشاهده خواهد شد که در آن موارد اختلاف با رنگ قرمز مشخص شده اند.



حال اگر روی Go To کلیک کنیم برنامه LAD/STL/FBD باز شده و در دو پنجره کنار هم از سطری که اختلاف شروع شده نمایش داده میشود.



۵– دستورات برنامه نویسی S7

مشتمل بر :

در این بخش دستورات برنامه نویسی به تفصیل بیان شده اند. قبل از شروع خواننده محترم لازم است به نکات زیر توجه داشته باشد:

۱. از آنجا که دستورات برنامه نویسی به زبان STL کامل تر از دو زبان LAD و FBD است ، STL بعنوان مبنا انتخاب شده است . با این وجود اگر بلاک های خاصی در LAD و FBD موجود بوده اند در انتهای هر بخش به آنها نیز اشاره شده است.

۲. برای اکثر دستورات معادل LAD و FBD نیز بیان شده است مگر در مواردی که بدلیل تشابه ارائه یکی از آنها کافی بنظر رسیده است.

۳. دستورات بسته به عملکرد در ۱٤ گروه طبق فهرست صفحه قبل دسته بندی و ارائه شده اند.

٤. برای هر یک از دستورات جدول زیر که نشان دهنده بیتهای Status Word است ترسیم شده است

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:									

در سطر دوم این جدول منظور از Writes نتیجه ای است که CPU در بیتهای مورد نظر مینویسید . علامتهایی که در این سطر زیر هر بیت بکار میرود یکی از علامات زیر خواهد بود :

علامت	شرح
0	اجرای دستور مقدار بیت مورد نظر را 0 میکند
1	اجرای دستور مقدار بیت مورد نظر را 1 میکند
x	اجرای دستور مقدار بیت مورد نظر را 0 یا 1میکند
-	اجرای دستور تاثیری روی بیت نمیگذارد (غیر مربوط)

۵-۱ دستورات Bit Logic

این دستورات همانطور که از نامشان پیداست عملیاتی را روی یک بیت انجام میدهند. دستوراتی که در این بخش تشریح میگردند عبارتند از :

- **A And**
- **AN And Not**
- **O Or**
- **ON Or Not**
- **X Exclusive Or**
- **XN Exclusive Or Not**
- **O And before Or**
- **A(And with Nesting Open**
- **AN(And Not with Nesting Open**
- **O(Or with Nesting Open**
- **ON(Or Not with Nesting Open**
- **X(Exclusive Or with Nesting Open**
- **XN(Exclusive Or Not with Nesting Open**
- **) Nesting Closed**
- **= Assign**
- **R Reset**
- **S Set**
- **NOT Negate RLO**
- **SET Set RLO (=1)**
- **CLR Clear RLO (=0)**
- **SAVE Save RLO in BR Register**
- **FN Edge Negative**
- **FP Edge Positive**

بیت آدرس داده شده میتواند بصورت زیر باشد:

Address	Data Type	Memory Area
<Bit>	Bool	I,Q,M,L,D

دستور STL : **A And**

فرمت: **A <Bit>**

شرح : دستور A چک میکند که آیا وضعیت بیت آدرس داده شده "1" است یا نه و سپس RLO را باوضعیت این بیت AND میکند.

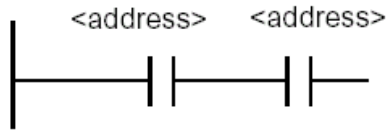
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

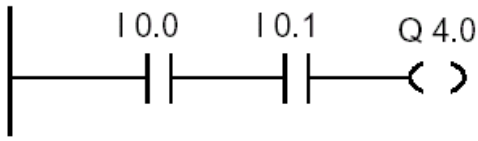
مـثال : در مثال روبرو ورودی I0.0 و I0.1 با یکدیگر And شده و نتیجه به خروجی Q 4.0 ارسال میشود. بدیهی است خروجی وقتی "1" خواهد شد.که هر دو ورودی "1" باشند.

```
A   I 0.0
A   I 0.1
=   Q 4.0
```

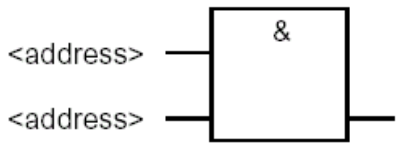
معادل LAD



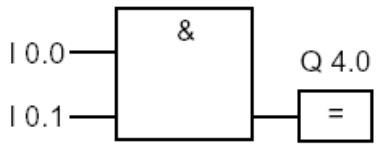
مثال



معادل FBD



مثال



دستور STL : **AN And Not**

فرمت: **AN <Bit>**

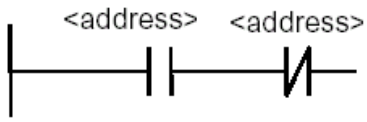
دستور AN چک میکند که آیا وضعیت بیت آدرس داده شده "0" است یا نه و سپس RLO را باوضعیت این بیت AND میکند.

وضعیت Status Word: مشابه دستور And

مثال : در مثال روبرو ورودی I0.0 و Not I0.1 با یکدیگر And شده و نتیجه به خروجی Q4.0 ارسال میشود. بدیهی است خروجی وقتی "1" خواهد شد.که ورودی اول "1" و ورودی دوم "0" باشد.

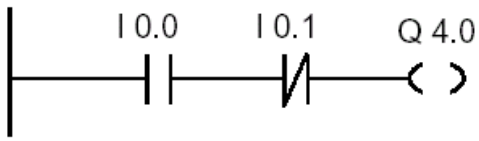
```
A   I 0.0
AN  I 0.1
=   Q 4.0
```

معادل LAD

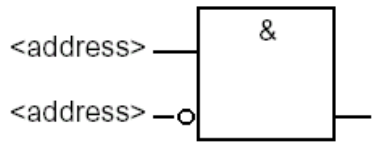


کنتاکت سمت چپ را **Normal Open** و کنتاکت سمت راست را **Normal Closed** گویند

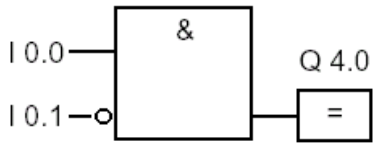
مثال



معادل FBD



مثال



دستور STL : **O Or**

فرمت: **O <Bit>**

شرح : دستور O چک میکند که آیا وضعیت بیت آدرس داده شده "1" است یا نه و سپس RLO را باوضعیت این بیت Or میکند.

وضعیت Status Word:

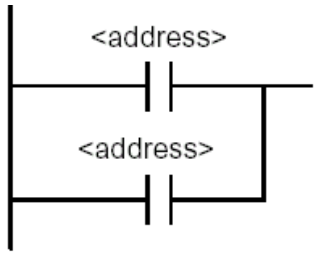
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	x	x	1

مثال :

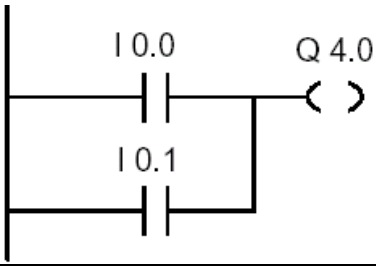
در مثال روبرو ورودی I0.0 و I0.1 با یکدیگر OR شده و نتیجه به خروجی Q4.0 ارسال میشود.بدیهی است هرکدام از این دو ورودی که "1" باشد خروجی "1" خواهد شد.

```
O   I 0.0
O   I 0.1
=   Q 4.0
```

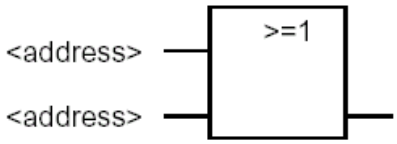
معادل LAD



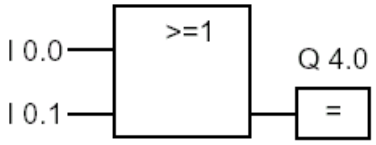
مثال



معادل FBD



مثال



دستور STL : **ON Or Not**

فرمت: **ON <Bit>**

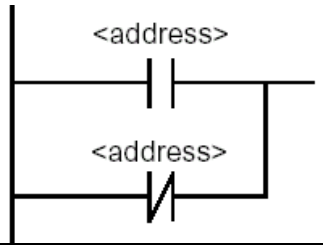
دستور ON چک میکند که آیا وضعیت بیت آدرس داده شده "0" است یا نه و سپس RLO را باوضعیت این بیت Or میکند.

وضعیت Status Word: مشابه دستور or

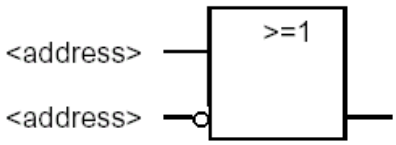
در مثال روبرو ورودی I0.0 و Not I0.1 با یکدیگر OR شده و نتیجه به خروجی Q4.0 ارسال میشود.بدیهی است اگر ورودی اول "1" یا ورودی دوم "0" باشد خروجی "1" خواهد شد.

```
O    I 0.0
ON   I 0.1
=    Q 4.0
```

معادل LAD



معادل FBD



دستور STL: X Exclusive Or

فرمت: X <Bit>

شرح : دستور x چک میکند که آیا وضعیت بیت آدرس داده شده "1" است یا نه و سپس RLO را باوضعیت این بیت XOR میکند. در واقع اگر یکی از این دو (RLO یا بیت مورد نظر) دارای وضعیت "1" باشند نتیجه "1" خواهد بود

وضعیت Status Word: مشابه دستور or

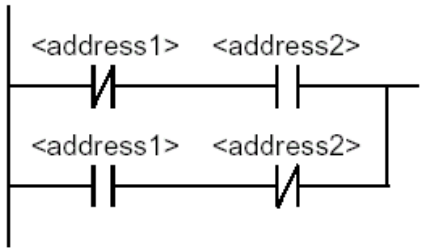
مثال :

```
X    I 0.0
X    I 0.1
=    Q 4.0
```

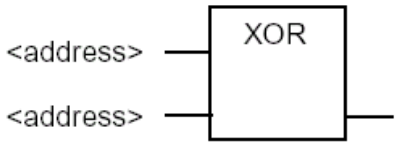
I 0.0	I 0.1	Q 4.0
0	0	0
0	1	1
1	0	1
1	1	0

با توجه به حالات مختلف ورودیها که در جدول آمده است خروجی "1" خواهد بود اگر فقط یکی از ورودیها "1" شود.در عمل یعنی اینکه خروجی فقط به یک کلید حساس است اگر دو کلید همزمان فشار داده شوند خروجی فعال نمیشود.

معادل LAD با ترکیب المانها ساخته میشود



معادل FBD



دستور STL: XN Exclusive Or Not

فرمت: XN <Bit>

دستور XN چک میکند که آیا وضعیت بیت آدرس داده شده "0" است یا نه و سپس RLO را باوضعیت این بیت XOR میکند. در واقع اگر یکی هر دو (RLO یا بیت مورد نظر) دارای وضعیت "1" یا هر دو دارای وضعیت "0" باشند نتیجه "1" خواهد بود.

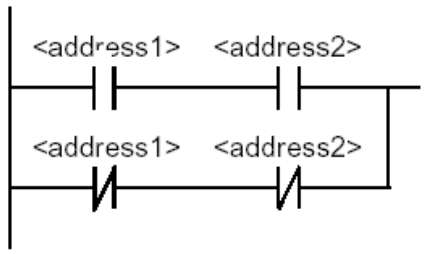
وضعیت Status Word: مشابه دستور or

```
X    I 0.0
XN   I 0.1
=    Q 4.0
```

I 0.0	I 0.1	Q 4.0
0	0	1
0	1	0
1	0	0
1	1	1

با توجه به حالات مختلف ورودیها که در جدول آمده است خروجی "1" خواهد بود اگر هر دو ورودی "1" یا هر ۲ ورودی "0" شود.

معادل LAD با ترکیب المانها ساخته میشود



معادل FBD

ندارد

دستور STL: A(And with Nesting Open

فرمت:

A(<Bit>

شرح:

دستور)A مقدار RLO و بیت OR (از بیتهای Status Word) را در Nesting Stack ذخیره میکند. ماکزیمم ۷ بار متوالی میتوان این مقادیر را در Nesting Stack ذخیره کرد یعنی ماکزیمم ۷ پرانتز تو در تو میتوان باز نمود.

وضعیت Status Word

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

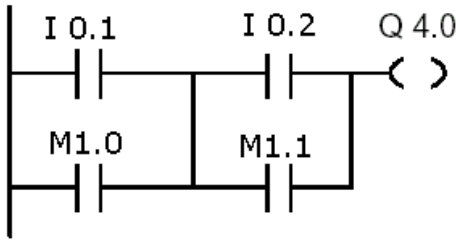
مثال:

در مثال روبرو ابتدا عملیات داخل پرانتز اول انجام میشود یعنی ورودی I0.0 و فلاگ M1.0 با هم OR میشوند. RLO منتجه به داخل Nesting Stack وارد میشود. سپس عملیات داخل پرانتز دوم انجام میشود و نهایتاً از AND شدن Nesting Stack و RLO جدید مقدار نهایی RLO ایجاد شده و به خروجی Q4.0 ارسال میشود.

```
A(
O    I 0.1
O    M1.0
)
A(
O    I 0.2
O    M1.1
)
=    Q 4.0
```

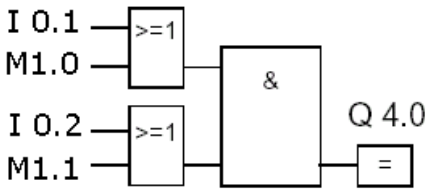
معادل LAD (مثال)

با ترکیب المانها ساخته میشود



معادل FBD (مثال)

با ترکیب المانها ساخته میشود



دستور STL: **AN(And Not with Nesting Open**

فرمت:

A(<Bit>

شرح:

این دستورمشابه)A است ولی فانکشن آن And Not میباشد.

دستور STL: **O(Or with Nesting Open**

فرمت:

O(<Bit>

شرح:

این دستورمشابه)A است ولی فانکشن آن Or میباشد.

دستور STL: **ON(Or Not with Nesting Open**

فرمت:

ON(<Bit>

شرح:

این دستورمشابه)O است ولی فانکشن آن Or Not میباشد.

دستور STL: **X(Exclusive Or with Nesting Open**

فرمت:

X(<Bit>

شرح:

این دستورمشابه)O است ولی فانکشن آن XOR میباشد.

دستور STL: **XN(Exclusive Or Not with Nesting Open**

فرمت:

X(<Bit>

شرح:

این دستورمشابه)X است ولی فانکشن آن XOR Not میباشد.

دستور STL: **) Nesting Close**

فرمت:

)

شرح:

دستور **(** مقادیر ذخیره شده در Nesting Stack را برداشته و به RLO و بیت OR (از بیتهای Status Word) انتقال میدهد بدیهی است اگر RLO قبلاً مقدار داشته طبق دستور نوشته شده با مقدار برداشته شده از Nesting Stack ترکیب میشود مثلاً And یا Or میشود در برنامه اگر از دستورات زیر که منجر به باز شدن Stack میشوند استفاده شده باشد. باید در انتها دستور بستن Stack یعنی) بکار رود.لازم است به تعداد پرانتزهای باز شده ، پرانتز بسته شده بکار ببریم.

A(
AN(
O(
ON(
X(
XN(

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	1	x	1

مثال:

در مثال روبرو ابتدا عملیات داخل پرانتز اول انجام میشود یعنی ورودی I0.0 و فلاگ M1.0 با هم OR میشوند . RLO منتجه به داخل Nesting Stack وارد میشود. سپس عملیات داخل پرانتز دوم انجام میشود و نهایتاً از AND شدن Nesting Stack و RLO جدید مقدار نهایی RLO ایجاد شده و به خروجی Q4.0 ارسال میشود.

```
A(
O     I 0.1
O     M1.0
)
A(
O     I 0.2
O     M1.1
)
=     Q 4.0
```

معادل LAD

مشابه دستور **A(**

معادل FBD

مشابه دستور **A(**

دستور STL: **= Assign**

فرمت: **= <Bit>**

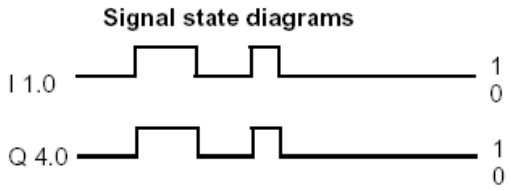
شرح: دستور = مقدار RLO را به بیت آدرس داده شده خروجی میفرستد اگر در برنامه از دستورات Master Control Relay که بعداً شرح داده خواهد شد استفاده شود و MCR=0 باشد دراینصورت با دستور = مقدار "0" به خروجی ارسال میشود و ربطی به مقدار RLO نخواهد داشت.

وضعیت Status Word:

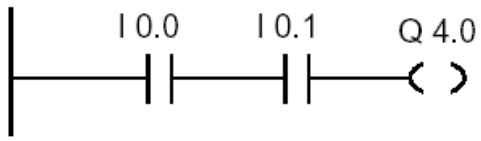
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	x	_	0

مثال: در مثال زیر ورودی I0.0 و I0.1 با یکدیگر And شده و نتیجه به خروجی Q4.0 ارسال میشود. بدیهی است خروجی وقتی "1" خواهد شد.که هر دو ورودی "1" باشند.

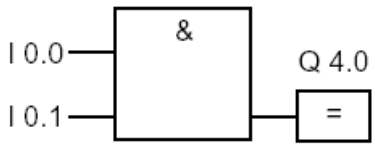
```
A   I 0.0
A   I 0.1
=   Q 4.0
```



معادل LAD (مثال)



معادل FBD (مثال)



دستور STL: **S Set**

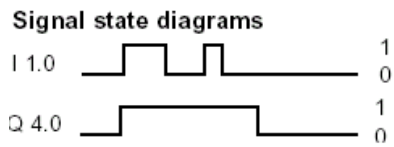
فرمت: **S <Bit>**

دستور S در صورتیکه مقدار RLO=1 باشد دراینصورت بیت خروجی آدرس داده شده را "1" میکند . اگر از دستورات Master Control Relay که بعداً شرح داده خواهد شد استفاده شود و MCR=0 باشد دراینصورت با دستور S مقدار خروجی تغییر نخواهد کرد.تفاوت دستور S و دستور = در اینست که دستور S تحت شرایط فوق خروجی را "1" میکند ولی دستور = عیناً مقدار RLO را به خروجی میفرستد بنابراین خروجی میتواند "0" یا "1" شود.دستور Sمیتواند حالت خود نگهدار را بدون نیاز به کنتاکت موازی ایجاد کند.

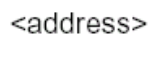
وضعیت Status Word: مشابه دستور Assign

در مثال زیر وقتی ورودی I0.0 یک باشد خروجی Q4.0 یک میشود و پس از آن 0 و 1 شدن I0.0 تاثیری روی خروجی ندارد.ولی اگر در ابتدا ورودی I0.0 یک نباشد خروجی Q4.0 تغییر نمیکند یعنی حالت قبلی خود را حفظ مینماید..

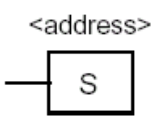
```
A   I 0.0
S   Q 4.0
```



معادل LAD



معادل FBD



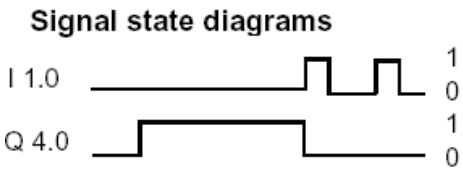
دستور STL: **R Reset**

فرمت: **R <Bit>**

شرح دستور R در صورتیکه مقدار RLO=1 باشد دراینصورت بیت خروجی آدرس داده شده را "0" میکند . اگر از دستورات Master Control Relay که بعداً شرح داده خواهد شد استفاده شود و MCR=0 باشد دراینصورت با دستور R مقدار خروجی تغییر نخواهد کرد.

وضعیت Status Word: مشابه دستور Assign

مثال : در مثال روبرو وقتی ورودی I0.0 یک باشد خروجی Q4.0 صفر میشود و پس از آن 0 و 1 شدن I0.0 تاثیری روی خروجی ندارد.ولی اگر در ابتدا ورودی I0.0 یک نباشد خروجی Q4.0 تغییر نمیکند یعنی حالت قبلی خود را حفظ مینماید..

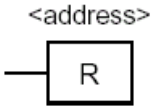



```
A    I 0.0
R    Q 4.0
```

معادل LAD

<address>
---(R)

معادل FBD



دستور STL: **NOT Negate RLO**

فرمت: **NOT**

دستور NOT مقدار RLO را عکس میکند یعنی اگر "0" است آنرا "1" و اگر "1" است آنرا "0" مینماید

وضعیت Status Word:

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	_	1	x	-

در مثال روبرو وقتی ورودی I0.0 یک باشد خروجی Q4.0 صفر میشود چون دستور NOT آنرا عکس میکند. همینطور اگر I0.0 صفر باشد خروجی یک خواهد بود.

```
A    I 0.0
NOT
=    Q 4.0
```

معادل LAD

---|NOT|---

معادل FBD

SET Set RLO (=1)

دستور STL:

فرمت:

SET

دستور SET مقدار RLO را "1" میکند بدون توجه به اینکه وضعیت قبلی آن چه بوده است.

وضعیت Status Word: مشابه دستور NOT

در مثال روبرو با دستور SET مقدار RLO یک شده و فلگ ها یک خواهند شد.

```
SET
=     M10.0
=     M15.1
```

STL Program	Signal State	Result of Logic Operation (RLO)
SET		1
= M 10.0	1	
= M 15.1	1	
= M 16.0	1	

معادل LAD

ندارد

معادل FBD

ندارد

CLR Clear RLO (=0)

دستور STL:

فرمت:

CLR

دستور CLR مقدار RLO را "0" میکند بدون توجه به اینکه وضعیت قبلی آن چه بوده است.

وضعیت Status Word:

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	0	0	0

در مثال روبرو با دستور CLR مقدار RLO صفر شده و فلگ ها صفر خواهند شد.

```
CLR
=     M10.1
=     M10.2
```

STL Program	Signal State	Result of Logic Operation (RLO)
CLR		0
= M 10.1	0	
= M 10.2	0	

معادل LAD

ندارد

معادل FBD

ندارد

دستور STL: **SAVE Save RLO in BR Register**

فرمت:

SAVE

شرح:

دستور SAVE مقدار RLO را در بیت BR (از بیت های Status Word) ذخیره میکند. در این شرایط بیت First Check یعنی FC/ ری ست نمیشود. از اینرو میتوان بیت BR را در Network جدید برنامه با دستور AND استفاده نمود.

یکی از کاربردهای دستور SAVE در بلاکهایی است که از برنامه اصلی فراخوان میشوند. اگر در انتهای این بلاکها و قبل از خروج از آنها دستور SAVE را بکار ببریم RLO در BR ذخیره شده و با بازگشت به برنامه اصلی اگر چه RLO تغییر میکند ولی BR همان مقدار قبلی را دارد. بدین طریق میتوان از اجرا شدن یا نشدن بلاک مطمئن شد بعبارت دیگر این روش برای عیب یابی بلاکها بکار میرود.

وضعیت **Status Word**

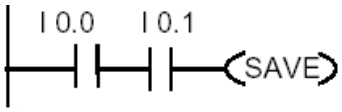
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	x	-	-	-	-	-	-	-	-

مثال:

```
A     I 0.0
A     I0.1
SAVE
```

معادل LAD

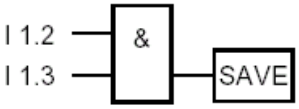
مثال



---(SAVE)

معادل FBD

مثال



FN Edge Negative

دستور STL :

فرمت:

FN <Bit>

Address	Data Type	Memory Area
<Bit>	Bool	I,Q,M,L,D

شرح :

دستور FN لبه پایین رونده RLO را آشکار میکند بعبارت دیگر وقتی RLO از "1" به "0" میرود این دستور آنرا تشخیص داده و نتیجه را با 1=RLO آشکار می سازد.

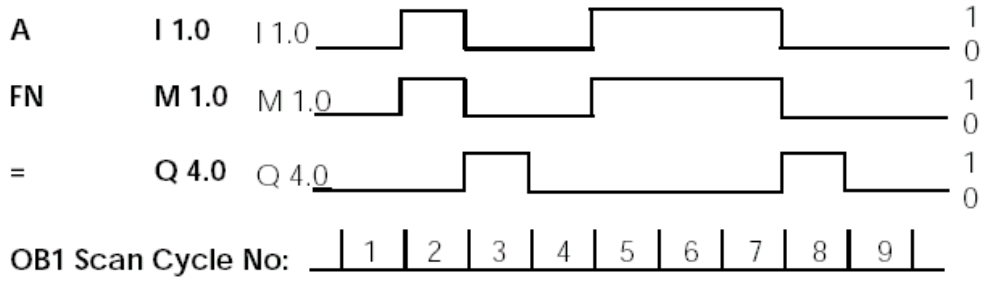
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	x	x	1

مثال :

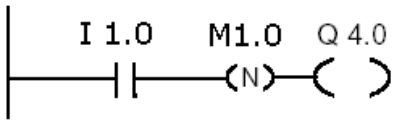
A	**I 1.0**
FN	**M1.0**
=	**Q 4.0**

در مثال روبرو به محض اینکه در سیکل اسکن جدید ورودی I1.0 از "1" به "0" میرود دستور FN آنرا تشخیص داده و خروجی Q4.0 یک میشود.بدیهی است وضعیت ورودی در سیکل قبلی در فلگ M1.0 ذخیره شده است.



معادل LAD

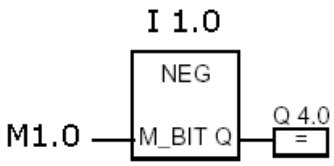
مثال



<address>
---(N)

معادل FBD

مثال



<address1>
NEG
M_BIT Q

FP Edge Positive

دستور STL :

فرمت:

FN <Bit>

Address	Data Type	Memory Area
<Bit>	Bool	I,Q,M,L,D

شرح :

دستور FP لبه بالا رونده RLO را آشکار میکند بعبارت دیگر وقتی RLO از "0" به "1" میرود این دستور آنرا تشخیص داده و نتیجه را با RLO=1 آشکار می سازد.

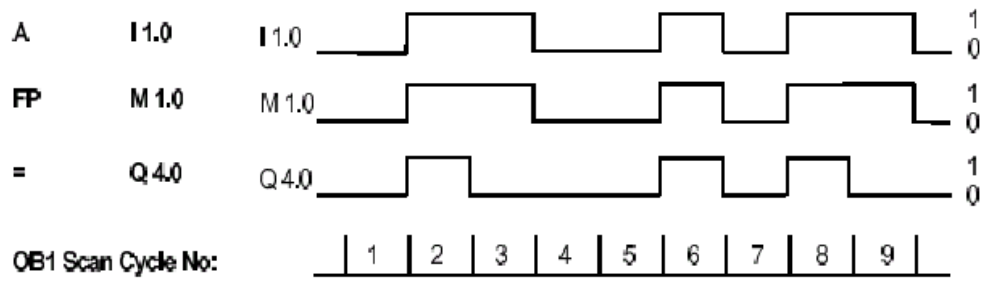
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	x	x	1

مثال :

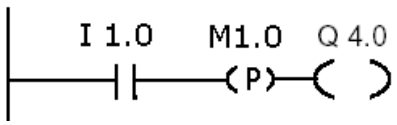
در مثال روبرو به محض اینکه در سیکل اسکن جدید ورودی I1.0 از "0" به "1" میرود دستور FP آنرا تشخیص داده و خروجی Q4.0 یک میشود. بدیهی است وضعیت ورودی در سیکل قبلی در فلگ M1.0 ذخیره شده است.

```
A     I 1.0
FN    M1.0
=     Q 4.0
```



معادل LAD

مثال

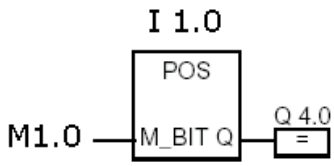


<address>

---(P)---

معادل FBD

مثال



<address1>

POS

M_BIT Q

علاوه بر آنچه در این بخش ذکر گردید در روش LAD و FBD المانهای دیگری نیز وجو دارند که معادل STL آنها چند دستور است معادل STL این موارد همان دستورات قبلی است ازاینرو نیازی به تکرار آنها در این قسمت نمیباشد.

دستور FBD

Midline Output

فرمت:

Address	Data Type	Memory Area
<Bit>	Bool	I,Q,M,L,D

شرح :

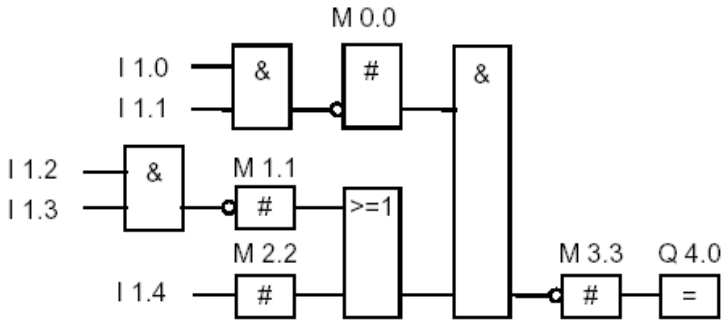
این دستور یک المان میانی برای ذخیره سازی RLO میباشد و آخرین نتیجه عملیات منطقی قبل از این بلوک را در خود ذخیره میسازد.

وضعیت Status Word

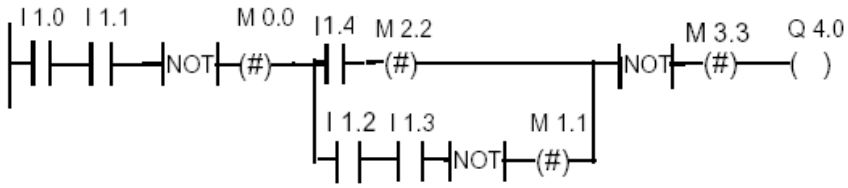
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	x	-	1

مثال :

در مثال زیر M0.0 برای ذخیره سازی نتیجه معکوس ناش از And دو ورودی I1.0 و I1.1 بکار رفته است همینطور M1.1 و M2.2 و M3.3 نیز برای ذخیره نتایج میان برنامه استفاده شده اند.



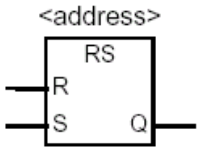
معادل LAD (مثال)



RS Reset_Set Flip Flop

دستور FBD

فرمت:



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	آدرسی که مشخص میکند کدام بیت ست یا ری ست شود
S	BOOL	I, Q, M, D, L, T, C	فعال کردن ست
R	BOOL	I, Q, M, D, L, T, C	فعال کردن ری ست
Q	BOOL	I, Q, M, D, L	خروجی

شرح :

RS فیلیپ فلاپ عمل ست و ری ست را وقتی که RLO =1 باشد اجرا میکند بنابر این اگر RLO=0 باشد این دستور اجرا نمیگردد.

عمل ری ست کردن بیت آدرس داده شده وقتی اتفاق می افتد که ورودی R=1 و ورودی S=0 باشد.

عمل ست کردن بیت آدرس داده شده وقتی اتفاق می افتد که ورودی R=0 و ورودی S=1 باشد. اگر هر دو ورودی 1 شوند نیز عمل ست انجام میشود.

وضعیت Status Word

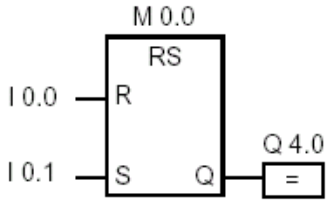
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

مثال :

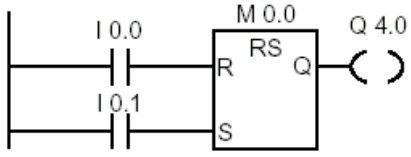
در مثال زیر اگر I0.0 یک و I0.1 صفر شود فلگ M0.0 ری ست شده و خروجی Q4.0 صفر خواهد شد.

اگر هر دو ورودی صفر شوند در اینصورت تغییری در وضعیت فلگ یا خروجی حاصل نمیشود.

اگر هر دو ورودی یک شوند در اینصورت فلگ یک شده و خروجی نیز یک میگردد.



معادل LAD (مثال)



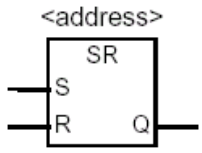
معادل STL (مثال)

```
A  I0.0
R  M0.0
A  I0.1
S  M0.0
=  Q4.0
```

SR Set_Reset Flip Flop

دستور FBD

فرمت:



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	آدرسی که مشخص میکند کدام بیت ست یا ری ست شود
S	BOOL	I, Q, M, D, L, T, C	فعال کردن ست
R	BOOL	I, Q, M, D, L, T, C	فعال کردن ری ست
Q	BOOL	I, Q, M, D, L	خروجی

شرح :

RS فیلیپ فلاپ عمل ست و ری ست را وقتی که RLO =1 باشد اجرا میکند بنابر این اگر RLO=0 باشد این دستور اجرا نمیگردد.

عمل ست کردن بیت آدرس داده شده وقتی اتفاق می افتد که ورودی R=0 و ورودی S=1 باشد.

عمل ری ست کردن بیت آدرس داده شده وقتی اتفاق می افتد که ورودی R=1 و ورودی S=0 باشد. اگر هر دو ورودی 1 شوند نیز عمل ری ست انجام میشود.

وضعیت Status Word

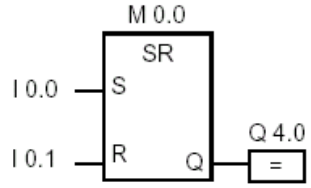
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

مثال :

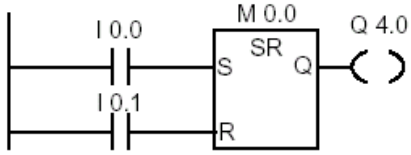
در مثال زیر اگر I0.0 یک و I0.1 صفر شود فلگ M0.0 ست شده و خروجی Q4.0 یک خواهد شد.

اگر هر دو ورودی صفر شوند در اینصورت تغییری در وضعیت فلگ یا خروجی حاصل نمیشود.

اگر هر دو ورودی یک شوند در اینصورت فلگ ری ست شده و خروجی نیز صفر میگردد.



معادل LAD (مثال)



معادل STL (مثال)

```
A   I   0.0
S   M   0.0
A   I   0.1
R   M   0.0
A   M   0.0
=   Q   4.0
```

۵-۲ دستورات مقایسه ای (Comparison Instruction)

این دستورات عملیات مقایسه را انجام میدهند مقاسیه در واقع بین محتویات دو آکومولاتور صورت میگیرد. علامتهای بکار رفته برای مقایسه در جدول زیر آمده اند:

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

دستوراتی که در این بخش تشریح میگردند عبارتند از :

- ? I Compare Integer (16-bit)
- ? D Compare Double Integer (32-bit)
- ? R Compare Floating-point Number (32-bit)

منظور از علامت ? علامتهای مقایسه است که در جدول بالا آورده شده اند.

تذکر : در این بخش بدلیل شباهت دستورات بکار رفته در LAD و FBD صرفاً به ارائه مثالهای معادل FBD اکتفا شده است.

دستور STL: ? I Compare Integer (16-bit)

فرمت:

==I , <>I , >I , <I , >=I , <=I

شرح:

دستورات مقایسه ای ۱۶ بیتی فوق محتویات آکومولاتور ACCU2-L را با محتویات آکومولاتور ACCU1-L مقایسه میکند. اگر نتیجه مقایسه درست بود RLO=1 و در غیر اینصورت RLO=0 میشود. بیت های CC1 و CC0 از بیتهای Status Word نیز مشخص کننده نتیجه واقعی مقایسه یعنی کوچکتر، بزرگتر یا مساوی بصورت جدول زیر هستند:

CC1	CC0	نتیجه
0	0	ACCU2-L = ACCU1-L
0	1	ACCU2-L < ACCU1-L
1	0	ACCU2-L > ACCU1-L

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	0	-	0	x	x	1

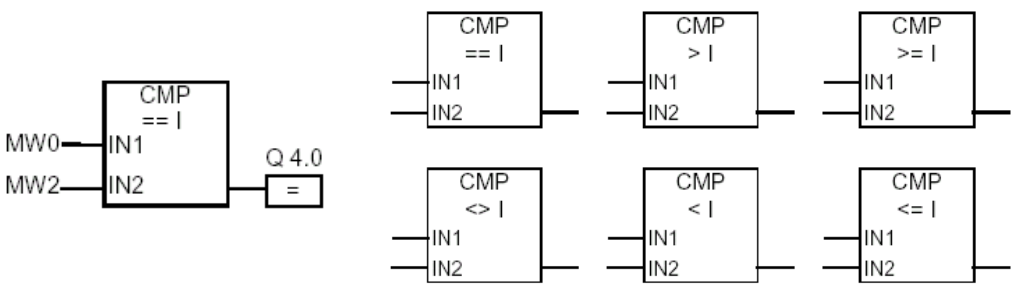
مثال:

در مثال روبرو مقدار MW0 به ACCU1-L بار میشود سپس این مقدار به ACCU2-L انتقال داده شده و مقدار MW2 به ACCU1-L بار میشود . مقایسه میشود که آیا مقدار ACCU2-L بزرگتر از مقدار ACCU1-L هست یانه ؟ اگر جواب مثبت بود RLO=1 میشود و خروجی Q4.0 یک میگردد.

```
L    MW0
L    MW2
>I
=    Q4.0
```

معادل FBD

مثال



دستور STL: ? D Compare Double Integer (32-bit)

فرمت:

==D , <>D , >D , <D , >=D , <=D

شرح:

دستورات مقایسه ای دو عدد صحیح ۳۲ بیتی فوق محتویات آکومولاتور ACCU2 را با محتویات آکومولاتور ACCU1 مقایسه میکند. اگر نتیجه مقایسه درست بود RLO=1 و در غیر اینصورت RLO=0 میشود. بیت های CC1 و CC0 از بیتهای Status Word نیز مشخص کننده نتیجه واقعی مقایسه یعنی کوچکتر ، بزرگتر یا مساوی بصورت جدول زیر هستند:

CC1	CC0	نتیجه
0	0	ACCU2 = ACCU1
0	1	ACCU2 < ACCU1
1	0	ACCU2 > ACCU1

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	-	0	x	x	1

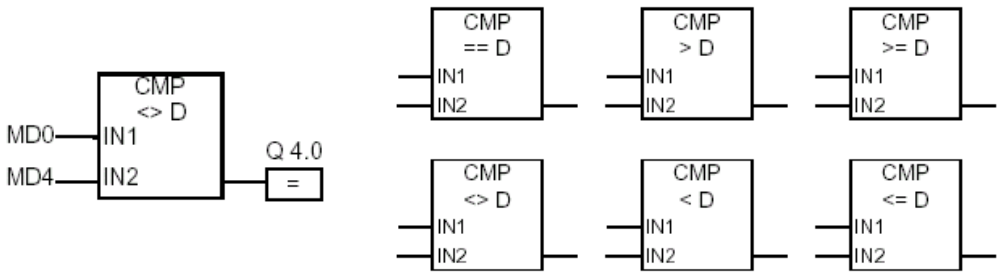
مثال:

در مثال روبرو مقدار MW0 به ACCU1 بار میشود سپس این مقدار به ACCU2 انتقال داده شده و مقدار MW2 به ACCU1 بار میشود . مقایسه میشود که آیا مقدار ACCU2 بزرگتر یا کوچکتر از مقدار ACCU1 هست یانه ؟ اگر جواب مثبت بود RLO=1 میشود و خروجی Q4.0 یک میگردد.

```
L     MD0
L     MD4
<>D
=     Q4.0
```

معادل FBD

مثال



دستور STL : ? R Compare Floating-Point (32-bit)

فرمت:

==R , <>R , >R , <R , >=R , <=R

شرح :

دستورات مقایسه ای اعداد اعشاری ۳۲ بیتی فوق محتویات آکومولاتور ACCU2 را با محتویات آکومولاتور ACCU1 مقایسه میکند. اگر نتیجه مقایسه درست بود RLO=1 و در غیر اینصورت RLO=0 میشود. بیت های CC1 و CC0 از بیتهای Status Word نیز مشخص کننده نتیجه واقعی مقایسه یعنی کوچکتر ، بزرگتر یا مساوی بصورت جدول زیر هستند:

CC1	CC0	نتیجه
0	0	ACCU2 = ACCU1
0	1	ACCU2 < ACCU1
1	0	ACCU2 > ACCU1

وضعیت Status Word

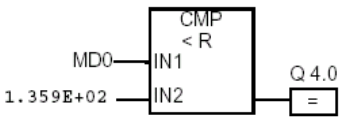
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	0	x	x	1

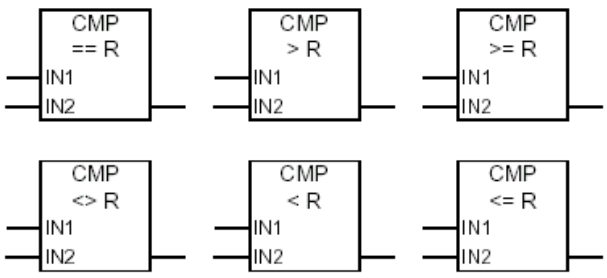
مثال :

در مثال روبرو مقدار MD0 به ACCU1 بار میشود سپس این مقدار به ACCU2 انتقال داده شده و عدد 135.9 به ACCU1 بار میشود . مقایسه میشود که آیا مقدار ACCU2 کوچکتر از مقدار ACCU1 هست یانه ؟ اگر جواب مثبت بود RLO=1 میشود و خروجی Q4.0 یک میگردد.

```
L     MD0
L     1.359E+02
<R
=     Q4.0
```

معادل FBD مثال





۳-۵ دستورات تبدیل (Conversion Instruction)

این دستورات عملیات تبدیل را انجام میدهند و میتوان آنها را به ٤ دسته زیر تقسیم نمود:

دسته اول : دستوراتی که مقادیر BCD یا Integer را به سایر فرمتهای عددی تبدیل مینمایند و عبارتند از :

- **BTI BCD to Integer (16-bit)**
- **ITB Integer (16-bit) to BCD**
- **BTD BCD to Integer (32-bit)**
- **ITD Integer (16-bit) to Double Integer (32-bit)**
- **DTB Double Integer (32-bit) to BCD**
- **DTR Double Integer (32-bit) to Floating-point (32-bit IEEE-FP)**

دسته دوم : دستوراتی که برای متمم کردن بکار میروند و عبارتند از :

- **INVI Ones Complement Integer (16-bit)**
- **INVD Ones Complement Double Integer (32-bit)**
- **NEGI Twos Complement Integer (16-bit)**
- **NEGD Twos Complement Double Integer (32-bit)**
- **NEGR Negate Floating-point Number (32-bit, IEEE-FP)**

دسته سوم : دستوراتی که ترتیب بیت ها را در آکومولاتور شماره ۱ تغییر میدهند و عبارتند از :

- **CAW Change Byte Sequence in ACCU 1-L (16-bit)**
- **CAD Change Byte Sequence in ACCU 1 (32-bit)**

دسته چهارم : دستوراتی که تبدیل روی اعداد اعشاری ۳۲ بیتی انجام میدهند و عبارتند از :

- **RND Round**
- **TRUNC Truncate**
- **RND+ Round to Upper Double Integer**
- **RND- Round to Lower Double Integer**

تذکر : در این بخش نیز بدلیل شباهت دستورات بکار رفته در LAD و FBD صرفاً به ارائه بلوکهای FBD اکتفا شده است.این بلوکها عیناً در دیاگرام نردبانی LAD بکار میروند.

BTI BCD to Integer (16-bit)

دستور STL :

فرمت:

BTI

شرح :

دستور BTI یک عدد BCD سه شماره ای را که در ACCU1-L بار شده به عدد صحیح (Integer) ۱۶ بیتی تبدیل مینماید. نتیجه در همان ACCU1-L ذخیره میشود بنابراین ACCU1-H و ACCU2 در طول تبدیل تغییری نمیکنند.

عدد BCD که در ACCU1-L بار میشود میتواند بین "999-" تا "999+" باشد که ۳ رقم آن در بیتهای 0 تا 11 و علامت آن در بیت 15 قرار میگیرد(1= Negative , 0=Positive). بیتهای 12 تا 14 در تبدیل بکار نمیروند. اگر یکی از ارقام دسیمال عدد BCD در رنج غیر مجاز 10 تا 15 واقع شود خطای BCDF در حین تبدیل ظاهر میشود که معمولاً CPU را به مد STOP میبرد و کد خطا به شماره Id:2521 در بافر تشخیص عیب CPU ذخیره میگردد. با طراحی و برنامه ریزی OB121 میتوان روی این خطا مدیریت کرد و مانع Stop شدن CPU گردید.

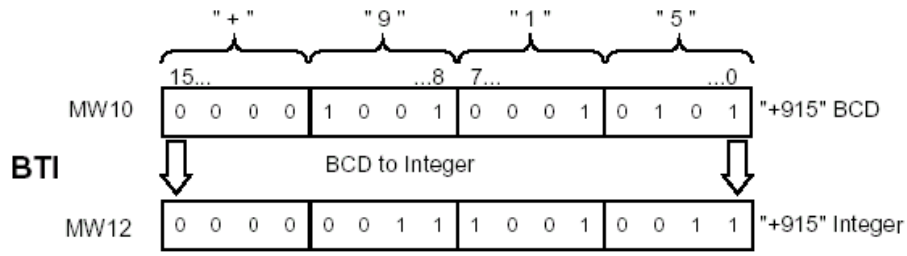
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

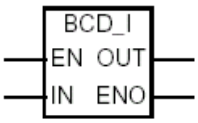
مثال :

```
L    MW10
BTI
T    MW12
```

در مثال روبرو عدد BCD که در MW10 موجود است به ACCU1-L بار میشود سپس این مقدار به Integer تبدیل شده و نتیجه از ACCU1-L به MW12 انتقال می یابد. در شکل زیر نحوه تبدیل عدد "915+" از BCD به Integer در آکومولاتور ACCU1-L نشان داده شده است.

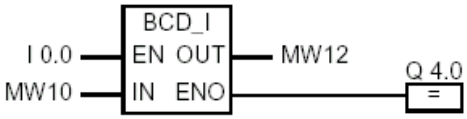


معادل FBD



مثال

تبدیل با I0.0=1 شروع میشود.اگر تبدیل انجام شد خروجی Q4.0=1 است و اگر سرریزی (overflow) اتفاق افتاد تبدیل انجام نشده و خروجی Q4.0 صفرمیشود.



ITB Integer (16-bit) to BCD

دستور STL:

فرمت:

ITB

شرح:

دستور ITB یک عدد صحیح ۱۶ بیتی را که در ACCU1-L بار شده به عدد BCD سه شماره ای تبدیل مینماید. نتیجه در همان ACCU1-L ذخیره میشود بنابرین ACCU1-H و ACCU2 در طول تبدیل تغییری نمیکنند.

در ACCU1-L بیتهای 0 تا 11 مقدار BCD را در خود دارند و بیتهای 12 تا 15 علامت را بصورت زیر نشان میدهند:

0000= Positive , 1111=Negative

عدد BCD میتواند بین "999-" تا "999+" باشد اگر عدد خارج از این رنج باشد بیتهای OV و OS از بیت های Status Word یک میشوند. این دستور تاثیری روی RLO ندارد.

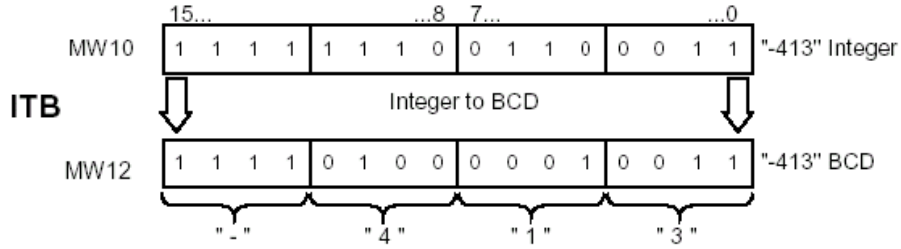
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	x	x	-	-	-	-

مثال:

در مثال روبرو عدد صحیح که در MW10 موجود است به ACCU1-L بار میشود سپس این مقدار به BCD تبدیل شده و نتیجه از ACCU1-L به MW12 انتقال می یابد. در شکل زیر نحوه تبدیل عدد "413-" از Integer به BCD در آکومولاتور ACCU1-L نشان داده شده است.

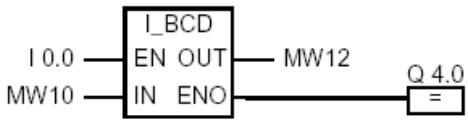
```
L    MW10
ITB
T    MW12
```

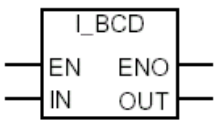


معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال قبل تغییر میکند





BTD BCD to Integer (32-bit)

دستور STL :

فرمت:

BTD

شرح :

دستور BTD یک عدد BCD هفت شماره ای را که در ACCU1 بار شده به عدد صحیح (Integer) ۳۲ بیتی تبدیل مینماید. نتیجه در همان ACCU1 ذخیره میشود بنابرین ACCU2 در طول تبدیل تغییری نمیکنند.

عدد BCD که در ACCU1 بار میشود میتواند بین "9,999,999-" تا "9,999,999+" باشد که ۷ رقم آن در بیتهای 0 تا 27 و علامت آن در بیت 31 قرار میگیرد (1= Negative , 0=Positive). بیتهای 28 تا 30 در تبدیل بکار نمیروند. اگر یکی از ارقام دسیمال عدد BCD در رنج غیر مجاز 10 تا 15 واقع شود خطای BCDF در حین تبدیل ظاهر میشود که معمولاً CPU را به مد STOP میبرد و کد خطا به شماره Id:2521 در بافر تشخیص عیب CPU ذخیره میگردد. با طراحی و برنامه ریزی OB121 میتوان روی این خطا مدیریت کرد و مانع Stop شدن CPU گردید.

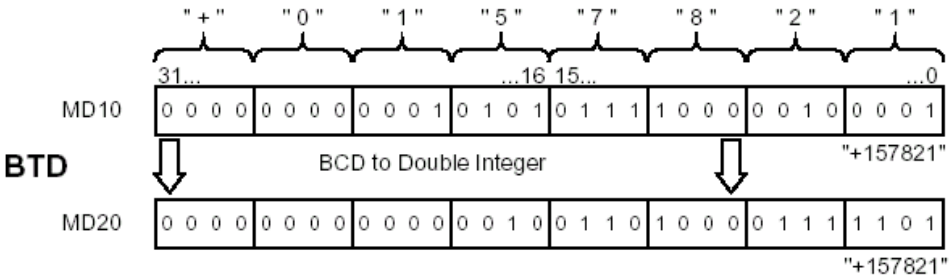
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

```
L     MD10
BTI
T     MD20
```

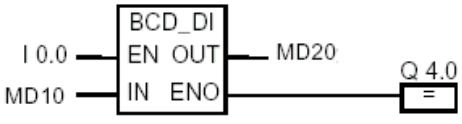
در مثال روبرو عدد BCD که در MD10 موجود است به ACCU1 بار میشود سپس این مقدار به Integer تبدیل شده و نتیجه از ACCU1 به MD20 انتقال می یابد. در شکل زیر نحوه تبدیل عدد "157821+" از BCD به Integer در آکومولاتور ACCU1 نشان داده شده است.

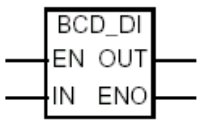


معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال دستورBTI تغییر میکند





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

دستور STL :

فرمت:

ITD

شرح :

دستور ITD یک عدد صحیح ۱۶ بیتی را که در ACCU1-L بار شده به عدد صحیح ۳۲ بیتی تبدیل مینماید. نتیجه در ACCU1 ذخیره میشود بنابرین ACCU2 در طول تبدیل تغییری نمیکنند.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو عدد صحیح۱۶ بیتی که در MW10 موجود است به ACCU1-L بار میشود سپس این مقدار به عدد صحیح ۳۲ بیتی تبدیل شده و نتیجه از ACCU1 به MD12 انتقال می یابد. در شکل زیر نحوه تبدیل عددصحیح "10-" از ۱۶ بیتی به ۳۲ بیتی در آکومولاتور ACCU1 نشان داده شده است.

```
L     MW10
ITB
T     MD12
```

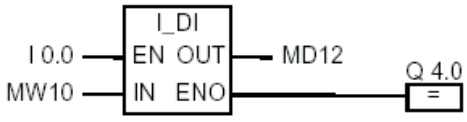
Example: MW10 = "-10" (Integer, 16-bit)

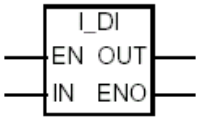
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **ITD**	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
after execution of **ITD**	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 or 1, bits are not used for conversion)							

معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال دستورBTI تغییر میکند





دستور STL: DTB Double Integer (32-bit) to BCD

فرمت:

DTB

شرح:

دستور DTB یک عدد صحیح ۳۲ بیتی را که در ACCU1 بار شده به عدد BCD هفت شماره ای تبدیل مینماید. نتیجه در همان ACCU1 ذخیره میشود بنابرین ACCU2 در طول تبدیل تغییری نمیکنند.

در ACCU1 بیتهای 0 تا 27 مقدار BCD را در خود دارند و بیتهای 28 تا 31 علامت را بصورت زیر نشان میدهند:

0000= Positive , 1111=Negative

عدد BCD میتواند بین "9,999,999-" تا "9,999,999+" باشد اگر عدد خارج از این رنج باشد بیتهای OV و OS از بیت های Status Word یک میشوند. این دستور تاثیری روی RLO ندارد.

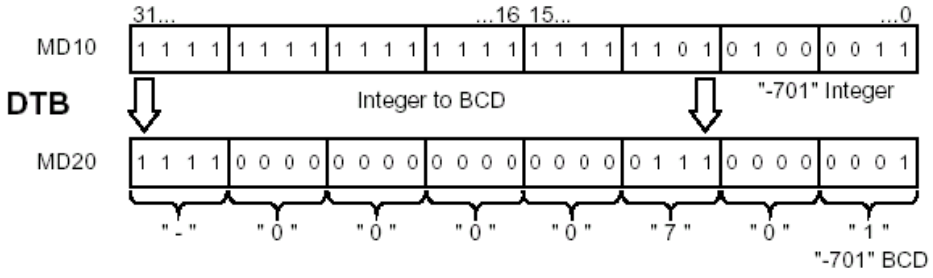
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

در مثال روبرو عدد صحیح که در MW10 موجود است به ACCU1-L بار میشود سپس این مقدار به BCD تبدیل شده و نتیجه از ACCU1-L به MW12 انتقال می یابد. در شکل زیر نحوه تبدیل عدد "701-" از Integer به BCD در آکومولاتور ACCU1-L نشان داده شده است.

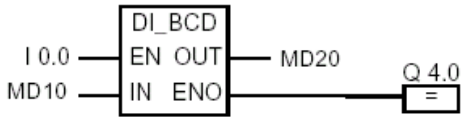
```
L    MD10
ITB
T    MD20
```

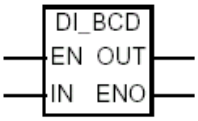


معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال دستورBTI تغییر میکند





دستور STL: DTR Double Integer (32-bit) to Floating –Point (32-bit)

فرمت:

DTR

شرح:

دستور DTR یک عدد صحیح ۳۲ بیتی را که در ACCU1 بار شده به عدد اعشاری ۳۲ بیتی (Real) که دقت بیشتر دارد تبدیل مینماید. نتیجه در همان ACCU1 ذخیره میشود بنابرین ACCU2 در طول تبدیل تغییری نمیکنند.

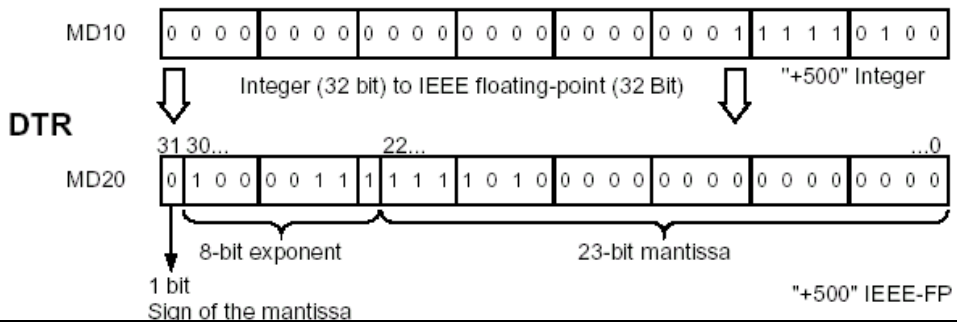
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

در مثال روبرو عدد صحیح ۳۲ بیتی که در MD10 موجود است به ACCU1 بار میشود سپس این مقدار به اعشاری تبدیل شده و نتیجه از ACCU1 به MD20 انتقال می یابد. در شکل زیر نحوه تبدیل عدد "500+" از Integer به Real در آکومولاتور ACCU1 نشان داده شده است.

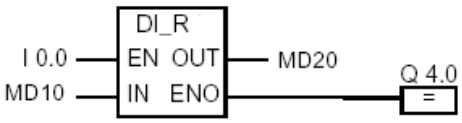
```
L     MD10
ITB
T     MD20
```

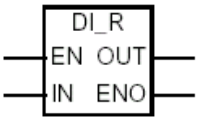


معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال دستورBTI تغییر میکند





INVI Ones Complement Integer (16-bit)

دستور STL:

فرمت:

INVI

شرح:

دستور INVI تک تک بیت های یک عدد صحیح 16 بیتی را که در ACCU1-L بار شده متمم یک میکند. بعبارت دیگر تمام "1" ها را به "0" و تمام "0" ها را به "1" تبدیل مینماید. نتیجه در همان آکومولاتور ACCU1-L ذخیره میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

در مثال روبرو عدد صحیح ۱۶ بیتی که در MW8 موجود است به ACCU1-L بار میشود سپس این مقدار متمم یک شده و نتیجه از ACCU1-L به MW10 انتقال می یابد. در شکل زیر مثالی از نحوه تبدیل آمده است.

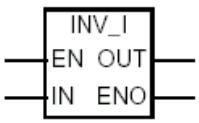
```
L    MW8
INVI
T    MW10
```

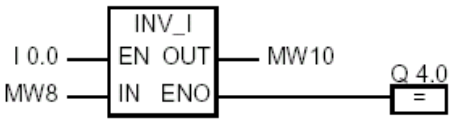
Contents	ACCU1-L			
Bit	15 . . .	. .	. .	. . . 0
before execution of **INVI**	0110	0011	1010	1110
after execution of **INVI**	1001	1100	0101	0001

معادل FBD

مثال

تبدیل با I0.0=1 شروع میشود.اگر تبدیل انجام شود خروجی Q4.0=1 است و اگر تبدیل انجام نشود خروجی Q4.0 صفرمیشود.





INVD Ones Complement Double Integer (32-bit)

دستور STL :

فرمت:

INVD

شرح :

دستور INVD تک بیت های یک عدد صحیح 32بیتی را که در ACCU1 بار شده متمم یک میکند. بعبارت دیگر تمام "1" ها را به "0" و تمام "0" ها را به "1" تبدیل مینماید. نتیجه در همان آکومولاتور ACCU1 ذخیره میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو عدد صحیح 32ی که در MD موجود است به ACCU1 بار میشود سپس این مقدار متمم یک شده و نتیجه از ACCU1 به MD12انتقال می یابد. در شکل زیر مثالی از نحوه تبدیل آمده است.

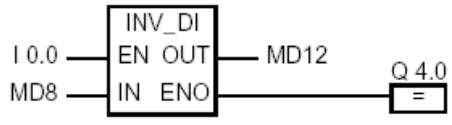
```
L    MD8
INVI
T    MD12
```

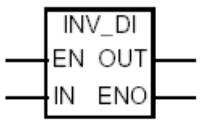
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **INVD**	0110	1111	1000	1100	0110	0011	1010	1110
after execution of **INVD**	1001	0000	0111	0011	1001	1100	0101	0001

معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال های قبل تغییر میکند





NEGI Twos Complement Integer (16-bit)

دستور STL :

فرمت:

NEGI

شرح :

دستور NEGI یک عدد صحیح 16 بیتی را که در ACCU1-L بار شده متمم دو میکند. بعبارت دیگر تمام "1" ها را به "0" و تمام "0" ها را به "1" تبدیل مینماید و سپس به نتیجه حاصل "1" اضافه میکند. نتیجه در همان آکومولاتور ACCU1-L ذخیره میشود. متمم دو در واقع منجر به منفی شدن عدد میگردد مثل اینکه آن را در منفی یک ضرب کرده باشیم. که برای عملیات حسابی (بویژه تفریق) مفید است.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

Status word generation	CC 1	CC 0	OV	OS
Result = 0	0	0	0	-
-32768 <= Result <= -1	0	1	0	-
32767 >= Result >= 1	1	0	0	-
Result = 32768	0	1	1	1

مثال :

در مثال روبرو عدد صحیح ۱۶ بیتی که در MW8 موجود است به ACCU1-L بار میشود سپس این مقدار متمم دو (یعنی منفی) شده و نتیجه از ACCU1-L به MW10 انتقال می یابد. در شکل زیر مثالی از نحوه تبدیل آمده است.

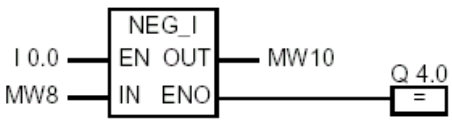
```
L     MW8
NEGI
T     MW10
```

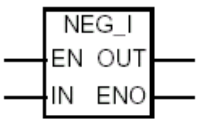
Contents	ACCU1-L			
Bit	15 . . .	. .	. .	. . . 0
before execution of **NEGI**	0101	1101	0011	1000
after execution of **NEGI**	1010	0010	1100	1000

معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال های قبل تغییر میکند





NEGD Twos Complement Double Integer (32-bit)

دستور STL :

فرمت:

NEGD

شرح :

دستور NEGI یک عدد صحیح 32 بیتی را که در ACCU1 بار شده متمم دو میکند. بعبارت دیگر تمام "1" ها را به "0" و تمام "0" ها را به "1" تبدیل مینماید و سپس به نتیجه حاصل "1" اضافه میکند. نتیجه در همان آکومولاتور ACCU1 ذخیره میشود. متمم دو در واقع منجر به منفی شدن عدد میگردد مثل اینکه آن را در منفی یک ضرب کرده باشیم. که برای عملیات حسابی (بویژه تفریق) مفید است.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

Status word generation	CC 1	CC 0	OV	OS
Result = 0	0	0	0	-
-2,147,483,648 <= Result <= -1	0	1	0	-
2,147,483,647 >= Result >= 1	1	0	0	-
Result = 2,147,483,648	0	1	1	1

مثال :

در مثال روبرو عدد صحیح ۳۲ بیتی که در MD8 موجود است به ACCU1 بار میشود سپس این مقدار متمم دو (یعنی منفی) شده و نتیجه از ACCU1 به MD12 انتقال می یابد. در شکل زیر مثالی از نحوه تبدیل آمده است.

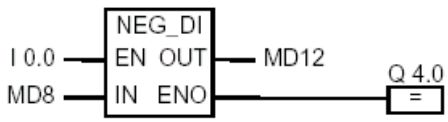
```
L    MD8
NEGI
T    MD12
```

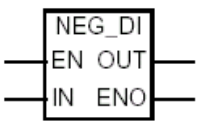
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **NEGD**	0101	1111	0110	0100	0101	1101	0011	1000
after execution of **NEGD**	1010	0000	1001	1011	1010	0010	1100	1000

معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال های قبل تغییر میکند





دستور STL:

NEGR Twos Complement Floating-Point (32-bit)

فرمت:

NEGR

شرح:

دستور NEGR یک عدد اعشاری 32 بیتی را که در ACCU1 بار شده را منفی میکند. در واقع این دستور بیت 31 در ACCU1 را که نشان دهنده علامت است متمم میکند("0" به "1" یا بعکس) نتیجه در همان آکومولاتور ACCU1 ذخیره میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

در مثال روبرو عدد اعشاری ۳۲ بیتی که در MD8 موجود است به ACCU1 بار میشود سپس این مقدار منفی شده و نتیجه از ACCU1 به MD12 انتقال می یابد. در زیر مثالی از نحوه تبدیل آمده است.

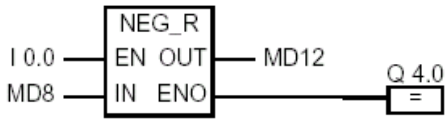
```
L     MD8
NEGR
T     MD12
```

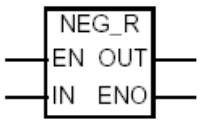
MD8 = + 6.234 ⟶ MD12 = - 6.234

معادل FBD

مثال

خروجی Q4.0=1 شبیه مثال های قبل تغییر میکند





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

دستور STL :

فرمت:

CAW

شرح : دستور CAW ترتیب بیتها را در ACCU1-L معکوس میکند . نتیجه در آکومولاتور ACCU1-L ذخیره میشود بنابراین ACCU1-H و ACCU2 در طول تبدیل تغییری نمیکنند. این دستور معادل LAD , FBD ندارد

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو مقدار ۱۶ بیتی که در MW10 موجود است به ACCU1-L بار میشود سپس ترتیب بیتهای آن مانند مثال زیر عوض شده و نتیجه به MW20 انتقال می یابد

```
L    MW10
CAW
T    MW20
```

Contents	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
before execution of CAW	value A	value B	value C	value D
after execution of CAW	value A	Value B	value D	value C

CAD Change Byte Sequence in ACCU1 (32-bit)

دستور STL :

فرمت:

CAD

شرح : دستور CAD ترتیب بیتها را در ACCU1 معکوس میکند . نتیجه در آکومولاتور ACCU1 ذخیره میشود بنابراین ACCU2 در طول تبدیل تغییری نمیکنند. معادل LAD و FBD ندارد و وضعیت Status Word در آن مشابه دستور CAW است.

مثال :

در مثال روبرو مقدار ۳۲ بیتی که در MD10 موجود است به ACCU1 بار میشود سپس ترتیب بیتهای آن مانند مثال زیر عوض شده و نتیجه به MD20 انتقال می یابد

```
L    MD10
CAD
T    MD20
```

Contents	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
before execution of CAD	value A	value B	value C	value D
after execution of CAD	value D	value C	value B	value A

دستور STL :

RND Round

فرمت:

RND

شرح :

دستور RND عدد اعشاری 32 بیتی را به عدد صحیح 32 بیتی (Double Integer) تبدیل کرده و با حذف قسمت اعشاری آنرا به نزدیکترین عدد صحیح گرد مینماید. اگر قسمت اعشاری دقیقاً بین دو عدد زوج و فرد واقع شده باشد (مانند 18.5 که بین 18 و 19 قرار دارد) در اینصورت دستور فوق عدد زوج را انتخاب مینماید. اگر عدد خارج از رنج مجاز باشد بیتهای OV و OS یک میگردند.

نتیجه تبدیل در آکومولاتور ACCU1 ذخیره میشود .

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	x	x	-	-	-	-

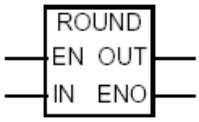
مثال :

در مثال روبرو عدد اعشاری ۳۲ بیتی که در MD8 موجود است به ACCU1 بار میشود سپس گرد شده و نتیجه به MD12 انتقال می یابد

```
L    MD8
RND
T    MD12
```

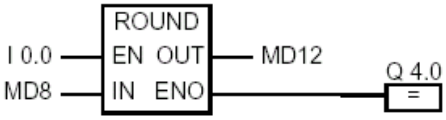
Value before conversion			Value after conversion
MD8 = "100.5"	=>	RND =>	MD12 = "+100"
MD8 = "-100.5"	=>	RND =>	MD12 = "-100"

معادل FBD



مثال

تبدیل با I0.0=1 شروع میشود.اگر تبدیل انجام شود خروجی Q4.0=1 است و اگر تبدیل انجام نشود مثلاً سرریزی اتفاق بیفتد خروجی Q4.0 صفرمیشود.



TRUNC Truncate

دستور STL:

فرمت:

TRUNC

شرح:

دستور TRUNC عدد اعشاری 32 بیتی را به عدد صحیح 32 بیتی (Double Integer) تبدیل کرده و با حذف قسمت اعشاری آنرا به عدد صحیح تبدیل مینماید.درواقع قسمت اعشاری به صفر تبدیل میشود. اگر عدد خارج از رنج مجاز باشد بیتهای OV و OS یک میگردند. نتیجه تبدیل در آکومولاتور ACCU1 ذخیره میشود .

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	x	x	-	-	-	-

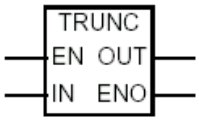
مثال:

در مثال روبرو عدد اعشاری ۳۲ بیتی که در MD8 موجود است به ACCU1 بار میشود سپس بخش اعشار آن حذف شده و نتیجه به MD12 انتقال می یابد

```
L     MD8
TRUNC
T     MD12
```

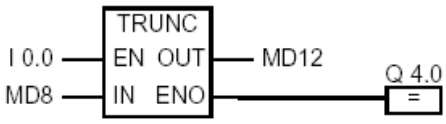
Value before conversion			Value after conversion
MD8 = "100.5"	=>	TRUNC =>	MD12 = "+100"
MD8 = "-100.5"	=>	TRUNC =>	MD12 = "-100"

معادل FBD



مثال

خروجی Q4.0=1 شبیه مثال صفحه قبل تغییر میکند.



RND+ Round to Upper Double Integer

دستور STL :

فرمت:

RND+

شرح :

دستور+ RND عدد اعشاری 32 بیتی را به عدد صحیح 32 بیتی (Double Integer) تبدیل کرده و با حذف قسمت اعشاری آنرا به عدد صحیح بالاتر یعنی عدد صحیحی که با عدد اعشاری مساوی یا از آن بزرگتر است گرد مینماید مثلاً 1.2+ به 2+ تبدیل میشود یا 1.5- به 1-گرد میشود. اگر عدد خارج از رنج مجاز باشد بیتهای OV و OS یک میگردند.

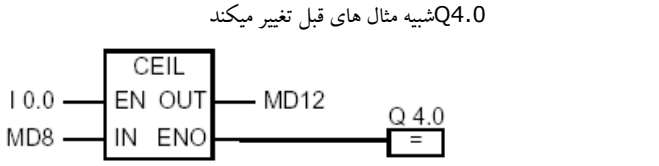
نتیجه تبدیل در آکومولاتور ACCU1 ذخیره میشود .

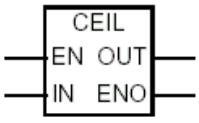
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	x	x	-	-	-	-

معادل FBD

مثال





RND- Round to Lower Double Integer

دستور STL :

فرمت:

RND-

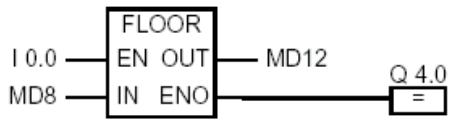
شرح :

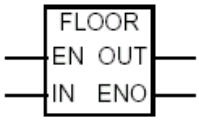
دستور- RND عدد اعشاری 32 بیتی را به عدد صحیح 32 بیتی (Double Integer) تبدیل کرده و با حذف قسمت اعشاری آنرا به عدد صحیح پایین تر یعنی عدد صحیحی که با عدد اعشاری مساوی یا از آن کوچکتر است گرد مینماید مثلاً 1.2+ به 1+ تبدیل میشود. اگر عدد خارج از رنج مجاز باشد بیتهای OV و OS یک میگردند. نتیجه تبدیل در آکومولاتور ACCU1 ذخیره میشود .

وضعیت Statud word شبیه دستور +RND است.

معادل FBD

مثال

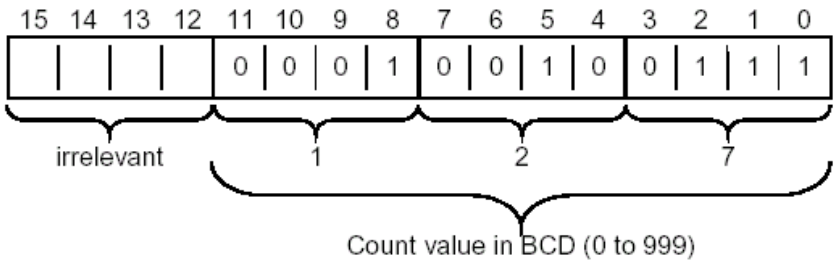


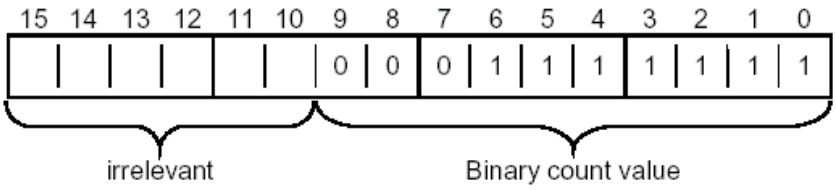


٤-٥ دستورات کانترها (Counter Instruction)

کانترها یا شمارنده ها المانهایی از زبان برنامه نویسی در STEP7 هستند که ناحیه خاصی از حافظه CPU را بخود اختصاص داده اند. در این ناحیه برای هر کانتر یک Word یعنی ١٦ بیت رزرو شده است.تعداد کانتر بستگی به نوع CPU دارد که در مشخصات فنی آن آورده میشود.

مقدار کانتر بصورت BCD و از 0 تا 999 میباشد (در عین حال میتوان معادل باینری آنرا نیز داشت) چون هر رقم ٤ بیت بخود اختصاص میدهد پس از ١٦ بیت فوق ١٢ بیت یعنی بیت های 0 تا 11 استفاده میشوند. شکل زیر عدد 127 را که در ناحیه حافظه یک کانتر قرار گرفته نشان میدهد:





دستورات کانتر عبارتند از :

- **FR Enable Counter (Free)**
- **L Load Current Counter Value into ACCU 1**
- **LC Load Current Counter Value into ACCU 1 as BCD**
- **R Reset Counter**
- **S Set Counter Preset Value**
- **CU Counter Up**
- **CD Counter Down**

تذکر : در این بخش نیز بدلیل شباهت دستورات بکار رفته در LAD و FBD صرفاً به ارائه بلوکهای FBD اکتفا شده است.این بلوکها عیناً در دیاگرام نردبانی LAD بکار میروند.

FR Enable Counter (Free)

دستور STL :

فرمت:

FR <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح :

وقتی RLO از "0" به "1" میرود دستور FR لبه را که برای شمارش افزایشی یا کاهشی کانتر بکار میرود تشخیص میدهد وکانتر را فعال میکند. این دستور آزاد است یعنی استفاده از آن الزامی نیست بدون آن نیز با تغییر RLO از "0" به "1" کانتر فعال میگردد.

میکند بعبارت دیگر وقتی RLO از "0" به "1" میرود این دستور آنرا تشخیص داده و نتیجه را با RLO=1 آشکار می سازد.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

مثال :

در مثال روبرو به محض اینکه در سیکل اسکن جدید ورودی I1.0 از "0" به "1" میرود دستور FR آنرا تشخیص داده و کانتر C3 را فعال میکند. بدیهی است هنوز به کانتر دستور شمارش بالا یا پایین داده نشده است.

```
A     I 1.0
FR    C3
```

.

معادل LAD **مثال**

ندارد –

معادل FBD **مثال**

ندارد –

L Load Current Counter Value into ACCU1

دستور STL :

فرمت:

L <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح :

دستور L ابتدا محتویات ACCU1 در ACCU2 ذخیره کرده سپس مقدار کانتری که آدرس داده شده است را از ناحیه حافظه کانتربصورت Integer به ACCU1-L بار میکند

وضعیت Status Word

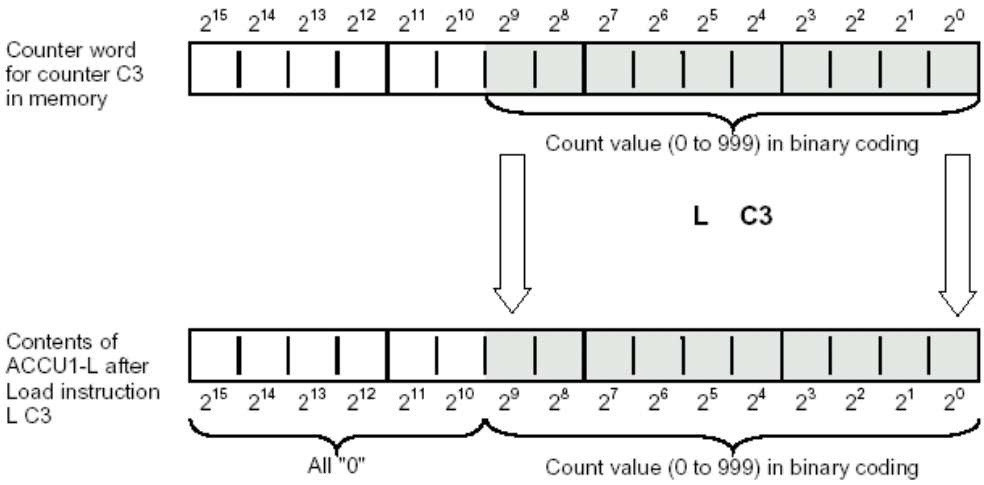
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو مقدار کانتر C3 مطابق شکل زیر بصورن باینری به ACCU1-L بار میشود.

L C3

.



معادل LAD **مثال**

ندارد –

معادل FBD **مثال**

ندارد –

LC Load Current Counter Value into ACCU1 as BCD

دستور STL :

فرمت:

LC <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح :

دستور LC ابتدا محتویات ACCU1 در ACCU2 ذخیره کرده سپس مقدار کانتری که آدرس داده شده است را از ناحیه حافظه کانتر بصورت BCD به ACCU1-L بار میکند

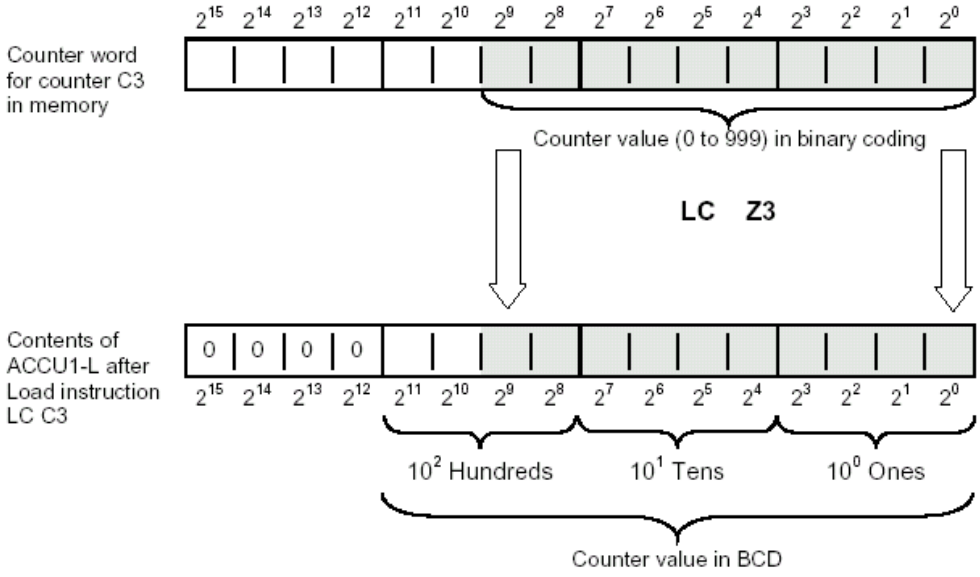
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو مقدار کانتر C3 بصورت BCD مطابق شکل زیر به ACCU1-L بار میشود..

LC C3



معادل LAD **مثال**

ندارد –

معادل FBD **مثال**

ندارد –

R Reset Counter

دستور STL :

فرمت:

R <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح : دستور R اگر RLO=1 باشد مقدار کانتر آدرس داده شده را با صفر جایگزین میکند.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

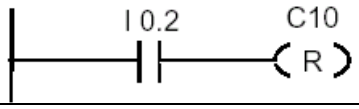
مثال :

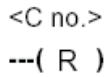
در مثال روبرو اگر I0.2=1 باشد مقدار کانتر C10 ری ست میشود.

```
A    I0.2
R    C10
```

معادل LAD

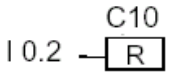
مثال





معادل FBD

مثال



S Set Counter Preset Value

دستور STL :

فرمت:

S <Counter>

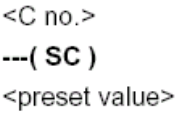
شرح : دستور S وقتی که RLO از "0" به "1" برود مقدار ACCU1-L را به کانتر آدرس داده شده میفرستد. این مقدار باید بصورت BCD بین 0 تا 999 باشد. وضعیت Status Word مشابه دستور ری ست فوق میباشد.

مثال :

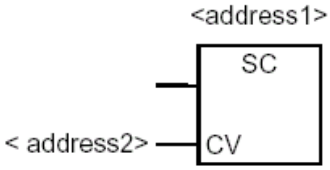
در مثال روبرو وضعیت سیگنال در I0.0 چک میشود سپس عدد 100 که بصورت BCD است به ACCU1-L بار میشود اگر RLO از "0" به "1" برود کانتر C5 با مقدار اولیه فوق ست میشود.

```
A    I0.0
L    C#100
S    C5
```

معادل LAD



معادل FBD



CU Counter Up

دستور STL :

فرمت:

CU <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح :

دستور CU وقتی که RLO از "0" به "1" برود مقدار کانتر آدرس داده شده را یکی افزایش میدهد. افزایش در صورتی امکان پذیر است که مقدار فعلی کانتر از 999 کمتر باشد. اگر مقدار کانتر به 999 برسد شمارش متوقف میگردد و تغییر وضعیت مجدد RLO تاثیری ندارد. لازم به ذکر است در این حالت بیت OV یک نخواهد شد چون عملاً سرریزی اتفاق نیفتاده است.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

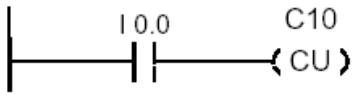
مثال :

در مثال روبرو وضعیت سیگنال در I0.0 چک میشود اگر RLO از "0" به "1" برود مقدار کانتر C10 یکی افزایش می یابد.

```
A     I0.0
CU    C10
```

معادل LAD

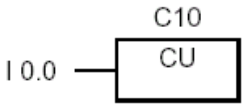
مثال

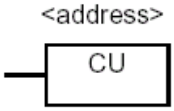



```
<C no.>
---( CU )
```

معادل FBD

مثال





CD Counter Down

دستور STL:

فرمت:

CD <Counter>

Address	Data Type	Memory Area
< Counter >	Counter	C

شرح:

دستور CD وقتی که RLO از "0" به "1" برود مقدار کانتر آدرس داده شده را یکی کاهش میدهد. کاهش در صورتی امکان پذیر است که مقدار فعلی کانتر از 0 بزرگتر باشد. اگر مقدار کانتر به 0 برسد شمارش متوقف میگردد و تغییر وضعیت مجدد RLO تاثیری ندارد.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

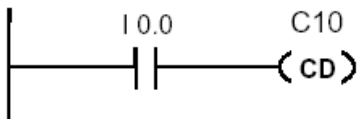
مثال:

در مثال روبرو وضعیت سیگنال در I0.1 چک میشود اگر RLO از "0" به "1" برود کانتر C10 با مقدار اولیه ۱۴ ست میشود سپس با تغییر I0.0 از "0" به "1" کانتر C10 یکی کاهش می یابد. اگر شمارش به صفر برسد کانتر متوقف شده و خروجی Q0.0 یک میشود.

```
L     C#14
A     I0.1
S     C10
A     I0.0
CD    C10
AN    C1
=     Q0.0
```

معادل LAD

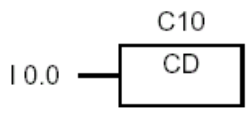
مثال



<C no.>
---(CD)

معادل FBD

مثال



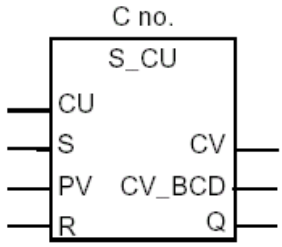
<address>
CD

سایر بلوکهای FBD

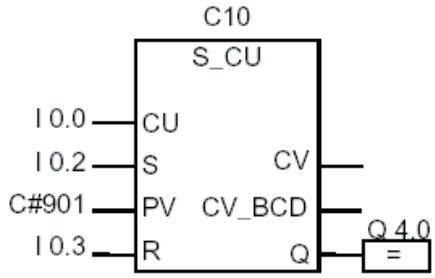
بلوک FBD :

S_CU Assign Parameter and Counter Up

فرمت:



مثال



Parameter	Data Type	Memory	شرح پارامتر
no.	COUNTER	C	شماره کانتر
CU	BOOL	I, Q, M, D, L	ورودی فعال کننده افزایش کانتر
S	BOOL	I, Q, M, D, L, T, C	ورودی برای ست کردن مقدار اولیه به کانتر
PV	WORD	I, Q, M, D, L or constant	مقدار اولیه بصورت BCD
R	BOOL	I, Q, M, D, L, T, C	ورودی ری ست کننده کانتر
CV	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (Hex)
CV_BCD	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (BCD)
Q	BOOL	I, Q, M, D, L	خروجی نمایش وضعیت کانتر

شرح :

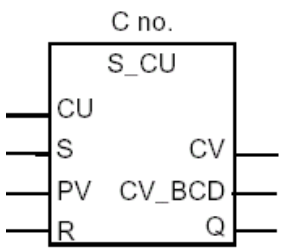
وقتی که RLO برای ورودی متصل به S از "0" به "1" برود کانتر با مقدار PV ست میشود. با لبه بالا رونده ورودی CU مقدار کانتر یکی افزایش می یابد. با فعال شدن ورودی R کانتر به مقدار صفر ری ست میشود. وقتی مقدار شمارش بزرگتر از صفر است خروجی Q یک و وقتی مقدار شمارش صفر شود این خروجی صفر میگردد.

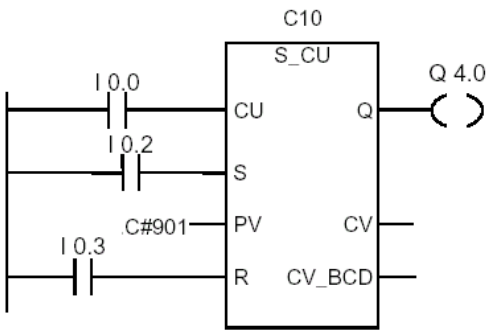
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

معادل LAD

مثال

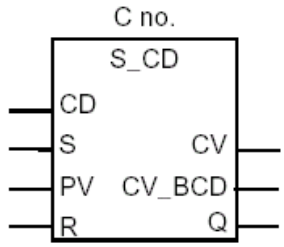




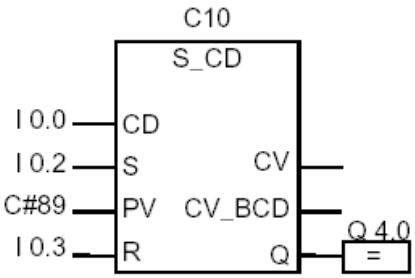
S_CD Assign Parameter and Counter Down

بلوک FBD :

فرمت:



مثال



Parameter	Data Type	Memory	شرح پارامتر
no.	COUNTER	C	شماره کانتر
CD	BOOL	I, Q, M, D, L	ورودی فعال کننده کاهش کانتر
S	BOOL	I, Q, M, D, L, T, C	ورودی برای ست کردن مقدار اولیه به کانتر
PV	WORD	I, Q, M, D, L or constant	مقدار اولیه بصورت BCD
R	BOOL	I, Q, M, D, L, T, C	ورودی ری ست کننده کانتر
CV	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (Hex)
CV_BCD	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (BCD)
Q	BOOL	I, Q, M, D, L	خروجی نمایش وضعیت کانتر

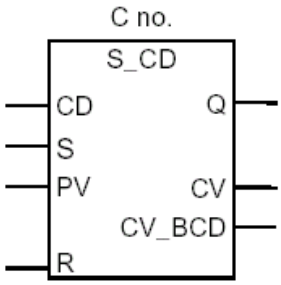
شرح :

وقتی که RLO برای ورودی متصل به S از "0" به "1" برود کانتر با مقدار PV ست میشود. با لبه بالا رونده ورودی CD مقدار کانتر یکی کاهش می یابد. با فعال شدن ورودی R کانتر به مقدار صفر ری ست میشود. وقتی مقدار شمارش بزرگتر از صفر است خروجی Q یک و وقتی مقدار شمارش صفر شود این خروجی صفر میگردد.

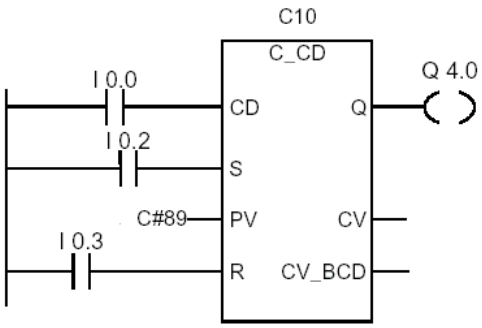
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

معادل LAD



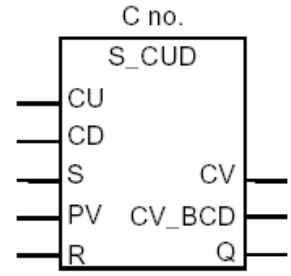
مثال



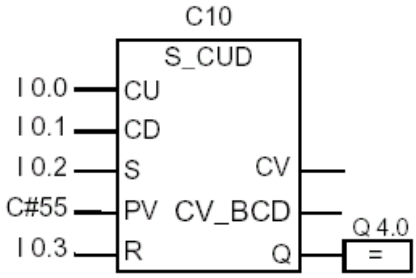
S_CUD Assign Parameter and Counter Up/Down

بلوک FBD:

فرمت:



مثال



Parameter	Data Type	Memory	شرح پارامتر
no.	COUNTER	C	شماره کانتر
CU	BOOL	I, Q, M, D, L	ورودی فعال کننده افزایش کانتر
CD	BOOL	I, Q, M, D, L	ورودی فعال کننده کاهش کانتر
S	BOOL	I, Q, M, D, L, T, C	ورودی برای ست کردن مقدار اولیه به کانتر
PV	WORD	I, Q, M, D, L or constant	مقدار اولیه بصورت BCD
R	BOOL	I, Q, M, D, L, T, C	ورودی ری ست کننده کانتر
CV	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (Hex)
CV_BCD	WORD	I, Q, M, D, L	مقدار لحظه ای کانتر بصورت باینری (BCD)
Q	BOOL	I, Q, M, D, L	خروجی نمایش وضعیت کانتر

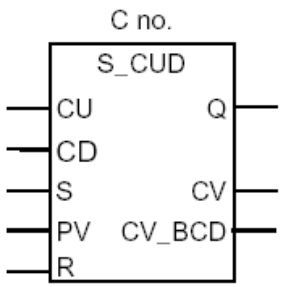
شرح:

وقتی که RLO برای ورودی متصل به S از "0" به "1" برود کانتر با مقدار PV ست میشود. . با لبه بالا رونده ورودی CU مقدار کانتر یکی افزایش می یابد و با لبه بالا رونده ورودی CD مقدار کانتر یکی کاهش می یابد. با فعال شدن ورودی R کانتر به مقدار صفر ریست میشود. وقتی مقدار شمارش بزرگتر از صفر است خروجی Q یک و وقتی مقدار شمارش صفر شود این خروجی صفر میگردد.

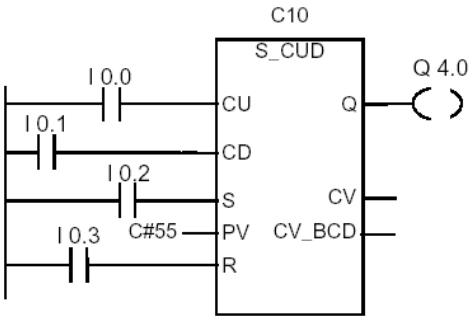
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	x	x	x	1

معادل LAD



مثال



۵-۵ دستورات دیتا بلاک ها (Data Block Instructions)

دیتا بلاک ها بر دو نوع هستند.

نوع اول : اشتراکی یا Shared DB که تمام بلاکهای برنامه نویسی مانند OB و FB و FC به آن دسترسی دارند.

نوع دوم : خاص یا Instance DB که خاص FB است و FB همراه با آن صدا زده میشود.

دستورات دیتا بلاک که مربوط به هر دو نوع DB فوق است و در صفحات بعد توضیح داده میشوند عبارتند از :

- **OPN** Open a Data Block
- **CDB** Exchange Shared DB and Instance DB
- **L DBLG** Load Length of Shared DB in ACCU 1
- **L DBNO** Load Number of Shared DB in ACCU 1
- **L DILG** Load Length of Instance DB in ACCU 1
- **L DINO** Load Number of Instance DB in ACCU 1

بجز دستور OPN سایر دستورات فوق معادل FBD و LAD ندارند.

OPN Open a Data Block

دستور STL :

فرمت:

OPN <Data Block>

Address	Data Block Type	Source Address
< Data Block >	DB , DI	1 to 65535

شرح :

دستور OPN برای باز کردن هر دو نوع دیتابلاک Shared و Instance بصورت زیر بکار میرود:

OPN DBn باز کردن دیتا بلاک Sharedشماره n

OPN DIn باز کردن دیتا بلاک Instanceشماره n

همزمان میتوان یک دیتا بلاک نوع Shared و یک دیتا بلاک نوع Instance را باز نمود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو دیتابلاک اشتراکی DB10 باز شده و از آن بیت با آدرس DBX0.0 خوانده شده و به خروجی Q4.0 ارسال میگردد.

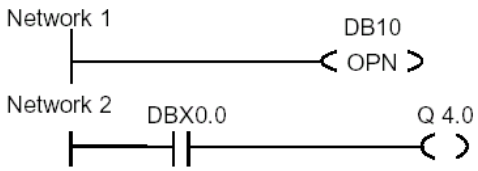
```
OPN   DB10
A     DBX0.0
=     Q4.0
```

معادل LAD

مثال

<DB no.> or <DI no.>

---(OPN)

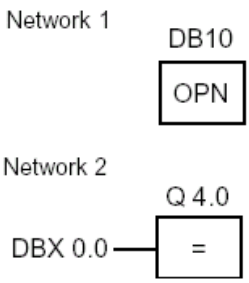


معادل FBD

مثال

<DB-Number> or
<DI-Number>

OPN



دستور STL :

CDB Exchange Shared DB and Instance DB

فرمت:

CDB

شرح :

دستور CDB رجیستر دیتابلاک Shared و Instance را با هم جابجا میکند. یعنی نوع Shared به نوع Instance تبدیل میشود و همینطور برعکس.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال روبرو قبل از عمل جابجایی دیتابلاک شماره ۱۰ از نوع Shared و دیتابلاک شماره ۲۰ از نوع Instance است ولی بعد از دستور CDB دیتابلاک شماره ۱۰ از نوع Instance و دیتابلاک شماره ۲۰ از نوع Shared خواهد بود.

```
OPN   DB10
OPN   DI20
CDB
```

دستور STL :

L DBLG Load Length of Shared DB in ACCU1

فرمت:

L DBLG

شرح :

دستور فوق ابتدا محتویات ACCU1 را در ACCU2 ذخیره کرده سپس طول دیتا بلاک Shared را به آکومولاتور ACCU1 بار میکند.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال رو برو ابتدا دیتابلاک شماره ۱۰ بصورت Shared باز میشود سپس طول آن به آکومولاتور بار شده و با عدد ذخیره شده در MD10 مقایسه میگردد اگر از آن کوچکتر بود به آدرس ERRO پرش مینماید.

```
OPN   DB10
L     DBLG
L     MD10
<     D
JC    ERRO
```

دستور STL :

L DBNO Load Number of Shared DB in ACCU1

فرمت:

```
L DBNO
```

شرح : دستور فوق ابتدا محتویات ACCU1 را در ACCU2 ذخیره کرده سپس شماره دیتا بلاک Shared را به آکومولاتور ACCU1 بار میکند وضعیت Status Wordمشابه صفحه قبل میباشد.

مثال :

در مثال رو برو ابتدا دیتابلاک شماره ۱۰ بصورت Shared باز میشود سپس شماره آن (یعنی عدد ۱۰) به آکومولاتور بار شده و از آنجا به MW0 ارسال میگردد.

```
OPN   DB10
L     DBNO
T     MW0
```

دستور STL :

L DILG Load Length of Instance DB in ACCU1

فرمت:

```
L DILG
```

شرح : دستور فوق ابتدا محتویات ACCU1 را در ACCU2 ذخیره کرده سپس طول دیتا بلاک Instance را به آکومولاتور ACCU1 بار میکند.

مثال :

در مثال رو برو ابتدا دیتابلاک شماره ۲۰ بصورت Instance باز میشود سپس طول آن به آکومولاتور بار شده و با عدد ذخیره شده در MW10 مقایسه میگردد اگر از آن کوچکتر بود به آدرس ERRO پرش مینماید.

```
OPN   DI20
L     DILG
L     MW10
<     I
JC    ERRO
```

دستور STL :

L DINO Load Number of Instance DB in ACCU1

فرمت:

```
L DINO
```

شرح : دستور فوق ابتدا محتویات ACCU1 را در ACCU2 ذخیره کرده سپس شماره دیتا بلاک Instance را به آکومولاتور ACCU1 بار میکند. وضعیت Status Wordمشابه صفحه قبل میباشد

مثال :

در مثال رو برو ابتدا دیتابلاک شماره ۲۰ بصورت Instance باز میشود سپس شماره آن (یعنی عدد ۲۰) به آکومولاتور بار شده و از آنجا به MW0 ارسال میگردد.

```
OPN   DI20
L     DBNO
T     MW0
```

٥-٦ دستورات کنترل لاجیک (Logic Control Instructions)

از دستورات پرش (jump) برای کنترل لاجیک برنامه میتوان استفاده کرد بگونه ای که در طول سیکل اسکن برنامه از روی برخی دستورات پرش کرده و آنها را اجرا نکند. بعلاوه میتوان دستور حلقه (LOOP) را برای اینکه بخشی از برنامه در یک سیکل اسکن چند بار اجرا شود استفاده نمود. دستورات پرش به آدرس محل پرش که Label نامیده میشود نیاز دارند همینطور دستور Loop.

نکاتی که در استفاده از دستورات jump و Loop باید مد نظر قرار گیرد عبارتند از :

- Label نباید از چهار کاراکتر بیشتر باشد.
- Label باید با حرف و نه با عدد شروع شود.
- نقطه ای که به آن پرش یا Loop انجام میشود با Label شروع شده و پس از آن علامت : قرار میگیرد (مثلاً Test:)
- پرش به جلو و عقب هردو امکان پذیر است.
- دستور پرش و Label مربوطه هردو باید در داخل یک بلاک باشند نه اینکه Label در بلاک دیگری تعریف شده باشد. همینطور برای دستور LOOP.
- نام Label باید در داخل بلاک منحصر بفرد باشد. استفاده از چند Label همنام در یک بلاک مجاز نیست.
- ماکزیمم فاصله بین دستور پرش و Label میتواند 32767 Word از Program Code باشد.

دستورات پرش را به ٤ دسته میتوان تقسیم کرد:

دسته اول : دستورات پرش بدون قید و شرط شامل :

- JU Jump Unconditional
- JL Jump to Labels

دسته دوم : دستورات پرش که مشروط به وضعیت RLO هستند شامل :

- JC Jump if RLO = 1
- JCN Jump if RLO = 0
- JCB Jump if RLO = 1 with BR
- JNB Jump if RLO = 0 with BR

دسته سوم : دستورات پرش که مشروط به وضعیت یک بیت از Status Word(بجز RLO) هستند شامل :

- JBI Jump if BR = 1
- JNBI Jump if BR = 0
- JO Jump if OV = 1
- JOS Jump if OS = 1

دسته چهارم : دستورات پرش که مشروط به نتیجه محاسبات هستند شامل :

- JZ Jump if Zero
- JN Jump if Not Zero
- JP Jump if Plus
- JM Jump if Minus
- JPZ Jump if Plus or Zero
- JMZ Jump if Minus or Zero
- JUO Jump if Unordered

JU Jump Unconditional

دستور STL :

فرمت:

JU <jump Label>

شرح :

دستور JU بدون توجه به وضعیت بیتهای Status Word به محل آدرس داده شده توسط Label پرش مینماید بهمین خاطر به آن دستور پرش بدون قید و شرط گفته میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال رو برو وقتی اجرای برنامه به دستور JU میرسد از روی دو سطر بعدی پرش کرده و به سطری که به آدرس FORW مشخص شده پرش مینماید.

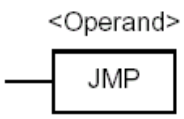
معمولاً وقتی در برنامه قبل از Label هایی که مربوط به دستورات jump هستند از دستور JU استفاده میشود زیرا اگر برنامه به سطر ماقبل این Label (در مثال روبرو DELE) رسید معنایش اینست که دستور پرش (در مثال روبرو JC DELE) اتفاق نیفتاده است و بهمین دلیل باید از روی دستورات مربوط به آن (که در این مثال بدنبال برچسب DELE) نوشته شده پرش بدون قید و شرط انجام شود.

```
        A    I1.0
        A    I1.2
        JC   DELE
        L    MB10
        INC  1
        T    MB10
        JU   FORW
DELE:   L    0
        T    MB10
FORW:   A    I2.1
```

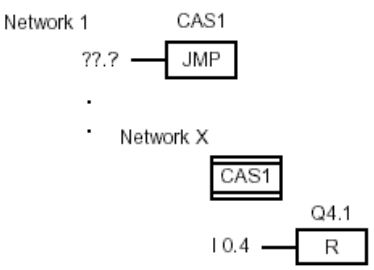
معادل LAD

مثال (به شکل Label دقت کنید)

معادل FBD



مثال



JL Jump to Labels

دستور STL :

فرمت:

JL <jump Label>

شرح :

دستور JL امکان برنامه ریزی برای چند پرش را در برنامه فراهم میسازد. بعد از دستور JL باید لیست محلهای پرش (ماکزیمم ۲۵۵ مورد) با دستور JU لیست شده باشد.پرش به اولین دستور JU وقتی اتفاق می افتد که ACCU1-L-L=0 باشد و پرش به دومین دستور JU وقتی اتفاق می افتد که ACCU1-L-L=1 باشد و بهمین ترتیب برای JU های بعدی باید ACCU1-L-L برترتیب مساوی 2، 3 ..... تا 254 باشد.اگر تعداد JU ها بیش از آخرین مقدار باشد دستور JL به اولین دستور بعد از انتهای لیست JU پرش میکند آدرس این سطر درجلوی JL نوشته میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال رو برو ابتدا محتوی MB0 به ACCU1-L-L بار میشود.

```
      L    MB0
      JL   LSTX
      JU   SEG0
      JU   SEG1
      JU   COMM
      JU   SEG3
LSTX: JU   COMM
SEG0: *
      *
      JU   COMM
SEG1: *
      *
      JU   COMM
SEG3: *
      *
COMM: *
```

اگر ACCU1-L-L>3 باشد به LSTX پرش میکند.

اگر ACCU1-L-L=0 باشد به SEG0 پرش میکند.

اگر ACCU1-L-L=1 باشد به SEG1 پرش میکند.

اگر ACCU1-L-L=2 باشد به COMM پرش میکند.

اگر ACCU1-L-L=3 باشد به SEG3 پرش میکند.

علت استفاده از JU در آخر هر قسمت قبلاً توضیح داده شد.

معادل LAD

مثال

ندارد

–

معادل FBD

مثال

ندارد

–

JC Jump if RLO=1

دستور STL :

فرمت:

JC <jump Label>

شرح : دستور JC در صورتیکه RLO=1 باشد به محل آدرس داده شده توسط Label پرش مینماید بهمین خاطر به آن دستور پرش مشروط (Conditional) گفته میشود.

وضعیت Status Word

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

مثال :

در مثال رو برو وقتی اجرای برنامه به دستور JC میرسد در صورتیکه RLO=1 باشد یعنی هر دو ورودی I1.0 و I1.2 یک باشند به آدرس JOVR پرش مینماید .

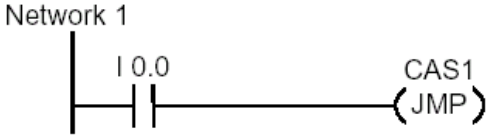
```
      A    I1.0
      A    I1.2
      JC   JOVR
      L    IW8
      T    MW22
JOVR: A    I2.1
```

معادل LAD

<label name>
---(JMP)

مثال

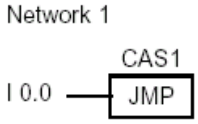
شبیه مثال ذکر شده برای JU است ولی پرش درصورتی انجام میشود که I0.0 یک باشد.



معادل FBD

<address>
JMP

مثال



JCN Jump if RLO=0

دستور STL :

فرمت:

JCN <jump Label>

شرح : دستور JCN در صورتیکه RLO=0باشد به محل آدرس داده شده توسط Label پرش مینماید یعنی عکس دستور JC است و بهمین خاطر به آن دستور پرش (Conditional Not) گفته میشود.وضعیت Status Word در آن مشابه دستور JC است. نام بلاکهای FBD و LAD در آن JMPN میباشد.

دستور STL : **JCB Jump if RLO=1 With BR**

فرمت:

JCB <jump Label>

شرح : دستور JCB مقدار RLO را در بیت BR از بیتهای Status Word کپی کرده و در صورتیکه RLO=1باشد به محل آدرس داده شده توسط Label پرش مینماید .وضعیت Status Word در آن مشابه دستور JC است

مثال :

در مثال رو برو وقتی اجرای برنامه به دستور JCB میرسد مقدار RLO را در BR ذخیره کرده و در صورتیکه RLO=1 باشد یعنی هر دو ورودی I1.0 و I1.2 یک باشند به آدرس JOVR پرش مینماید .

```
      A    I1.0
      A    I1.2
      JCB  JOVR
      L    IW8
      T    MW22
JOVR: A    I2.1
```

معادل LAD و FBD

مثال

سمبل خاص ندارد ولی میتوان آنرا با سمبلهای JMP و BR ایجاد کرد.

-

دستور STL : **JNB Jump if RLO=0 With BR**

فرمت:

JNB <jump Label>

شرح : دستور JNB مقدار RLO را در بیت BR از بیتهای Status Word کپی کرده و در صورتیکه RLO=0باشد به محل آدرس داده شده توسط Label پرش مینماید . وضعیت Status Word در آن مشابه دستور JC است

مثال :

در مثال رو برو وقتی اجرای برنامه به دستور JNB میرسد در صورتیکه RLO=0 باشد یعنی یکی از دو ورودی I1.0 و I1.2 صفر باشد به آدرس JOVR پرش مینماید .

```
      A    I1.0
      A    I1.2
      JNB  JOVR
      L    IW8
      T    MW22
JOVR: A    I2.1
```

معادل LAD و FBD

مثال

سمبل خاص ندارد ولی میتوان آنرا با سمبلهای JMPN و BR ایجاد کرد.

-

JBI Jump if BR=1

دستور STL :

فرمت:

JBI <jump Label>

شرح : دستور JBI در صورتیکه BR=1 باشد به محل آدرس داده شده توسط Label پرش مینماید

وضعیت Status Word

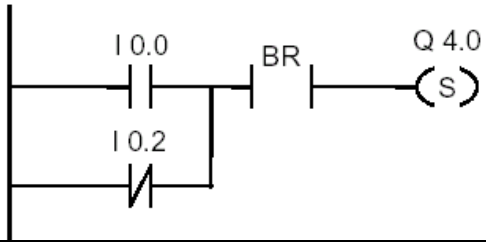
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	1	-	0

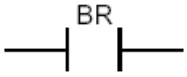
مثال :

در مثال رو برو وقتی اجرای برنامه به دستور JBI میرسد در صورتیکه BR=1 باشد (یعنی قبلاً در جایی از برنامه یا در انتهای بلاک FC1 با دستور SAVE مقدار یک در BR ذخیره شده باشد) به آدرس Test پرش مینماید .

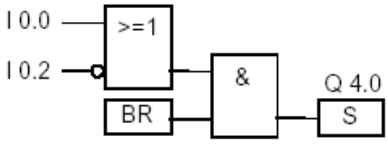
```
      CALL   FC1
      JBI    TEST
      *
      JU     COMM
Test: A      I2.2
      *
COMM: =
```

معادل LAD (دستور پرش مستقیم ندارد ولی میتوان BR را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان BR را چک کرد و سپس با JMP پرش نمود) **مثال**



JNBI Jump if BR=0

دستور STL :

فرمت:

JNBI <jump Label>

شرح : دستور JNBI در صورتیکه BR=0 باشد به محل آدرس داده شده توسط Label پرش مینماید . وضعیت status Word مانند دستور JBI میباشد.

معادل LAD و FBD **مثال**

شبیه دستور JBI -

دستور STL : JO Jump if OV=1

فرمت:

JO <jump Label>

شرح :

دستور JO در صورتیکه بیت OV از بیتهای Status Word یک باشد یعنی سرریزی (Overflow) اتفاق افتاده باشد به محل آدرس داده شده توسط Label پرش مینماید . توصیه میشود بعد از دستورات محاسباتی برای کنترل اینکه نتیجه در رنج مجاز هست یا نه از دستور JO یا دستور JOS استفاده شود.

وضعیت Status Word

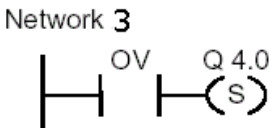
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

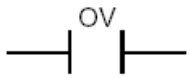
مثال :

در مثال رو برو اگر حاصل ضرب محتوی MW10 در عدد ۳ از حد مجاز بیشتر شود بیت OV=1 میشود ووقتی اجرای برنامه به دستور JO میرسد به آدرس OVER پرش مینماید .

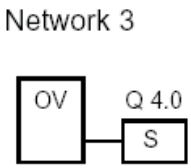
```
      L    MW10
      L    3
      *I
      JO   OVER
      T    MW10
      *
      JU   NEXT
OVER: AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

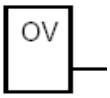
معادل LAD (دستور پرش مستقیم ندارد ولی میتوان OV را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان OV را چک کرد و سپس با JMP پرش نمود) **مثال**





JOS Jump if OS=1

دستور STL :

فرمت:

JOS <jump Label>

شرح :

دستور JOS در صورتیکه بیت OS از بیتهای Status Word یک باشد یعنی قبلاً سرریزی (Overflow) اتفاق افتاده باشد به محل آدرس داده شده توسط Label پرش مینماید . توصیه میشود بعد از دستورات محاسباتی برای کنترل اینکه نتیجه در رنج مجاز هست یا نه از دستور JO یا دستور JOS استفاده شود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

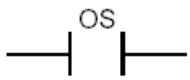
مثال :

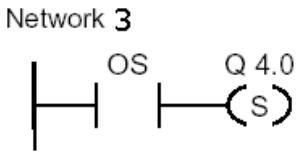
در مثال رو برو اگر حاصل ضرب محتوی MW10 در عدد ۳ از حد مجاز بیشتر شودیا اگر نتیجه تقسیم MW200 بر 50 از حد مجاز تجاوز کند بیت OS=1 میشود ووقتی اجرای برنامه به دستور JOS میرسد به آدرس OVER پرش مینماید .

اگر بجای دستور JOS در این برنامه دستور JO بکار میرفت سرریزی را فقط در عملیات آخر یعنی عملیات تقسیم میدید و سرریزی عملیات ضرب کنترل نمیشد ولی با دستور OS حتی اگر در عملیات قبلی سرریزی اتفاق افتاده باشد برنامه قابل کنترل است.

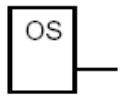
```
      L    MW10
      L    3
      *I
      L    MW200
      L    50
      /I
      JOS  OVER
      T    MW10
      *
      JU   NEXT
OVER: AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

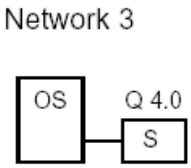
معادل LAD (دستور پرش مستقیم ندارد ولی میتوان OS را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان OS را چک کرد و سپس با JMP پرش نمود) **مثال**





JZ Jump if Zero

دستور STL:

فرمت:

JZ <jump Label>

شرح:

دستور JZ در صورتیکه نتیجه محاسبات صفر باشد به محل آدرس داده شده توسط Label پرش مینماید. صفر بودن نتیجه محاسبات با بیتهای زیر از Status Word مشخص میگردد:

CC1=0	CC0=0

وضعیت Status Word

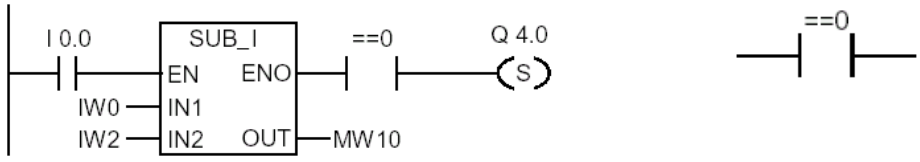
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

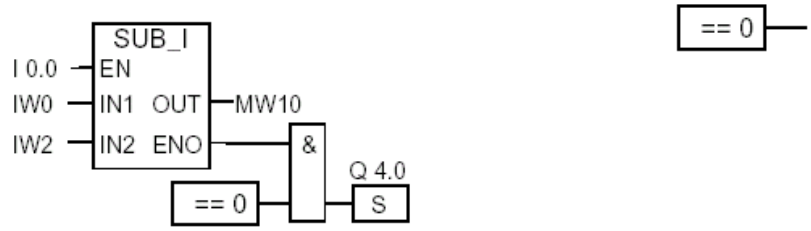
در مثال رو برو اگر نتیجه تفریق MW10 و عدد 30 صفر شود وقتی اجرای برنامه به دستور JZ میرسد به آدرس ZERO پرش مینماید.

```
      L    MW10
      L    30
      -I
      JZ   ZERO
      T    MW10
      *
      JU   NEXT
ZERO: AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

معادل LAD (دستور پرش مستقیم ندارد ولی میتوان صفر بودن را چک کرد و سپس با JMP پرش نمود) **مثال**



معادل FBD (دستور پرش مستقیم ندارد ولی میتوان صفر بودن را چک کرد و سپس با JMP پرش نمود) **مثال**



JN Jump if Not Zero

دستور STL :

فرمت:

JN <jump Label>

شرح :

دستور JN در صورتیکه نتیجه محاسبات صفر نباشد به محل آدرس داده شده توسط Label پرش مینماید . صفر نبودن نتیجه محاسبات به معنی بزرگتر یا کوچکتر از صفر بودن است که با بیتهای زیر از Status Word مشخص میگردد:

وقتی نتیجه کوچکتر از صفر است	CC0=1	CC1=0
وقتی نتیجه بزرگتر از صفر است	CC0=0	CC1=1

وضعیت Status Word

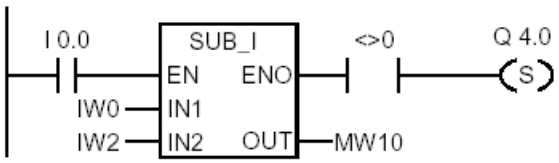
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

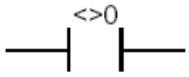
مثال :

در مثال رو برو اگر نتیجه تفریق MW10 و عدد 30 صفر نشود وقتی اجرای برنامه به دستور JN میرسد به آدرس NOZE پرش مینماید .

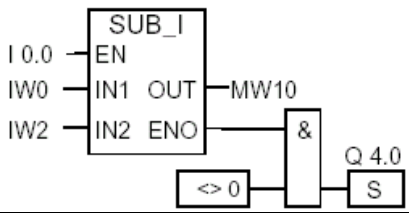
```
      L    MW10
      L    30
      -I
      JN   NOZE
      T    MW10
      *
      JU   NEXT
NOZE: AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

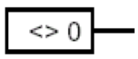
معادل LAD (دستور پرش مستقیم ندارد ولی میتوان صفر نبودن را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان صفر نبودن را چک کرد و سپس با JMP پرش نمود) **مثال**





JP Jump if Plus

دستور STL:

فرمت:

JP <jump Label>

شرح:

دستور JP در صورتیکه نتیجه محاسبات بزرگتر از صفر باشد به محل آدرس داده شده توسط Label پرش مینماید . مثبت بودن نتیجه محاسبات با بیتهای زیر از Status Word مشخص میگردد:

CC1=1	CC0=0

وضعیت Status Word

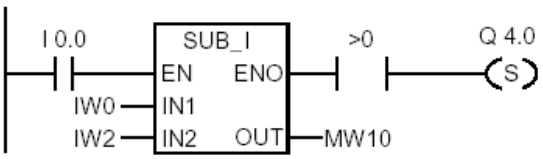
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

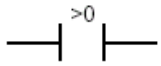
مثال:

در مثال رو برو اگر نتیجه تفریق MW10 و عدد 30 مثبت باشد وقتی اجرای برنامه به دستور JP میرسد به آدرس POS پرش مینماید .

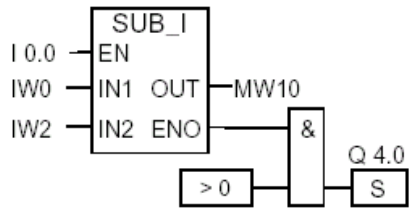
```
      L    MW10
      L    30
      -I
      JP   POS
      T    MW10
      *
      JU   NEXT
POS:  AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

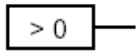
معادل LAD (دستور پرش مستقیم ندارد ولی میتوان مثبت بودن را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان مثبت بودن را چک کرد و سپس با JMP پرش نمود) **مثال**





JM Jump if Minus

دستور STL:

فرمت:

JM <jump Label>

شرح:

دستور JM در صورتیکه نتیجه محاسبات کوچکتر از صفر باشد به محل آدرس داده شده توسط Label پرش مینماید. منفی بودن نتیجه محاسبات با بیتهای زیر از Status Word مشخص میگردد:

CC1=0	CC0=1

وضعیت Status Word

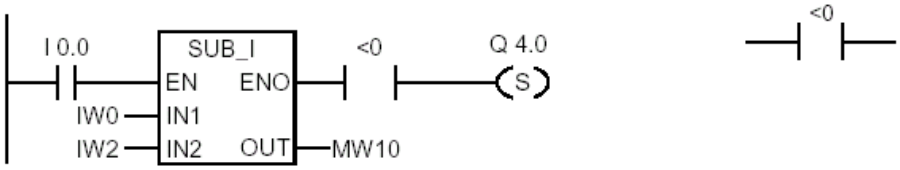
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

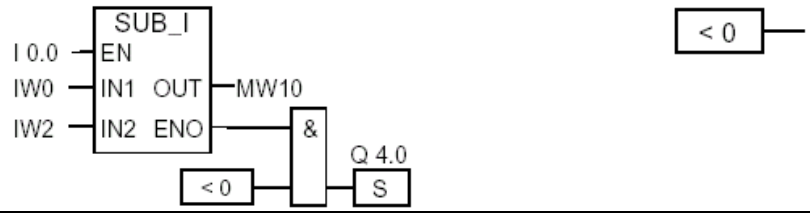
در مثال رو برو اگر نتیجه تفریق MW10 و عدد 30 منفی باشد وقتی اجرای برنامه به دستور JM میرسد به آدرس NEG پرش مینماید.

```
      L    MW10
      L    30
      -I
      JM   NEG
      T    MW10
      *
      JU   NEXT
NEG:  AN   M4.0
      S    Q4.0
NEXT: NOP  0
```

معادل LAD (دستور پرش مستقیم ندارد ولی میتوان منفی بودن را چک کرد و سپس با JMP پرش نمود) **مثال**



معادل FBD (دستور پرش مستقیم ندارد ولی میتوان منفی بودن را چک کرد و سپس با JMP پرش نمود) **مثال**



JPZ Jump if Plus Or Zero

دستور STL :

فرمت:

JPZ <jump Label>

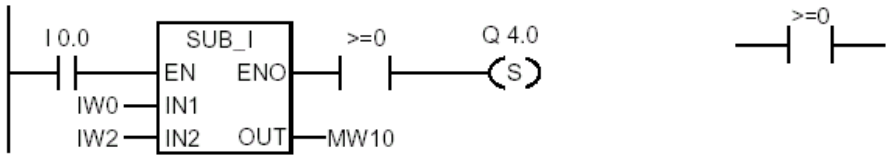
شرح :

دستور JPZ در صورتیکه نتیجه محاسبات بزرگتر یا مساوی صفر باشد به محل آدرس داده شده توسط Label پرش مینماید . مثبت یا صفر بودن نتیجه محاسبات با بیتهای زیر از Status Word مشخص میگردد:

وقتی نتیجه صفر است	CC0=0	CC1=0
وقتی نتیجه بزرگتر از صفر است	CC0=0	CC1=1

معادل LAD و FBD (دستور پرش مستقیم ندارد ولی میتوان مثبت یا صفر بودن را چک کرد و سپس با JMP پرش نمود)

مثال LAD



JMZ Jump if Plus Or Zero

دستور STL :

فرمت:

JMZ <jump Label>

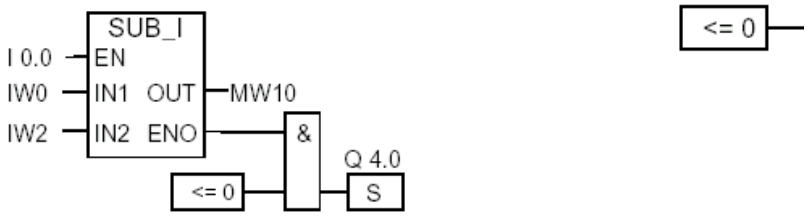
شرح :

دستور JMZ در صورتیکه نتیجه محاسبات کوچکتر یا مساوی صفر باشد به محل آدرس داده شده توسط Label پرش مینماید . منفی یا صفر بودن نتیجه محاسبات با بیتهای زیر از Status Word مشخص میگردد:

وقتی نتیجه صفر است	CC0=0	CC1=0
وقتی نتیجه بزرگتر از صفر است	CC0=0	CC1=1

معادل LAD و FBD (دستور پرش مستقیم ندارد ولی میتوان منفی یا صفر بودن را چک کرد و سپس با JMP پرش نمود)

مثال FBD



JUO Jump if Unordered

دستور STL:

فرمت:

JUO <jump Label>

شرح:

دستور JUO در صورتیکه در صورتی که بیتهای CC1 و CC0 از Status Word هر دو یک باشند به محل آدرس داده شده توسط Label پرش مینماید . این حالت وقتی اتفاق می افتد که :

- تقسیم بر صفر انجام شود
- دستور غیر مجازی بکار رود
- فرمت غیر مجاز برای عدد اعشاری استفاده شود

وضعیت Status Word

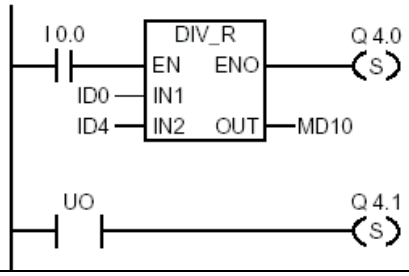
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

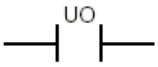
مثال:

در مثال رو برو اگر ID4=0 باشد تقسیم بر صفر اتفاق می افتد و وقتی اجرای برنامه به دستور JUO میرسد به آدرس ERRO پرش مینماید.

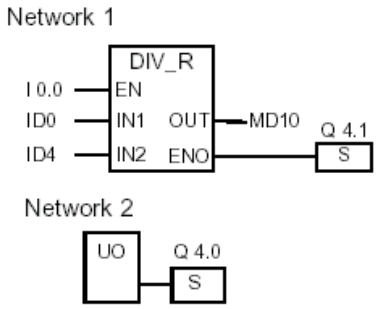
```
      L    ID0
      L    ID4
     /D
    JUO   ERRO
      T    MD10
     JU    NEXT
ERRO: S    Q4.1
NEXT: NOP  0
```

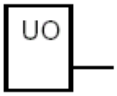
معادل LAD (دستور پرش مستقیم ندارد ولی میتوان Unorder را چک کرد و سپس با JMP پرش نمود) **مثال**





معادل FBD (دستور پرش مستقیم ندارد ولی میتوان Unorder بودن را چک کرد و سپس با JMP پرش نمود) **مثال**





دستور STL : **LOOP Loop**

فرمت:

LOOP <jump Label>

شرح :

دستور LOOP از مقدار ACCU1-L یکی کم میکند و در صورتی که نتیجه مخالف صفر باشد پرش انجام میدهد. این پرش ادامه می یابد تا زمانی که ACCU1-L=0 شود.

عدد مربوط به تعداد تکرار لوپ قبل از حلقه به آکومولاتور فوق بار میشود.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال :

در مثال رو برو عدد 5 بعنوان شاخص حلقه به آکومولاتور بار میشود سپس حلقه که هر بار مقدار MD20 را با خودش جمع میکند پنج بار تکرار میگردد .

```
      *
      L    5
NEXT: T    MW10
      L    MD20
      +    D
      T    MD20
      L    MW10
      LOOP NEXT
      *
```

معادل LAD و FBD (دستور پرش مستقیم ندارد ولی میتوان منفی یا صفر بودن را چک کرد و سپس با JMP پرش نمود)

۵-۷ دستورات محاسبات عدد صحیح (Integer Math Instructions)

دستورات محاسباتی آکومولاتورهای ۱ و ۲ را با هم ترکیب کرده و نتیجه را در آکومولاتور ۱ ذخیره میسازد. برای این منظور ابتدا عدد اول به آکومولاتور ۱ بار میشود سپس محتویات آکومولاتور ۱ به آکومولاتور ۲ شیفت پیدا میکند و عدد دوم به آکوملاتور ۲ بار میشود بعداز انجام عمل محاسباتی نتیجه در آکومولاتور ۱ ذخیره شده و آکومولاتور ۲ همچنان مقدار قبلی خود که در واقع همان عدد اولی است را حفظ مینماید.در حالتی که CPU دارای ٤ آکومولاتور است محتویات آکومولائور ۳ در آکومولاتور ۲ کپی میشود و محتویات آکومولائور ٤ در آکومولاتور ۳ کپی میگردد. دستوراتی که عملیات محاسباتی را روی عدد صحیح انجام میدهند عبارتند از :

- **+I** Add ACCU 1 and ACCU 2 as Integer (16-bit)
- **-I** Subtract ACCU 1 from ACCU 2 as Integer (16-bit)
- ***I** Multiply ACCU 1 and ACCU 2 as Integer (16-bit)
- **/I** Divide ACCU 2 by ACCU 1 as Integer (16-bit)
- **+** Add Integer Constant (16, 32 Bit)
- **+D** Add ACCU 1 and ACCU 2 as Double Integer (32-bit)
- **-D** Subtract ACCU 1 from ACCU 2 as Double Integer (32-bit)
- ***D** Multiply ACCU 1 and ACCU 2 as Double Integer (32-bit)
- **/D** Divide ACCU 2 by ACCU 1 as Double Integer (32-bit)
- **MOD** Division Remainder Double Integer (32-bit)

این دستورات بیتهای OS , OV , CC1,CC0 را تحت تاثیر قرار میدهند مطابق جدول و جداول صفحه بعد زیر:

وقتی نتیجه در رنج مجاز است	CC 1	CC 0	OV	OS
0 (zero)	0	0	0	*
16 bits: -32 768 <= result< 0 (negative number)	0	1	0	*
32 bits: -2 147 483 648 <=result < 0 (negative number)				
16 bits: 32 767 >= result > 0 (positive number)	1	0	0	*
32 bits: 2 147 483 647 >= result > 0 (positive number)				

*: در این حالت بیت osتحت تاثیر قرار نمیگیرد.

وقتی نتیجه در رنج مجاز نیست	CC 1	CC 0	OV	OS
Underflow (addition) 16 bits: result = -65536 32 bits: result = -4 294 967 296	0	0	1	1
Underflow (multiplication) 16 bits: result < -32 768 (negative number) 32 bits: result < -2 147 483 648 (negative number)	0	1	1	1
Overflow (addition, subtraction) 16 bits: result > 32 767 (positive number) 32 bits: result > 2 147 483 647 (positive number)	0	1	1	1
Overflow (multiplication, division) 16 bits: result > 32 767 (positive number) 32 bits: result > 2 147 483 647 (positive number)	1	0	1	1
Underflow (addition, subtraction) 16 bits: result < -32. 768 (negative number) 32 bits: result < -2 147 483 648 (negative number)	1	0	1	1
Division by 0	1	1	1	1

Operation	CC 1	CC 0	OV	OS
+D: result = -4 294 967 296	0	0	1	1
/D or MOD: division by 0	1	1	1	1

دستور STL :

+I Add ACCU1 And ACCU2 as Integer(16-Bit)

فرمت:

+I

شرح :

دستور +I محتویات ACCU1-L و ACCU2-L را با هم جمع کرده و نتیجه را در ACCU1-L ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۱۶ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

وضعیت Status Word

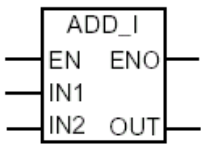
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال :

در مثال رو برو مقادیر MW0 و MW2 با هم جمع شده و نتیجه در MW10 ذخیره میشود.

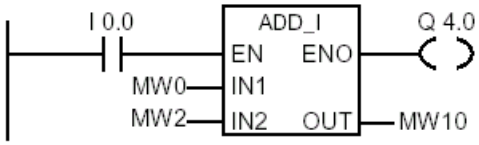
```
L   MW0
L   MW2
+   I
T   MW10
```

معادل LAD

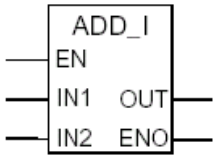


مثال

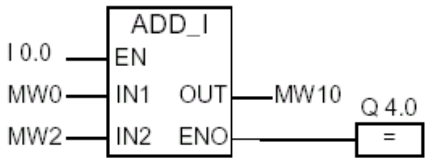
با I0.0=1عمل جمع انجام شده درحالت عادی خروجی Q4.0 یک است ودر صورتیکه نتیجه خارج از رنج باشد (مثلا بزرگتر از 32768)خروجی Q4.0 صفر میشود.



معادل FBD



مثال

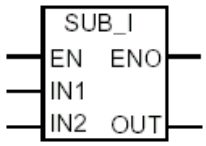


دستور STL : **-I Subtract ACCU1 from ACCU2 as Integer(16-Bit)**

فرمت: **-I**

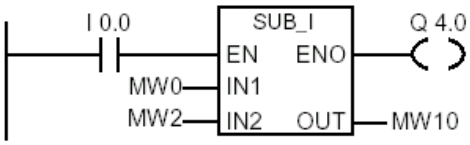
شرح : دستور -I مقدار ACCU1-L را ازمقدار ACCU2-L کم کرده و نتیجه را در ACCU1-L ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۱۶ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل LAD



مثال

با I0.0=1عمل تفریق انجام شده درحالت عادی خروجی Q4.0یک است ودر صورتیکه نتیجه خارج از رنج باشد خروجی Q4.0 صفر میشود.

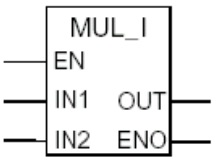


دستور STL : ***I Multiply ACCU1 And ACCU2 as Integer(16-Bit)**

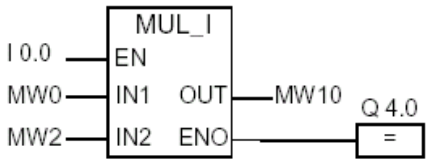
فرمت: ***I**

شرح : دستور *I محتویات ACCU1-L و ACCU2-L را در هم ضرب کرده و نتیجه را در ACCU1-L ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۱۶ بیتی هستند. تاثیر آن روی Status Word شبیه دستور قبلی است.

معادل FBD



مثال



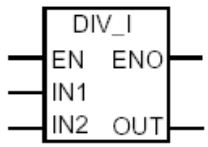
دستور STL : **/I Divide ACCU2 by ACCU1 as Integer(16-Bit)**

فرمت:

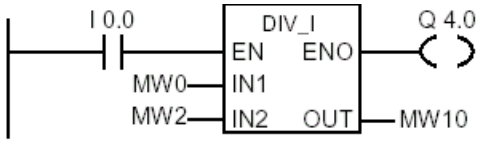
/ I

شرح : دستور / I مقدار ACCU2-L را برمقدار ACCU1-L تقسیم کرده خارج قسمت در ACCU1-L و باقیمانده در ACCU1-H ذخیره میشود. تاثیر آن روی Status Word شبیه دستور بالاست..

معادل LAD



مثال



دستور STL: **+ Add Integer Constant (16 , 32Bit)**

فرمت:

+

شرح:

دستور + عدد صحیح ثابت را با محتویات ACCU1-L جمع کرده و نتیجه را در ACCU1-L ذخیره میکند. محتوای ACCU2-L در طول این عمل تغییر نمیکند. این دستور برای دونوع عدد صحیح زیر بکار میرود:

عدد صحیح ۱۶ بیتی بین -32768 تا +32767

عدد صحیح ۳۲ بیتی بین -2,147,483,648 تا +2,147,483,647

وضعیت Status Word

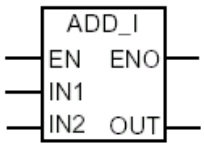
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

در مثال رو برو مقادیر MW0 و عدد ۲۵۸۹ با هم جمع شده و نتیجه در MW10 ذخیره میشود.

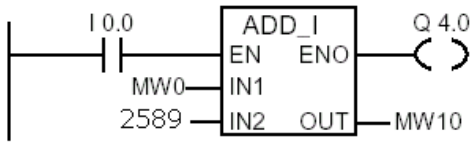
```
L   MW0
+   2589
T   MW10
```

معادل LAD

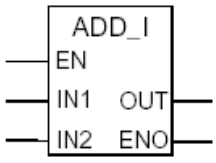


مثال

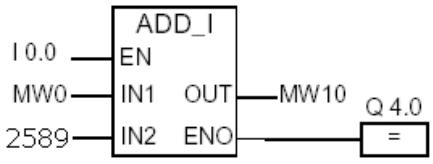
با I0.0=1عمل جمع انجام شده درحالت عادی خروجی Q4.0 یک است ودر صورتیکه نتیجه خارج از رنج باشد خروجی Q4.0 صفر میشود.



معادل FBD



مثال



+D Add ACCU1 And ACCU2 as Double Integer(32-Bit)

دستور STL:

فرمت:

+D

شرح:

دستور +D محتویات ACCU1 و ACCU2 را با هم جمع کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

وضعیت Status Word

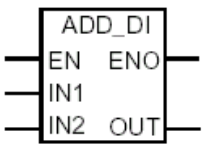
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

در مثال رو برو مقادیر MD0 و MD4 با هم جمع شده و نتیجه در MD10 ذخیره میشود.

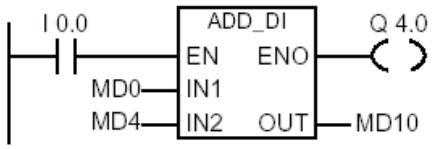
```
L   MD0
L   MD4
+   D
T   MD10
```

معادل LAD

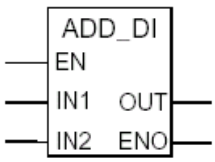


مثال

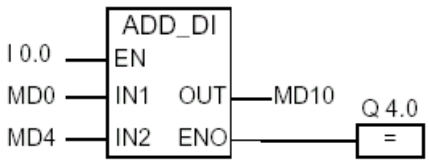
با I0.0=1 عمل جمع انجام شده درحالت عادی خروجی Q4.0 یک است ودر صورتیکه نتیجه خارج از رنج باشد خروجی Q4.0 صفر میشود.



معادل FBD



مثال



دستور STL : **-D Subtruct ACCU1 from ACCU2 as Double Integer(32-Bit)**

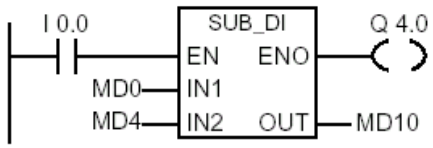
فرمت:

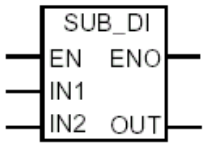
-D

شرح : دستور -D محتویات ACCU1 را از ACCU2 کم کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد

معادل LAD

مثال





دستور STL : ***D Multiply ACCU1 And ACCU2 as Double Integer(32-Bit)**

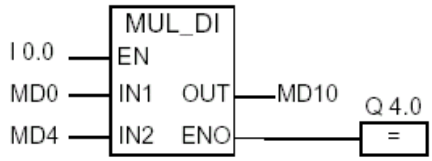
فرمت:

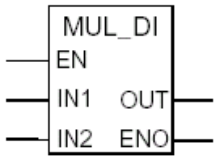
***D**

شرح : دستور *D محتویات ACCU1 و ACCU2 را در هم ضرب کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد صحیح ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل FBD

مثال





دستور STL : **/D Divide ACCU2 by ACCU1 as Double Integer(32-Bit)**

فرمت:

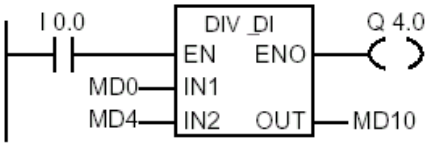
/D

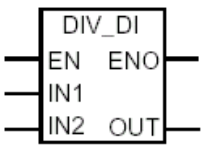
شرح :

دستور /D محتوی ACCU2 را بر محتوی ACCU1 تقسیم میکند خارج قسمت در ACCU1 و باقیمانده در ACCU2 ذخیره میشود. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل LAD

مثال





دستور STL:

MOD Division Remainder Double Integer(32-Bit)

فرمت:

MOD

شرح:

دستور MOD برای محاسبه باقیمانده تقسیم دو عدد صحیح ۳۲ بیتی بکار میرود. این دستور محتوی ACCU2 را بر محتوی ACCU1 تقسیم کرده و باقیمانده را در ACCU1 ذخیره میکند. خارج قسمت ذخیره نمیگردد. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

وضعیت Status Word

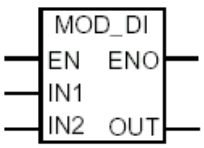
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

در مثال رو برو مقدار MD0 بر مقدار MD4 تقسیم شده و باقیمانده در MD10 ذخیره میشود. بعنوان نمونه اگر MD0=13 و MD4=4 باشد MD10=1 خواهد بود.

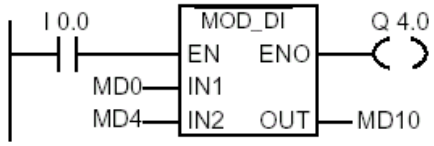
```
L   MD0
L   MD4
/   D
T   MD10
```

معادل LAD

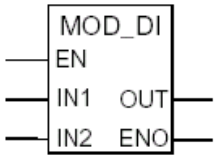


مثال

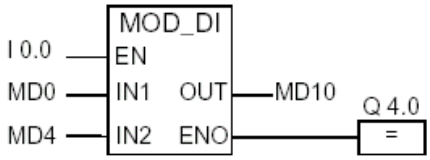
با I0.0=1 باقیمانده تقسیم محاسبه شده درحالت عادی خروجی Q4.0 یک است ودر صورتیکه نتیجه خارج از رنج باشد خروجی Q4.0 صفر میشود.



معادل FBD



مثال



۵-۸ دستورات محاسبات عدد اعشاری (Floating-Point Math Instructions)

دستورات محاسبات اعشاری نیز شبیه دستورات محاسباتی عدد صحیح آکومولاتورهای ۱ و ۲ را با هم ترکیب کرده و نتیجه را در آکومولاتور ۱ ذخیره میسازد. برای این منظور ابتدا عدد اول به آکومولاتور ۱ بار میشود سپس محتویات آکومولاتور ۱ به آکومولاتور ۲ شیفت پیدا میکند و عدد دوم به آکومولاتور ۲ بار میشود بعداز انجام عمل محاسباتی نتیجه در آکومولاتور ۱ ذخیره شده و آکومولاتور ۲ همچنان مقدار قبلی خود که در واقع همان عدد اولی است را حفظ مینماید.در حالتی که CPU دارای ٤ آکومولاتور است محتویات آکومولاتور ۳ در آکومولاتور ۲ کپی میشود و محتویات آکومولائور ٤ در آکومولاتور ۳ کپی میگردد. دستوراتی که عملیات محاسباتی را روی عدد اعشاری انجام میدهند عبارتند از :

- **+R** Add ACCU 1 and ACCU
- **-R** Subtract ACCU 1 from ACCU 2
- ***R** Multiply ACCU 1 and ACCU 2
- **/R** Divide ACCU 2 by ACCU 1

علاوه بر دستورات فوق توابع ریاضی استاندارد را نیز میتوان برای اعداد اعشاری بکار برد که عبارتند از :

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

این دستورات بیتهای OS , OV , CC1,CC0 را تحت تاثیر قرار میدهند مطابق جداول زیر:

وقتی نتیجه در رنج مجاز است	CC 1	CC 0	OV	OS
+0, -0 (Null)	0	0	0	*
-3.402823E+38 < result < -1.175494E-38 (negative number)	0	1	0	*
+1.175494E-38 < result < 3.402824E+38 (positive number)	1	0	0	*

*: در این حالت بیت os تحت تاثیر قرار نمیگیرد.

وقتی نتیجه در رنج مجاز نیست	CC 1	CC 0	OV	OS
Underflow -1.175494E-38 < result < - 1.401298E-45 (negative number)	0	0	1	1
Underflow +1.401298E-45 < result < +1.175494E-38 (positive number)	0	0	1	1
Overflow Result < -3.402823E+38 (negative number)	0	1	1	1
Overflow Result > 3.402823E+38 (positive number)	1	0	1	1
Not a valid floating-point number or illegal instruction (input value outside the valid range)	1	1	1	1

دستور STL: +R Add ACCU1 And ACCU2 as Floating-Point (32-Bit)

فرمت:

+R

شرح:

دستور +R محتویات ACCU1 و ACCU2 را با هم جمع کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد اعشاری ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

وضعیت Status Word

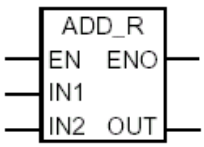
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

در مثال رو برو مقادیر اعشاری MD0 و MD4 با هم جمع شده و نتیجه در MD10 ذخیره میشود.

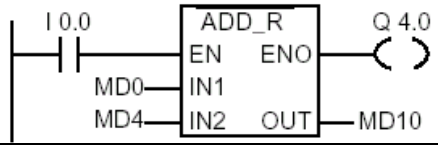
```
L   MD0
L   MD4
+   R
T   MD10
```

معادل LAD

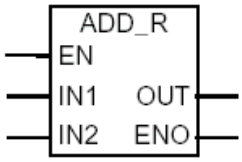


مثال

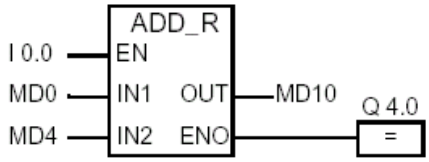
با I0.0=1 عمل جمع انجام شده درحالت عادی خروجی Q4.0 یک است ودر صورتیکه نتیجه خارج از رنج باشد خروجی Q4.0 صفر میشود.



معادل FBD



مثال



دستور STL :

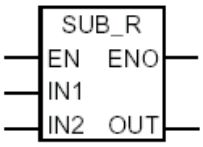
-R Subtruct ACCU1 from ACCU2 as Floating-Point (32-Bit)

فرمت: -R

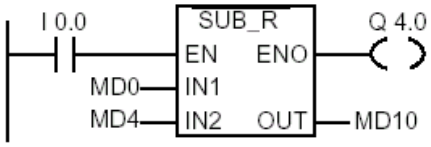
شرح :

دستور -R محتویات ACCU1 را از ACCU2 کم کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد اعشاری ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل LAD



مثال



دستور STL :

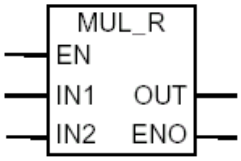
*R Multiply ACCU1 And ACCU2 as Floating-Point (32-Bit)

فرمت: *R

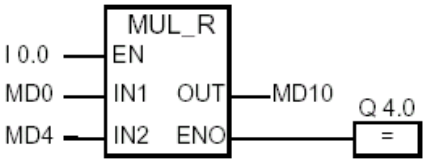
شرح :

دستور R* محتویات ACCU1 و ACCU2 را در هم ضرب کرده و نتیجه را در ACCU1 ذخیره میکند. هر دو آکومولاتور فوق محتوی اعداد اعشاری ۳۲ بیتی هستند. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل FBD



مثال



دستور STL :

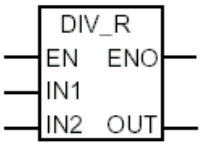
/R Divide ACCU2 by ACCU1 as Floating-Point (32-Bit)

فرمت: /R

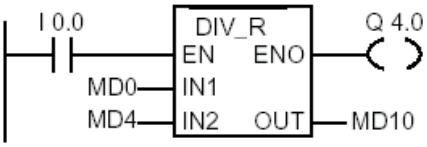
شرح :

دستور R/ محتوی ACCU2 را بر محتوی ACCU1 تقسیم میکند خارج قسمت در ACCU1 و باقیمانده در ACCU2 ذخیره میشود. انجام دستور تاثیری روی RLO نمیگذارد ولی بیتهای CC1,CC0, OV,OS را تغییر میدهد.

معادل LAD



مثال



دستور STL:

ABS Absolute Value of Floating-Point (32-Bit)

فرمت:

ABS

شرح:

دستور ABS قدر مطلق محتوی ACCU1 را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور تاثیری روی بیتهای Staut Word نمیگذارد .

وضعیت Status Word

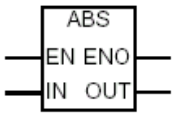
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

مثال:

در مثال رو برو قدر مطلق مقدار MD8 در MD10 ذخیره میشود.

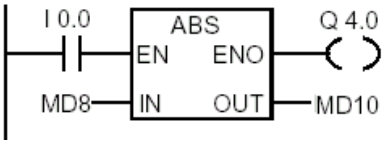
```
L    MD8
ABS
T    MD10
```

معادل LAD

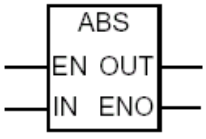


مثال

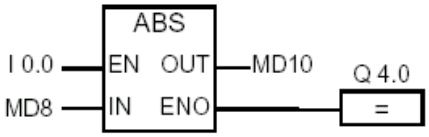
با I0.0=1 فانکشن انجام شده ودر صورتیکه تبدیل انجام نشود خروجی Q4.0 صفر میشود.



معادل FBD



مثال



SQR Square of Floating-Point (32-Bit)

دستور STL:

فرمت:

SQR

شرح: دستور SQR توان دوم محتوی ACCU1 را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

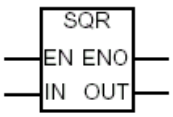
در مثال رو برو مجذور مقدار MD0 در MD10 ذخیره میشود.

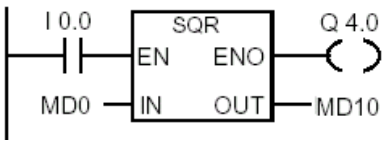
```
L    MD0
SQR
T    MD10
```

معادل LAD

مثال

با I0.0=1 فانکشن انجام شده ودر صورتیکه ورودی اعشاری نباشد یا تبدیل انجام نشود خروجی Q4.0 صفر میشود.





SQRT Square Root of Floating-Point (32-Bit)

دستور STL:

فرمت:

SQRT

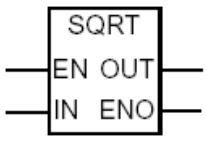
شرح:

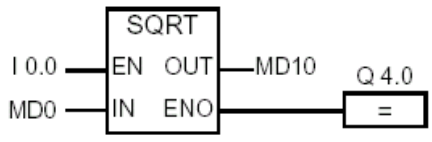
دستور SQRT ریشه دوم محتوی ACCU1 را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

معادل FBD

مثال

با I0.0=1 فانکشن انجام شده ودر صورتیکه منفی باشد یا اعشاری نباشد خروجی Q4.0 صفر میشود.





EXP Exponential Value of Floating-Point (32-Bit)

دستور STL:

فرمت:

EXP

شرح:

دستور EXP مقدار e به توان محتوی ACCU1 را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

وضعیت Status Word

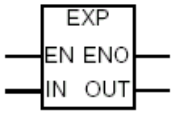
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال:

در مثال رو برو e به توان مقدار MD0 در MD10 ذخیره میشود.

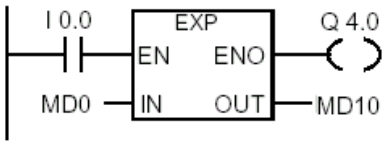
```
L    MD0
EXP
T    MD10
```

معادل LAD



مثال

با I0.0=1فانکشن انجام شده ودر صورتیکه ورودی اعشاری نباشد یا تبدیل انجام نشود خروجی Q4.0 صفر میشود.



LN Natural Logarithm of Floating-Point (32-Bit)

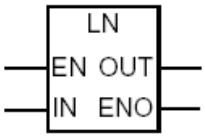
دستور STL:

فرمت:

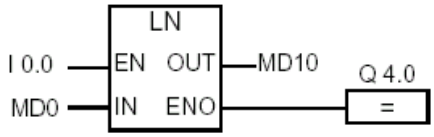
LN

شرح: دستور LN مقدر لگاریتم طبیعی محتوی ACCU1 را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد

معادل FBD



مثال



دستور STL:

SIN Sine of Angles as Floating-Point (32-Bit)

فرمت:

SIN

شرح :

دستور SIN مقدار سینوس محتوی ACCU1 که برحسب رادیان فرض میشود را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

وضعیت Status Word

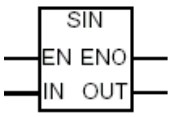
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال :

در مثال رو برو سینوس مقدار MD0 در MD10 ذخیره میشود.

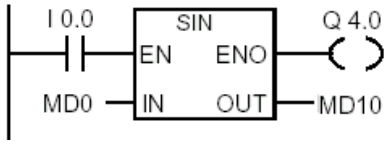
```
L    MD0
SIN
T    MD10
```

معادل LAD

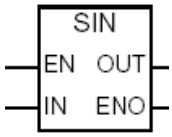


مثال

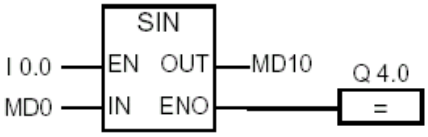
با I0.0=1فانکشن انجام شده ودر صورتیکه ورودی اعشاری نباشد یا تبدیل انجام نشود خروجی Q4.0 صفر میشود.



معادل FBD



مثال



دستور STL:

COS Cosine of Angles as Floating-Point (32-Bit)

فرمت:

COS

شرح:

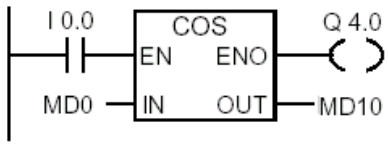
دستور COS مقدار کسینوس محتوی ACCU1 که برحسب رادیان فرض میشود را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

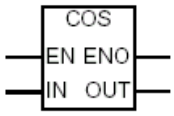
مثال:

در مثال رو برو کسینوس مقدار MD0 در MD10 ذخیره میشود.

```
L    MD0
COS
T    MD10
```

معادل LAD **مثال**





دستور STL:

TAN Tangent of Angles as Floating-Point (32-Bit)

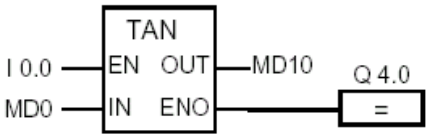
فرمت:

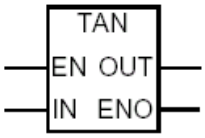
TAN

شرح:

دستور TAN مقدار تانژانت محتوی ACCU1 که برحسب رادیان فرض میشود را محاسبه کرده و نتیجه را درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

معادل FBD **مثال**





ASIN Arc Sine of Floating-Point (32-Bit)

دستور STL :

فرمت:

ASIN

شرح :

دستور ASIN مقدار آرک سینوس محتوی ACCU1 را محاسبه کرده و نتیجه را که در واقع بصورت رادیان است درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

وضعیت Status Word

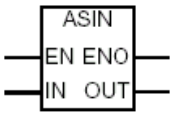
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	x	-	-	-	-

مثال :

در مثال رو برو آرک سینوس مقدار MD0 در MD10 ذخیره میشود.

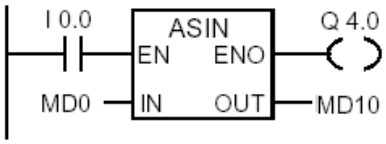
```
L    MD0
ASIN
T    MD10
```

معادل LAD

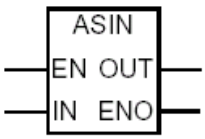


مثال

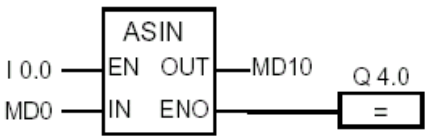
با I0.0=1فانکشن انجام شده ودر صورتیکه ورودی اعشاری نباشد یا تبدیل انجام نشود خروجی Q4.0 صفر میشود.



معادل FBD



مثال



دستور STL: ACOS Arc Cosine of Floating-Point (32-Bit)

فرمت:

ACOS

شرح :

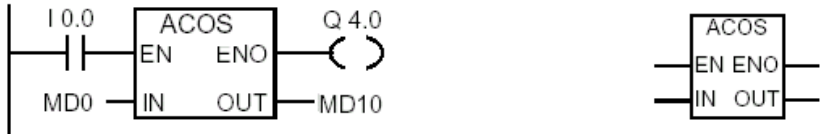
دستور ACOS مقدار آرک کسینوس محتوی ACCU1 را محاسبه کرده و نتیجه را که در واقع بصورت رادیان است درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

مثال :

در مثال رو برو آرک کسینوس مقدار MD0 در MD10 ذخیره میشود.

```
L    MD0
ACOS
T    MD10
```

معادل LAD **مثال**



دستور STL: ATAN Arc Tangent of Floating-Point (32-Bit)

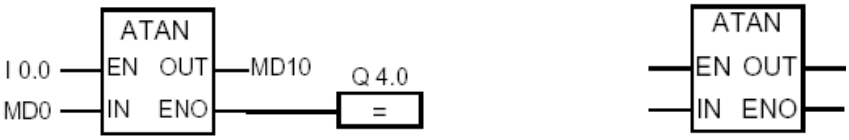
فرمت:

ATAN

شرح :

دستور ATAN مقدار آرک تانژانت محتوی ACCU1 را محاسبه کرده و نتیجه را که در واقع بصورت رادیان است درخود ACCU1 ذخیره میکند. انجام دستور ، بیتهای CC1,CC0, OV,OS را تحت تاثیر قرار میدهد.

معادل FBD **مثال**



۵-۹ دستورات بارگذاری و انتقال (Load and Transfer Instructions)

دستورات Load و Transfer امکان تبادل و جابجایی اطلاعات را بین مدولهای ورودی/خروجی و نواحی حافظه CPU یا بین خود نواحی حافظه CPU را فراهم می سازند. در هر سیکل اسکن CPU این دستورات را بدون قید و شرط و بدون توجه به RLO اجرا میسازد. لیست این دستورات بشرح زیر میباشد:

- **L** Load
- **L STW** Load Status Word into ACCU 1
- **LAR1 AR2** Load Address Register 1 from Address Register 2
- **LAR1 <D>** Load Address Register 1 with Double Integer (32-bit Pointer)
- **LAR1** Load Address Register 1 from ACCU 1
- **LAR2 <D>** Load Address Register 2 with Double Integer (32-bit Pointer)
- **LAR2** Load Address Register 2 from ACCU 1

- **T** Transfer
- **T STW** Transfer ACCU 1 into Status Word
- **TAR1 AR2** Transfer Address Register 1 to Address Register 2
- **TAR1 <D>** Transfer Address Register 1 to Destination (32-bit Pointer)
- **TAR2 <D>** Transfer Address Register 2 to Destination (32-bit Pointer)
- **TAR1** Transfer Address Register 1 to ACCU 1
- **TAR2** Transfer Address Register 2 to ACCU 1
- **CAR** Exchange Address Register 1 with Address Register 2

تذکر: دستورات Load و Transfer سمبل مجزایی در زبانهای LAD و FBD ندارند. وقتی آدرسی را در جلوی ورودی یک بلاک معرفی میکنیم معنای Load دارد و همینطور وقتی آدرسی را در جلوی خروجی یک بلاک قرار میدهیم معنای Transfer دارد.بنابراین در این قسمت معادل FBD و LAD برای دستورات آورده نشده است.

L Load

دستور STL :

فرمت:

L <Address>

Address	Data type	Memory area	Source address
<address>	BYTE	Q , I , PI , M , L ,D, Pointer, Parameter	0...65535
	WORD		0...65534
	DWORD		0...65532

شرح :

دستور L ابتدا محتوای ACCU1 به ACCU2 شیفت کرده و ACCU1 را به "0" ری ست میکند پس از آن دیتای آدرس داده شده را به ACCU1 بار میکند.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

چند مثال :

L IB10 از مدول ورودی یک بایت با آدرس IB10 به ACCU1-L-L بار میشود.

L MB120 از ناحیه حافظه یک بایت با آدرس MB120 به ACCU1-L-L بار میشود.

L DBB12 از ناحیه دیتا بلاکی که قبلاً باز شده یک بایت با آدرس DBB12 به ACCU1-L-L بار میشود.

L DIW15 از ناحیه دیتا بلاک Instance یک Word با آدرس DIW15 به ACCU1-L بار میشود.

L LD252 از ناحیه حافظه Local Data یک Dword با آدرس LD252 به ACCU1 بار میشود.

L P#I8.7 Pointer که آدرس بیت ورودی I8.7 را مشخص میکند به ACCU1 بار میشود.

L OTTO پارامتر "OTTO" به ACCU1 بار میشود.

محتوای آکومولاتور ۱ برای دستور Load

Contents of ACCU 1	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
before execution of load instruction	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
after execution of **L MB10** (L <Byte>)	00000000	00000000	00000000	<MB10>
after execution of **L MW10** (L <word>)	00000000	00000000	<MB10>	<MB11>
after execution of **L MD10** (L <double word>)	<MB10>	<MB11>	<MB12>	<MB13>
X = "1" or "0"				

L STW Load Status Word into ACCU1

دستور STL:

فرمت:

L STW

شرح:

دستور L STW محتوای رجیستر Status Word را به ACCU1 بار میکند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد. محتوای آکومولاتور ۱ پس از اجرای دستور L STW بصورت زیر خواهد بود.

Bit	31-9	8	7	6	5	4	3	2	1	0
Content:	0	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

L AR1 Load Address Register1 From ACCU1

دستور STL:

فرمت:

LAR1

شرح:

دستور LAR1 رجیستر AR1 را با محتوای ACCU1 پر میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

L AR1 <D> Load Address Register1 with Double Integer (32-Bit)

دستور STL:

فرمت:

LAR1 <D>

Address	Data type	Memory area	Source address
<D>	DWORD Pointer Constant	D , M , L	0...65532

شرح:

دستور LAR1 محتوای رجیستر AR1 را با Dword آدرس داده شده جایگزین میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

چند مثال:

LAR1 DBD20	محتوای DBD20 از دیتا بلاک اشتراکی که قبلاً باز شده به AR1 انتقال می یابد.
LAR1 DID30	محتوای DID30 از دیتا بلاک Instance که قبلاً باز شده به AR1 انتقال می یابد.
LAR1 LD180	از ناحیه حافظه Local Data یک Dword با آدرس LD180 به AR1 انتقال می یابد.
LAR1 MD24	از ناحیه حافظه یک Dword با آدرس MD24 به AR1 انتقال می یابد.
LAR1 P#M100.0	Pointer که آدرس بیت حافظه M100.0 را مشخص میکند به AR1 انتقال می یابد.

دستور STL :

LAR1 AR2 Load Address Register1 from Address Register2

فرمت:

LAR1 AR2

شرح :

دستور LAR1 AR2 محتوی رجیستر AR2 را به رجیستر AR1 میفرستد. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

دستور STL :

L AR2 Load Address Register2 From ACCU1

فرمت:

LAR2

شرح :

دستور LAR2 رجیستر AR2 را با محتوای ACCU1 پر میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

دستور STL :

L AR2 <D> Load Address Register2 with Double Integer (32-Bit)

فرمت:

LAR2 <D>

Address	Data type	Memory area	Source address
<D>	DWORD Pointer Constant	D , M , L	0...65532

شرح :

دستور LAR2 محتوای رجیستر AR2 را با Dword آدرس داده شده جایگزین میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

چند مثال :

LAR2 DBD20 محتوای DBD20 از دیتا بلاک اشتراکی که قبلاً باز شده به AR2 انتقال می یابد.

LAR2 DID30 محتوای DID30 از دیتا بلاک Instance که قبلاً باز شده به AR2 انتقال می یابد.

LAR2 LD180 از ناحیه حافظه Local Data یک Dword با آدرس LD180 به AR2 انتقال می یابد.

LAR2 MD24 از ناحیه حافظه یک Dword با آدرس MD24 به AR2 انتقال می یابد.

LAR2 P#M100.0 Pointer که آدرس بیت حافظه M100.0 را مشخص میکند به AR2 انتقال می یابد.

دستور STL: T Transfer

فرمت:

T <Address>

Address	Data type	Memory area	Source address
<address>	BYTE	Q , I , PQ , , M , L , D	0...65535
	WORD		0...65534
	DWORD		0...65532

شرح :

دستور T محتوای ACCU1 را به آدرس مشخص شده انتقال میدهد. اینکه چه مقدار بایت از ACCU1 کپی شود بستگی به سایز آدرس مشخص شده دارد . میفرستد اگر در برنامه از دستورات Master Control Relay که بعداً شرح داده خواهد شد استفاده شود و MCR=0 باشد دراینصورت با دستور T مقدار "0" به آدرس مورد نظر ارسال میشود و ربطی به مقدار ACCU1 نخواهد داشت.

این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

چند مثال :

T QB10 محتوای ACCU1-L-L به بایت خروجی QB10 ارسال میشود.

T MW120 محتوای ACCU1-L به word ناحیه حافظه با آدرس MW120 ارسال میشود.

T DBD2 محتوای ACCU1 به Dword آدرس داده شده DBD2 از دیتا بلاکی که قبلاً باز شده ارسال میشود..

دستور STL: T STW Transfer ACCU1 into Status Word

فرمت:

T STW

شرح :

دستور T STW بیتهای 0تا 8 آکومولاتور ۱ را به رجیستر Status Word می فرستد. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا میشود ولی همه آنها را تحت تاثیر قرار میدهد.

تذکر : برای CPU های S7-300 دستور فوق بیتهای OR,STA,/FC را تغییر نمیدهد یعنی فقط بیتهای 1,4,5,6,7,8 عوض میشوند.

دستور STL: CAR Exchange Address Register1 With Address Register2

فرمت:

CAR

شرح :

دستور CAR محتویات رجیسترهای AR2 و AR1 را با هم عوض (جابجا) میکند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

TAR1 Transfer Address Register1 to ACCU1

دستور STL :

فرمت:

TAR1

شرح :

دستور TAR1 محتوای رجیستر AR1 را به ACCU1 میفرستد. قبل از این کار مقدار ACCU1 به ACCU2 انتقال می یابد. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

TAR1<D> Transfer Address Register1 To Destination (32-Bit Pointer)

دستور STL :

فرمت:

TAR1 <D>

Address	Data type	Memory area	Source address
<D>	DWORD	D , M , L	0...65532

شرح :

دستور TAR1 محتوای رجیستر AR1 را به آدرس مشخص شده که میتواند متغیر حافظه (M) یا دیتای دیتا بلاک (D) یا دیتای محلی (L) باشد ارسال میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند.این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

چند مثال :

TAR1 DBD20 محتوای AR1 به DBD20 از دیتا بلاک اشتراکی که قبلاً باز شده انتقال می یابد.

TAR1 DID30 محتوای AR1 به DID30 از دیتا بلاک Instance که قبلاً باز شده انتقال می یابد.

TAR1 LD180 محتوای AR1 به ناحیه Local Data یک Dword با آدرس LD180 انتقال می یابد.

TAR1 MD24 محتوای AR1 به ناحیه حافظه یک Dword با آدرس MD24 انتقال می یابد.

TAR1 AR2 Transfer Address Register1 to Address Register2

دستور STL :

فرمت:

TAR1 AR2

شرح :

دستور TAR1 AR2 محتوی رجیستر AR1 را به رجیستر AR2 میفرستد. ACCU1 و ACCU2 بدون تغییر باقی میمانند. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

TAR2 Transfer Address Register1 to Address Register2

دستور STL :

فرمت:

TAR2

شرح :

دستور TAR2 محتوای رجیستر AR2 را به ACCU1 میفرستد. قبل از این کار مقدار ACCU1 به ACCU2 انتقال می یابد.. این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

TAR2<D> Transfer Address Register2 To Destination (32-Bit Pointer)

دستور STL :

فرمت:

TAR2 <D>

Address	Data type	Memory area	Source address
<D>	DWORD	D , M , L	0...65532

شرح :

دستور TAR2 محتوای رجیستر AR2 را به آدرس مشخص شده که میتواند متغیر حافظه (M) یا دیتای دیتا بلاک (D) یا دیتای محلی (L) باشد ارسال میکند. ACCU1 و ACCU2 بدون تغییر باقی میمانند.این دستور بدون توجه به وضعیت بیتهای Status Word اجرا شده و آنها را نیز تحت تاثیر قرار نمیدهد.

چند مثال :

TAR1 DBD20 محتوای AR2 به DBD20 از دیتا بلاک اشتراکی که قبلاً باز شده انتقال می یابد.

TAR1 DID30 محتوای AR2 به DID30 از دیتا بلاک Instance که قبلاً باز شده انتقال می یابد.

TAR1 LD180 محتوای AR2 به ناحیه Local Data یک Dword با آدرس LD180 انتقال می یابد.

TAR1 MD24 محتوای AR2 به ناحیه حافظه یک Dword با آدرس MD24 انتقال می یابد.

۲۱۵

۵-۱۰ دستورات کنترل برنامه (Program Control Instructions)

دستورات کنترل برنامه که در این بخش مورد بحث قرار میگیرند عبارتند از :

.

- **BE** Block End
- **BEC** Block End Conditional
- **BEU** Block End Unconditional
- **CALL** Block Call
- **CC** Conditional Call
- **UC** Unconditional Call
- **Call FB**
- **Call FC**
- **Call SFB**
- **Call SFC**
- **Call Multiple Instance**
- **Call Block from a Library**
- **MCR** (Master Control Relay)
- **MCR(** Save RLO in MCR Stack, Begin MCR
- **)MCR** End MCR
- **MCRA** Activate MCR Area
- **MCRD** Deactivate MCR Area

BE Block End
BEU Block End Unconditional

دستورات STL :

فرمت:

BE , BEU

شرح :

دستورات BE , BEU اسکن (اجرای) برنامه در بلاک فعلی را قطع میکند و به بلاک ماقبل که این بلاک از داخل آن فراخوان (Call) شده و به دستور بعد از فراخوانی بلاک برمیگردد. دیتاهای محلی(Local Data) مربوط به بلاک فعلی رها شده و دیتاهای محلی بلاک اصلی فعال میشوند. دیتا بلاکهایی که قبل از دستور Call باز شده بودند مجدداً باز میشوند (re-opened) این دو دستور مشابه بوده و هیچ قید و شرطی ندارند و برنامه به محض رسیدن به آنها قطع میشود.اگر از روی این دستورات پرش انجام شود اجرای بلاک هیچگاه خاتمه نمی یابد.

وضعیت Status Word

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

مثال :

در مثال اگر I1.0=1 باشد برنامه بدون انجام عمل خاصی ادامه می یابد ولی بمحض اینکه I1.0=0 شد دستورات بعد از JC اجرا شده و با رسیدن به دستور BE اجرای برنامه خاتمه یافته و به بلاک ماقبل برمیگردد.

```
      A    I1.0
      JC   NEXT
      *
      BE
NEXT: NOP  0
```

معادل LAD و FBD

ندارد

مثال

–

دستور STL :

BEC Block End Conditional

فرمت:

BEC

شرح :

دستور BEC از نظر عملکرد شبیه دستور BE است ولی در صورتی اجرا میشود که RLO قبل از آن "1" باشد.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	x	0	1	1	0

مثال :

در مثال اگر I1.0=1 باشد اجرای برنامه خاتمه یافته و به بلاک ماقبل برمیگردد ولی در صورتی که I1.0=0 باشد دستورات بعد از BEC اجرا خواهند شد.

```
      A    I1.0
      BEC
      L    IW4
      T    MW10
```

معادل LAD و FBD

ندارد

مثال

–

CALL Block Call

دستورات STL :

فرمت:

CALL < logic block identifier>

شرح :

دستور CALL برای فراخوانی بلاکهای FC و FB و SFB و SFC یا سایر بلاکهای استاندارد که از قبل توسط زیمنس برنامه نویسی شده بکار میرود.این دستور بلاکی که در جلوی آن بصورت مستقیم یا بصورت سمبلیک آدرس داده شده را بدون هیچ قید و شرطی و مستقل از وضعیت RLO صدا میزند. یعنی کنترل اجرای برنامه را به آن بلاک میبرد. FB و SFB باید حتماً همراه با ذکر نام DB صدا زده شوند. DB های فوق یا باید قبلاً موجود بوده و یا در همان لحظه ای که برنامه نوشته میشود با پیغامی که توسط Step7 داده میشود ایجاد شوند جدول زیر نحوه صدا زدن بلاکهای مختلف را نشان میدهد:

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

اگر برای بلاکها از اسامی سمبلیک استفاده شود این اسامی باید قبلاً در Symbol Table تعریف شده باشند.

وقتی یک بلاک در برنامه STL با دستور CALL صدا زده میشود بطور اتوماتیک لیستی از متغیرها مانند مثال زیر در زیر آن ظاهر میگردد

Formal Parameter متغیر هایی هستند که قبلاً در موقع ایجاد بلاک بعنوان ورودی و خروجی بلاک تعریف شده اند.

Actual Parameter مقادیر یا آدرسهایی هستند که برنامه نویس باید آنها را تعیین کند.

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

در عین حال ممکن است برای بلاکی ورودی و خروجی تعریف نشده باشد برای اینگونه بلاکها لیست فوق با دستور CALL ظاهر نمیشود.

اگر FB صدا زده شود مقادیر و آدرسهایی که بعنوان Actual Parameter تعریف میشوند در موقع اجرای برنامه در دیتا بلاک Instance مشخص شده در دستور Call ذخیره میشوند. مثال :

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

ادامه دستور CALL

وضعیت Status Word

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

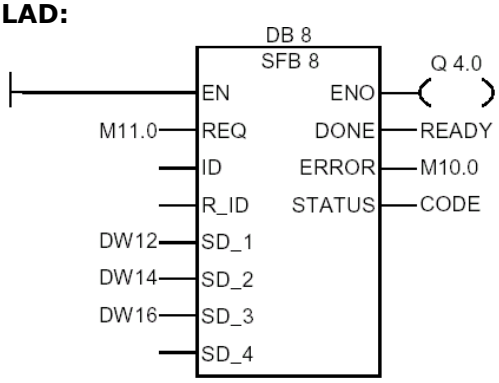
معادل LAD و FBD

FC و FB مورد نظر ابتدا باید توسط برنامه LAD/STL/FBD ایجاد شوند و پارامترهای ورودی و خروجی آنها تعریف شود . پس از آن با باز گشت به بلاک اصلی FC یا FB فوق را در پنجره المانهای LAD یا FBD در برنامه خواهیم دید که با دوبار کلیک روی آن بصورت وارد برنامه میشود.

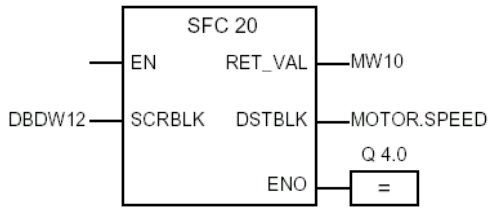
بلاکهای SFC و SFB از قبل نوشته شده هستند و میتوان آنها را در پنجره مزبور بویژه در زیر مجموعه Library پیدا نمود. مثال روبرو یک SFB را در برنامه LAD و یک SFC رادر برنامه FBD نشان میدهد.

مثال

LAD:



FBD:



Call Multiple Instance

دستور STL :

فرمت:

CALL # Variable Name

شرح :

Multiple Instance به منظور دسترسی چند FB به یک DB طراحی شده است و منجر به صرفه جویی در حافظه CPU میگردد. در این روش FB اصلی با یک DB لینک است و همراه با آن صدا زده میشود مثلاً FB1,DB1 اگر فرض کنیم چند FB دیگر مثلاً FB2 FB3 نیز بخواهند از DB1 همزمان استفاده کنند در اینصورت در جدول Declaration مربوط به FB1 سایر FB ها را با نام دلخواه و توسط متغیری از نوع Static معرفی مینماییم مثال :

Decl.	Name	Type
Stat	Test2	FB2
Stat	Test3	FB3

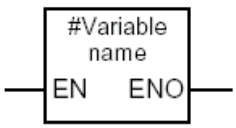
پس از تکمیل این جدول میتوانیم در داخل FB1 دو بلاک FB دیگر را بصورت زیر صدا بزنیم و نیازی به ذکر نام DB در آنها نیست.

```
CALL # TEST2
CALL # TEST3
```

وضعیت Status Word

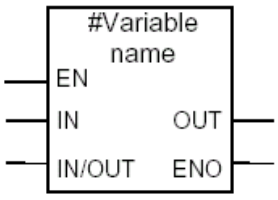
	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	0	0	x	x	x

معادل LAD

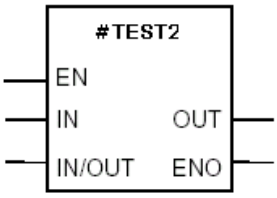


مثال

معادل FBD



مثال



CC Conditional Call

دستور STL:

فرمت:

CC < logic block identifier>

شرح:

دستور CC برای صدا زدن بلاک های FC و SFC بصورت شرطی بکار میرود یعنی در صورتی که RLO=1 باشد بلاک مورد نظر صدا زده میشود. در این روش پارامترهای بلاک در زیر دستور CC ظاهر نمیشوند به این معنی که با این روش امکان دادن دیتا به بلاک یا گرفتن دیتا از بلاک وجود ندارد.پس در مواردی که بلاک نیاز به ورودی و خروجی ندارد قابل استفاده است.

وضعیت Status Word

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

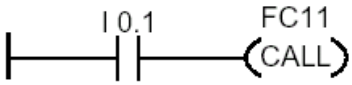
مثال:

در مثال روبرو اگر I1.0=1 باشد عمل فراخوانی بلاک FC6 انجام میشود.

```
A    I1.0
CC   FC6
A    M3.0
```

معادل LAD

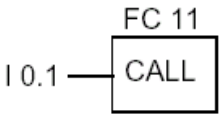
مثال



<FC/SFC no.>
---(CALL)

معادل FBD

مثال



<FC-/SFC number>

UC Unconditional Call

دستور STL:

فرمت:

UC < logic block identifier>

شرح:

دستور UC برای صدا زدن بلاک های FC و SFC بدون قید و شرط و بدون توجه به وضعیت RLO بکار میرود. در این روش مانند دستور CCپارامترهای بلاک در زیر دستور ظاهر نمیشوند یعنی با این روش امکان دادن دیتا به بلاک یا گرفتن دیتا از بلاک وجود ندارد.پس در مواردی که بلاک نیاز به ورودی و خروجی ندارد قابل استفاده است.

وضعیت Status Word

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

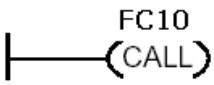
مثال:

در مثال روبرو ورودی I1.0 هر چه باشد عمل فراخوانی بلاک FC6 انجام میشود.

```
A    I1.0
UC   FC6
```

معادل LAD

مثال

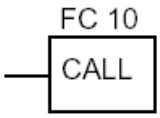


<FC/SFC no.>

---(CALL)

معادل FBD

مثال



<FC-/SFC number>

MCR Master Control Relay

دستور STL :

اخطار **بمنظور جلوگیری از بروز هر گونه حادثه هرگز دستورات MCR را جایگزین مدارات سخت افزاری قطع اضطراری نکنید.**

شرح :

دستورات MCR همانند یک سوئیچ اصلی برای قطع و وصل تغذیه عمل میکنند. MCR دستورات زیر را تحت تاثیر قرار میدهد:

- **=** <bit>
- **S** <bit>
- **R** <bit>
- **T** <byte>, **T** <word>, **T** <double word>

اگر MCR =0 باشد شبیه حالت قطع کلید عمل میکند وخروجی ها یا بیتهایی که در دستورهای = و T معرفی شده اند را صفر میکند. یعنی این دستورات عملاً در برنامه کاری انجام نمیدهند. بعلاوه خروجی ها یا بیتهایی که با دستورات S و R کار میکنند نیز آخرین وضعیت خود را حفظ کرده و دیگر ست یا ری ست نمیشوند. اگر MCR=1 شود برنامه کار عادی خود را دنبال میکند یعنی مانند وضعیت وصل شدن کلید تغذیه. نتیجه بحث فوق در جدول زیر آمده است.

Signal State of MCR	= <bit>	S <bit>, R <bit>	T <byte>, T <word> T <double word>
0 ("OFF")	بیت آدرس داده شده 0 میشود	بیتهای آدرس داده شده تغییر نخواهند کرد و آخرین وضعیت خود را حفظ میکنند.	بیت آدرس داده شده 0 میشود
1 ("ON")	پردازش برنامه بصورت عادی انجام میشود	پردازش برنامه بصورت عادی انجام میشود	پردازش برنامه بصورت عادی انجام میشود

MCRمتشکل از چند دستور است که به ترتیب زیر نوشته میشوند:

- **MCRA** Activate MCR Area
- **MCR(** Begin MCA Area
- **)MCR** End MCR Area
- **MCRD** Deactivate MCR Area

MCR با MCRA فعال و با MCRD غیر فعال میشود. هیندو را باید بصورت جفتی بکار برد و نمیتوان صرفاً از یکی از آنها استفاده نمود. این دو دوستور بدون توجه به بیتهای Status Word اجرا شده و تاثیری نیز روی آنها نمیگذارند.برنامه بین دو دستور)MCR و MCR(نوشته میشود که به آن ناحیه MCR میگویند . دستور)MCR این ناحیه را باز کرده و RLO را در پشته MCR ذخیره میسازد . اگر RLO=1 باشد در اینصورت MCR =1 یعنی ON است و پدازش برنامه عادی است یعنی MCR روی آن تاثیری نمیگذارد. ولی وقتی RLO = 0 شد در اینصورت MCR=0 یعنی OFF خواهد شد و خروجیهای طبق جدول بالا تغییر خواهند کرد. ناحیه MCR که با دستور)MCR باز شده با دستور MCR(بسته میشود و این دو دستور با هم بکار میروند اصطلاحاً Nested هستند. میتوان آنها را تودر تو و ماکزیمم تا ۸ مرحله بکار برد ولی تعداد)MCR ها باید با تعداد MCR(ها برابر باشد. در غیر اینصورت خطای پشته MCR بصورت MCRF ظاهر خواهد شد. Status Word با دستورات)MCR و MCR(بصورت زیر تحت تاثیر قرار میگیرد.

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

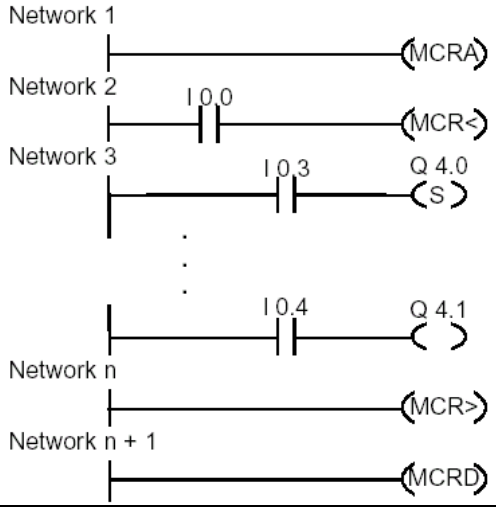
مثال :

در مثال روبرو تا زمانی که ورودی I 1.0=1 است پردازش برنامه عادی انجام میشود ولی بمحض اینکه I1.0 =0 شد MCR فعال شده و خروجی های Q8.0 و Qw20 که در داخل ناحیه MCR قرار دارند بدون توجه به دستورات ماقبل آنها صفر میشوند.

بدیهی است دستورات دو سطرانتهای برنامه که خارج از ناحیه MCR قرار دارند تحت تاثیر MCR قرار نمیگیرند.

```
MCRA
     A    I 1.0
MCR(
     A    I 4.0
     =    Q 8.0
     L    MW 20
     T    QW 10
)MCR
MCRD
     A    I 1.1
     =    Q 8.1
```

معادل LAD **مثال**



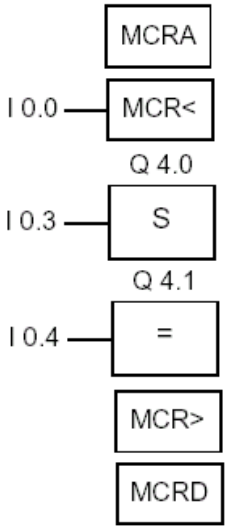
---(MCRA)

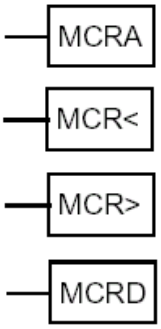
---(MCR<)

---(MCR>)

---(MCRD)

معادل FBD **مثال**





۵-۱۱ دستورات شیفت و چرخش (Shift and Rotate Instructions)

الف) دستورات شیفت

دستورات شیفت برای جابجایی بیت به بیت محتویات آکومولاتور ۱ به چپ یا راست بکار میروند. شیفت به چپ معادل عمل ضرب است. شیفت به اندازه n بیت به چپ معادل ضرب محتوای آکومولاتور در عدد 2n است بعنوان مثال با یک شیفت چپ عدد ۲ برابر میشود. شیفت به راست معادل عمل تقسیم است. شیفت به اندازه n بیت به به راست معادل تقسیم محتوای آکومولاتور بر عدد 2n است بعنوان مثال با یک شیفت راست عدد نصف میشود .

در هر بار شیفت چپ یک 0 از سمت راست وارد بیت اول آکومولاتور میگردد و در هر بار شیفت راست بسته به مقدار بیت آخر (یعنی بیت پانزدهم که علامت عدد را نشان میدهSign Bit) 0 یا 1 از سمت چپ وارد آخرین بیت آکومولاتور میشود.بعبارت دیگر اگر عدد منفی باشد 1 و اگر عدد مثبت باشد 0 وارد آکومولاتور میشود. در هر دو نوع شیفت آخرین بیت شیفت شده به CC1 (از بیت های Status Word) بار میشود و بیتهای CC0 , OV به صفر ری ست میشوند. میتوان از بیت CC1 برای پرش(Jump) استفاده کرد.

دستورات شیفت غیر مشروط هستند یعنی اجرای آنها به وضعیت بیتهای Status Word بستگی ندارد. این دستورات پس از اجرا بر RLO تاثیر نمیگذارند.وضعیت Status Word برای تمام دستورات شیفت بصورت زیر است:

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	x	x	0	0	1	-	0

دستورات شیفت عبارتند از :

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

ب) دستورات چرخش

دستورات Rotate برای چرخش بیت به بیت محتویات آکومولاتور ۱ به چپ یا راست بکار میروند. فرق این دستورات با دستورات شیفت آنست که در Shift از یکطرف 0 وارد آکومولاتور شده و جایگزین اولین یا آخرین بیت میگردد ولی در Rotate آخرین بیت بجای اولین بیت می نشیند و از بیرون چیزی وارد آکومولاتور نمیشود. آخرین بیت چرخش داده شده به CC1 (از بیت های Status Word) بار میشود و بیتهای CC0 , OV به صفر ری ست میشوند. میتوان از بیت CC1 برای پرش(Jump) استفاده کرد.

دستورات چرخش عبارتند از :

- **RLD** Rotate Left Double Word (32-bit)
- **RRD** Rotate Right Double Word (32-bit)
- **RLDA** Rotate ACCU 1 Left via CC 1 (32-bit)
- **RRDA** Rotate ACCU 1 Right via CC 1 (32-bit)

SSI Shift Sign Integer (16-Bit)

دستور STL :

فرمت: **SSI < Number>** یا **SSI**

شرح :

دستور SSI برای شیفت راست یک عدد صحیح علامت دار (مثبت یا منفی) بکار میرود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت راست شیفت میدهد. محتوای آکومولاتور ACCU1-H تغییر نمی کند. اگر SSI به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SSI<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 15 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال۱ :

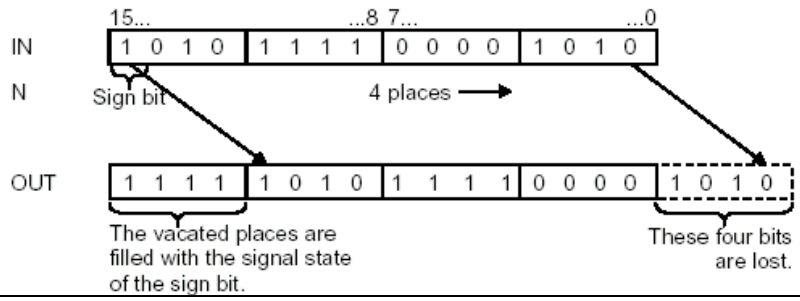
اگر محتوای آکومولاتور ۱ عدد 10 باشد با یکبار شیفت راست همانطور که در شکل زیر نشان داده شده است نتیجه عدد 5 میباشد (یعنی تقسیم بر ۲ اتفاق می افتد)

```
L   +10
SSI 1
T   QW0
```

بیت	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
قبل از شیفت	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
بعد از شیفت	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

مثال۲ :

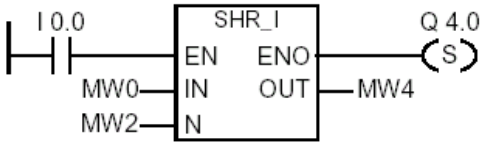
شکل روبرو محتوای آکومولاتور ACCU1-L را قبل و بعد از ٤ شیفت راست نشان میدهد. چون عدد اولیه منفی بوده 1 وارد آکومولاتور شده است.

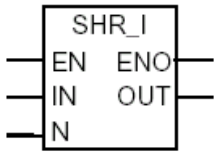


معادل LAD

مثال

MW0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW2 به راست شیفت پیدا کرده نتیجه به خروجی MW4 ارسال شده و خروجی Q4.0 ست میشود.

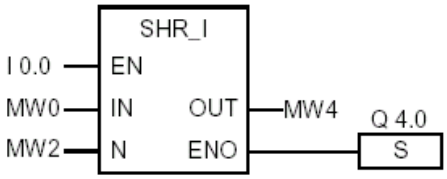


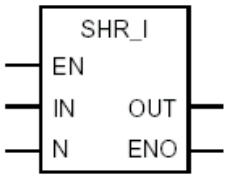


معادل FBD

مثال

مشابه توضیحات ارائه شده برای مثال LAD





SSD Shift Sign Double Integer (32-Bit)

دستور STL :

فرمت:

SSD یا SSD < Number>

شرح :

دستور SSD برای شیفت راست یک عدد صحیح ۳۲ بیتی علامت دار (مثبت یا منفی) بکار میرود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست شیفت میدهد. اگر SSD به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SSD<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 15 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال۱ :

شکل زیر نشان میدهد که چگونه محتوای آکومولاتور ۱ با دستور SSD7 هفت بار به راست شیفت پیدا کرده است.

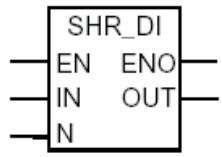
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **SSD 7**	1000	1111	0110	0100	0101	1101	0011	1011
after execution of **SSD 7**	1111	1111	0001	1110	1100	1000	1011	1010

مثال۲ :

```
L    +3
L    MD20
SSD
JP   Next
```

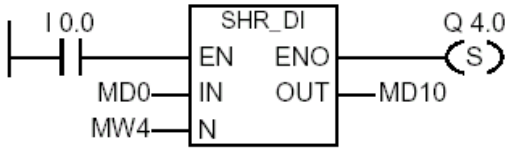
در مثال روبرو ابتدا عدد 3 به آکومولاتور ۱ بار میشود و سپس به آکومولاتور ۲ انتقال یافته و بجای آن مقدار MD20 به آکومولاتور ۱ میرود. پس از آن با دستور SSD محتوای ACCU1 سه بار (یعنی به اندازه مقدار آکومولاتور ۲) به راست شیفت پیدا میکند. در صورتیکه آخرین بیت شیفت شده به بیرون 1 باشد CC1=1 شده و چون با دستور شیفت CC0=0 میشود بنابراین وضعیت ایندو بیت معادل نمایش یک عدد مثبت است و دستور JP به آدرس Next پرش میکند.

معادل LAD

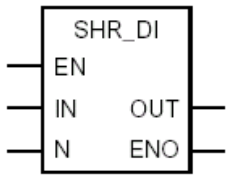


مثال

MD0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW4 به راست شیفت پیدا کرده نتیجه به خروجی MD10 ارسال شده و خروجی Q4.0 ست میشود.

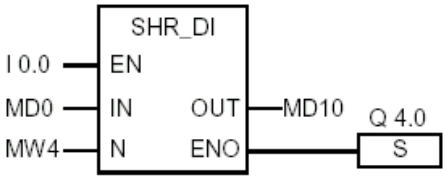


معادل FBD



مثال

مشابه توضیحات ارائه شده برای مثال LAD



SLW Shift Left Word (16-Bit)

دستور STL :

فرمت:

SLW یا SLW < Number>

شرح :

دستور SLW برای شیفت چپ یک Word بکار میرود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت چپ شیفت میدهد. محتوای آکومولاتور ACCU1-H تغییر نمی کند. اگر SLW به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SLW<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 15 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال۱ :

اگر محتوای آکومولاتور ۱ عدد 3 باشد با یکبار شیفت چپ همانطور که در شکل زیر نشان داده شده است نتیجه عدد ٦ میباشد (یعنی در ۲ ضرب شده است) که به MW0 ارسال میشود

```
L   +3
SLW1
T   MW0
```

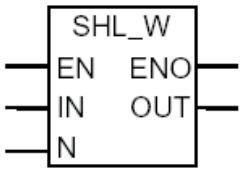
بیت	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
قبل از شیفت	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
بعد از شیفت	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

مثال۲ :

در جدول زیر محتوای آکومولاتور که با دستور SLW5 پنج بار به چپ شیفت شده نشان داده شده است.

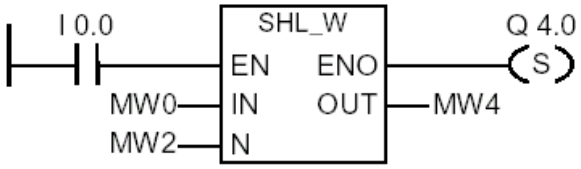
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **SLW 5**	0101	1111	0110	0100	0101	1101	0011	1011
after execution of **SLW 5**	0101	1111	0110	0100	1010	0111	0110	0000

معادل LAD

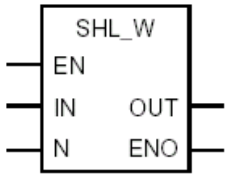


مثال

MW0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW2 به چپ شیفت پیدا کرده نتیجه به خروجی MW4 ارسال شده و خروجی Q4.0 ست میشود.

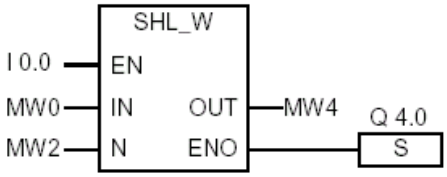


معادل FBD



مثال

مشابه توضیحات ارائه شده برای مثال LAD



SRW Shift Right Word (16-Bit)

دستور STL :

فرمت:

SRW یا SRW < Number>

شرح :

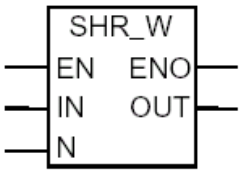
دستور SRW برای شیفت راست یک Word بکار میرود و محتوای آکومولاتور ACCU1-L را بیت به بیت به سمت راست شیفت میدهد. محتوای آکومولاتور ACCU1-H تغییر نمی کند. اگر SRW به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SRW<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 15 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال ۱ :

در جدول زیر محتوای آکومولاتور که با دستور SRW6 شش بار به راست شیفت شده نشان داده شده است. این کار معادل تقسیم بر ۱۲ میباشد.

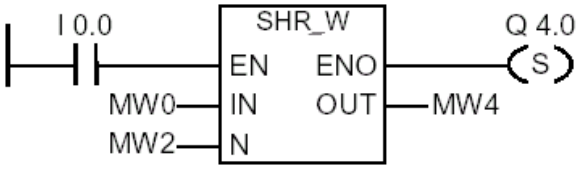
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **SRW 6**	0101	1111	0110	0100	0101	1101	0011	1011
after execution of **SRW 6**	0101	1111	0110	0100	0000	0001	0111	0100

معادل LAD

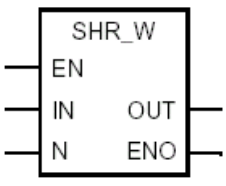


مثال

MW0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW2 به راست شیفت پیدا کرده نتیجه به خروجی MW4 ارسال شده و خروجی Q4.0 ست میشود.

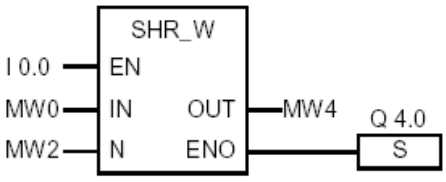


معادل FBD



مثال

مشابه توضیحات ارائه شده برای مثال LAD



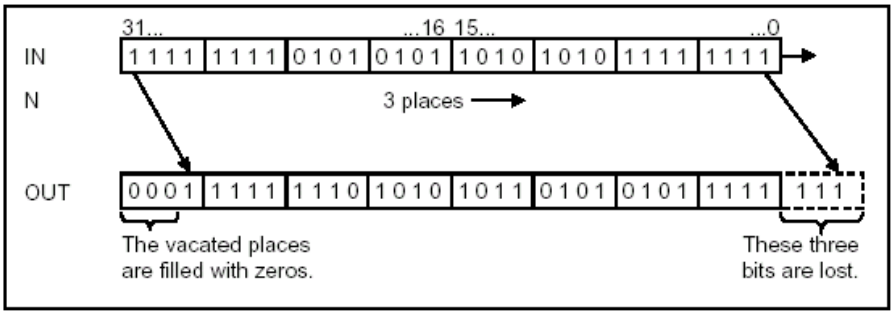
دستور STL : SLD Shift Left Double Word (32-Bit)

فرمت: **SLD** یا **SLD < Number>**

شرح :

دستور SLD برای شیفت چپ یک DWord بکار میرود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت چپ شیفت میدهد. اگر SLD به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SLD<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 32 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال : شکل زیر۳ بار شیفت چپ یک Dword را نشان میدهد.



معادل LAD و FBD : بلوک SHL_DW است که شکل آن مشابه SHL_W میباشد.

دستور STL : SRD Shift Right Double Word (32-Bit)

فرمت: **SRD** یا **SRD < Number>**

شرح :

دستور SRD برای شیفت راست یک DWord بکار میرود و محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست شیفت میدهد. اگر SRD به تنهایی بکار رود تعداد شیفت به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت SRD<Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 32 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال :

```
L   +3
L   MD20
SRD
JP  Next
```

در مثال روبرو ابتدا عدد 3 به آکومولاتور ۱ بار میشود و سپس به آکومولاتور ۲ انتقال یافته و بجای آن مقدار MD20 به آکومولاتور ۱ میرود. پس از آن با دستور SRD محتوای ACCU1 سه بار (یعنی به اندازه مقدار آکومولاتور ۲) به راست شیفت پیدا میکند. در صورتیکه آخرین بیت شیفت شده به بیرون 1 باشد CC1=1 شده و چون با دستور شیفت CC0=0 میشود بنابراین وضعیت ایندو بیت معادل نمایش یک عدد مثبت است و دستور JP به آدرس Next پرش میکند.

معادل LAD و FBD : بلوک SHR_DW است که شکل آن مشابه SHR_W میباشد.

RLD Rotate Left Double Word (32-Bit)

دستور STL :

فرمت:

RLD یا **RLD < Number>**

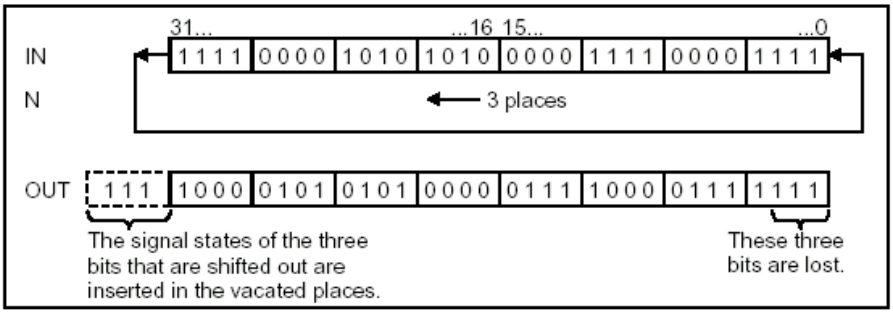
شرح :

دستور RLD محتوای آکومولاتور ACCU1 را بیت به بیت به سمت چپ چرخش میدهد.. اگر RLD به تنهایی بکار رود تعداد چرخش به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت RLD <Number> بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 32 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال۱ : جدول زیر محتوای آکومولاتور۱ را قبل و بعد از ٤ بار چرخش به چپ نشان میدهد

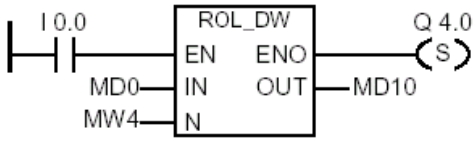
Contents	ACCU1-H				ACCU1-L			
Bit	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **RLD 4**	0101	1111	0110	0100	0101	1101	0011	1011
after execution of **RLD 4**	1111	0110	0100	0101	1101	0011	1011	0101

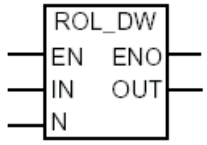
مثال۲ : شکل زیر محتوای آکومولاتور۱ را قبل و بعد از ۳ بار چرخش به چپ نشان میدهد



معادل LAD **مثال**

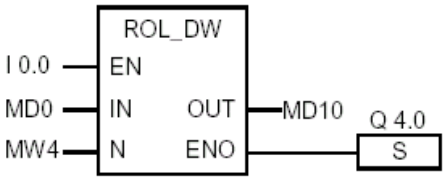
MD0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW4 به چپ چرخش پیدا کرده نتیجه به خروجی MD10 ارسال شده و خروجی Q4.0 ست میشود.

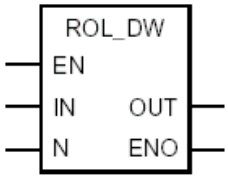




معادل FBD **مثال**

مشابه توضیحات ارائه شده برای مثال LAD





RRD Rotate Right Double Word (32-Bit)

دستور STL :

فرمت:

RRD یا RRD < Number>

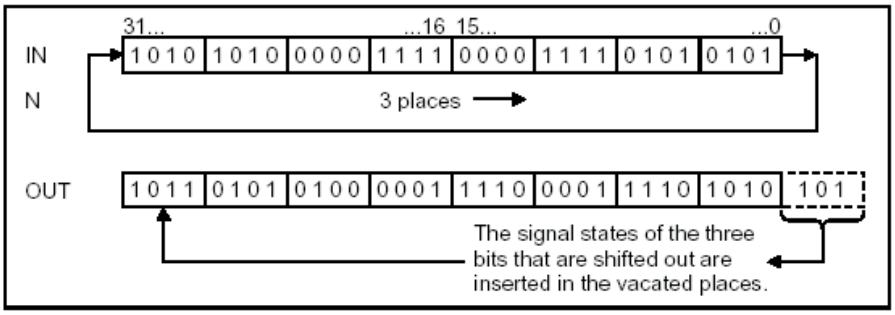
شرح :

دستور RRD محتوای آکومولاتور ACCU1 را بیت به بیت به سمت راست چرخش میدهد. اگر RRD به تنهایی بکار رود تعداد چرخش به اندازه عددی است که قبلاً به ACCU2-L-L با ر شده است و اگر دستور بصورت <Number> RLD بکار رود تعداد شیفت به اندازه عدد صحیحی است که با Number مشخص میشود. عدد میتواند بین 0 تا 32 باشد. اگر عدد 0 باشد عملاً محتوای آکومولاتور ۱ تغییر نمیکند و معادل دستور NOP یعنی No operation است.

مثال۱ : جدول زیر محتوای آکومولاتور ۱ را قبل و بعد از ٤ بار چرخش به راست نشان میدهد

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

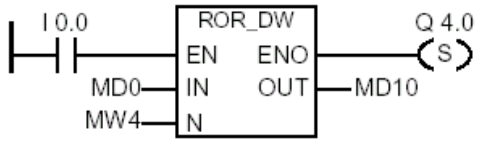
مثال۲ : شکل زیر محتوای آکومولاتور۱ را قبل و بعد از ۳ بار چرخش به راست نشان میدهد

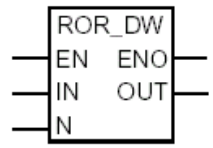


معادل LAD

مثال

MD0 به آکومولاتور۱ بار شده و با 1 شدن I0.0 باندازه عدد MW4 به راست چرخش پیدا کرده نتیجه به خروجی MD10 ارسال شده و خروجی Q4.0 ست میشود.

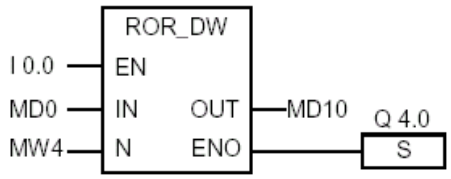


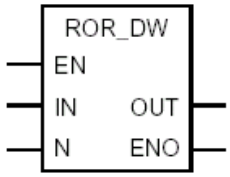


معادل FBD

مثال

مشابه توضیحات ارائه شده برای مثال LAD





دستور STL: RLDA Rotate Left Double Word Via CC1 (32-Bit)

فرمت:

RLDA

شرح :

این دستور Dword موجود در آکومولاتور ACCU1 را فقط یکبار بیت به بیت به سمت چپ و ازطریق بیت CC1 چرخش میدهد. بعبار دیگر آخرین بیت یعنی بیت ۳۲ از ACCu1 به CC1 رفته و مقدار CC1 به اولین بیت آکومولاتور ۱ بار میشود. این دستور بیت های CC0 , OV را صفر میکند ولی مقدار CC1 با توجه به توضیح فوق میتواند صفر یا یک باشد.

مثال : جدول زیر محتوای آکولاتور ۱ را قبل و بعد از اجرای دستور RLDA نشان میدهد.

Contents	CC 1	ACCU1-H				ACCU1-L			
Bit		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **RLDA**	**X**	**0**101	1111	0110	0100	0101	1101	0011	1011
after execution of **RLDA**	**0**	1011	1110	1100	1000	1011	1010	0111	011**X**
	(X = 0 or 1, previous signal state of CC 1)								

معادل LAD و FBD : ندارد .

دستور STL: RRDA Rotate Right Double Word Via CC1 (32-Bit)

فرمت:

RRDA

شرح :

این دستور Dword موجود در آکومولاتور ACCU1 را فقط یکبار بیت به بیت به سمت راست و ازطریق بیت CC1 چرخش میدهد. بعبار دیگر اولین بیت آکومولاتو۱ به CC1 رفته و مقدار CC1 به آخرین بیت آکومولاتور ۱ بار میشود. این دستور بیت های , OV CC0 را صفر میکند ولی مقدار CC1 با توجه به توضیح فوق میتواند صفر یا یک باشد.

مثال : جدول زیر محتوای آکولاتور ۱ را قبل و بعد از اجرای دستور RRDA نشان میدهد.

Contents	CC 1	ACCU1-H				ACCU1-L			
Bit		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
before execution of **RRDA**	**X**	0101	1111	0110	0100	0101	1101	0011	101**1**
after execution of **RRDA**	**1**	**X**010	1111	1011	0010	0010	1110	1001	1101
	(X = 0 or 1, previous signal state of CC 1)								

معادل LAD و FBD : ندارد

۱۲-۵ دستورات تایمرها (Timer Instructions)

هر CPU تعداد تایمر مشخصی را ساپورت میکند این تعداد در مشخصات فنی CPU ذکر شده است.بعنوان مثال CPU برای هر تایمر در حافظه CPU یک Word (۱۶ بیت) رزرو شده است که بیت های 0 تا 9 برای مقدار زمان بکار میرود. زمان بصورت باینری در این بیت ها ذخیره شده و پس از راه اندازی تایمر شروع به کم شدن میکند تا اینکه به صفر برسد. مقدار زمان را باید قبل از راه اندازی تایمر مشخص و به حافظه بار کرد فرمت این زمان میتواند به یکی از دو روش زیر باشد:

۱. W#16#txyz که در آن :

t پله زمانی است مثلا ثانیه

xyz مقدار زمان بصورت BCD

این روش معمولاً استفاده نمیشود. با این وجود مثالی برای فهم بیشتر در صفحه ۲۷۹ ارائه شده است.

۲. S5T#aH_bM_cS_dMS که در آن :

H ساعت

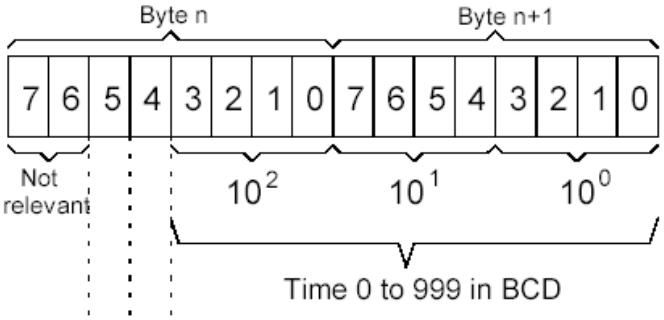
M دقیقه

S ثانیه

MS میلی ثانیه

a , b, c اعداد مربوط به موارد فوق هستند

ماکزیمم زمانی که میتوان استفاده کرد 2H_46M_30S یا 9990 ثانیه میباشد.



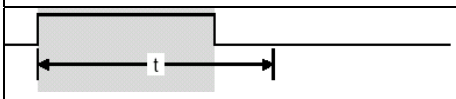
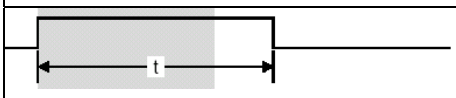
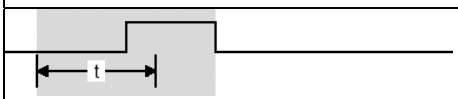
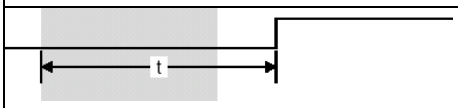
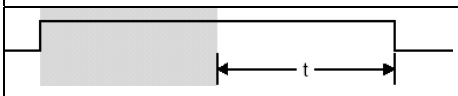
		Time base
0	0	0.01 s
0	1	0.1 s
1	0	1 s
1	1	10 s

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

بسته به نوع پله زمانی انتخاب شده که به آن Resolution هم میگویند ، رنج زمانی مشخصی برای تایمر قابل اسفاده خواهد بود طبق جدول زیر :

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

وقتی تایمر شروع به کار میکند محتوی آکومولاتور ۱ بعنوان مقدار زمان مورد استفاده قرار میگیرد. مقدار زمان بصورت BCD در بیت های 0 تا 11 ذخیره میگردد . بیت های 12 و 13 پله های زمانی را بصورت باینری نمایش میدهد. شکل صفحه قبل عدد 127 را بعنوان مقدار اولیه تایمر و با پله زمانی 1 ثانیه نمایش میدهد.تایمرها انواع مختلف دارند که فانکشن آنها با هم متفاوت است جدول زیر انواع آنها را با هم مقایسه کرده و میتواند راهنمای کاربران برای انتخاب تایمر مناسب باشد.

نوع تایمر	شرح	وضعیت ورودی فعال کننده تایمر
Pulse Timer	با یک شدن ورودی بکار می افتد و خروجی آن تا وقتی شرایط زیر برقرار است یک باقی می ماند: ۱)زمان t سپری نشده باشد ۲) ورودی 1 باشد	
Extended Pulse Timer	با یک شدن ورودی بکار می افتد و خروجی آن تا وقتی زمان t سپری نشده یک باقی میماند حتی اگر ورودی صفر شود.	
On-Delay Timer	با یک شدن ورودی بکار می افتد و خروجی آن ابتدا صفر و پس از گذشت زمان t بشرط اینکه ورودی هنوز یک باشد یک میشود و با صفر شدن خروجی صفر میگردد.	
Retentive On-Delay Timer	با یک شدن ورودی بکار می افتد و خروجی آن ابتدا صفر و پس از گذشت زمان t حتی اگر ورودی صفر شده باشد یک میشود و یک باقی میماند.	
Off-Delay Timer	با یک شدن ورودی بکار می افتد ولی زمان t از وقتی شروع میشود که ورودی صفر شود. پس از آن به اندازه t خروجی یک باقی میماند.	

لیست دستورات تایمرها بشرح زیر است:

- **FR** Enable Timer (Free)
- **L** Load Current Timer Value into ACCU 1 as Integer
- **LC** Load Current Timer Value into ACCU 1 as BCD
- **R** Reset Timer
- **SD** On-Delay Timer
- **SE** Extended Pulse Timer
- **SF** Off-Delay Timer
- **SP** Pulse Timer
- **SS** Retentive On-Delay Timer

FR Enable Timer (free)

دستور STL :

فرمت:

FR <timer>

شرح :

دستور FR وقتی که RLO از 0 به 1 میرود لبه سیگنال را تشخیص داده و تایمر مربوطه را فعال میکند. برای راه اندازی تایمر معمولاً نیازی به استفاده از این دستور نیست فقط برا ی تریگر مجدد تایمری که در حال کار است بکار میرود و آنرا مجددا با مقدار زمان اولیه Restart میکند. این دستور معادل FBD و LAD ندارد.

وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

مثال :

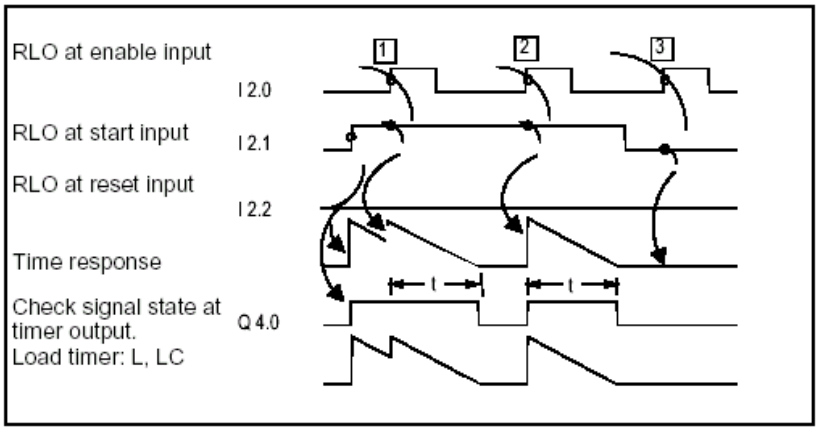
در برنامه روبرو بدون توجه به وضعیت I2.0 و صرفاً با یک شدن ورودی I2.1 تایمر T1 با زمان 10 ثانیه فعال میشود و با صفر شدن I2.1 تایمر غیر فعال میگردد تحت این شرایط تاثیر دستور FR که لبه سیگنال را وقتی ورودی I2.0 از صفر به یک میرود تشخیص میدهد بصورت زیر خواهد بود که در شکل نیز نمایش داده شده است.

```
A   I2.0
FR  T1
A   I2.1
L   S5T#10S
SP  T1
A   I2.2
R   T1
A   T1
=   Q4.0
L   T1
T   MW10
```

۱) وقتی تایمر در حال کار است و RLO در ورودی I2.0 از صفر به یک میرود تایمر T1 بطور کامل ری استارت میشود یعنی مجدداً با زمان ۱۰ ثانیه شروع بکار میکند. در این شرایط اگر RLO در ورودی I2.0 از یک به صفر برگردد تاثیری روی کار تایمر ندارد.

۲) وقتی تایمر در حال کار نیست یعنی زمان خاتمه یافته و ورودی I2.1 هنوز یک است در این شرایط اگر RLO در ورودی I2.0 از صفر به یک برود تایمر T1 با زمان ۱۰ ثانیه شروع بکار میکند. در این شرایط نیز اگر RLO در ورودی I2.0 از یک به صفر برگردد تاثیری روی کار تایمر ندارد.

۳) وقتی ورودی I2.1 صفر میشود تایمر ازکار می افتد در این شرایط اگر RLO در ورودی I2.0 از صفر به یک برود تاثیری روی تایمر T1 ندارد یعنی تایمر به کار نمی افتد.



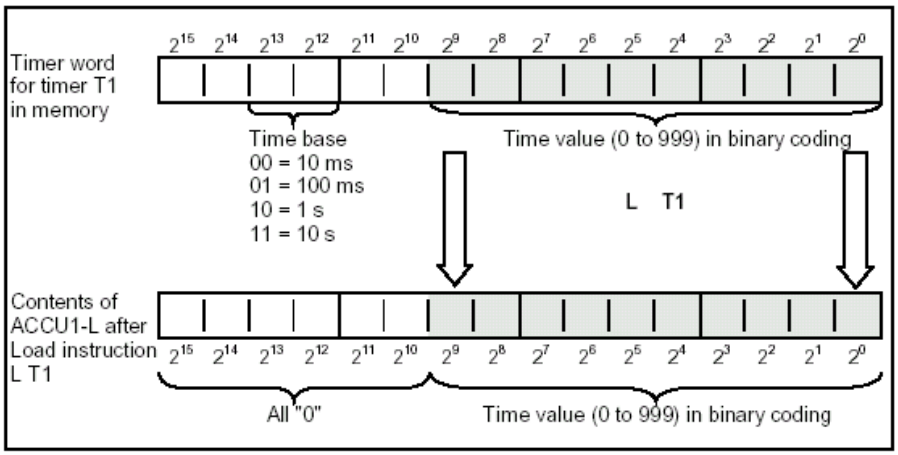
L Load Current Time Value as Integer

دستور STL :

فرمت: **L < Timer >**

شرح : این دستور ابتدا محتوی ACCU1 را به ACCU2 انتقال داده سپس مقدار فعلی زمان تایمر را از حافظه CPU را بصورت عدد صحیح به ACCU1-L بار میکند. باید توجه داشت که پله های زمانی (Time Base) که در حافظه CPU موجود است به آکومولاتور بار نمیشود. این دستور تاثیری روی بیتهای Status Word ندارد.

مثال: در مثال روبرو تایمر t1 همانطور که در شکل زیر نشان داده شده است به آکومولاتور بار میشود. **L T1**



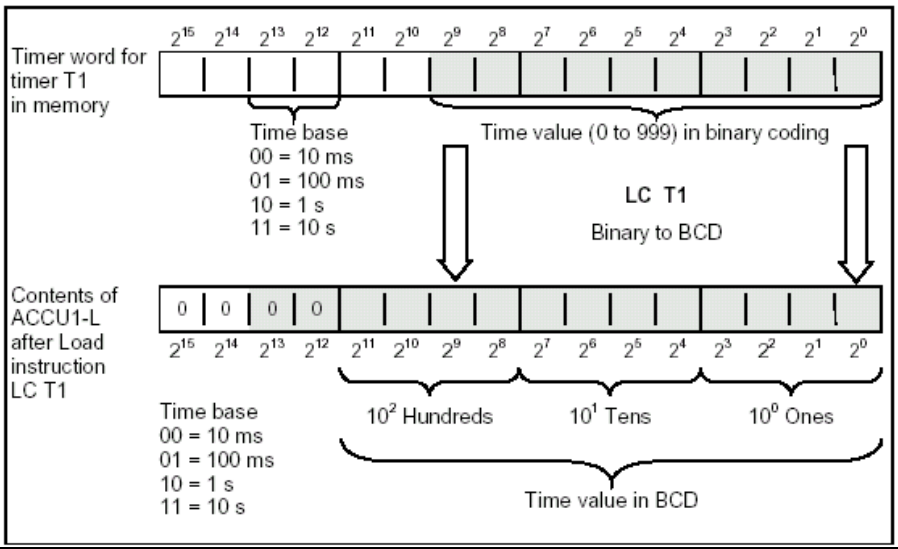
معادل LAD و FBD : همانطور که در بلوکهای مربوط به هر تایمر در صفحات بعد نشان داده شده است هر تایمر یک خروجی BI دارد که مقدار زمان فعلی را بصورت عدد صحیح نشان میدهد.

LC Load Current Time Value as BCD

دستور STL :

فرمت: **LC < Timer >**

شرح : این دستور مشابه دستور قبلی است با این تفاوت که مقدار زمان جاری را بصورت BCD به آکومولاتور بار میکند.(شکل زیر)



SP Pulse Timer

دستور STL :

فرمت: **SP < Timer>**

شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود با زمان تعیین شده راه اندازی میکند. اگر RLO از یک به صفر برگردد تایمر قطع میشود هرچند زمان آن به پایان نرسیده باشد.به این تایمر One Shot نیز گفته میشود.

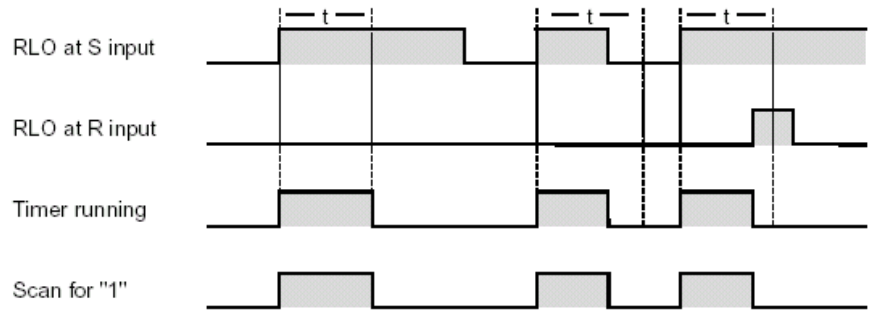
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

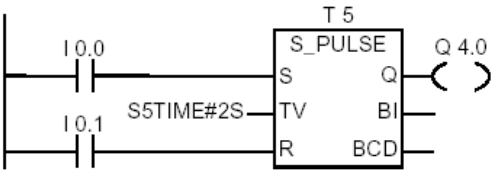
مثال :

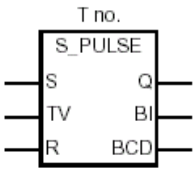
در مثال روبرو تایمر T5 وقتی ورودی I0.0 از صفر به یک برود با زمان ۲ ثانیه بکار می افتد . در طول ۲ ثانیه اگر ورودی I0.0 از یک به صفر برگردد تایمر غیر فعال میشود . فعال بودن یا نبودن تایمر در خروجی Q4.0 دیده میشود. این تایمر وقتی ورودی I0.1 از صفر به یک میرود ری ست میشود.

```
A   I0.0
L   S5T#2S
SP  T5
A   I0.1
R   T5
A   T5
=   Q4.0
```



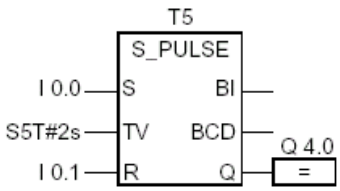
معادل LAD **مثال**

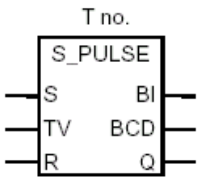




ورودی S تایمر را فعال و ورودی R آنرا ری ست میکند. مقدار زمان به ورودی TV داده میشود. زمان فعلی تایمر را بصورت باینری در خروجی BI و بصورت BCD در خروجی BCD میتوان دید.خروجی Q فعال بودن یا نبودن تایمر را نشان میدهد

معادل FBD **مثال**





SE Extended Pulse Timer

دستور STL :

فرمت: **SE < Timer>**

شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود با زمان تعیین شده راه اندازی میکند. تا وقتی زمان تایمر سپری نشده باشد حتی اگر RLO از یک به صفر برگردد تایمر قطع نمیشود.

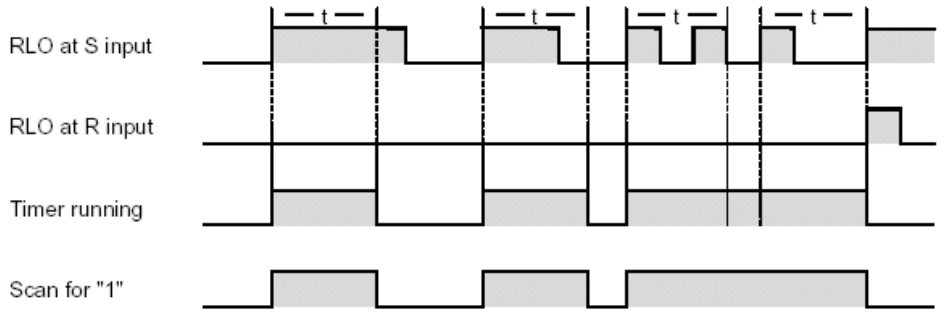
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

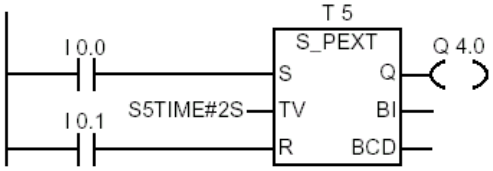
مثال :

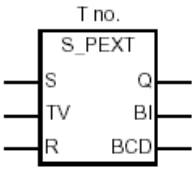
در مثال روبرو تایمر T5 وقتی ورودی I0.0 از صفر به یک برود با زمان ۲ ثانیه بکار می افتد در طول ۲ ثانیه اگر ورودی I0.0 از یک به صفر برگردد تاثیری روی تایمر ندارد. فعال بودن یا نبودن تایمر در خروجی Q4.0 دیده میشود. این تایمر وقتی ورودی I0.1 از صفر به یک میرود ری ست میشود.

```
A   I0.0
L   S5T#2S
SE  T5
A   I0.1
R   T5

A   T5
=   Q4.0
```



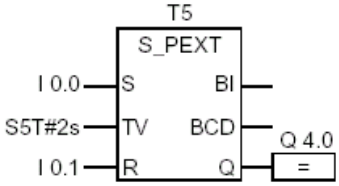
معادل LAD **مثال**

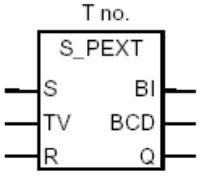




ورودی و خروجی ها شبیه آنچه برای Pulse Timer توضیح داده شد میباشند.

معادل FBD **مثال**





SD On-Delay Timer

دستور STL :

فرمت: **SD < Timer>**

شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود با زمان تعیین شده راه اندازی میکند خروجی آن ابتدا صفر و پس از گذشت زمان t بشرط اینکه ورودی هنوز یک باشد یک میشود و با صفر شدن خروجی صفر میگردد (تایمر تاخیر در وصل)

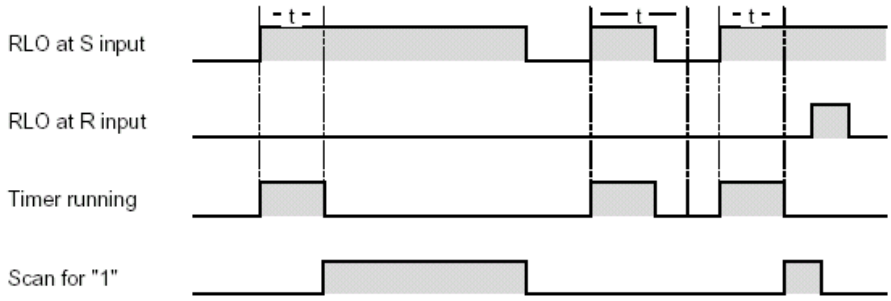
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

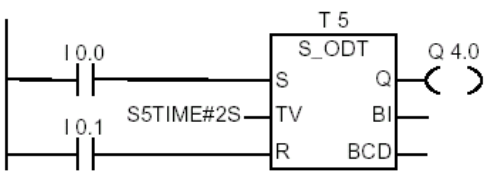
مثال :

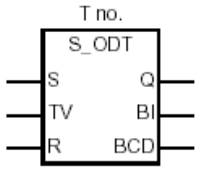
در مثال روبرو تایمر T5 وقتی ورودی I0.0 از صفر به یک برود شروع بکار میکند خروجی آن ابتدا صفر و پس از گذشت زمان ۲ ثانیه بشرط اینکه هنوز ورودی I0.0 یک باشد خروجی تایمر یک میشود. فعال بودن یا نبودن تایمر در خروجی Q4.0 دیده میشود. این تایمر وقتی ورودی I0.1 از صفر به یک میرود ری ست میشود.

```
A   I0.0
L   S5T#2S
SD  T5
A   I0.1
R   T5
A   T5
=   Q4.0
```



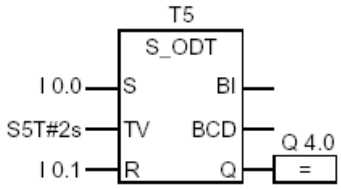
معادل LAD **مثال**

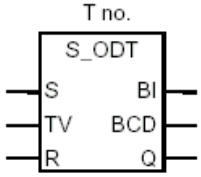




ورودی و خروجی ها شبیه آنچه برای Pulse Timer توضیح داده شد میباشند.

معادل FBD **مثال**





SS Retentive On-Delay Timer

دستور STL :

فرمت: **SS < Timer>**

شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود با زمان تعیین شده راه اندازی میکند خروجی آن ابتدا صفر و پس از گذشت زمان t حتی اگر ورودی صفر شده باشد یک میشود و یک باقی میماند.

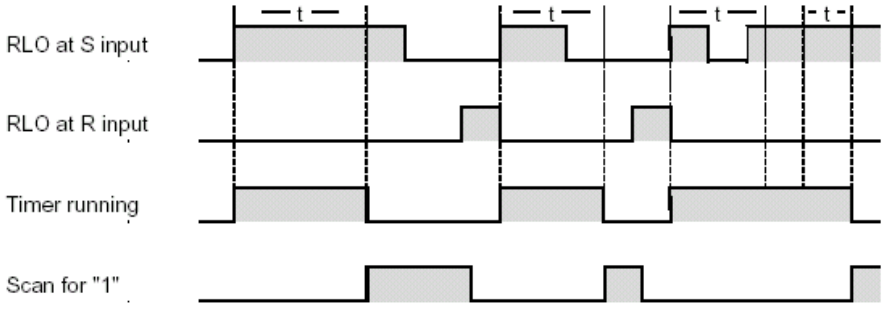
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

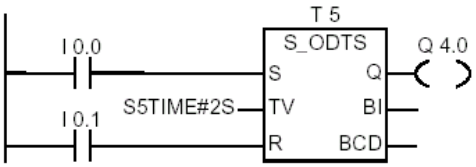
مثال :

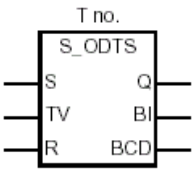
در مثال روبرو تایمر T5 وقتی ورودی I0.0 از صفر به یک برود شروع بکار میکند خروجی آن ابتدا صفر و پس از گذشت زمان ۲ ثانیه بشرط اینکه هنوز ورودی I0.0 یک باشد خروجی تایمر یک میشود ویک باقی میماند. فعال بودن یا نبودن تایمر در خروجی Q4.0 دیده میشود. این تایمر وقتی ورودی I0.1 از صفر به یک میرود ری ست میشود.

```
A   I0.0
L   S5T#2S
SS  T5
A   I0.1
R   T5
A   T5
=   Q4.0
```



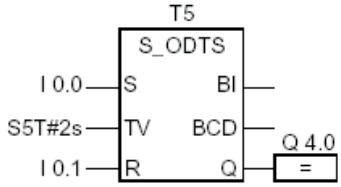
معادل LAD **مثال**

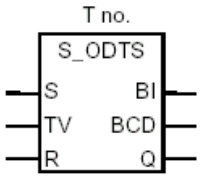




ورودی و خروجی ها شبیه آنچه برای Pulse Timer توضیح داده شد میباشند.

معادل FBD **مثال**





SF Off-Delay Timer

دستور STL :

فرمت: **SF < Timer>**

شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود با زمان تعیین شده راه اندازی میکند ولی زمان t از وقتی شروع میشود که ورودی صفر شود. پس از آن به اندازه t خروجی یک باقی میماند. (تایمر تاخیر در قطع)

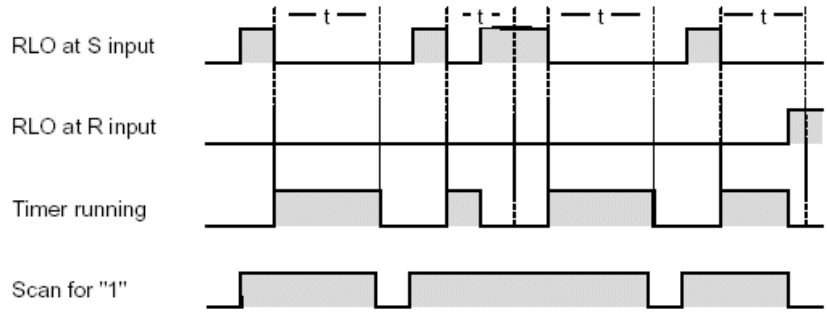
وضعیت Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	0	-	-	0

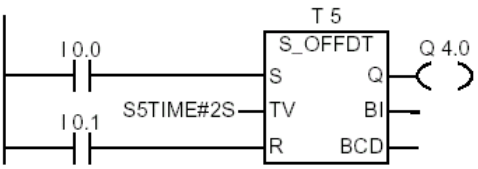
مثال :

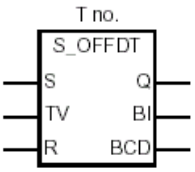
در مثال روبرو تایمر T5 وقتی ورودی I0.0 از یک به صفر برگردد شروع بکار میکند خروجی آن در طول مدت ۲ ثانیه یک است. فعال بودن یا نبودن تایمر در خروجی Q4.0 دیده میشود. این تایمر وقتی ورودی I0.1 از صفر به یک میرود ری ست میشود.

```
A   I0.0
L   S5T#2S
SF  T5
A   I0.1
R   T5
A   T5
=   Q4.0
```



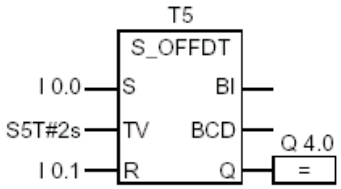
معادل LAD **مثال**

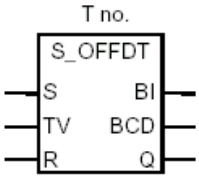




ورودی و خروجی ها شبیه آنچه برای Pulse Timer توضیح داده شد میباشند.

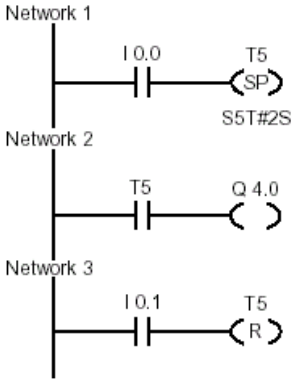
معادل FBD **مثال**





در روش LAD و FBD المانها و بلوکهای دیگری نیز برای تایمرها وجود دارند که میتوانند جایگزین المانهایی که تاکنون گفته شد گردند. این المانها در زیر توضیح داده شده است. معادل STL این موارد همان دستورات قبلی است ازاینرو نیازی به تکرار آنها در این قسمت نمیباشد.

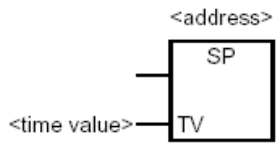
دستور LAD	Pulse Timer Coil	---(SP)



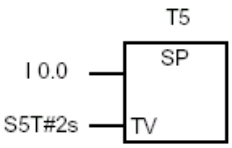
شرح :این دستور تایمر آدرس داده شده را وقتی RLO از صفر به یک برود بصورت Pulse Timer راه اندازی میکند. نحوه عملکرد این تایمر قبلاً توضیح داده شده است. المان LAD بصورت زریر است و مثالی از کاربرد آن در شکل روبرو آورده شده است. درز این مثال تایمر با ورودی I0.0 راه اندازی و با ورودی I0.1 ری ست میشود. فعال بودن یا نبودن تایمر را میتوان در خروجی Q4.0 مشاهده کرد.

<T no..>
---(SP)
<time value>

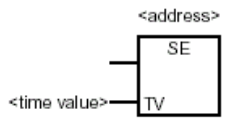
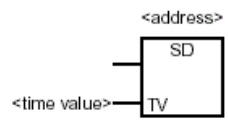
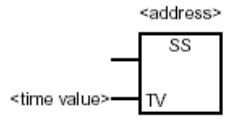
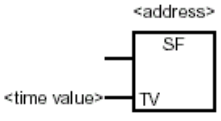
معادل FBD



مثال



سایر انواع تایمرها نیز المانهایی شبیه المانهای بالا دارند . که صرفاً به شکل المان آنها در زیر اشاره شده است..

المانLAD	عنوان	بلوک FBD
<T no.> ---(SE) <time value>	**ExtendedPulse Timer**	
<T no.> ---(SD) <time value>	**On-Delay Timer**	
<T no.> ---(SS) <time value>	**Retentive On-Delay Timer**	
<T no.> ---(SF) <time value>	**Off-Delay Timer**	

۵-۱۳ دستورات عملیات منطقی روی Word (Word Logic Instructions)

این دستورات یک جفت Word یا یک جفت Dword را بیت به بیت مطابق دستور ذکر شده بصورت منطق بولی (Boolean Logic) با هم ترکیب میکند بدیهی است هر کدام از جفت های فوق باید در یکی از دو آکومولاتور موجود باشند. برای Word فقط بیتهای بخش Low Word آکومولاتورها با هم ترکیب میشوند و برای Dword تمام بیتهای دو آکومولاتور با هم ترکیب میشوند.در هر دو حالت نتیجه در آکومولاتور ۱ ذخیره شده و مقدار قبلی این آکومولاتور از بین میرود.

این دستورات بیتهای OV و CC0 از بیتهای Status Word را صفر کرده ولی وضعیت بیت CC1 بستگی به نتیجه عملیات دارد. اگر نتیجه مخالف 0 بود CC1=1 و اگر نتیجه 0 بود CC1=0 خواهد بود.

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	x	0	0	-	-	-	-	-

لیست این دستورات بصورت زیر است :

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

دستور STL: AW And Word (16-bit)

فرمت:

AW
AW < Constant>

شرح : دستور AW محتوای آکومولاتور ACCU1-L را با محتوای ACCU2-L بیت به بیت AND میکند و نتیجه را در ACCU1-L ذخیره مینماید. آکومولاتورهای ACCU1-H و ACCU2-H بدون تغییر باقی میمانند.

دستور AW <Constant> شبیه دستور AW است ولی محتوای ACCU1-L را با مقدار ثابت 16 بیتی بیت به بیت AND میکند

مثال۱ :

در مثال روبرو مقادیری که در ورودیهای IW20 و IW22 موجود هستند بیت به بیت ترکیب شده و نتیجه به MW8 انتقال می یابد

```
L   IW20
L   IW22
AW
T   MW8
```

مثال۲ :

در مثال روبرو مقدار MW0 و مقدار ثابت W#16#000F بیت به بیت ترکیب شده و نتیجه به MW2 انتقال می یابد با فرض اینکه MW0 برابر با عدد هگز 5555 باشد نتیجه ترکیب در جدول زیر آمده است.

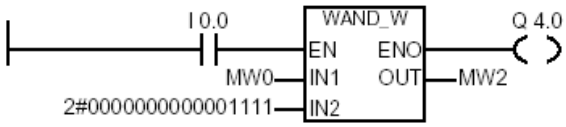
```
L   MW0
L   W#16#000F
AW
T   MW2
```

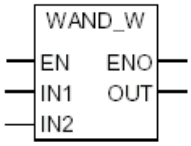
MW0	=	01010101 01010101
W#16#000F	=	00000000 00001111
MW2	=	00000000 00000101

معادل LAD

مثال

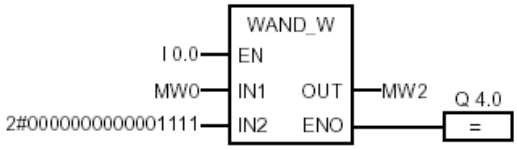
در این مثال خروجی Q4.0 وقتی دستور اجرا شود یک میشود

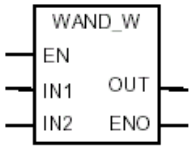




معادل FBD

مثال





OW Or Word (16-bit)

دستور STL:

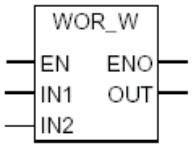
فرمت:

OW
OW < Constant>

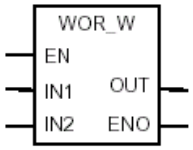
شرح :این دستور مشابه دستور AW است فقط عملکرد آن Or میباشد. مثال زیر Or شدن بیت به بیت دو آکومولاتور نشان میدهد.

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

المان LAD



المان FBD



XW Exclusive Or Word (16-bit)

دستور STL:

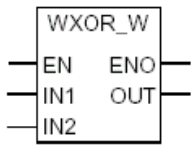
فرمت:

XW
XW < Constant>

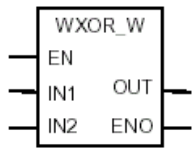
شرح :این دستور مشابه دستورات قبلی است ولی عملکرد آن XORمیباشد. مثال زیر XOR شدن بیت به بیت دو آکومولاتور نشان میدهد.
حاصل ترکیب وقتی 1 است که وضعیت دوبیت متفاوت باشد یعنی یکی 0 و دیگری 1 باشد.

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

المان LAD



المان FBD



AD And Double Word (32-bit)

دستور STL:

فرمت:

AD
AD < Constant>

شرح :دستور AD محتوای آکومولاتور ACCU1 را با محتوای ACCU2 بیت به بیت AND میکند و نتیجه را در ACCU1 ذخیره مینماید.

دستور AD <Constant> شبیه دستور AD است ولی محتوای ACCU1 را با مقدار ثابت 32 بیتی بیت به بیت AND میکند

مثال۱ :

در مثال روبرو مقادیری که در ورودیهای ID20 و ID24 موجود هستند بیت به بیت ترکیب شده و نتیجه به MD8 انتقال می یابد

```
L    ID20
L    ID24
AD
T    MD8
```

مثال۲ :

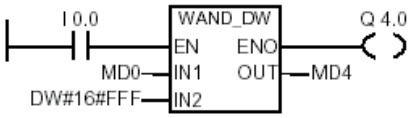
در مثال روبرو مقدار MD و مقدار ثابت DW#16#00000FFF بیت به بیت ترکیب شده و نتیجه به MD4 انتقال می یابد با فرض اینکه MD0 برابر با عدد هگز 55555555 باشد نتیجه ترکیب در جدول زیر آمده است.

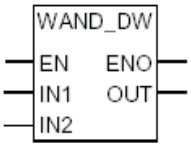
```
L    MD0
L    DW#16#FFF
AD
T    MD4
```

MD0	=	0101010101010101	0101010101010101
DW#16#FFF	=	0000000000000000	0000111111111111
MD4	=	0000000000000000	0000010101010101

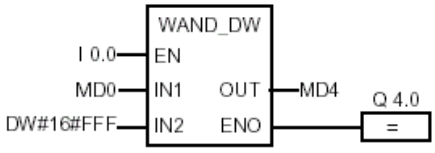
معادل LAD **مثال**

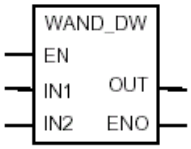
در این مثال خروجی Q4.0 وقتی دستور اجرا شود یک میشود





معادل FBD **مثال**





OD Or Double Word (32-bit)

دستور STL:

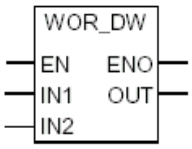
فرمت:

OD
OD < Constant>

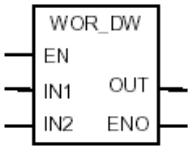
شرح :این دستور مشابه دستور AD است فقط عملکرد آن Or میباشد. مثال زیر Or شدن بیت به بیت دو آکومولاتور نشان میدهد.

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

المان LAD



المان FBD



XD Exclusive Or Double Word (32-bit)

دستور STL:

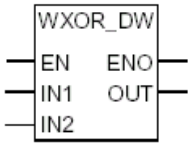
فرمت:

XD
XD < Constant>

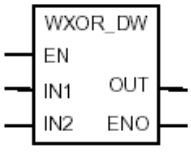
شرح :این دستور مشابه دستورات قبلی است ولی عملکرد آن XORمیباشد. مثال زیر XOR شدن بیت به بیت دو آکومولاتور نشان میدهد.
حاصل ترکیب وقتی 1 است که وضعیت دوبیت متفاوت باشد یعنی یکی 0 و دیگری 1 باشد.

Bit	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
ACCU 1 before execution of **XOD**	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 or 32-bit constant	1111	0011	1000	0101	0111	0110	1011	0101
Result (ACCU 1) after execution of **XOD**	1010	0011	0111	1001	1111	0011	1000	1110

المان LAD



المان FBD



۱۴-۵ دستورات آکومولاتوری (Accumulator Instructions)

قبل از بحث روی دستورات آکومولاتوری خلاصه چند مطلب را در ارتباط با آکومولاتورها مرور میکنیم :

- اکثر CPU های S7 دارای ۲ آکومولاتور و بعضی دارای ٤ آکومولاتور هستند.
- هر آکومولاتور ۳۲ بیتی است.
- ساختار هر آکومولاتور مانند شکل زیر است که برای آکومولاتور ۱ رسم شده است:

31 24	23 16	15 8	7 0
ACCU1			
ACCU1-H		ACCU1-L	
ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
High Word - High Byte	High Word - Low Byte	Low Word - High Byte	Low Word - Low Byte

- دستورات آکومولاتور بدون توجه به وضعیت Status Word اجرا شده و روی آنها تاثیر نمیگذارد.

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
Writes:	-	-	-	-	-	-	-	-	-

دستورات آکومولاتوری عبارتند از :

- **TAK** Toggle ACCU 1 with ACCU 2
- **PUSH** CPU with Two ACCUs
- **PUSH** CPU with Four ACCUs
- **POP** CPU with Two ACCUs
- **POP** CPU with Four ACCUs
- **ENT** Enter ACCU Stack
- **LEAVE** Leave ACCU Stack
- **INC** Increment ACCU 1-L-L
- **DEC** Decrement ACCU 1-L-L
- **+AR1** Add ACCU 1 to Address Register 1
- **+AR2** Add ACCU 1 to Address Register 2
- **BLD** Program Display Instruction (Null)
- **NOP 0** Null Instruction
- **NOP 1** Null Instruction

تذکر : دستورات آکومولاتوری فقط به فرم STL هستند و معادل LAD و FBD ندارند.

TAK Toggle ACCU1 with ACCU2

دستور STL :

فرمت: **TAK**

شرح :دستور TAK محتویات دو آکومولاتور ۱ و ۲ را با هم جابجا میکند. در CPU های چهار آکومولاتوری آکومولاتورهای ۳ و ٤ تغییر نمیکنند.

مثال۱ :

در مثال روبرو طبق شکل زیر محتوای دو آکومولاتور با هم جابجا میشوند

```
L   MW10
L   MW12
TAK
```

Contents	ACCU 1	ACCU 2
before executing **TAK** instruction	<MW12>	<MW10>
after executing **TAK** instruction	<MW**10**>	<MW**12**>

مثال۲ :

در این مثال با استفاده از دستور TAK همیشه عدد کوچکتر از عدد بزرگتر کم میشود. مقدار Mw10 به آکومولاتور ۲ و مقدار MW12 به آکومولاتور ۱ میرود. اگر MW10>MW12 بود پرش کرده و آنها را از هم کم میکند و نتیجه یعنی MW10- MW12 را به MW4 میفرستد اما اگر Mw10<MW12 بود در اینصورت با دستور TAK محتوای دو آکومولاتور عوض شده و عدد کوچکتر در MW12 قرار میگیرد سپس عمل تفریق انجام می شود.

```
      L   MW10
      L   MW12
      >   I
      JP  NEXT
      TAK
NEXT: -   I
      T   MW4
```

POP

دستور STL :

فرمت: **POP**

شرح :دستور POP برای CPU های دو آکومولاتوری محتویات آکومولاتور ۲ را به آکومولاتور ۱ کپی میکند.این دستور در CPU های چهار آکومولاتوری آکومولاتور ۲ را به آکومولاتور ۱ و آکومولاتور ٤ را به آکومولاتور ۳ و آکومولاتور ۳ را به آکومولاتور ۲ کپی مینماید

برای CPU های ۲ آکومولاتوری:

Contents	ACCU 1	ACCU 2
before executing **POP** instruction	value A	value B
after executing **POP** instruction	value B	value B

برای CPU های ٤ آکومولاتوری:

Contents	ACCU 1	ACCU 2	ACCU 3	ACCU 4
before executing **POP** instruction	value A	value B	value C	value D
after executing **POP** instruction	value B	value C	value D	value D

مثال۱ :

در مثال روبرو عدد ۱۰ به آکومولاتور ۲ و عدد ۲۰ به آکومولاتوز ۱ انتقال می یابد. ابتدا عدد ۲۰ به MD0 ارسال شده سپس با دستور POP عدد ۱۰ جایگزین عدد ۲۰ در آکومولاتور ۱ میگردد و از آنجا به MD4 انتقال می یابد.

```
L   +10
L   +20
T   MD0
POP
T   MD4
```

دستور STL: PUSH

فرمت:
PUSH

شرح :دستور PUSH برای CPU های دو آکومولاتوری محتویات آکومولاتور ۱ را به آکومولاتور ۲ کپی میکند.این دستور در CPU های چهار آکومولاتوری آکومولاتور ۱ را به آکومولاتور ۲ ، آکومولاتور ۲ را به آکومولاتور ۳ و آکومولاتور ۳ را به آکومولاتور ٤ کپی مینماید

برای CPU های ۲ آکومولاتوری:

Contents	ACCU 1	ACCU 2
before executing **PUSH** instruction	value A	value B
after executing **PUSH** instruction	value A	value A

برای CPU های ٤ آکومولاتوری:

Contents	ACCU 1	ACCU 2	ACCU 3	ACCU 4
before executing **PUSH** instruction	value A	value B	value C	value D
after executing **PUSH** instruction	value A	value A	value B	value C

مثال۱ :

در مثال روبرو مقدار MW10 که در آکومولاتور ۱ وجود دادرد به آکومولاتور ۲ کپی میشود.

```
L    MW10
PUSH
```

دستور STL: ENT Enter ACCU Stack

فرمت:
ENT

شرح :دستور ENT که فقط برای CPU های چهار آکومولاتوری بکار میرود محتوی آکومولاتور ۳ را به ٤ و ۲ را به ۳ کپی میکند. اگر این دستور قبل از دستور Load بکار رود نتایج میانی در آکومولاتور ۳ ذخیره میشوند.

مثال۱ :

در این مثال نتیجه جمع DBD0 و DBD4 در ACCU1 موجود است. وقتی DBD8 بار میشود نتیجه جمع قبلی به ACCU2 رفته و DBD8 در ACCU1 قرار میگیرد.در این مرحله اگر دستور L DBD12 بکار رود نتیجه جمع قبلی از بین میرود.برای جلوگیری از این کار دستور ENT را بکار برده تا نتیجه جمع را از آکومولاتور ۲ به آکومولاتور ۳ بفرستد. پس از آن DBD12 در آکومولاتور ۱ قرار گرفته و از DBD8 کم شده نتیجه در آکومولاتور ۱ ریخته میشود.دستور R/ محتوی آکومولاتور ۲ یعنی (DBD0+DBD4) را بر محتوی آکومولاتور ۱ یعنی (DBD8-DBD12) تقسیم میکند. نتیجه در آکومولاتور ۱ ذخیره شده و از آنجا به DBD16 انتقال می یابد.

```
L    DBD0
L    DBD4
+    R
L    DBD8
ENT
L    DBD12
-    R
/    R
T    DBD16
```

دستور STL: LEAVE ACCU Stack

فرمت:
LEAVE

شرح :دستور LEAVE که فقط برای CPU های چهار آکومولاتوری بکار میرود محتوی آکومولاتور ۳ را به ۲ و ٤ را به ۳ کپی میکند. اگر این دستور قبل از دستورات شیفت و چرخش بکار رود و آکومولاتورها با هم ترکیب شوند در اینحالت شبیه یک فانکشن محاسباتی عمل میکند. آکومولاتورهای ۱ و ٤ تغییری نمیکنند.

دستور STL : **INC Increment ACCU-1—L-L**

فرمت: **INC <8 bit integer >**

شرح :دستور INC به مقدار ACCU-1-L-L یکی اضافه میکند و نتیجه را در همین بخش از آکومولاتور ۱ ذخیره مینماید. سایر بخشهای آکومولاتور ۱ و نیز آکومولاتور ۲ بدون تغییر باقی میمانند. این دستور فقط برای مقادیر ۸ بیتی بکار میرود. یعنی اعداد صحیح مثبت که بین صفر و ماکزیمم ۲۵۵ قرار دارند. بدیهی است برای اضافه کردن مقادیر ۱۶ بیتی از دستور I+ و برای مقادیر ۳۲ بیتی از دستور D+ استفاده خواهد شد.

مثال۱ :

در مثال روبرو مقدار MB22 مرتباً یکی اضافه و نتیجه در همان MB22 ذخیره میشود.

```
L    MB22
INC
T    MB22
```

دستور STL : **DEC Decrement ACCU-1—L-L**

فرمت: **DEC <8 bit integer >**

شرح :دستور DEC از مقدار ACCU-1-L-L یکی کم میکند و نتیجه را در همین بخش از آکومولاتور ۱ ذخیره مینماید. سایر بخشهای آکومولاتور ۱ و نیز آکومولاتور ۲ بدون تغییر باقی میمانند. این دستور فقط برای مقادیر ۸ بیتی بکار میرود. یعنی اعداد صحیح مثبت که بین صفر و ماکزیمم ۲۵۵ قرار دارند. بدیهی است برای اضافه کردن مقادیر ۱۶ بیتی از دستور I- و برای مقادیر ۳۲ بیتی از دستور D- استفاده خواهد شد.

مثال۱ :

در مثال روبرو مقدار MB22 مرتباً یکی کم و نتیجه در همان MB22 ذخیره میشود.

```
L    MB22
DEC
T    MB22
```

دستور STL : **+AR1 Add ACCU1 to Address Register 1**

فرمت:

+AR1
+AR1 <P#byte.bit>

شرح :دستور AR1+ مقدار صحیح ۱۶ بیتی ذخیره شده در ACCU-1-L را با AR1 جمع میکند. این مقدار باید در رنج صحیح و بین 32768- تا 32767+ باشد.

دستور AR1+ <P#byte.bit> مقدار آفستی که باید با AR1 جمع شود را مشخص مینماید.

مثال۱ :

در این مثال عدد ۳۰۰ با مقدار AR1 جمع میشود

```
L    +300
+AR1
```

مثال۲ :

در این مثال آفست 300.0 به AR1 اضافه میشود.

```
+AR1    P#300.0
```

دستور STL: +AR2 Add ACCU1 to Address Register2

فرمت:

```
+AR2
+AR2  <P#byte.bit>
```

شرح :این دستور مشابه دستور AR1+ است فقط برای AR2 بکار میرود.

مثال ۱ :

در این مثال عدد ۳۰۰ با مقدار AR2 جمع میشود

```
L     +300
+AR2
```

مثال ۲ :

در این مثال آفست 300.0 به AR2 اضافه میشود.

```
+AR2    P#300.0
```

دستور STL: BLD Program Display Instruction (Null)

فرمت:

```
BLD  <Number >
```

شرح :دستور BLD کار خاصی انجام نمیدهد وفقط برای نمایش توسط PG بکار میرود. وقتی برنامه LAD یا FBD بصورت STL نمایش داده میشود این دستور بطور اتوماتیک با عدد های تعیین شده توسط سیستم (بین BLD0 تا BLD255) روی صفحه نمایش داده میشود.

دستور STL: NOP0 Null Instruction

فرمت:

```
NOP0
```

شرح :دستور NOP0 همانطور که از نامش یعنی (No Operation) پیداست کار خاصی انجام نمیدهد و در برنامه نویسی در صورت لزوم بکار میرود. این دستور دارای Bit Pattern با شانزده صفر است.

دستور STL: NOP1 Null Instruction

فرمت:

```
NOP0
```

شرح :دستور NOP1 نیز شبیه NOP0 کار خاصی انجام نمیدهد. این دستور دارای Bit Pattern با شانزده یک است.

۶– ارائه چند مثال برنامه نویسی

مشتمل بر :

۶–۱ تولید پالس مداوم (کنترل نشده)

۶–۲ تولید پالس مداوم (کنترل شده)

۶–۳ کنترل نوار نقاله از دو طرف

۶–٤ تشخیص جهت حرکت نوار نقاله

۶–۵ کنترل قطع شدن دو موتور با شافت هم محور

۶–۶ تنظیم زمان روشن بودن اجاق برقی توسط اپراتور

۶–۷ کنترل زمان روشنایی در راه پله

۶–۸ ایجاد پالس با پهنای دلخواه توسط SFB3

۶–۹ محاسبه n فاکتوریل

۶–۱۰ آدرس دهی متغیر با استفاده از Address Register

۶–۱۱ کنترل وضعیت انبار

۶–۱۲ کنترل ترتیبی روشن شدن دو نوار نقاله

۶–۱۳ راه اندازی و توقف موتور القایی ۳ فاز

۶–۱٤ مثالی جامع از یک برنامه کاربردی

٦-١ تولید پالس مداوم (کنترل نشده)

برنامه زیر بطور مداوم مقدار موجود در MW100 که ابتدا صفر است را یکی افزایش داده و نتیجه را روی خود MW100 میریزد. اگر بیت با کم ترین ارزش این word یعنی بیت M101.0 را به خروجی Q0.0 بفرستیم می بینیم که مرتباً 0 و 1 میشود.

```
 L  MW100
 L  1
+I
 T  MW100
 A  M101.0
 =  Q0.0
```

MW100															
MB100								MB101							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

با توجه به شکل فوق که بیتهای MW100 را نشان میدهد میتوان جدول زیر که تغییرات سایر بیتها از بایت MB101 را نشان میدهد ملاحظه کرد. اگر به ستونهای جدول دقت کنیم می بینیم که فرکانس صفر و یک شدن بیت ها نسبت به بیت ماقبل مرتباً نصف میشود.

	MB101							
Scan Cycle	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1
10	0	0	0	0	1	0	1	0
11	0	0	0	0	1	0	1	1
12	0	0	0	0	1	1	0	0

ارائه چند مثال برنامه نویسی

۶-۲ تولید پالس مداوم (کنترل شده)

در برنامه قبل فرکانس پالس کنترل نشده و تابعی از زمان سیکل اسکن CPU بود . میتوان با اضافه کردن تایمر به برنامه فرکانس پالس را کنترل نمود. در برنامه زیر با استفاده از یک تایمر Extended Pulse هر ۲۵۰میلی ثانیه یکبار در خروجی Q0.0 پالس خواهیم داشت.در این برنامه در طول زمانی که تایمر فعال است مقدار MW100 یکی افزایش می یابد ولی وقتی تایمر غیر فعال میشود (یعنی ۲۵۰ میلی ثانیه) خاتمه می یابد بلاک با دستور BEC خاتمه می یابد و Mw100 افزایش پیدا نمیکند. دستور NOT وقتی که تایمر غیر فعال یعنی RLO=0 میشود مجدداً RLO را یک کرده و بدنبال آن دستور BEC بلاک را خاتمه میدهد.

```
AN      M0.0
L       S5T#250MS
SE      T1
NOT
BEC
L       MW100
L       1
+I
T       MW100
A       M101.0
=       Q0.0
```

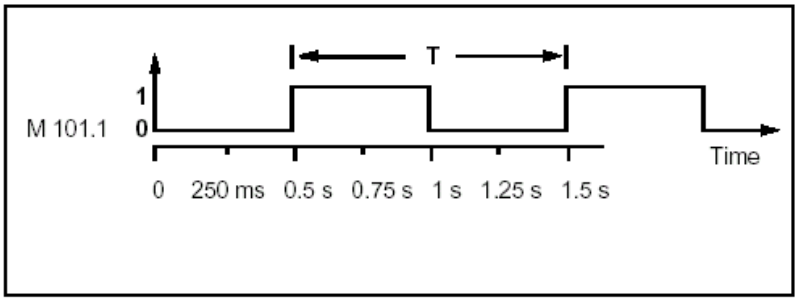
اگر بخواهیم فرکانس پالس را تعیین کنیم کافیست با توجه به رابطه F=1/T زمان صفر و یک شدن را جمع زده و معکوس نماییم. بعنوان مثال بیت M101.0 به اندازه ۲۵۰ میلی ثانیه صفر و باندازه ۲۵۰ میلی ثانیه صفر است پس فرکانس آن 1/(250+250) که معادل 2HZ خواهد بود. فرکانس بیتهای دیگر بترتیب طبق جدول زیر نصف فرکانس بیت قبلی است.

Bits of MB100	Frequency in Hertz	Duration	
M 101.0	2.0	0.5 s	(250 ms on / 250 ms off)
M 101.1	1.0	1 s	(0.5 s on / 0.5 s off)
M 101.2	0.5	2 s	(1 s on / 1 s off)
M 101.3	0.25	4 s	(2 s on / 2 s off)
M 101.4	0.125	8 s	(4 s on / 4 s off)
M 101.5	0.0625	16 s	(8 s on / 8 s off)
M 101.6	0.03125	32 s	(16 s on / 16 s off)
M 101.7	0.015625	64 s	(32 s on / 32 s off)

شکل زیر وضعیت بیت M101.1 را نشان میدهد. که فرکانس آن 1HZ میباشد.

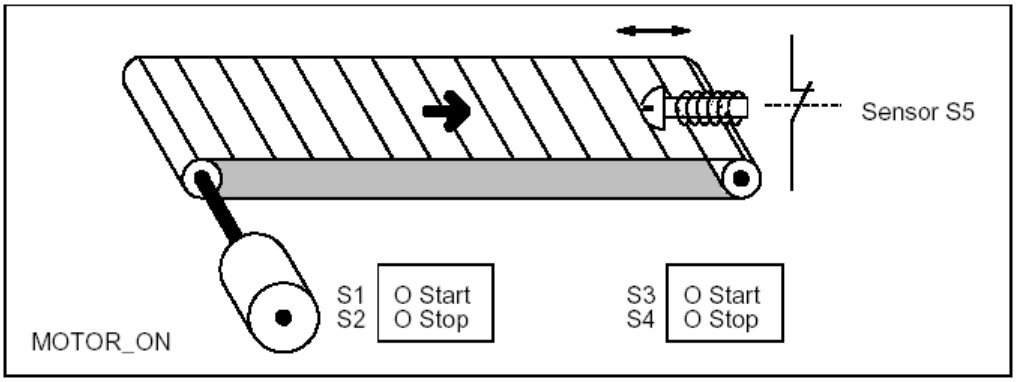
Signal state of Bit 1 of MB 101 (M 101.1)

Frequency = 1/T = 1/1 s = 1 Hz



٦-۳ کنترل نوار نقاله از دو طرف

نوار نقاله شکل زیر توسط کلیدهای Start و Stop که در دو طرف نوار قرار گرفته روشن و خاموش میشود. بعلاوه وقتی سنسور نرمال بسته (NC) حضور قطعه را حس کند نوار از کار می افتد :



ابتدا در جدول سمبلها آدرس ها رابصورت زیر تعریف میکنیم:

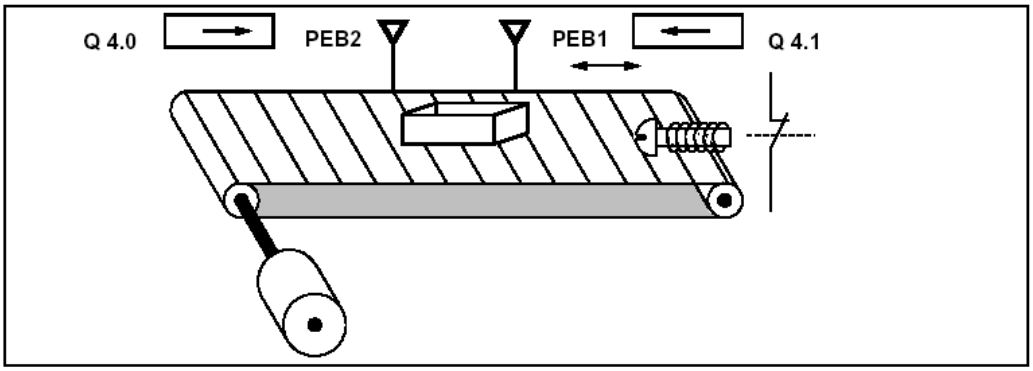
سمبل	آدرس
S1	I 0.0
S2	I 0.1
S3	I 1.0
S4	I 1.1
S5	I 2.0
MOTOR	Q 0.0

برنامه به زبان STL بصورت زیر خواهد بود :

```
O    "S1"
O    "S3"
S    "MOTOR"
O    "S2"
O    "S4"
ON   "S5"
R    "MOTOR"
```

٤-٦ تشخیص جهت حرکت نوار نقاله

سنسورهای فتوالکتریک PEB1 و PEB2 که هر دو نرمال باز (NO) هستند برای تشخیص جهت حرکت نوار نقاله شکل زیر طراحی شده اند. این سنسورها حضور جسم را تشخیص میدهند. میخواهیم وقتی نوار به سمت راست حرکت کند لامپ Q4.0 و وقتی نوار به سمت چپ حرکت می کند لامپ Q4.1 روشن شود.



ابتدا در جدول سمبلها آدرس ها بصورت زیر تعریف میکنیم

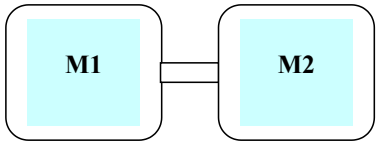
سمبل	آدرس
PEB2	I 0.0
PEB1	I 0.1
RIGHT	Q 4.0
LEFT	Q 4.1

برنامه زیر با تشخیص لبه بالا رونده سیگنال سنسورها جهت حرکت نوار را مشخص میکند:

```
NETWORK 1
      A    "PEB2"
     FP    M 0.0
     AN    "PEB1"
      S    "LEFT"

NETWORK 2
      A    "PEB1"
     FP    M0.1
     AN    "PEB2"
      S    "RIGHT"

NETWORK 3
     AN    "PEB2"
     AN    "PEB1"
      R    "RIGHT"
      R    "LEFT"
```

٦-٥ کنترل قطع شدن دو موتور با شافت هم محور

شافت دو موتور الکتریکی مانند شکل زیر بصورت مکانیکی بهم متصل شده و مشترکا سیستمی را می چرخانند. اگر تغذیه برق یکی از موتور ها قطع شود بار روی موتور دیگر می افتد و موتور دوم نیز به این بار اضافه میشود که وضعیت خطرناکی برای موتور اول است .
برنامه ای بنویسید که در صورت وجود این شرایط چراغ سیگنال Q0.0 روشن شده و برق هر دو موتور قطع شود.



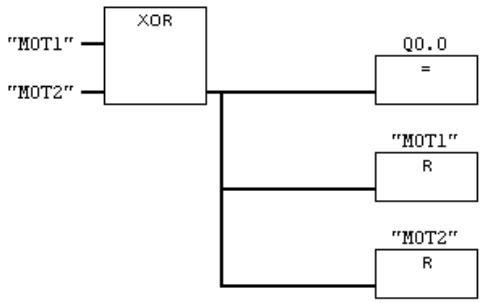
روش اول :

```
O(
A     "MOT1"
AN    "MOT2"
)
O(
AN    "MOT1"
A     "MOT2"
)
=     Q0.0
R     "MOT1"
R     "MOT2"
```

روش دوم : استفاده از Exclusive OR بصورت زیر:

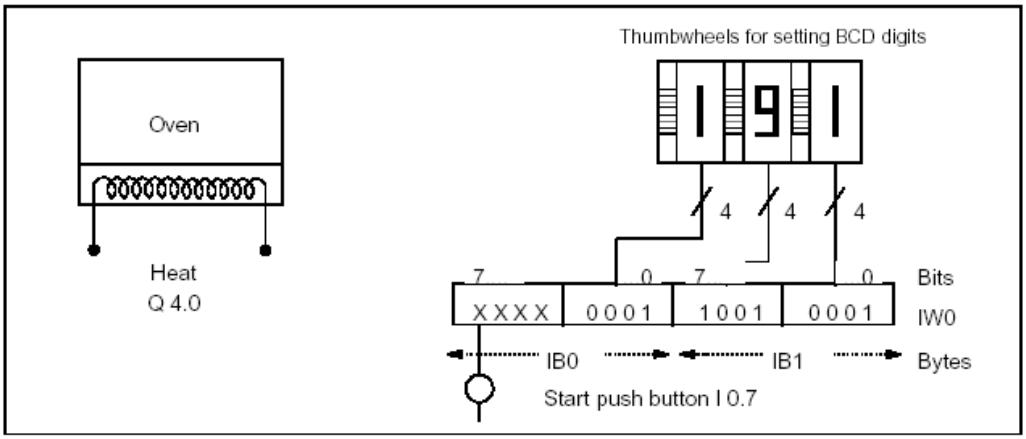
```
X     "MOT1"
X     "MOT2"
=     Q0.0
R     "MOT1"
R     "MOT2"
```

که با FBD بصورت زیر خواهد بود:



٦-٦ تنظیم زمان روشن بودن اجاق برقی

شروع کار گرم شدن اجاق برقی شکل زیر با فشار دادن شستی Start است. اپراتور مدت زمانی که لازم است اجاق روشن باشد را با استفاده از تنظیم سه عدد با کلید چرخشی (Thumbwheel) انجام میدهد.مقدار تنظیم شده برحسب ثانیه است یعنی در شکل زیر زمان روی ۱۹۱ ثانیه تنظیم شده است. برنامه مربوطه را بنویسید:



هر کدام از سگمنتهای تنظیم کننده چرخشی یک عدد BCD بین 0 تا 9 است . یعنی هر یک از شماره ها ٤ بیت نیاز دارد که طبق جدول زیر به ورودی IW0 داده شده است:

برای رقم یکان	ورودی های I1.0 تا I1.3
برای رقم دهگان	ورودی های I1.4 تا I1.7
برای رقم صدگان	ورودی های I0.0 تا I0.3

برای فهم بیشتر برنامه به توضیحات صفحه ۲۳۲ در بخش ۵ مراجعه نمایید. روش راه اندازی تایمر به روش اول ذکر شده در آن جا میباشد.

A T1 **= Q4.0** **BEC**	با روشن شدن تایمر اجاق را روشن و با خاموش شدن تایمر اجاق را خاموش کن در زمان روشن بودن تایمر بلاک را با دستور BEC در همین جا ختم کن . این دستور از راه اندازی مجدد تایمر وقتی که شستی I 0.7 مجددا فشار داده شود جلوگیری میکند.
L IW0	مقدار زمان بصورت BCD وارد آکومولاتور ۱ میشود.
AW W#16#0FFF	برای اینکه بتوان پله زمانی را در کنار زمان فوق وارد آکومولاتور کرد ابتدا تک تک ۱۲ بیت مربوط به زمان را با 1 وسایر بیتهای آکومولاتور را با صفر ANDمیکنیم.
OW W#16#2000	میخواهیم پله زمانی برحسب ثانیه باشد پس باید بیتهای ۱۲ و ۱۳ آکومولاتور را "10" کنیم پس مقدار هگز 2000 را با مقدار قبلی OR میکنیم
A I 0.7 **SE T1**	با فشار دادن شستی تایمر T1 رابه صورت Extended Pulse راه اندازی کن

٦-۷ کنترل زمان روشنایی در راه پله

لامپ راه پله یک ساختمان پنج طبقه توسط شستی راهروی هر طبقه لازم است بصورت زیر کنترل شود:

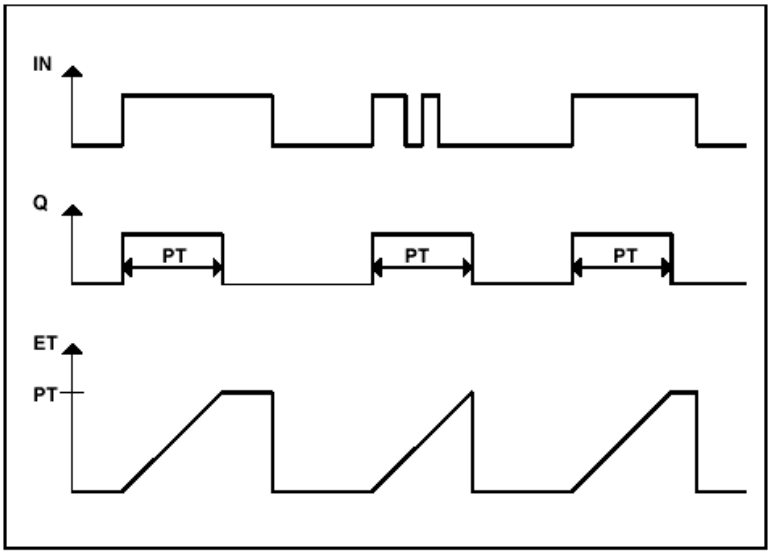
- لامپ راهرو توسط هرکدام از شستی های راهرو ها روشن شود.
- لامپ بمدت ۳۰ ثانیه روشن و پس از آن خاموش گردد.
- اگر در طول ۳۰ ثانیه شستی دیگری فشار داده شود زمان ۳۰ ثانیه از همان لحظه شروع گردد.

```
O    I 0.0
O    I 0.1
O    I 0.2
O    I 0.3
O    I 0.4
S    Q 0.0
L    S5T#30S
SD   T1
R    Q0.0
```

٦-۸ ایجاد پالس با پهنای دلخواه توسط SFB3

فانکشن SFB3 یا با نام سمبلیک "TP" برای تولید پالس با پهنای دلخواه طراحی شده است

با لبه بالارونده ورودی IN پالسی با پهنای تنظیم شده در ورودی PT در خروجی Q ظاهر میشود.شکل زیر ورودی IN و خروجی Q را همراه با خروجی ET یا (Expired Time) نشان میدهد.



در مثال زیر در بلاک OB1 فانکشن بلاک SFB3 صدا زده شده و با زمان ٦ ثانیه تنظیم شده است.

```
CALL SFB3,DB3
    IN: I 0.0
    PT : T#6S
    Q : Q0.0
    ET : MD0
```

بدیهی است برای داشتن پالس مداوم لازم است ورودی مانند شکل بالا مرتبا 0 و 1 شود.

۶-۹ محاسبه n فاکتوریل

برنامه زیر با استفاده از حلقه LOOP مقدار فاکتوریل عددی که به IW0 داده میشود را محاسبه و نتیجه را در MW20 ذخیره میسازد.

```
        L     1
        T     MW20
        L     IW0
TEST:   T     MW0
        L     MW20
        *     I
        T     MW20
        L     MW0
LOOP    TEST
```

۶-۱۰ آدرس دهی متغیر با استفاده از Address Register

برنامه زیر مقدار IW0 را به QW0 که آدرس آن از طریق رجیستر AR1 مشخص میشود انتقال می یابد:

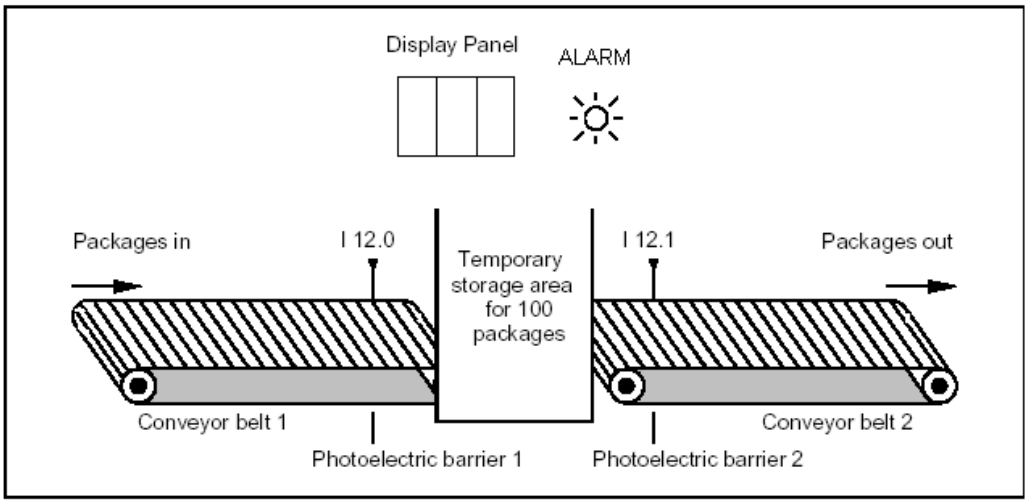
L	IW0	IW0 به آکومولاتور ۱ انتقال می یابد
L	0	IW0 به آکومولاتور ۲ و 0 به آکومولاتور ۱ انتقال می یابد
LAR1		0 از آکومولاتور ۱ به AR1 انتقال می یابد
TAK		IW0 در آکومولاتور ۱ قرار میگیرد
T	QW [AR1,P#0.0]	محتوای آکومولاتور ۱ به خروجی QW با آدرس مشخص شده در AR1 منتقل میشود یعنی به QW0 مقدار آفست در اینجا صفر است ولی اگر داشتیم P#2.0 دراینصورت آدرس نهایی که ازجمع AR1 و مقدار آفست بدست می آید 2 میشد یعنی QW2

با استفاده از منطق فوق برنامه زیر هر بار عدد ۱۶ یعنی ۲ بایت را به AR1 اضافه میکند تا اینکه به عدد ۱۶۰ برسد و مقدار IW0 را به خروجی های QW0 تا QW20 منتقل میسازد.

```
        L   0
        T   MW0
ABC:    L   MW0
        L   160
        <=  I
        JCN END
        L   IW0
        L   MW0
      LAR1
       TAK
        T   QW [AR1,P#0.0]
        L   MW0
        L   16
        +   I
        T   MW0
       JU   ABC
END: NOP   0
```

۶-۱۱ کنترل وضعیت انبار

در شکل زیر سیستمی با دو نوار نقاله و یک انبار موقت میانی نشان داده شده است. نوار ۱ بسته های محصول را به این انبار تحویل میدهد. سنسور فتوالکتریک I 12.0 که در انتهای نوار ۱ قرار دارد تعداد این بسته ها را میشمارد. ظرفیت انبار ۱۰۰ بسته است . نوار ۲ بسته ها را از انبار موقت به بیرون انتقال میدهد. فتوسلی که در ابتدای نوار ۲ قرار گرفته تعداد بسته های خروجی از انبار را میشمارد.



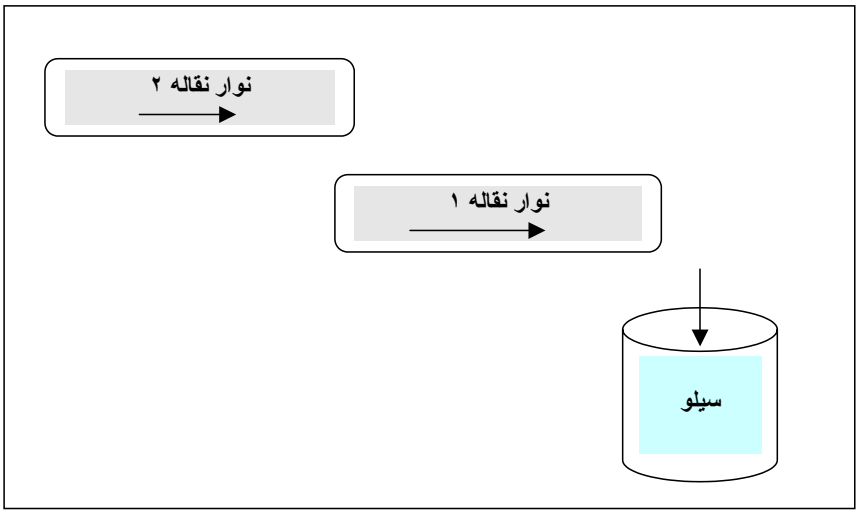
برنامه ای بنویسید که :

- تعداد بسته های موجود در انبار روی نمایشگر BCD (یعنی QW2) ظاهر شود.در هنگام روشن شدن سیستم موجودی انبار صفر فرض میشود.
- اگر تعداد بسته ها از ۹۰ بیشتر شد چراغ آلارم (Q0.1) روشن شود.
- اگر تعداد بسته ها به ۱۰۰ رسید موتور نوار ۱(Q0.0) قطع شود و وقتی کمتر از ۱۰۰ شد روشن شود.

```
A     I 12.0
CU    C1
A     I 12.1
CD    C1
LC    C1
T     QW2
L     C#90
>=    I
=     Q 0.1
LC    C1
L     C#100
<     I
=     Q0.0
```

۱۲-۶ کنترل ترتیبی روشن شدن دو نوار نقاله

ترتیب روشن شدن دو نوارنقاله شکل زیر باید به اینگونه باشد که با زدن شستی Start ابتدا نوار ۱ و با ۱۰ ثانیه تاخیر نوار ۲ روشن شود.



برنامه بصورت زیر خواهد بود:

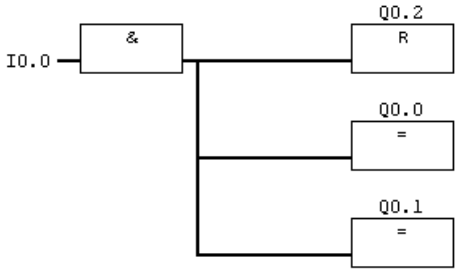
```
NETWORK 1
      A(
       O     "Start"
       O     "Conveyor1"
       )
       =     "Conveyor1"

NETWORK 2
       A     "Conveyor1"
       L     S5T#10S
      SP     T1
      AN     T1
       A     "Conveyor1"
       =     "Conveyor2"
```

٦-۱۳ راه اندازی و توقف موتور القایی ۳ فاز

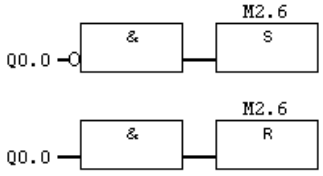
موتور القایی سه فاز با 1 شدن ورودی I0.0 روشن و با 0 شدن همین کلید پس از گذشت ۵ ثانیه متوقف میشود. با توجه به خروجی های زیر برنامه مربوطه را بصورت FBD بنویسید.

- Q0.0 : کنتاکتور تغذ یه موتور
- Q0.1 : لامپ نمایش روشن بودن موتور
- Q0.2 : لامپ نمایش خاموش بودن موتور
- Q0.1 : لامپ نمایش در حال متوقف شدن موتور (روشن در طول ۵ ثانیه)

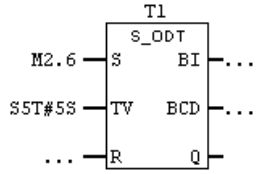
Network 1 : Title:



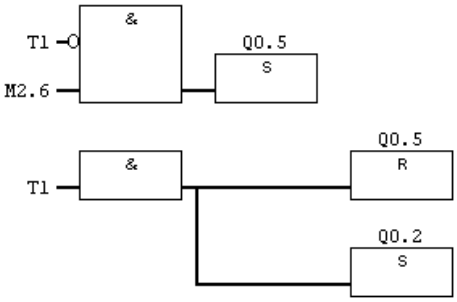
Network 2 : Title:



Network 3 : Title:

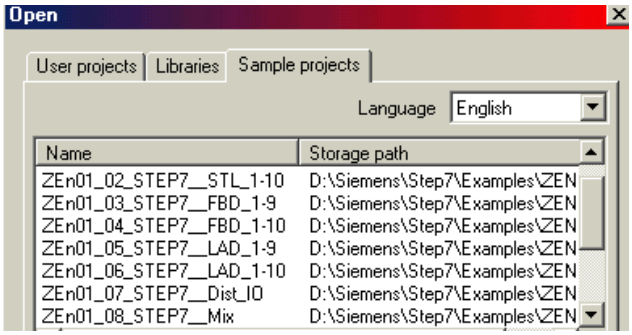


Network 4 : Title:



٦-١٤ مثالی جامع از یک برنامه کاربردی

در STEP7 برنامه هایی بصورت نمونه وجود دارند که توسط منوی File> Open در Simatic Manager لیست آنها طبق شکل زیر در بخش Sample Projects نمایش داده میشود.



در اینجا به تشریح یکی از این برنامه ها به نام ZEn01_08_STEP7_Mix که مربوط به یک سیستم مخلوط کن میباشد میپردازیم.

برای این منظور قدمهای زیر بترتیب برداشته میشوند.

١. توصیف فرآیند
٢. اجزای کنترلی سیستم
٣. لاجیک سیستم
٤. کنترل اپراتوری
٥. بلاکهای مورد نیاز
٦. تعریف جدول سمبلها
٧. طراحی FB
٨. ایجاد DB ها
٩. طراحی FC
١٠. طراحی OB

١- توصیف فرآیند

همانطور که در شکل صفحه بعد مشاهده میشود دو نوع سیال مختلف توسط ورودیهای A و B داخل تانکی ریخته میشوند. این دو مایع در داخل تانک توسط یک همزن مخلوط میگردند. محصول نهایی از طریق یک ولو تخلیه به بیرون انتقال می یابد.

٢- اجزای کنترلی سیستم

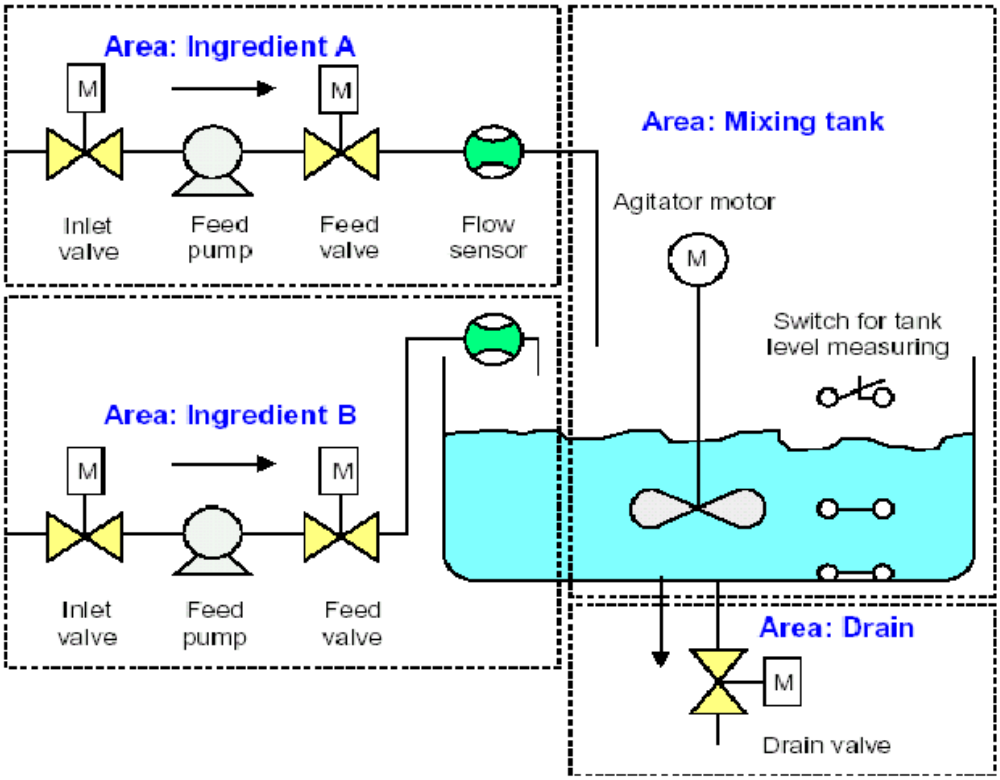
مطابق شکل فرآیند به ٤ ناحیه تقسیم شده است که عناصر کنترلی آنها عبارتند از:

نواحی ورودی A و B:

- ولو ورودی (Inlet Valve)
- پمپ تغذیه (Feed Pump)
- ولو تغذیه (Feed Valve)
- حس کننده دبی (Flow Sensor)

ناحیه تخلیه:

- سولنوئید ولو تخلیه (Drain Valve)



ناحیه تانک مخلوط کننده :

- موتور همزن
- سنسور ماکزیمم بودن سطح تانک از نوع نرمال بسته
- سنسور مینیمم بودن سطح تانک از نوع نرمال باز
- سنسور خالی بودن تانک از نوع نرمال باز

۳- لاجیک سیستم

لاجیک نواحی ورودی:

- اگر تانک پر شود موتورپمپ های تغذیه باید قطع شوند.
- وقتی ولو تخلیه باز است باید موتور پمپ های تغذیه فعال باشند.
- ولوهای قبل و بعد از موتور پمپهای تغذیه باید یک ثانیه بعد از استارت موتور پمپ ها باز شوند.
- ولوهای قبل و بعد از موتور پمپهای تغذیه باید وقتی پمپ های مزبور قطع میشوند بلافاصله بسته شوند (برای جلوگیری از نشتی پمپ که در اینحالت سیگنال از سنسور فلو می آید)
- سنسور فلو ۷ ثانیه بعد از استارت پمپ مقدار واقعی فلو را ارسال نماید.
- اگر سنسور فلو وقتی که پمپها فعال هستند مقدار فلوی صفر را بفرستد باید پمپ ها هر چه سریعتر قطع شوند.
- تعداد دفعات روشن شدن هر پمپ جهت مقاصد تعمیراتی شمارش شود.

لاجیک ناحیه تانک مخلوط کننده :

- اگر سطح تانک به مینیمم رسید موتور همزن قطع شود.
- اگر ولو تخلیه باز شد موتور همزن قطع شود.
- در صورتی که ۱۰ ثانیه بعد از فرمان استارت موتور فوق سیگنال فیدبک سرعت موتور دریافت نشد موتور قطع شود.
- تعداد دفعات روشن شدن موتور جهت مقاصد تعمیراتی شمارش شود.

لاجیک ناحیه تخلیه :

- باز و بسته شدن ولو تخلیه توسط فرمان انجام میشود.
- اگر تانک خالی شود باید ولو تخلیه بسته شده و پیغام "Tank Empty" تولید گردد.
- وقتی موتور همزن کار میکند باید ولو تخلیه بسته باشد.

٤- کنترل اپراتوری

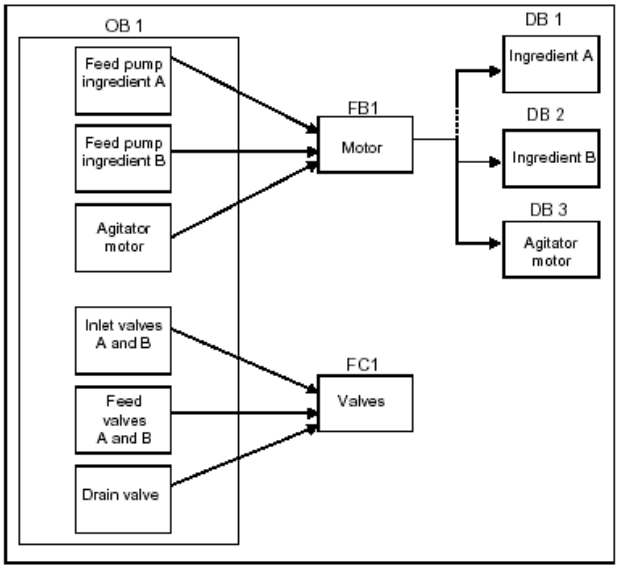
میز کنترل با وسایل زیر تجهیز شده است:

- سوئیچهای کنترلی Stop / Start
- سوئیچ Stop اضطراری
- لامپ های نشان دهنده وضعیت فرآیند
- سوئیچ ری ست کردن شمارنده تعمیرات

٥- بلاکهای مورد نیاز

شکل زیر ساختار کلی بلاکهای مورد نیاز را نشان میدهد نکات قابل توجه عبارتند از :

- برای موتورها بعلت یکسان بودن سیگنالهایی مانند On، Off و Count و ... از فانکشن بلاک FB استفاده شده است.
- دیتا بلاکهای DB1، DB2 و DB3 بصورت Instance و همگی مربوط به FB1 هستند . مقادیر واقعی و دیتاهای مربوط به هر موتور جداگانه در یکی از آنها ذخیره میشود.
- برای ولوها چون صرفاً فانکشن Open یا Close مورد نیاز بوده از یک FC استفاده شده است.



٦- جدول سمبل ها

با استفاده از Symbol Table جدول سمبلها را بصورت زیر تعریف مینماییم:

Symbole	Address	Data Type	Comment
Agitator	Q 8.0	BOOL	Activates the agitator
Agitator_fault	Q 8.3	BOOL	Display lamp for "Agitator motor fault"
Agitator_maint	Q 8.4	BOOL	Display lamp for "Agitator motor maintenance"
Agitator_off	Q 8.2	BOOL	Display lamp for "Agitator OFF"
Agitator_on	Q 8.1	BOOL	Display lamp for "Agitator ON"
Agitator_running	I 1.0	BOOL	Feedback signal from the agitator motor
Agitator_start	I 1.1	BOOL	Start pushbutton agitator
Agitator_stop	I 1.2	BOOL	Stop pushbutton agitator
DB_agitator	DB 3	FB 1	Instance DB for controlling agitator motor
DB_feed_pump_A	DB 1	FB 1	Instance DB for controlling feed pump A
DB_feed_pump_B	DB 2	FB 1	Instance DB for controlling feed pump B
Drain	Q 9.5	BOOL	Activates the drain valve
Drain_closed	I 0.7	BOOL	Pushbutton for closing drain valve
Drain_closed_disp	Q 9.7	BOOL	Display lamp for "Drain valve closed"
Drain_open	I 0.6	BOOL	Pushbutton for opening drain valve
Drain_open_disp	Q 9.6	BOOL	Display lamp for "Drain valve open"
EMER_STOP_off	I 1.6	BOOL	EMERGENCY STOP switch
Feed_pump_A	Q 4.4	BOOL	Activates the feed pump for ingredient A
Feed_pump_A_fault	Q 4.5	BOOL	Display lamp for "Feed pump A fault"
Feed_pump_A_maint	Q 4.6	BOOL	Display lamp for "Feed pump A maintenance"
Feed_pump_A_off	Q 4.3	BOOL	Display lamp for "Feed pump OFF ingredient A"
Feed_pump_A_on	Q 4.2	BOOL	Display lamp for "Feed pump ON ingredient A"
Feed_pump_A_start	I 0.0	BOOL	Start pushbutton feed pump for ingredient A
Feed_pump_A_stop	I 0.1	BOOL	Stop pushbutton feed pump for ingredient A
Feed_pump_B	Q 5.4	BOOL	Activates the feed pump for ingredient B
Feed_pump_B_fault	Q 5.5	BOOL	Display lamp for "Feed pump B fault"
Feed_pump_B_maint	Q 5.6	BOOL	Display lamp for "Feed pump B maintenance"
Feed_pump_B_off	Q 5.3	BOOL	Display lamp for "Feed pump OFF ingredient B"
Feed_pump_B_on	Q 5.2	BOOL	Display lamp for "Feed pump ON ingredient B"
Feed_pump_B_start	I 0.3	BOOL	Start pushbutton feed pump for ingredient B
Feed_pump_B_stop	I 0.4	BOOL	Stop pushbutton feed pump for ingredient B
Feed_valve_A	Q 4.1	BOOL	Activates the feed valve for ingredient A
Feed_valve_B	Q 5.1	BOOL	Activates the feed valve for ingredient B
Flow_A	I 0.2	BOOL	Ingredient A flows
Flow_B	I 0.5	BOOL	Ingredient B flows
Inlet_valve_A	Q 4.0	BOOL	Activates the inlet valve for ingredient A
Inlet_valve_B	Q 5.0	BOOL	Activates the inlet valve for ingredient B
Motor_block	FB 1	FB 1	FB for controlling pumps and agitator motor
Reset_maint	I 1.7	BOOL	Reset pushbutton for maintenance display (all
Tank_above_min	I 1.4	BOOL	Sensor "Mixing tank above minimum level"
Tank_below_max	I 1.3	BOOL	Sensor "Mixing tank not full"
Tank_empty_disp	Q 9.2	BOOL	Display lamp for "Mixing tank empty"
Tank_max_disp	Q 9.0	BOOL	Display lamp for "Mixing tank full"
Tank_min_disp	Q 9.1	BOOL	Display lamp for "Mixing tank below minimum l
Tank_not_empty	I 1.5	BOOL	Sensor "Mixing tank not empty"
Valve_block	FC 1	FC 1	FC for controlling valves

۷- طراحی Function Block (FB)

همانطور که میدانیم لازم است FB قبل از OB ایجاد شود . بدین منظور در پوشه بلاکها ، فانکشن بلاکی بنام FB1 ایجاد کرده و روی آن کلیک میکنیم تا توسط برنامه LAD/STL/FBD باز شود .

با توجه به اینکه یک FB برای همه موتورها منظور شده لازم است ابتدا سیگنالها و مواردی که برای همه موتورها بصورت عمومی بکار میرود را مشخص کنیم.یعنی ورودی ها ، خروجیها ی این فانکشن همراه با متغیرهای استاتیک یا احیاناً متغیرهای موقتی که در آن استفاده میشود. در این پروژه موارد فوق عبارتند از :

ورودی برای راه اندازی موتور (Start)

ورودی برای قطع موتور (Stop)

سیگنالی که نشان دهد موتور کار میکند بعنوان ورودی (Response)

طول زمانی که برای دریافت سیگنال فعال شدن موتور از لحظه استارت تعیین می شود. (Response_Time) این زمان بعنوان ورودی برای کنترل استفاده میشود یعنی تایمر ی به این اندازه صبر میکند.

تایمر برای قطع در صورتی که پس از زمان فوق سیگنال موتور دریافت نشده باشد(Timer_NO)

خروجی بعنوان فالت برای نمایش خطای فوق (Fault)

متغیر استاتیک برای ذخیره سازی مقدار تایمر (Timer_BCD)

خروجی برای نمایش فعال بودن موتور (Start_Dsp)

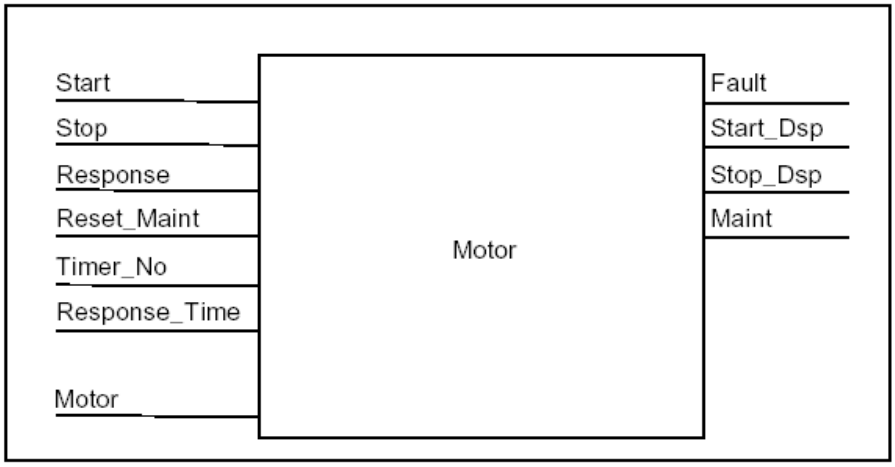
خروجی برای نمایش قطع بودن موتور (Stop_Dsp)

متغیر استاتیک برای ذخیره سازی تعداد دفعات استارت موتور (Starts)

خروجی برای وقتی که تعداد دفعات فوق به ۵۰ برسد برای تعمیرات (Maint)

ورودی از میز اپراتور برای ری ست کردن کانتر تعمیرات (Reset_Maint)

شکل زیر شمای کلی بلاک موتور رابا ورودی خروجی های مربوطه نمایش میدهد

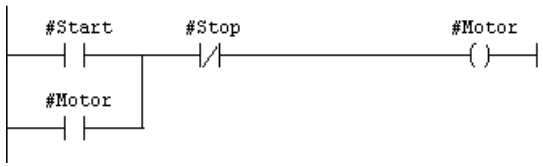


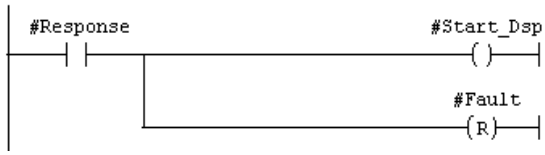
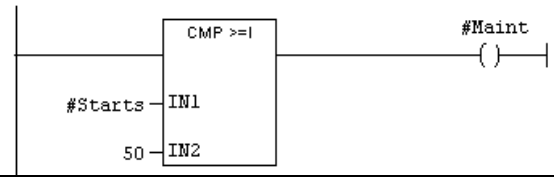
پس از طراحی این فانکشن بلاک اگردر OB1 از کاتالوگ کنار پنجره برنامه FB1 را در LAD بکار ببریم شکل فوق را خواهیم دید.

پارامتر ها را در جدول بالای FB تعریف میکنیم . لازم بذکر است ورودی ها(in) ، خروجی ها(out) و متغیرهای استاتیک (STAT)در دیتابلاک ذخیره میشوند ولی متغیرهای موقت (Temp) در LStack ذخیره شده و پس از پایان اجرای FB از بین میروند.

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Start	BOOL	FALSE	
0.1	in	Stop	BOOL	FALSE	
0.2	in	Response	BOOL	FALSE	
0.3	in	Reset_Maint	BOOL	FALSE	
2.0	in	Timer_No	TIMER		
4.0	in	Response_Time	S5TIME	S5T#0MS	
6.0	out	Fault	BOOL	FALSE	
6.1	out	Start_Dsp	BOOL	FALSE	
6.2	out	Stop_Dsp	BOOL	FALSE	
6.3	out	Maint	BOOL	FALSE	
8.0	in_out	Motor	BOOL	FALSE	
10.0	stat	Time_bin	WORD	W#16#0	
12.0	stat	Time_BCD	WORD	W#16#0	
14.0	stat	Starts	INT	0	
16.0	stat	Start_Edge	BOOL	FALSE	
	temp				

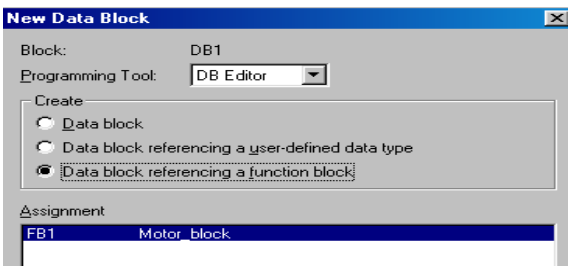
اکنون در قسمت Code Section برنامه را با توجه به لاجیک مورد نیاز مینویسیم این برنامه بصورت STL همراه با معادل LAD(در صورت وجود) در زیر ارائه شده است:

LAD	STL
سیگنالهای راه اندازی و قطع با یکدیگر AND شده و خروجی لازم را برای موتور ایجاد کرده اند . سیگنال خروجی با سیگنال راه اندازی OR شده تا پس از تحریک استارت بصورت خود نگهدار باقی بماند و نیازی به ۱ بودن سیگنال Start نباشد.	
	**Network 1 Start/stop and latching** A(O #Start O #Motor) AN #Stop = #Motor
با ۱ شدن سیگنال موتور در صورتی که پس از یک زمان تاخیری مشخص پاسخ موتور دریافت نشود سیگنال خطا فعال و سیگنال موتور غیر فعال میگردد. زمان تاخیر فوق الذکر در دیتابلاک ذخیره میگردد.	
-	**Network 2 Startup monitoring** A #Motor L #Response_Time SD #Timer_No AN #Motor R #Timer_No L #Timer_No T #Timer_bin LC #Timer_No T #Timer_BCD A #Timer_No AN #Response S #Fault R #Motor

روشن شدن موتور برای اپراتور نمایش داده میشود	
	**Network 3 Start lamp and fault reset** A #Response = #Start_Dsp R #Fault
توقف موتور برای اپراتور نمایش داده میشود	
	**Network 4 Stop lamp** AN #Response = #Stop_Dsp
تعداد دفعات روشن شدن موتور در دیتا بلاک ذخیره میشود	
-	**Network 5 Counting the starts** A #Motor FP #Start_Edge JCN lab1 L #Starts + 1 T #Starts lab1: NOP 0
اگر تعداد دفعات فوق به پنجاه برسد چراغ تعمیرات برای اپراتور روشن میشود	
	**Network 6 Maintenance lamp** L #Starts L 50 >=I = #Maint
اگر اپراتور کلید ری ست را فشار بدهد تعداد دفعات فوق صفر شده و چراغ خاموش میشود	
-	**Network 7 Reset counter for number of starts** A #Reset_Maint A #Maint JCN END L 0 T #Starts END: NOP 0

۸- ایجاد DB ها

در پوشه بلاکها سه DB به نامهای DB3 , DB2, DB1 ایجاد میکنیم . سپس روی تک تک آنها کلیک کرده و مطابق شکل زیرآن را به FB1 لینک مینماییم تا از نوع Instance باشند. پس از آن با باز شدن DB دیتاهایی که در جدول بالای FB تعریف کردیم را خواهیم دید .



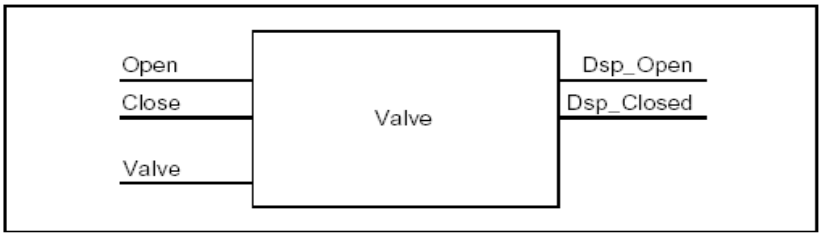
۹- طراحی فانکشن FC

همانند FB1 چون FC1 نیز از OB1 فراخوانده میشود باید قبل از OB1 طراحی و ایجاد شود . بدین منظور در پوشه بلاکها ، فانکشنی بنام FC1 ایجاد کرده و روی آن کلیک میکنیم تا توسط برنامه LAD/STL/FBD باز شود .

FC1 برای ولوهای ورودی ، تغذیه و تخلیه استفاده میشود طبق شکل زیر در این فانکشن برای هر ولو ورودی Open و Close را داریم ورودی Valve برای خود نگهدار شدن (Latching) استفاده میشود.

فانکشن FC1 وضعیت های باز یا بسته بودن ولو را بعنوان خروجی تولید کرده و در اختیار OB1 قرار میدهد.

بدیهی است وقتی FC1 از OB1 فرا خوانده میشود همانطور که در قسمت بعد خواهیم دید اینترلاکهایی که اجازه باز و بسته شدن ولو را میدهند در OB1 طراحی شده و نتیجه آنها بعنوان فرمان استارت به FC فرستاده میشود. بعلاوه در OB1 تعیین میشودکه خروجی های تولید شده توسط FC (باز و بسته بودن ولو) به کدام خروجی فیزیکی (لامپ نمایش) فرستاده شود.

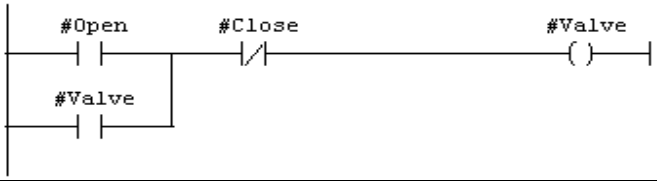


این پارامترها در جدول بالای FC بصورت زیر تعریف میگردند:

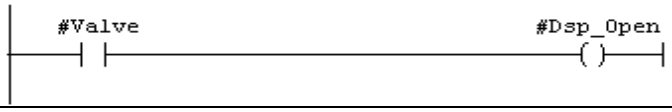
Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Open	BOOL		
0.1	in	Close	BOOL		
2.0	out	Dsp_Open	BOOL		
2.1	out	Dsp_Closed	BOOL		
4.0	in_out	Valve	BOOL		
	temp				

برنامه FC1 بصورت LAD در شکل زیر نشان داده شده و مفهوم هر بخش از آن بسادگی قابل فهم است.

Network 1 : Open/Close and Latching



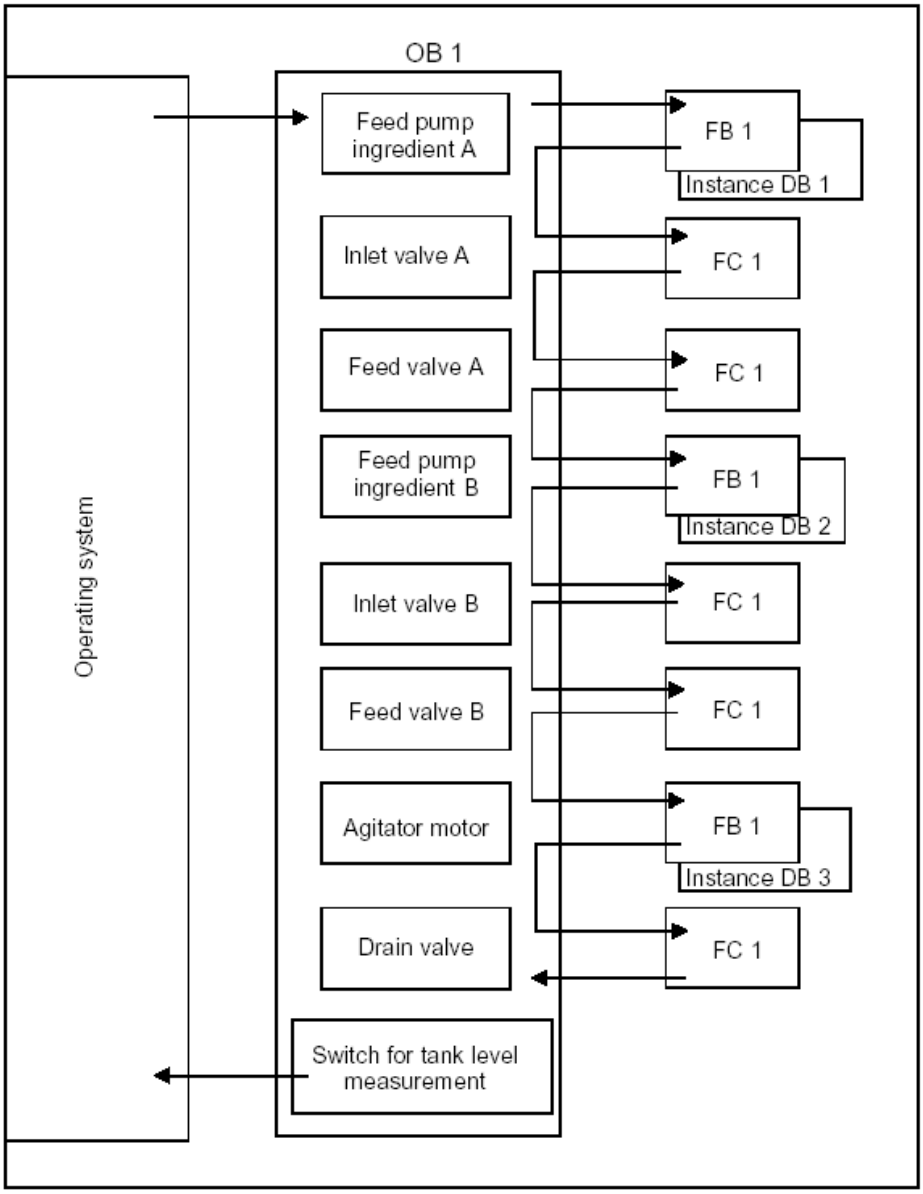
Network 2 : Display "Valve open"



Network 3 : Display "Valve Closed"

۱۰ - طراحی OB1

برنامه OB1 که بصورت سیکلی اجرا میشود با ساختار زیر تقسیم بندی و طراحی شده است :



در پوشه بلاکها روی OB1 که معمولاً از قبل موجود است کلیک مینماییم . پس از باز شدن بلاک در جدول بالای آن لیستی از متغیرهایی را می بینیم که مربوط به خود سیستم است . در انتهای آنها متغیرهای موقت مورد نظر را برای استفاده در برنامه OB1 وارد میکنیم . این متغیرها از Enable_motor مطابق شکل زیرشروع شده اند.

Address	Declaration	Name	Type	Init:	Comment
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), B
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB1)
2.0	temp	OB1_PRIORITY	BYTE		1 (Priority of 1 is lowest)
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 sca
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (mil
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (mil
12.0	temp	OB1_DATE_TIME	DATE_AND_TI		Date and time OB1 started
20.0	temp	Enable_Motor	BOOL		
20.1	temp	Enable_Valve	BOOL		
20.2	temp	Start_Fulfilled	BOOL		
20.3	temp	Stop_Fulfilled	BOOL		
20.4	temp	Inlet_Valve_A_Open	BOOL		
20.5	temp	Inlet_Valve_A_Closed	BOOL		
20.6	temp	Feed_Valve_A_Open	BOOL		
20.7	temp	Feed_Valve_A_Closed	BOOL		
21.0	temp	Inlet_Valve_B_Open	BOOL		
21.1	temp	Inlet_Valve_B_Closed	BOOL		
21.2	temp	Feed_Valve_B_Open	BOOL		
21.3	temp	Feed_Valve_B_Closed	BOOL		
21.4	temp	Open_Drain	BOOL		
21.5	temp	Close_Drain	BOOL		
21.6	temp	Close_Valve_Fulfille	BOOL		

متن برنامه بصورت STL در صفحات بعد با اسامی سمبلیک و بدون اسامی سمبلیک آورده شده است. در برنامه LAD/STL/FBD هرگاه بخواهیم برنامه را با یا بدون سمبل ببینیم کافی است از منوی View > Display With استفاده کنیم.

لازم به ذکر است در دستورات متن بلاکها اسامی سمبلیک داخل " " قرار میگیرند. کلماتی که با علامت # شروع شده اند مربوط به متغیرها و دیتاهای محلی بلاک هستند.

STL Program Without Symbolic Name	STL Program With Symbolic Name
	Network 1 Interlocks for feed pump A
A I 1.6 A I 1.3 AN Q 9.5 = #Enable_Motor	A "EMER_STOP_off" A "Tank_below_max" AN "Drain" = #Enable_Motor
	Network 2 Calling FB Motor for ingredient A
A I 0.0 A #Enable_Motor = #Start_Fulfilled A(O I 0.1 ON #Enable_Motor) = #Stop_Fulfilled CALL FB 1 , DB1 Start :=#Start_Fulfilled Stop :=#Stop_Fulfilled Response :=I0.2 Reset_Maint :=I1.7 Timer_No :=T12 Response_Time:=S5T#7S Fault :=Q4.5 Start_Dsp :=Q4.2 Stop_Dsp :=Q4.3 Maint :=Q4.6 Motor :=Q4.4	A "Feed_pump_A_start" A #Enable_Motor = #Start_Fulfilled A(O "Feed_pump_A_stop" ON #Enable_Motor) = #Stop_Fulfilled CALL "Motor_block", "DB_feed_pump_A" Start :=#Start_Fulfilled Stop :=#Stop_Fullfilled Response :="Flow_A" Reset_Maint :="Reset_maint" Timer_No :=T12 Reponse_Time:=S5T#7S Fault :="Feed_pump_A_fault" Start_Dsp :="Feed_pump_A_on" Stop_Dsp :="Feed_pump_A_off" Maint :="Feed_pump_A_maint" Motor :="Feed_pump_A"
	Network 3 Delaying the valve enable ingredient A
A Q 4.4 L S5T#1S SD T 13 AN Q 4.4 R T 13 A T 13 = #Enable_Valve	A "Feed_pump_A" L S5T#1S SD T 13 AN "Feed_pump_A" R T 13 A T 13 = #Enable_Valve
	Network 4 Inlet valve control for ingredient A
AN I 0.2 AN Q 4.4 = #Close_Valve_Fulfilled CALL FC 1 Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Feed_Valve_A_Open Dsp_Closed:=#Feed_Valve_A_Closed Valve :=Q4.1	AN "Flow_A" AN "Feed_pump_A" = #Close_Valve_Fulfilled CALL "Valve_block" Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Inlet_Valve_A_Open Dsp_Closed:=#Inlet_Valve_A_Closed Valve :="Inlet_Valve_A"
	Network 5 Feed valve control for ingredient A
AN I 0.2 AN Q 4.4 = #Close_Valve_Fulfilled CALL FC 1 Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Feed_Valve_A_Open Dsp_Closed:=#Feed_Valve_A_Closed Valve :=Q4.1	AN "Flow_A" AN "Feed_pump_A" = #Close_Valve_Fulfilled CALL "Valve_block" Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Feed_Valve_A_Open Dsp_Closed:=#Feed_Valve_A_Closed Valve :="Feed_Valve_A"

STL Program Without Symbolic Name	STL Program With Symbolic Name
	Network 6 Interlocks for feed pump B
A I 1.6 A I 1.3 AN Q 9.5 = #Enable_Motor	A "EMER_STOP_off" A "Tank_below_max" AN "Drain" = "Enable_Motor
	Network 7 Calling FB Motor for ingredient B
A I 0.3 A #Enable_Motor = #Start_Fulfilled A(O I 0.4 ON #Enable_Motor) = #Stop_Fulfilled CALL FB 1 , DB2 Start :=#Start_Fulfilled Stop :=#Stop_Fulfilled Response :=I0.5 Reset_Maint :=I1.7 Timer_No :=T14 Response_Time:=S5T#7S Fault :=Q5.5 Start_Dsp :=Q5.2 Stop_Dsp :=Q5.3 Maint :=Q5.6 Motor :=Q5.4	A "Feed_pump_B_start" A #Enable_Motor = #Start_Fulfilled A(O "Feed_pump_B_stop" ON #Enable_Motor) = #Stop_Fulfilled CALL "Motor_block", "DB_feed_pump_B" Start :=#Start_Fulfilled Stop :=#Stop_Fullfilled Response :="Flow_B" Reset_Maint :="Reset_maint" Timer_No :=T14 Reponse_Time:=S5T#7S Fault :="Feed_pump_B_fault" Start_Dsp :="Feed_pump_B_on" Stop_Dsp :="Feed_pump_B_off" Maint :="Feed_pump_B_maint" Motor :="Feed_pump_B"
	Network 8 Delaying the valve enable ingredient B
A Q 5.4 L S5T#1S SD T 15 AN Q 5.4 R T 15 A T 15 = #Enable_Valve	A "Feed_pump_B" L S5T#1S SD T 15 AN "Feed_pump_B" R T 15 A T 15 = #Enable_Valve
	Network 9 Inlet valve control for ingredient B
AN I 0.5 AN Q 5.4 = #Close_Valve_Fulfilled CALL FC 1 Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Inlet_Valve_B_Open Dsp_Closed:=#Inlet_Valve_B_Closed Valve :=Q5.0	AN "Flow_B" AN "Feed_pump_B" = #Close_Valve_Fulfilled CALL "Valve_block" Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Inlet_Valve_B_Open Dsp_Closed:=#Inlet_Valve_B_Closed Valve :="Inlet_Valve_B"
	Network 10 Feed valve control for ingredient B
AN I 0.5 AN Q 5.4 = #Close_Valve_Fulfilled CALL FC 1 Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Feed_Valve_B_Open Dsp_Closed:=#Feed_Valve_B_Closed Valve :=Q5.1	AN "Flow_B" AN "Feed_pump_B" = #Close_Valve_Fulfilled CALL "Valve_block" Open :=#Enable_Valve Close :=#Close_Valve_Fulfilled Dsp_Open :=#Feed_Valve_B_Open Dsp_Closed:=#Feed_Valve_B_Closed Valve :="Feed_Valve_B"

STL Program Without Symbolic Name	STL Program With Symbolic Name
	Network 11 Interlocks for agitator
A I 1.6 A I 1.4 AN Q 9.5 = #Enable_Motor	A "EMER_STOP_off" A "Tank_above_min" AN "Drain" = #Enable_Motor
	Network 12 Calling FB Motor for agitator
A I 1.1 A #Enable_Motor = #Start_Fulfilled A(O I 1.2 ON #Enable_Motor) = #Stop_Fulfilled CALL FB 1 , DB3 Start :=#Start_Fulfilled Stop :=#Stop_Fulfilled Response :=I1.0 Reset_Maint :=I1.7 Timer_No :=T16 Response_Time:=S5T#10S Fault :=Q8.3 Start_Dsp :=Q8.1 Stop_Dsp :=Q8.2 Maint :=Q8.4 Motor :=Q8.0	A "Agitator_start" A #Enable_Motor = #Start_Fulfilled A(O "Agitator_stop" ON #Enable_Motor) = #Stop_Fulfilled CALL "Motor_block", "DB_Agitator" Start :=#Start_Fulfilled Stop :=#Stop_Fullfilled Response :="Agitator_running" Reset_Maint :="Reset_maint" Timer_No :=T16 Reponse_Time:=S5T#10S Fault :="Agitator_fault" Start_Dsp :="Agitator_on" Stop_Dsp :="Agitator_off" Maint :="Agitator_maint" Motor :="Agitator"
	Network 13 Interlocks for drain valve
A I 1.6 A I 1.5 AN Q 8.0 = #Enable_Valve	A "EMER_STOP_off" A "Tank_not_empty" AN "Agitator" = "Enable_Valve
	Network 14 Drain valve control
A I 0.6 A #Enable_Valve = #Open_Drain A(O I 0.7 ON #Enable_Valve) = #Close_Drain CALL FC 1 Open :=#Open_Drain Close :=#Close_Drain Dsp_Open :=Q9.6 Dsp_Closed:=Q9.7 Valve :=Q9.5	A "Drain_open" A #Enable_Valve = #Open_Drain A(O "Drain_closed" ON #Enable_Valve) = #Close_Drain CALL "Valve_block" Open :=#Open_Drain Close :=#Close_Drain Dsp_Open :="Drain_open_disp" Dsp_Closed :="Drain_closed_disp" Valve :="Drain"
	Network 15 Tank level display
AN I 1.3 = Q 9.0 AN I 1.4 = Q 9.1 AN I 1.5 = Q 9.2	AN "Tank_below_max" = "Tank_max_disp" AN "Tank_above_min" = "Tank_min_disp" AN "Tank_not_empty" = "Tank_empty_disp"

۷– ارتباط On-Line با PLC

مشتمل بر :

۷–۱ Download کردن به PLC و Upload کردن از آن

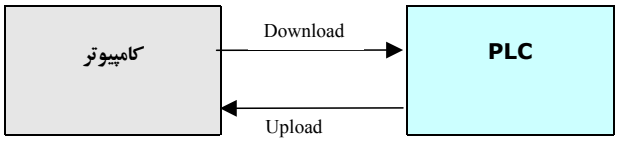
۷–۲ ارتباط On-Line با PLC از طریق Simatic Manager

۷–۳ ارتباط On-Line با PLC از طریق Hwconfig

۷–٤ ارتباط On-Line با PLC از طریق LAD/STL/FBD

۱-۷ Download کردن به PLC و Upload از آن

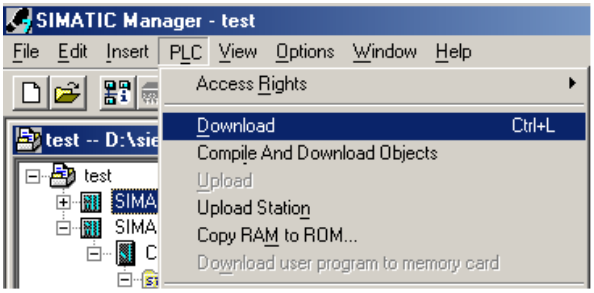
انتقال اطلاعات از کامپیوتر به PLC را Download و عکس آن یعنی انتقال از PLC به کامپیوتر را Upload گویند.

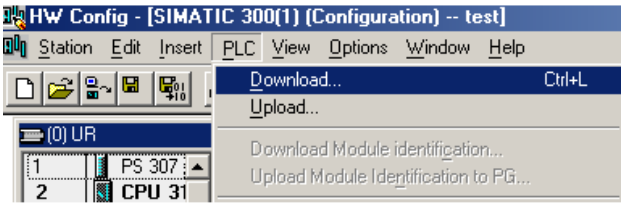


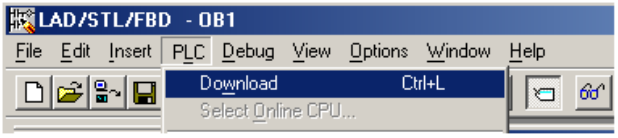
در بدو امر حافظه PLC خالیست و لازم است اطلاعات به آن Download شود. ولی در پروژه های اتوماسیون که طراح آن جدا از کاربر است و معمولا اطلاعات قبلا به PLC ارسال شده است کاربر قبل از هرگونه اقدامی لازم است عمل Upload را انجام داده و اطلاعات را به کامپیوتر منتقل نماید. پس از آن مبادرت به Download کردن یا ری ست کردن PLC بنماید.

با توضیحات فوق و با توجه به آنچه در بخش اول کتاب در مورد نحوه ایجاد ارتباط On-Line بیان شد بحث را پی میگیریم.

اگر به منوهای Simatic Manger و Hwconfig و LAD/STL/FBD نگاهی بیندازیم می بینیم که در منوی PLC آنها انتخاب به Download و Upload وجود دارد. شکل زیر:







باید توجه داشت که :

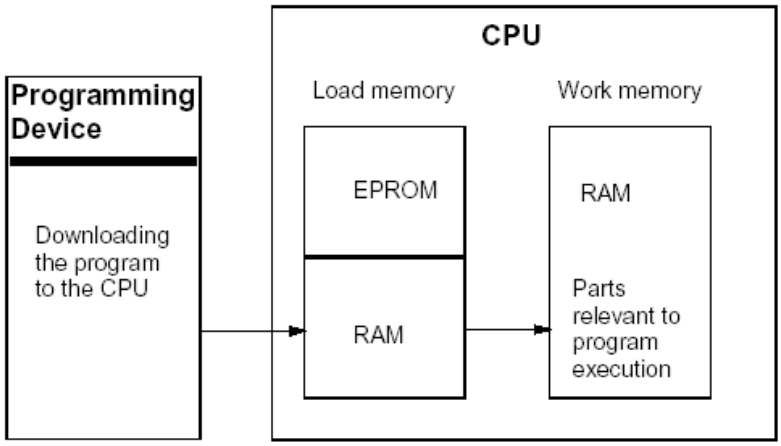
- از LAD/STL/FBD فقط میتوانیم برنامه را به PLC بفرستیم.
- از Hwconfig فقط میتوانیم اطلاعات سخت افزاری را به PLC بفرستیم یا از آن بگیریم.
- از Simatic Manager میتوانیم کل اطلاعات Station شامل سخت افزار و نرم افزار را به PLC بفرستیم یا از آن بگیریم.

آیکون Download بصورت  و آیکون Upload بصورت  در Toolbars برنامه ها نیز موجود است.

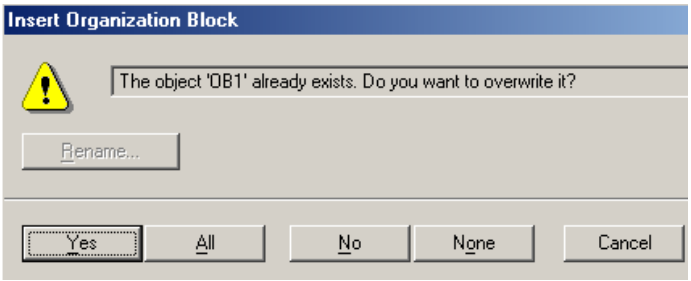
Download کردن

در ارتباط با Download اطلاعات به PLC نکات زیر را باید مد نظر قرار داد:

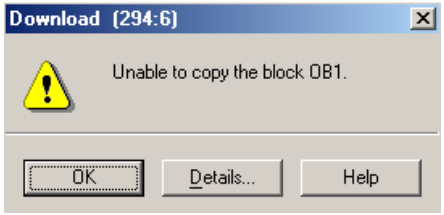
- اطلاعات پس از دانلود مانند شکل زیر ابتدا به Load Memory منتقل شده و از آنجا بخشهای اجرایی مورد نیاز به Work Memory ارسال میشوند.



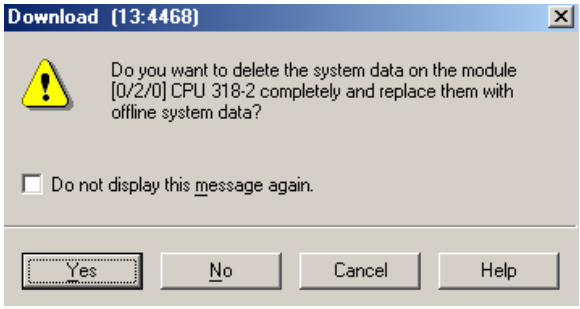
- جدول سمبلها و بلاک های UDT و VATقابل دانلود کردن نیستند و روی هارد کامپیوتر باقی میمانند .
- بهتر است قبل از دانلود کردن حافظه PLC بطور کامل ری ست گردد تا بلاکهای غیر ضرور پاک شده و کل اطلاعات از نو ارسال شود.
- اگر حافظه ری ست نشود و بلاکی همنام آنچه قبلا در حافظه وجود داشته بخواهد دانلود شود. پیغامی مانند شکل زیر ظاهر میشود که در صورت انتخاب Yes بلاک Overwrite میگردد.



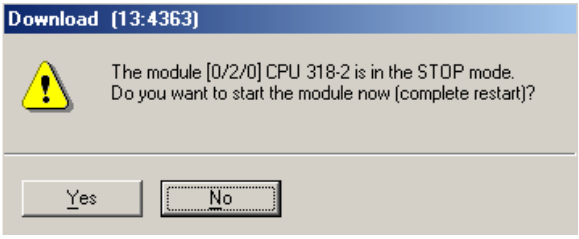
- در هنگام دانلود کردن باید توجه داشت که PLC در مد RUN نباشد. در غیر اینصورت پیغام زیر ظاهر میگردد. ارسال اطلاعات به PLC فقط در مد STOP و RUN-P امکان پذیر است.



- اگر اطلاعات سخت افزار یعنی System Data که در پوشه بلاک موجود است به PLC که از قبل پیکر بندی شده و این دیتا ها به آن دانلود شده ارسال شود برای جایگزین کردن مجدد آنها در پنجره ای بصورت زیر سوال میکند:



- اگردر مد RUN-P یعنی در حالیکه PLC مشغول اجرای برنامه است System Data به PLC دانلود شود نیاز به توقف PLCپیش می آید و برای اینکار سوال میکند . درصورت تایید برای راه اندازی مجدد نیز پیغام زیر ظاهر میشود.

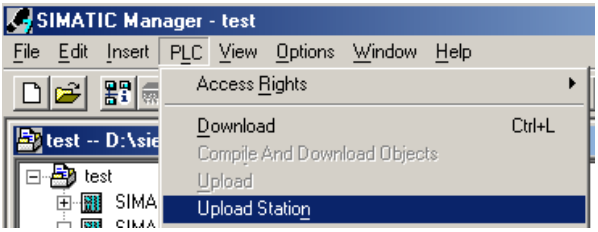


- وقتی در LAD/STL/FBD در حالیکه بلاک باز است عمل دانلود را انجام بدهیم اطلاعات موجود در بلاک هرچند ذخیره نشده باشند به PLC منتقل میشود. ولی اگر بلاک از طریق پوشه Blocks در Simatic Manager دانلود شود صرفا آخرین اطلاعات ذخیره شده در بلاک منتقل میگردد.
- در برنامه نویسی ساختار یافته که بلاک هایی از داخل بلاکهای دیگر صدا زده میشوند . لازم است کل بلاکها به PLC دانلود شود. در غیر اینصورت با فراخوانی بلاکی که در حافظه موجود نیست PLC به مد Stop میرود و چراغ SF روشن میگردد.
- برای انتقال کل بلاکها به PLC کافی است در Simatic Manager ابتدا روی پوشه Blocks کلیک کرده سپس عمل دانلود را انجام دهیم. همینطور میتوان تعدادی از بلاک ها را در پوشه فوق انتخاب کرد و دانلود نمود.
- در پنجره Simatic Manager میتوان برای دانلود بجای منوهای برنامه با استفاده از کلیک راست ماوس نیز این کار را انجام داد.
- اگر ساختار سخت افزار پیکر بندی شده در Step7 با آنچه بصورت واقعی وجود دارد تفاوت داشته باشد. در هنگام دانلود با پیغام خطا مواجه میشویم. و بعضا ممکن است PLC نیز با وجود خطاهای مزبور راه اندازی نشود. برای جلوگیری از بروز این شرایط توصیه میشود پیکر بندی سخت افزار قبل از دانلود با کدهای زیمنس که روی مدولها نوشته شده تطبیق داده شود و در صورت نیاز اصلاحات لازم در پیکر بندی بعمل آید.

Upload کردن

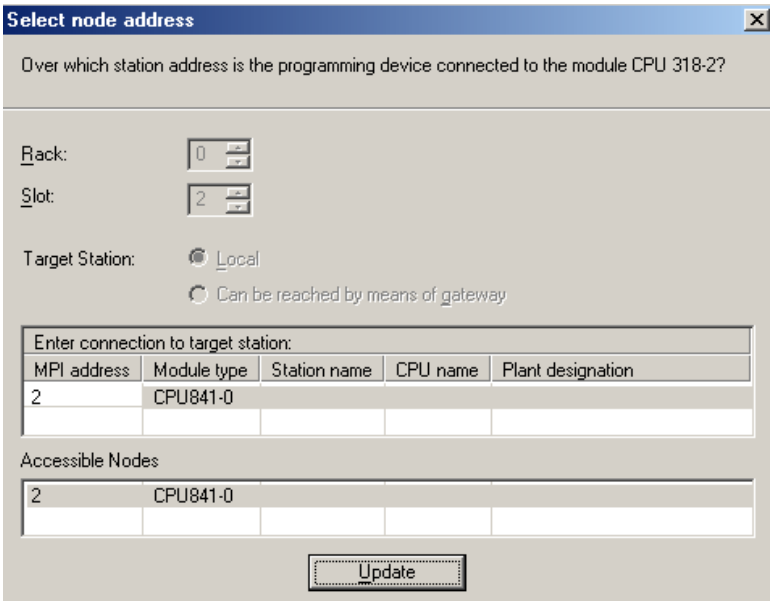
در ارتباط با Upload کردن اطلاعات از PLC نکات زیر را باید مد نظر قرار داد:

- Upload کردن در تمام مدهای کاری PLC امکان پذیر است.
- برای Upload کردن کل اطلاعات موجود در PLC لازم است از Simatic Manager و منوی PLC>upload station استفاده کنیم.



- بهتر است قبل از Upload کردن در Simatic Manager یک پروژه جدید ایجاد کرده و اطلاعات را به آن وارد نماییم.
- پس از ایجاد پروژه جدید و انتخاب Upload Station از منوی PLC پنجره ای مانند شکل زیر باز میشود. در این پنجره میتوان آدرس MPI مربوط به PLC را وارد کرد. اگر فقط یک PLC داشته باشیم این آدرس معمولا ۲ می باشد ولی درحالتیکه چند PLC روی شبکه داریم یا از آدرس ۲ فوق مطمئن نیستیم بهتر است روی View کلیک کرده تا تمام PLC های قابل دسترس را در لیست ببینیم.

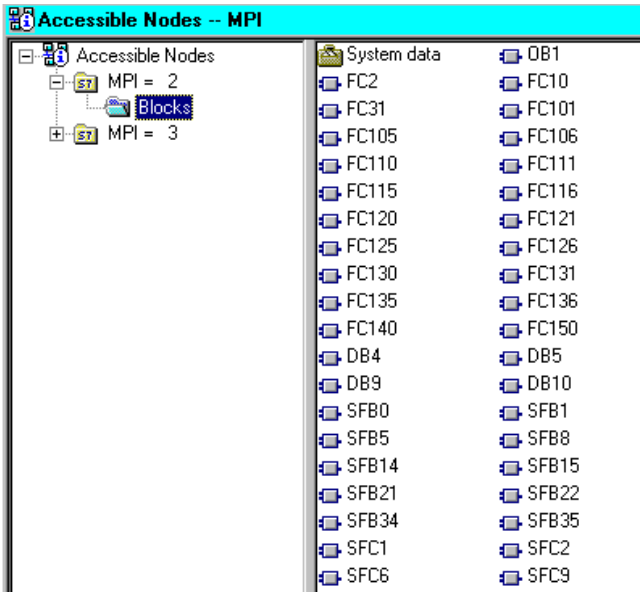
با انتخاب PLC مورد نظر و OK کردن عمل Upload انجام میگیرد و جزئیات Station را در پنجره SimatiC Manager مشاهده خواهیم کرد.



- وقتی Upload از یک PLC که قبلا پیکر بندی و برنامه نویسی شده است برای اولین بار انجام میشود لازم است اطلاعات دریافت شده را در محل مطمئنی ذخیره و نگهداری نمود. بعلاوه در این حالت بهتر است پروژه Archive گردد.
- VAT و UDT و جدول سمبلها همانطور که قابل دانلود نیستند. در بلاک های Upload شده نیز ظاهر نمیشوند.
- اسامی سمبلیک که برای متغیرهای محلی بلاک ها تعریف شده از بین میرود و با اسامی Temp نمایش داده میشوند.

۷-۲ ارتباط On-Line با PLC از طریق Simatic Manager

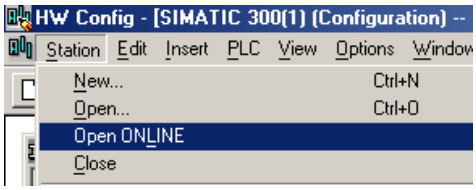
برای مشاهده بلاک های موجود در PLC میتوان از منوی PLC>Display Accessible Nodes استفاده کرد. پنجره ای مانند شکل زیر ظاهر میشود.



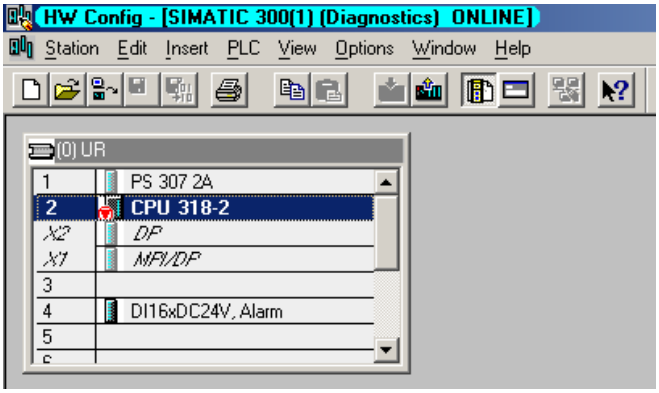
- در سمت چپ این پنجره PLC های قابل دسترسی از روی شبکه با آدر سهای MPI نمایش داده میشوند. اگر فقط یک PLC داشته باشیم این آدرس معمولا ۲ میباشد.
- با کلیک کردن روی هر آدرس MPI مورد نظر لیستی از بلاکهای موجود در PLC در پنجره سمت راست ظاهر میشود. در این روش اطلاعات سخت افزاری قابل دسترسی نیست.
- در لیست بلاکها علاوه بر بلاک های تهیه شده توسط کاربر بلاکهای دیگری از نوع بلاکهای سیستم مشاهده میکنیم که توسط خود PLC اضافه شده اند.
- در همین پنجره میتوان بلاک یا بلاکهایی را Delete کرد . در واقع بلاک از حافظه PLC و نه از روی کامپیوتر پاک میشود.
- همینطور میتوان بلاکی را از یک پروژه که در Simatic Manager بصورت Off-Line باز است کپی کرد و در پنجره فوق Paste نمود اینکار شبیه دانلود کردن بلاک مزبور به PLC میباشد.
- در Simatic Manager بعلاوه میتوان از منوی View > On line استفاده نمود.که در این حالت علاوه بر بلاک ها وضعیت سخت افزار نیز نشان داده میشود. در این حالت لازم است برای دیدن آخرین وضعیت مرتبا عمل Update را ازطریق منوی View و یا کلید F5 انجام دهیم.

۷-۳ ارتباط On-Line با PLC از طریق Hwconfig

با استفاده از منوی Station > open Online (شکل زیر) میتوان سخت افزار سیستم را بصورت On-Line مشاهده نمود .



در حالت On-Line پنجره Hwconfig بصورت شکل زیر ظاهر میشود:

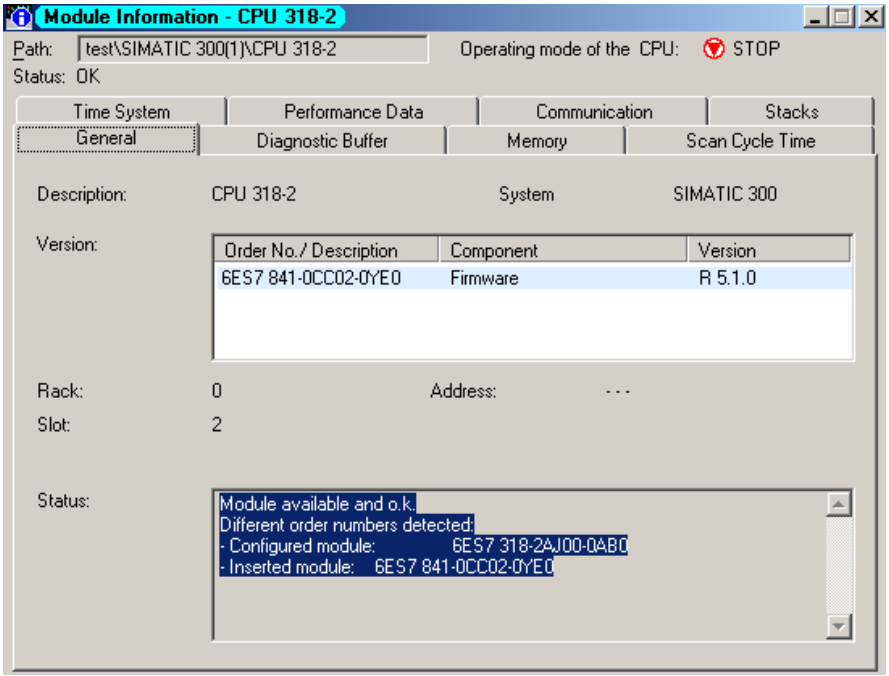


در اینحالت در کنار المانهای سخت افزاری ممکن است یکی از علامتهای زیر ظاهر شود که نشان دهنده وضعیت فعلی آن مدول میباشد.

Meaning	Symbol
STARTUP	
STOP	
RUN	
HOLD	

علامتهای دیگری نیز وجود دارند که بیانگر وجود اشکال هستند و در بحث عیب یابی در جلد دوم به آنها پرداخته میشود.

اگر در Hwconfig که بصورت On-Line باز است روی CPU در رک کلیک کنیم پنجره ای باز میشود که اطلاعات آن با حالت Off-Line متفاوت است مانند شکل زیر :



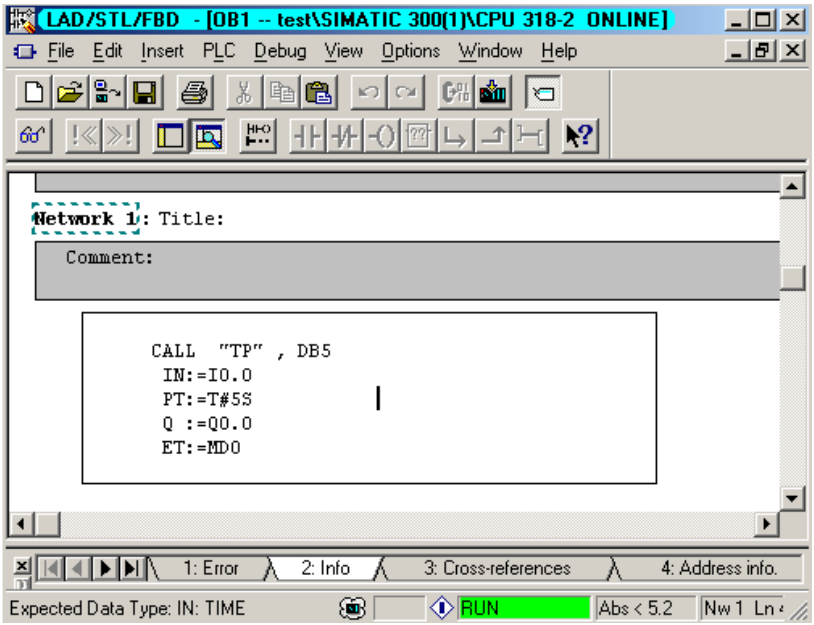
این پنجره در Simatic Manager نیز توسط منوی **PLC> Module Information** فعال میگردد

جدول زیر به اختصار بخشهای مختلف این پنجره را معرفی مینماید:با توجه به اهمیت این پنجره برای خطا یابی در جلد دوم کتاب به تفصیل در مورد آن صحبت خواهد شد.

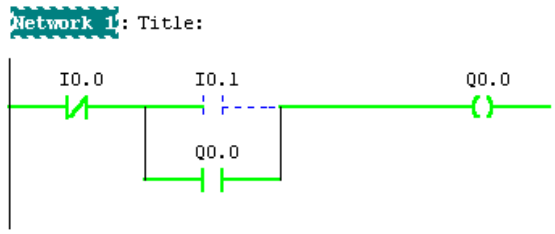
بخش	کاربرد
General	مشاهده مشخصات کلی CPU
Diagnostic Buffer	برای شناسایی علت توقف CPU و ارزیابی مدولهای مرتبط با این فالت
Memory	برای فشرده سازی حافظه
Scan Cycle Time	مشاهده زمان سیکل اسکن واقعی CPU
Time System	برای نمایش و تنظیم کردن زمان و تاریخ مدول و چک سنکرون بودن زمان
Performance Data	معرفی مشخصات عملکردی CPU مانند سایز Load Memory یا PII و...
Communication	اینکه چه تعداد ارتباط با CPU امکان پذیر است و چه تعداد هم اکنون اشغال شده است.
Stacks	برای تعیین علت توقف CPU و عیب یابی با استفاده از اطلاعات Istack و Bstack

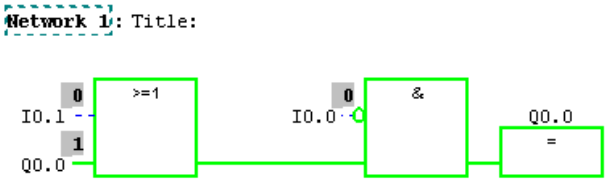
۷-٤ ارتباط On-Line با PLC از طریق LAD/STL/FBD

با استفاده از منوی File >Open OnLine یا از طریق آیکون عینک Monitor  میتوان بلاک را بصورت On-Line مشاهده نمود. البته آیکون فوق در صورتی ظاهر میشود که بلاک به PLC دانلود شده و هیچ تفاوتی بین بلاک موجود در PLC و بلاک روی کامپیوتر نباشد.پس از ارتباط On-Line پنجره به شکل زیر در می آید . مد RUN با رنگ سبز و مد Stop با رنگ قرمز در پایین پنجره ظاهر میشود.

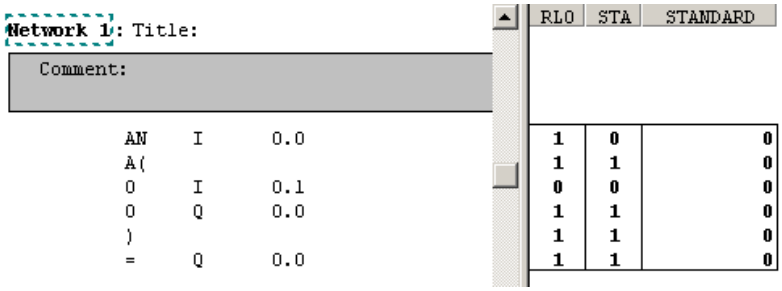


اگر برنامه بصورت LAD یا FBD باشد وضعیت سیگنالها میتوان مشاهده کرد هر جا اتصال برقرار است با رنگ سبز مانند شکل های زیر ظاهر میشود.

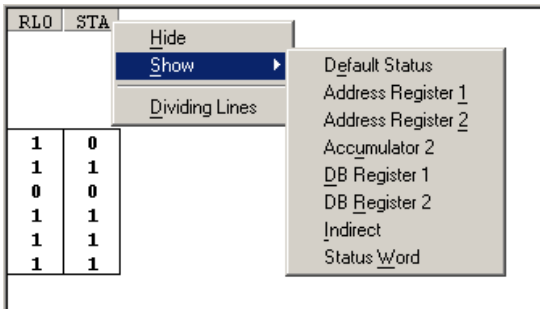




اگر برنامه بصورت STL باشد بجای دیدن اتصالات وضعیت سیگنالها را میتوان مشاهده کرد. بطور معمول RLO و STA از بیت های Status Word در جلوی هر سطر از برنامه مشاهده میشوند که برای دستورات Bit Logic مقدار دارند. برای سیگنالهای غیر لاجیک ستون Standard مقدار HEX را نشان میدهد.



اگر در پنجره سمت راست کلیک کنیم میتوانیم انتخاب کنیم که چه اطلاعات دیگری در حالت On line نشان داده شوند.(شکل زیر)



استفاده از مد Online برای فهم برنامه ، تست برنامه و نیز خطا یابی مفید است.

ضمیمه ۱

لیست بخشهای مختلف استاندارد IEC1131

IEC 61131-1 General Information

TABLE OF CONTENT - IEC 61131-1
(incl. page number)

IEC 61131-2 - Equipment Requirements and Tests

Table of Content IEC 61131-2
(incl. page numbers)

2.17	EMC (electromagnetic compatibility)
2.18	Enclosed Equipment
2.19	Enclosure
2.20	Equipment class
2.21	Equipment Under Test (EUT)
2.22	External Wiring
2.23	Field wiring
2.24	Functional Earthing Conductor
2.25	Hand-Held Equipment
2.26	Hazardous Live
2.27	Immunity (to a disturbance)
2.28	Immunity Type Test (immunity test)
2.29	Insulation
2.30	Interface
2.31	Internal Wiring
2.32	Isolated (devices, circuits)
2.33	Live Part
2.34	Mains Power Supply
2.35	Material Group
2.36	Micro-Environment:
2.37	Module
2.38	Multi-Channel Module
2.39	Normal Use
2.40	Normal Condition
2.41	Open Equipment
2.42	Operator
2.43	Overvoltage Category (of a circuit or within an electrical system)
2.44	Permanent Installation
2.45	Pollution Degree (in the micro-environment)
2.46	Port
2.47	Portable Equipment
2.48	Protective Conductor
2.49	Protective Extra-Low Voltage (PELV) Circuit
2.50	Protective Impedance
2.51	Recurring Peak Voltage
2.52	Routine Test
2.53	Safety Extra-Low Voltage Circuit (SELV circuit)
2.54	Service Personnel
2.55	Total Output Current (of an output module)
2.56	Type Test
2.57	Unit
2.58	Withstand Type Test (withstand test)
2.59	Working Voltage
3	Normal Service Conditions and Requirements
3.1	Climatic Conditions and Requirements
3.2	Mechanical Service Conditions and Requirements
3.3	Transport and Storage Conditions and Requirments
3.4	Electrical Service Conditions and Requirements
3.5	Special Conditions and Requirements
4	Functional Requirements
4.1	Functional Power Supply and Memory Back-up Requirements
4.2	Digital I/Os
4.3	Analog I/Os
4.4	Communication Interface Requirements
4.5	Main Processing Unit(s) and Memory(ies) of the PLC-system Requirements
4.6	Remote Input/Output Stations (RIOSs) Requirements

4.7	Peripherals (PADTs, TEs, HMIs) Requirements
4.8	PLC-system Self-Tests and Diagnostics Requirements
4.9	Functional Earthing
4.10	Mounting Requirments
4.11	General Marking Requirements
4.12	Requirements for Normal Service and Functional Type Tests and Verifications
4.13	Requirements for Information on Normal Service and Function
5	Normal Service and Functional Type Tests and Verifications
5.1	Climatic Tests
5.2	Mechanical Tests
5.3	Verification of Special Functional Requirements for Power Ports and Memory Back-up -- Special Immunity Limits for Power Ports
5.4	Verification of Input/Output Requirements
5.5	Verification of Communication Interface Requirements
5.6	Verification of MPU Requirements
5.7	Verification of Remote I/O Stations
5.8	Verification of Peripheral (PADTs, TEs, HMIs) Requirements
5.9	Verification of PLC-system Self-Tests and Diagnostics
5.10	Verification of Markings and Manufacturer's Documentation
6	General Information to be Provided by Manufacturer
6.1	Information on Type and Content of Documentation
6.2	Information on Compliance With This Standard
6.3	Information on Reliability
6.4	Information on Other Conditions
6.5	Information on Shipping and Storage
6.6	Information on AC and DC Power Supply
6.7	Information on Digital Inputs (current sinking)
6.8	Information on Digital Outputs for Alternating Currents (current sourcing)
6.9	Information on Digital Outputs for Direct Current (current sourcing)
6.10	Information on Analog Inputs
6.11	Information on Analog Outputs
6.12	Information on Communication Interfaces
6.13	Information on Main Processing Unit(s) and Memory(ies) of the PLC-system
6.14	Information on Remote Input/Output Stations (RIOSs)
6.15	Information on Peripherals (PADTs, TEs, HMIs)
6.16	Information on Self-Tests and Diagnostics
7	Electromagnetic Compatibility (EMC) Requirements
7.1	General
7.2	Emission Requirements
7.3	EMC Immunity Requirements
7.4	Requirements for EMC Tests and Verifications
7.5	Requirements for Information on EMC
8	Electromagnetic Compatibility (EMC) Type Tests and Verifications
8.1	Electromagnetic Compatibility Related Tests
8.2	Test Environment
8.3	Measurement of Radiated Interference
8.4	Measurement of Conducted Interference
8.5	Electrostatic Discharge
8.6	Radio Frequency Electromagnetic Field - Amplitude Modulated
8.7	Power Frequency Magnetic Fields
8.8	Fast Transient Bursts
8.9	High Energy Surges
8.10	Conducted Radio Frequency Interference
8.11	Damped Oscillatory Wave (for Zone C only)
8.12	Voltage Drops and Interruptions Power Ports Type Tests and Verifications
9	Electromagnetic Compatibility (EMC) Information to be Provided by Manufacturer

10	Safety Requirements
10.1	Protection Against Electrical Shock
10.2	Protection Against the Spread of Fire
10.3	Limited Power Circuits
10.4	Clearance and Creepage Distances Requirements
10.5	Flame Retardant Requirements for Non-Metallic Materials
10.6	Temperature Limits
10.7	Enclosures
10.8	Field Wiring Terminals Constructional Requirements
10.9	Provisions for Protective Earthing
10.10	Wiring
10.11	Switching Devices
10.12	Components
10.13	Battery Requirements
10.14	Maximum Voltage and Minimum Voltage
10.15	Markings and Identification
10.16	Requirements for Safety Type Tests and Verifications
10.17	Requirements for Safety Routine Tests and Verifications
10.18	Requirements for Information on Safety
11	Safety Type Tests and Verifications
11.1	Safety Related Mechanical Tests and Verifications
11.2	Safety Related Electrical Tests
11.3	Single Fault Condition Test – General
12	Safety Routine Tests
12.1	Dielectric Withstand Test
12.2	Dielectric Withstand Verification Test
12.3	Protective Earthing Test
13	Safety Information to be Provided by the Manufacturer
13.1	Information on Evaluation of Enclosures for Open Equipment (power dissipation)
13.2	Information on Mechanical Terminal Connection
Annex A	(Informative) Illustration of PLC-system Hardware Definitions
Annex B	(Informative) Digital Input Standard Operating Range Equations
Annex C	(Normative) Test Tools
C.1	Jointed Test Finger
C.2	Test Pins
Annex D	(Informative) Zone C EMC Immunity Levels
Annex E	(Informative) Overvoltage Example
Annex F	(Informative) Bibliography

IEC 61131-3 Programming Languages

TABLE OF CONTENT IEC 61131-3

1.	General
1.1	Scope
1.2	Normative references
1.3	Definitions
1.4	Overview and general requirements
1.5	Compliance
2.	Common elements
2.1	Use of printed characters

2.2	External representation of data
2.3	Data types
2.4	Variables
2.5	Program organization units
2.6	Sequential Function Chart (SFC) elements
2.7	Configuration elements
3.	Textual languages
3.1	Common elements
3.2	Instruction list (IL)
3.3	Structured Text (ST)
4.	Graphic languages
4.1	Common elements
4.2	Ladder Diagram (LD)
4.3	Function Block Diagram (FBD)
ANNEX A	Specification method for textual languages (normative)
A.1	Syntax
A.2	Semantics
ANNEX B	Formal specifications of language elements (normative)
B.0	Programming model
B.1	Common elements
B.2	Language IL (Instruction List)
B.3	Language ST (Structured Text)
ANNEX C	Delimiters and Keywords (normative)
ANNEX D	Implementation-dependent parameters (normative)
ANNEX E	Error Conditions (normative)
ANNEX F	Examples (informative)
F.1	Function WEIGH
F.2	Function block CMD_MONITOR
F.3	Function block FWD_REV_MON
F.4	Function block STACK_INT
F.5	Function block MIX_2_BRIX
F.6	Analog signal processing
F.7	Program GRAVEL
F.8	Program AGV
F.9	Use of enumerated data types
F.10	Function block RTC (Real Time Clock)
F.11	Function block ALRM_INT
ANNEX G	Index (informative)
ANNEX H	Reference character set (informative)

IEC 61131- 4 User Guidelines

TABLE OF CONTENT IEC 61131-4

1	General
1.1	Scope
1.2	References
1.3	Use of this report
2	Definitions
2.1	Application program (user program)
2.2	Automated system·

2.3	PLC-system
2.4	Programmable controller (PLC)
2.5	Programmable controller system (PLC-system)
3	Key elements of IEC 61131-1, Part 1: General information
3.1	PLC hardware model
3.2	Basic functional structure of a PLC system
3.3	CPU function
3.4	Programming languages
3.5	Availability and reliability
3.6	Re-start of PLC operating system
3.7	Documents supplied to the user
4	Key elements of IEC 61131-2: Part 2 - Equipment requirements and testing
4.1	Scope of Part 2
4.2	Definitions
4.3	Functional requirements
4.4	Electromagnetic compatibility (EMC) requirements
4.5	Safety requirements for PLC equipment
4.6	Information to be provided by the manufacturer
4.7	Compliance with IEC 61131-2 standard
5	Key elements of IEC61131-3: Part 3 - Programming Languages
5.1	Technical contents of Part 3
5.2	Common Elements
5.3	Programming languages
5.4	Applications program development
5.5	Implementation
6	Key Elements of IEC61131-5, Part 5 - Communication
6.1	Scope of Part 5
6.2	Technical contents of IEC61131-5 and the PLC model
6.3	Communication Model
6.4	PLC communication services
6.5	Application functions
6.6	Communication function blocks
6.7	Compliance
7.	Key elements of IEC61131-7, Part 7 - Fuzzy Logic programming
7.1	General outline of Part 7
7.2	Integration of fuzzy control application into the programmable controllers
7.3	Fuzzy Control Language (FCL)
7.4	Exchange of fuzzy control programs
7.5	Compliance
8	General rules for installation
8.1	Environmental conditions
8.2	Field wiring
8.3	Electromagnetic compatibility
8.4	User system markings
9	PLC in Functional safety applications
9.1	Using a PLC in a safety-related application
9.2	Safety-related system – Safety requirements
9.3	PLC requirements in a safety-related system
9.4	Integration of PLC into safety-related system
Annex A	User checklists
A.1	Equipment data
A.2	General information checklist
A.3	Equipment requirements checklist
A.4	Checklist on programming languages
A.5	Checklist on communication
A.6	Checklist on Fuzzy control language

IEC 61131-5 Messaging service specification (Communication)

TABLE OF CONTENT IEC 61131-5

Annex A	
A.1	(normative) Mapping to ISO/IEC 9506-5
A.2	Application specific functions
A.3	PC object mapping
A.4	Communication function block mapping to MMS objects and services
A.5	Implementation specific features and parameters
Annex B	(normative) PC behavior using ISO/IEC 9506-2
B.1	PC communications mapping to MMS
B.2	Implementation specific features and parameters
Annex C	(informative)
Index	

IEC 61131-6 reserve for future use

IEC 61131-7 Fuzzy Control Programming

Detail not available

IEC 61131-8 Guidelines for the application and implementation of programming languages

TABLE OF CONTENT IEC 61131-8

1.	General
1.1	Scope
1.2	Normative references
1.3	Overview
2.	Introduction to IEC 61131-3
2.1	General considerations
2.2	Overcoming historical limitations
2.3	Basic features in IEC 61131-3
2.4	New features in the second edition of IEC 61131-3
2.5	Software engineering considerations
3.	Application guidelines
3.1	Use of data types
3.2	Data passing
3.3	Use of function blocks
3.4	Differences between function block instances and functions
3.5	Use of indirectly referenced function block instances
3.6	Recursion within programmable controller programming languages
3.7	Single and multiple invocation
3.8	Language specific features
3.9	Use of SFC elements
3.10	Scheduling, concurrency, and synchronization mechanisms
3.11	Communication facilities in ISO/IEC 9506/5 and IEC 61131-5

3.12	Recommended programming practices
4.	Implementation guidelines
4.1	Resource allocation
4.2	Implementation of data types
4.3	Execution of functions and function blocks
4.4	Implementation of Sequential Function Charts (SFCs)
4.5	Task scheduling
4.6	Error handling
4.7	System interface
4.8	Compliance
4.9	Compatibility with IEC 617-12, 617-13, and 848
5.	Programming support environment (PSE) requirements
5.1	User interface
5.2	Programming of programs, functions and function blocks
5.3	Application design and configuration
5.4	Separate compilation
5.5	Separation of interface and body
5.6	Linking of configuration elements with programs
5.7	Library management
5.8	Analysis tools
5.8.1	Simulation and debugging
5.8.2	Performance estimation
5.8.3	Feedback loop analysis
5.8.4	SFC analysis
5.9	Documentation requirements
5.10	Security of data and programs
5.11	On-line facilities
Annex A	Changes to IEC 61131-3 2nd Edition
A.1	Reasons for the 2nd edition of part 3
A.2	Corrigendum
A .3	Amendment
A.3.1	Numeric literals (2.2.1) – typed literals
A.3.2	Elementary data types (2.3.1) - double-byte strings
A.3.3	Derived data types (2.3.3) - enumerated data types
A.3.4	Single element variables (2.4.1.1) - 'wild-card' direct addresses
A.3.5	Declaration (2.4.3) - Temporary variables
A.3.6	Type assignment (2.4.3.1) - RETAIN and NON_RETAIN Variable attributes
A.3.7	Function (2.5.1) – Use EN/ENO
A.3.8	Declaration (2.5.1.3) - Function invocation with VAR_IN_OUT
A.3.9	Type conversion functions (2.5.1.5.1)
A.3.10	Functions of time data types (2.5.1.5.6)
A.3.11	Function blocks (2.5.2) - Extended initialisation facilities
A.3.12	Pulse action qualifiers (2.6.4.4)
A.3.13	Action control (2.6.4.5)
A.3.14	Configuration initialisation (2.7.1)
A.3.15	Instruction List (3.2)
A.3.16	Formal specification of language elements (Annex B)
A.3.17	Further amendments
ANNEX B	SOFTWARE QUALITY MEASURES
ANNEX C	INDEX

ضمیمه ۲

مقادیرمعادل ورودی و خروجی های آنالوگ

Digitized for Analog Input and Output Values

مفهوم Resolution در کارتهای آنالوگ

کارت آنالوگ ممکن است دارای یکی از Resolution های مندرج در جدول زیر باشد که این بستگی به نوع کارت آنالوگ دارد.

Resolution in Bits (+ Sign)	Units		Analog Value	
	Decimal	Hex	High-Order Byte	Low-Order Byte
8	128	80	VZ 0 0 0 0 0 0 0	1 x x x x x x x
9	64	40	VZ 0 0 0 0 0 0 0	0 1 x x x x x x
10	32	20	VZ 0 0 0 0 0 0 0	0 0 1 x x x x x
11	16	10	VZ 0 0 0 0 0 0 0	0 0 0 1 x x x x
12	8	8	VZ 0 0 0 0 0 0 0	0 0 0 0 1 x x x
13	4	4	VZ 0 0 0 0 0 0 0	0 0 0 0 0 1 x x
14	2	2	VZ 0 0 0 0 0 0 0	0 0 0 0 0 0 1 x
15	1	1	VZ 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1

بیت VZ برای علامت (0 برای مثبت و 1 برای منفی) رزرو شده است و بیت هایی که با X نشان داده شده به معنای صفر است. با در نظر داشتن این موضوع میبینیم که :

- در Resolution ۱۵ بیتی پله ها یکی یکی افزایش پیدا میکنند بنابراین بیشترین دقت را داریم.
- در Resolution ۱٤ بیتی پله ها دوتا دوتا افزایش پیدا میکنند چون بیت اول همیشه صفر است بنابراین دقت کمتری نسبت به ۱۵ بیتی دارد.
- بهمین ترتیب اگر مقادیر را دنبال کنیم به Resolution ۸ بیتی میرسیم که پله هایش ۱۲۸ تایی است چون هفت بیت اول همیشه صفر است . بنابراین کمترین دقت را خواهیم داشت.

معادل دسیمال و هگز مقادیر آنالوگ بسته به نوع ورودی دارد که در جداول زیر آورده شده اند.

۱- مقادیر معادل ورودیهای آنالوگ

مقادیر معادل سیگنالهای ولتاژ و جریان

Voltage Ranges							
Measuring Range ± 80 mV	Measuring Range ±250 mV	Measuring Range ±500 mV	Measuring Range ±1 V	Measuring Range ±2.5 V	Units		Range
					Decimal	Hex	
> 94.071	> 293.97	> 587.94	> 1.175	> 2.9397	32767	^{7FFF}H	Overflow
94.071 : 80.003	293.97 : 250.01	587.94 : 500.02	1.175 : 1.00004	2.9397 : 2.5001	32511 : 27649	^{7EFF}H : ^{6C01}H	Overrange
80.000 60.000 : – 60.000 – 80.000	250.00 187.50 : – 187.50 – 250.00	500.00 375.00 : – 375.00 – 500.00	1.000 0.750 : – 0.750 –1.000	2.500 1.875 : – 1.875 – 2.500	27648 20736 : –20736 –27648	^{6C00}H ^{5100}H : ^{AF00}H ^{9400}H	Nominal range
– 80.003 : – 94.74	– 250.01 : – 293.98	– 500.02 : – 587.96	– 1.00004 : – 1.175	– 2.5001 : – 2.93398	–27649 : –32512	^{93FF}H : ^{8100}H	Underrange
≤ 94.074	≤ 293.98	≤ 587.96	≤ 1.175	≤ 2.93398	–32768	^{8000}H	Underflow

ادامه مقادیر معادل سیگنالهای ولتاژ و جریان

Voltage and Current Measuring Ranges						
Measuring Range ± 5 V	Measuring Range ± 10 V و ± 10 mA	Measuring Range ± 3.2 mA	Measuring Range ± 20 mA	Units		Range
				Decimal	Hex	
> 5.8794	> 11.7589	> 3.7628	> 23.515	32767	7FFFH	Overflow
5.8794 : 5.0002	11.7589 : 10.0004	3.7628 : 3.2001	23.515 : 20.0007	32511 : 27649	7EFFH : 6C01H	Overrange
5.00 3.75 : – 3.75 – 5.00	10.00 7.50 : – 7.50 – 10.00	3.200 2.400 : – 2.400 – 3.200	20.000 14.998 : – 14.998 – 20.000	27648 20736 : –20736 –27648	6C00H 5100H : AF00H 9400H	Nominal range
– 5.0002 : – 5.8796	– 10.0004 : – 11.759	– 3.2001 : – 3.7629	– 20.0007 : – 23.516	–27649 : –32512	93FFH : 8100H	Underrange
≤ 5.8796	≤ 11.759	≤ 3.7629	≤ 23.516	–32768	8000H	Underflow

Voltage and Current Measuring Ranges					
Measuring Range 1 to 5 V	Measuring Range 0 20 mA	Measuring Range 4 to 20 mA	Units		Range
			Decimal	Hex	
> 5.7036	> 23.515	> 22.810	32767	7FFFH	Overflow
5.7036 : 5.0001	23.515 : 20.0007	22.810 : 20.0005	32511 : 27649	7EFFH : 6C01H	Overrange
5.000 4.000 : 1.000	20.000 14.998 : 0.000	20.000 16.000 : 4.000	27648 20736 : 0	6C00H 5100H : 0H	Nominal range
0.9999 : 0.2963	-0.0007 : –3.5185	3.9995 : 1.1852	–1 : –4864	FFFFH : ED00H	Underrange
< 0.2963	≤ 3.5185	< 1.1852	–32768	8000H	Underflow

مقادیر معادل سیگنالهای سنسورهای مقاومتی

Resistance-Type Sensors					
Measuring Range 150 Ω	Measuring Range 300 Ω	Measuring Range 600 Ω	Units		Range
			Decimal	Hex	
> 176.383	> 352.767	> 705.534	32767	7FFFH	Overflow
176.383 : 150.005	352.767 : 300.011	705.534 : 600.022	32511 : 27649	7EFFH : 6C01H	Overrange
150.000 112.500 : 0.000	300.000 225.000 : 0.000	600.000 450.000 : 0.000	27648 20736 : 0	6C00H 5100H : 0H	Nominal range
(negative values physically not possible)			–1 : –4864	FFFFH : ED00H	Underrange
-	-	-	–32768	8000H	Underflow

Standard Temperature Range, Pt 100			
Standard Temperature Range Pt 100 850 °C	Units		Range
	Decimal	Hex	
> 1000.0	32767	7FFFH	Overflow
1000.0 : 850.1	10000 : 8501	2710H : 2135H	Overrange
850.0 : –200.0	8500 : –2000	2134H : F830H	Nominal range
–200.1 : –243.0	–2001 : –2430	F82FH : F682H	Underrange
≤ 243.0	–32768	8000H	Underflow

Climate Temperature Range, Pt 100			
Standard Temperature Range Pt 100 130 °C	Units		Range
	Decimal	Hex	
> 155.00	32767	7FFFH	Overflow
155.00 : 130.01	15500 : 13001	3C8CH : 32C9H	Overrange
130.00 : –120.00	13000 : –12000	32C8H : D120H	Nominal range
–120.01 : –145.00	–12001 : –14500	D11FH : C75CH	Underrange
≤ 145.00	–32768	8000H	Underflow

ادامه مقادیر معادل سیگنالهای سنسورهای مقاومتی

Standard Temperature Range, Ni 100			
Standard Temperature Range Ni 100 250 ºC	Units		Range
	Decimal	Hex	
>295.0	32767	7FFFH	Overflow
295.0 : 250.1	2950 : 2501	B86H : 9C5H	Overrange
250.0 : –60.0	2500 : –600	9C4H : FDA8H	Nominal range
–60.1 : –105.0	–601 : –1050	FDA7H : FBE6H	Underrange
≤ 105,0	–32768	8000H	Underflow

Climate Temperature Range, Ni 100			
Standard Temperature Range Ni 100 250 ºC	Units		Range
	Decimal	Hex	
>295.00	32767	7FFFH	Overflow
295.00 : 250.01	29500 : 25001	733CH : 61A9H	Overrange
250.00 : –60.00	25000 : –6000	61A8H : E890H	Nominal range
–60.01 :	–6001 :	E88FH :	Underrange
–105.00	–10500	D6FCH	
≤ 105.00	–32768	8000H	Underflow

ترموکوپل ها

Thermocouple Type K			
Temperature Range in ºC Type K	Units		Range
	Decimal	Hex	
>1622	32767	7FFFH	Overflow
1622	16220	3FSCH	
:	:	:	Overrange
1373	13730	35A2H	
1372	13720	3598H	
:	:	:	Nominal range
–270	–2700	F574H	
≤ 270	≤ 2700	<F574H	Underrange

In the case of incorrect wiring (e. g. polarity reversal or open inputs) or of a sensor error in the negative range (e. g. incorrect thermocouple type), the analog input module signals underflow below F0C5H and Outputs 8000H

ادامه ترموکوپل ها

Thermocouple Type N			
Temperature Range in °C Type N	Units		Range
	Decimal	Hexadecimal	
>1550	32767	7C8CH	Overflow
1550 : 1301	15500 : 13010	3C8CH : 32D2H	Overrange
1300 : –270	13000 : –2700	32C8H : F574H	Nominal range
≤ 270	≤ 2700	<F574H	Underrange

Thermocouple Type J			
Temperature Range in°C Type J	Units		Range
	Decimal	Hexadecimal	
>1450	32767	7FFFH	Overflow
1450	14500	38A4H	
:	:	:	Overrange
1201	12010	2EEAH	
1200	12000	2EE0H	
:	:	:	Nominal range
–210.0	–2100	F7CCH	
≤ 210	≤ 2100	<F7CCH	Underrange

Thermocouple Type E			
Temperature Range in°C Type E	Units		Range
	Decimal	Hexadecimal	
>1201	32767	7FFFH	Overflow
1200	12000	2EE0H	
:	:	:	Overrange
1001	10010	271AH	
1000	10000	2710H	
:	:	:	Nominal range
–270	–2700	F574H	
≤ 271	≤ 2700	<F574H	Underrange

Thermocouple Type L			
Temperature Range in°C Type L	Units		Range
	Decimal	Hexadecimal	
>1150	**32767**	7FFFH	**Overflow**
1150	**11500**	2CECH	
:	**:**	:	**Overrange**
901	**9010**	2332H	
900	**9000**	2328H	
:	**:**	:	**Nominal range**
–200	**–2000**	F830H	
≤ 200	**≤ 2000**	<F830H	**Underrange**

In the case of incorrect wiring (e. g. polarity reversal or open inputs) or of a sensor error in the negative range (e. g. incorrect thermocouple type), the analog input module signals underflow below F0C5H and Outputs 8000H

۲- مقادیر معادل خروجی های آنالوگ

Voltage Output Ranges					
Output Range 0 to 10 V	Output Range 1 to 5 V	Output Range ±10 V	Units		Range
			Decimal	Hex	
0	0	0	>32511	>7EFFH	Overflow
11.7589 : 10.0004	5.8794 : 5.0002	11.7589 : 10.0004	32511 : 27649	7EFFH : 6C01H	Overrange
10.0000 : 0	5.0000 : 1.0000	10.0000 : 0 : – 10.0000	27648 : 0 : – 6912 – 6913 : – 27648	6C00H : 0H : E500H E4FFH : 9400H	Nomial range
0	:0.9999 0	10.0004 : – 11.7589	– 27649 : – 32512	93FFH : 8100H	Underrange
		0	≤ 32512	<8100H	Underflow

Current Output Ranges					
Output Range 0 to 20 mA	Output Range 4 to 20 mA	Output Range ±20 mA	Units		Range
			Decimal	Hex	
0	0	0	>32511	>7EFFH	Overflow
23.515 : 20.0007	22.81 : 20.005	23.515 : 20.0007	32511 : 27649	7EFFH : 6C01H	Overrange
20.000 : 0	20.000 : 4.000	20.000 0 : – 20.000	27648 : 0 : – 6912 – 6913 : – 27648	6C00H : 0H : E500H E4FFH : 9400H	Nominal range
0	3.9995 0	: – 23.515	– 27649 : – 32512	93FFH : 8100H	Underrange
	0	0	≤ 32512	<8100H	Underflow

ضمیمه ۳

لیست دستورات STL
در S7-300 و S7-400

English Mnemonics	Program Elements Catalog	Description
+	Integer math Instruction	Add Integer Constant (16, 32-Bit)
=	Bit logic Instruction	Assign
)	Bit logic Instruction	Nesting Closed
+AR1	Accumulator	AR1 Add ACCU 1 to Address Register 1
+AR2	Accumulator	AR2 Add ACCU 1 to Address Register 2
+D	Integer math Instruction	Add ACCU 1 and ACCU 2 as Double Integer (32-Bit)
-D	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Double Integer (32-Bit)
*D	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Double Integer(32-Bit)
/D	Integer math Instruction	Compare Double Integer (32-Bit) ==, <>, >, <, >=, <=
?D	Compare	Add ACCU 1 and ACCU 2 as Integer (16-Bit)
+I	Integer math Instruction	Subtract ACCU 1 from ACCU 2 as Integer (16-Bit)
-I	Integer math Instruction	Multiply ACCU 1 and ACCU 2 as Integer (16-Bit)
*I	Integer math Instruction	Divide ACCU 2 by ACCU 1 as Integer (16-Bit)
/I	Integer math Instruction	Compare Integer (16-Bit) ==, <>, >, <, >=, <=
?I	Compare	Add ACCU 1 and ACCU 2 as a Floating-Point Number (32-Bit IEEE-FP)
+R	Floating point Instruction	Subtract ACCU 1 from ACCU 2 as a Floating-Point Number (32-Bit IEEE-FP)
-R	Floating point Instruction	Multiply ACCU 1 and ACCU 2 as Floating-Point Number (32-Bit IEEE-FP)
*R	Floating point Instruction	Divide ACCU 2 by ACCU 1 as a Floating-Point Number (32-Bit IEEE-FP)
/R	Floating point Instruction	Number (32-Bit IEEE-FP)
?R	Compare	Compare Floating-Point Number (32-Bit) ==, <>, >, <, >=, <=
A	Bit logic Instruction	And
A(	Bit logic Instruction	And with Nesting Open
ABS	Floating point Instruction	Absolute Value of a Floating-Point Number (32-Bit IEEE-FP)
ACOS	Floating point Instruction	Generate the Arc Cosine of a Floating-Point Number (32-Bit)
AD	Word logic Instruction	AND Double Word (32-Bit)
AN	Bit logic Instruction	And Not
AN(	Bit logic Instruction	And Not with Nesting Open
ASIN	Floating point Instruction	Generate the Arc Sine of a Floating-Point Number (32-Bit)
ATAN	Floating point Instruction	Generate the Arc Tangent of a Floating-Point Number (32-Bit)
AW	Word logic Instruction	AND Word (16-Bit)
BE	Program control	Block End
BEC	Program control	Block End Conditional
BEU	Program control	Block End Unconditional
BLD	Program control	Program Display Instruction (Null)
BTD	Convert	BCD to Integer (32-Bit)
BTI	Convert	BCD to Integer (16-Bit)
CAD	Convert	Change Byte Sequence in ACCU 1 (32-Bit)
CALL	Program control	Block Call
CALL	Program control	Call Multiple Instance

English Mnemonics	Program Elements Catalog	Description
CALL	Program control	Call Block from a Library
CAR	Load/Transfer	Exchange Address Register 1 with Address Register 2
CAW	Convert	Change Byte Sequence in ACCU 1-L (16-Bit)
CC	Program control	Conditional Call
CD	Counters	Counter Down
CDB	Convert	Exchange Shared DB and Instance DB
CLR	Bit logic Instruction	Clear RLO (=0)
COS	Floating point Instruction	Generate the Cosine of Angles as Floating-Point Numbers (32-Bit)
CU	Counters	Counter Up
DEC	Accumulator	Decrement ACCU 1-L-L
DTB	Convert	Double Integer (32-Bit) to BCD
DTR	Convert	Convert Double Integer (32-Bit) to Floating-Point (32-Bit IEEE-FP)
ENT	Accumulator	Enter ACCU Stack
EXP	Floating point Instruction	Generate the Exponential Value of a Floating-Point Number (32-Bit)
FN	Bit logic Instruction	Edge Negative
FP	Bit logic Instruction	Edge Positive
FR	Counters	Enable Counter (Free) (free, FR C 0 to C 255)
FR	Timers	Enable Timer (Free)
INC	Accumulator	Increment ACCU 1-L-L
INVD	Convert	Ones Complement Double Integer (32-Bit)
INVI	Convert	Ones Complement Integer (16-Bit)
ITB	Convert	Integer (16-Bit) to BCD
ITD	Convert	Integer (16-Bit) to Double Integer (32-Bit)
JBI	Jumps	Jump if BR = 1
JC	Jumps	Jump if RLO = 1
JCB	Jumps	JCB SPBB Jumps Jump if RLO = 1 with BR
JCN	Jumps	Jump if RLO = 0
JL	Jumps	Jumps Jump to Labels
JM	Jumps	Jump if Minus
JMZ	Jumps	Jump if Minus or Zero
JN	Jumps	Jump if Not Zero
JNB	Jumps	Jump if RLO = 0 with BR
JNBI	Jumps	Jump if BR = 0
JO	Jumps	Jump if OV = 1
JOS	Jumps	Jump if OS = 1
JP	Jumps	Jump if Plus
JPZ	Jumps	Jump if Plus or Zero
JU	Jumps	Jump Unconditional
JUO	Jumps	Jump if Unordered
JZ	Jumps	Jump if Zero
L	Load/Transfer	Load
L DBLG	Load/Transfer	Load Length of Shared DB in ACCU 1
L DBNO	Load/Transfer	Load Number of Shared DB in ACCU 1

English Mnemonics	Program Elements Catalog	Description
L DILG	Load/Transfer	Load Length of Instance DB in ACCU 1
L DINO	Load/Transfer	Load Number of Instance DB in ACCU 1
L STW	Load/Transfer	Load Status Word into ACCU 1
L	Timers	Load Current Timer Value into ACCU 1 as Integer (the current timer value can be a number from 0 to 255, for example, L T 32)
L	Counters	Load Current Counter Value into ACCU 1 (the current counter value can be a number from 0 to 255, for example, L C 15)
LAR1	Load/Transfer	Load Address Register 1 from ACCU 1
LAR1 <D>	Load/Transfer	Load Address Register 1 with Double Integer (32-Bit Pointer)
LAR1 AR2	Load/Transfer	Load Address Register 1 from Address Register 2
LAR2	Load/Transfer	Load Address Register 2 from ACCU 1
LAR2 <D>	Load/Transfer	Load Address Register 2 with Double Integer (32-Bit Pointer)
LC	Counters	Load Current Counter Value into ACCU 1 as BCD (the current timer value can be a number from 0 to 255, for example, LC C 15)
LC	Timers	Load Current Timer Value into ACCU 1 as BCD (the current counter alue can be a number from 0 to 255, for example, LC T 32)
LEAVE	Accumulator	Leave ACCU Stack
LN	Floating point Instruction	Generate the Natural Logarithm of a Floating-Point Number (32-Bit)
LOOP	Jumps	Loop
MCR(	Program control	Save RLO in MCR Stack, Begin MCR
)MCR	Program control	End MCR
MCRA	Program control	Activate MCR Area
MCRD	Program control	Deactivate MCR Area
MOD	Integer math Instruction	Division Remainder Double Integer (32-Bit)
NEGD	Convert	Twos Complement Double Integer (32-Bit)
NEGI	Convert	Twos Complement Integer (16-Bit)
NEGR	Convert	Negate Floating-Point Number (32-Bit, IEEE-FP)
NOP 0	Accumulator	Null Instruction
NOP 1	Accumulator	Null Instruction
NOT	Bit logic Instruction	Negate RLO
O	Bit logic Instruction	Or
O(	Bit logic Instruction	Or with Nesting Open
OD	Bit logic Instruction	OR Double Word (32-Bit)
ON	Bit logic Instruction	Or Not
ON(	Bit logic Instruction	Or Not with Nesting Open
OPN	DB call	Open a Data Block
OW	Word logic Instruction	OR Word (16-Bit)
POP	Accumulator	CPU with Two ACCUs
POP	Accumulator	CPU with Four ACCUs
POP	Accumulator	CPU with Two ACCUs
PUSH	Accumulator	CPU with Four ACCUs
R	Bit logic Instruction	Reset
R	Counters	Reset Counter (the current counter can be a number from 0 to 255, for example, R C 15)
R	Timers	Reset Timer (the current timer can be a number from 0 to 255, for example, R T 32)

English Mnemonics	Program Elements Catalog	Description
RLD	Shift/Rotate	Rotate Left Double Word (32-Bit)
RLDA	Shift/Rotate	Rotate ACCU 1 Left via CC 1 (32-Bit)
RND	Convert	Round
RND-	Convert	Round to Lower Double Integer
RND+	Convert	Round to Upper Double Integer
RRD	Shift/Rotate	Rotate Right Double Word (32-Bit)
RRDA	Shift/Rotate	Rotate ACCU 1 Right via CC 1 (32-Bit)
S	Bit logic Instruction	Set
S	Counters	Set Counter Preset Value (the current counter can be a number from 0 to 255, for example, S C 15)
SAVE	Bit logic Instruction	Save RLO in BR Register
SD	Timers	On-Delay Timer
SE	Timers	Extended Pulse Timer
SET	Bit logic Instruction	Set
SF	Timers	Off-Delay Timer
SIN	Floating point Instruction	Generate the Sine of Angles as Floating-Point Numbers (32-Bit)
SLD	Shift/Rotate	Shift Left Double Word (32-Bit)
SLW	Shift/Rotate	Shift Left Word (16-Bit)
SP	Timers	Pulse Timer
SQR	Floating point Instruction	Generate the Square of a Floating-Point Number (32-Bit)
SQRT	Floating point Instruction	Generate the Square Root of a Floating-Point Number (32-Bit)
SRD	Shift/Rotate	Shift Right Double Word (32-Bit)
SRW	Shift/Rotate	Shift Right Word (16-Bit)
SS	Timers	Retentive On-Delay Timer
SSD	Shift/Rotate	Shift Sign Double Integer (32-Bit)
SSI	Shift/Rotate	Shift Sign Integer (16-Bit)
T	Load/Transfer	Transfer
T	Load/Transfer	Transfer ACCU 1 into Status Word
TAK	Accumulator	Toggle ACCU 1 with ACCU 2
TAN	Floating point Instruction	Generate the Tangent of Angles as Floating-Point Numbers (32-Bit)
TAR1	Load/Transfer	Transfer Address Register 1 to ACCU 1
TAR1	Load/Transfer	Transfer Address Register 1 to Destination (32-Bit Pointer)
TAR1	Load/Transfer	Transfer Address Register 1 to Address Register 2
TAR2	Load/Transfer	Transfer Address Register 2 to ACCU 1
TAR2	Load/Transfer	Transfer Address Register 2 to Destination (32-Bit Pointer)
TRUNC	Convert	Truncate
UC	Program control	Unconditional Call
X	Bit logic Instruction	Exclusive Or
X(	Bit logic Instruction	Exclusive Or with Nesting Open
XN	Bit logic Instruction	Exclusive Or Not
XN(	Bit logic Instruction	Exclusive Or Not with Nesting Open
XOD	Word logic Instruction	Exclusive OR Double Word (32-Bit)
XOW	Word logic Instruction	Exclusive OR Word (16-Bit)

ضمیمه ۴

زمان اجرای دستورات و فانکشن های S7-400

۱- زمان اجرای دستورات

۲- زمان اجرای SFC ها

۳- زمان اجرای SFB ها

زمان اجرای دستورات و فانکشن های S7-400

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Bit Logic Instructions							
A/AN		AND/AND NOT					
	I/Q a.b	Input/output	1*/2	0.2/0.3	0.1	0.08	0.1
	M a.b	Bit memory	1**/2	0.2/0.3	0.1	0.08	0.1
	L a.b	Local data bit	2	0.3	0.1	0.08	0.1
	DBX a.b	Data bit	2	0.3	0.1	0.08	0.1
	DIX a.b	Instance data bit	2	0.3	0.1	0.08	0.1
	c [d]	Memory-indirect, area-internal	2	0.3+	0.1+	0.08+	0.1+
	c [AR1,m]	Register-ind., area-internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	c [AR2,m]	Register-ind., area-internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
O/ON		OR/OR NOT					
	I/Q a.b	Input/output	1*/2	0.2/0.3	0.1	0.08	0.1
	M a.b	Bit memory	1**/2	0.2/0.3	0.1	0.08	0.1
	L a.b	Local data bit	2	0.3	0.1	0.08	0.1
	DBX a.b	Data bit	2	0.3	0.1	0.08	0.1
	DIX a.b	Instance data bit	2	0.3	0.1	0.08	0.1
	c [d]	Memory-indirect, area-internal	2	0.3+	0.1+	0.08+	0.1+
	c [AR1,m]	Register-ind., area-internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	c [AR2,m]	Register-ind., area-internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
X/XN		Exclusive OR/ Exclusive OR NOT					
	I/Q a.b	Input/output	2	0.3	0.1	0.08	0.1
	M a.b	Bit memory	2	0.3	0.1	0.08	0.1
	L a.b	Local data bit	2	0.3	0.1	0.08	0.1
	DBX a.b	Data bit	2	0.3	0.1	0.08	0.1
	DIX a.b	Instance data bit	2	0.3	0.1	0.08	0.1
	c [d]	Memory-indirect, area-internal	2	0.3+	0.1+	0.08+	0.1+
	c [AR1,m]	Register-ind., area-internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	c [AR2,m]	Register-ind., area-internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
Bit Logic Instructions with Parenthetical Expressions							
A(		AND left parenthesis	1	0.1	0.1	0.08	0.1
AN(		AND NOT left parenthesis	1	0.1	0.1	0.08	0.1
O(		OR left parenthesis	1	0.1	0.1	0.08	0.1
ON(		OR NOT left parenthesis	1	0.1	0.1	0.08	0.1
X(		Exclusive OR left parenthesis	1	0.1	0.1	0.08	0.1
XN(		Exclusive OR NOT left parenthesis	1	0.1	0.1	0.08	0.1
)		Right parenthesis, removing an entry from the nesting stack.	1	0.1	0.1	0.08	0.1

\+ Plus time required for loading the address of the instruction (see page 20)
* With direct instruction addressing; Address area 0 to 127
** With direct instruction addressing; Address area 0 to 255

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
ORing of AND Instructions							
O		ORing of AND operations according to the rule: AND before OR	1	0.1	0.1	0.08	0.1
Logic Instructions with Timers and Counters							
A/AN		AND/AND NOT					
	T f	Timer	1[1)]/2	0,3	0.1	0.08	0.1
	T [e]	Timer, memory-indirect addressing	2	0.3+	0.1+	0.08+	0.1+
	C f	Counter	1[1)]/2	0,3	0.1	0.08	0.1
	C [e]	Counter, memory-indirect addressing	2	0.3+	0.1+	0.08+	0.1+
	Timer para.	Timer/counter (addressing via parameter)	2	0.3+	0.1+	0.08+	0.1+
	Counter para.			0.3+	0.1+	0.08+	0.1+
O/ON		OR/OR NOT					
	T f	Timer	1[1)]/2	0.3	0.1	0.08	0.1
	T [e]	Timer, memory-indirect addr.	2	0.3+	0.1+	0.08+	0.1+
	C f	Counter	1[1)]/2	0.3	0.1	0.08	0.1
	C [e]	Counter, memory-indirect addressing	2	0.3+	0.1+	0.08+	0.1+
	Timer para.	Timer/counter (addressing via parameter)	2	0.3+	0.1+	0.08+	0.1+
	Counter para.			0.3+	0.1+	0.08+	0.1+
X/XN		EXCLUSIVE OR/EXCLUSIVE OR NOT					
	T f	Timer	2	0.3	0.1	0.08	0.1
	T [e]	Timer, memory-indirect addr.	2	0.3+	0.1+	0.08+	0.1+
	C f	Counter	2	0.3	0.1	0.08	0.1
	C [e]	Counter, mem.-indirect addr.	2	0.3+	0.1+	0.08+	0.1+
	Timer para.	EXCLUSIVE OR timer/counter (addressing	2	0.3+	0.1+	0.08+	0.1+
	Counter para.	via parameter)		0.3+	0.1+	0.08+	0.1+
Word Logic Instructions with the Contents of Accumulator 1							
AW		AND ACCU2-L	1	0.1	0.1	0.08	0.1
AW	W#16#p	AND 16-bit constant	2	0.2	0.1	0.08	0.1
OW		OR ACCU2-L	1	0.1	0.1	0.08	0.1
OW	W#16#p	OR 16-bit constant	2	0.2	0.1	0.08	0.1
XOW		EXCLUSIVE OR ACCU2-L	1	0.1	0.1	0.08	0.1
XOW	W#16#p	EXCLUSIVE OR 16-bit constant	2	0.2	0.1	0.08	0.1
AD		AND ACCU2	1	0.1	0.1	0.08	0.1
AD	DW#16#p	AND 32-bit constant	3	0.3	0.15	0.12	0.15
OD		OR ACCU2	1	0.1	0.1	0.08	0.1
OD	DW#16#p	OR 32-bit constant	3	0.3	0.15	0.12	0.15
XOD		EXCLUSIVE OR ACCU2	1	0.1	0.1	0.08	0.1
XOD	DW#16#p	EXCLUSIVE OR 32-bit constant	3	0.3	0.15	0.12	0.15

\+ Plus time required for loading the address of the instruction (see page 20)

1) With direct instruction addressing ;Address area 0 to 255

Instr.	Address ID		Description	Length in Words	Execution Time in µs			
					CPU 412	CPU 414	CPU 416	CPU 417
Evaluating Conditions Using AND, OR and EXCLUSIVE OR								
A/AN O/ON X/XN			AND/AND NOT OR/OR-NOT EXCLUSIVE OR/ EXCLUSIVE-OR-NOT					
	==0		Result=0 (A1=0 and A0=0)	1	0.1	0.1	0.08	0.1
	>0		Result>0 (CC1=1 and CC0=0)	1	0.1	0.1	0.08	0.1
	<0		Result<0 (CC1=0 and CC0=1)	1	0.1	0.1	0.08	0.1
	<>0		Result<>0 ((CC1=0 and CC0=1) or (CC1=1 and CC0=0))	1	0.1	0.1	0.08	0.1
	>=0		Result>=0 ((CC1=1 and CC0=0) or (CC1=0 and CC0=0))	1	0.1	0.1	0.08	0.1
	<=0		Result<=0 ((CC1=0 and CC0=1) or (CC1=0 and CC0=0))	1	0.1	0.1	0.08	0.1
	UO		Unordered math instruction (CC1=1 and CC0=1)	1	0.1	0.1	0.08	0.1
	OS		AND OS=1	1	0.1	0.1	0.08	0.1
	BR		AND BR=1	1	0.1	0.1	0.08	0.1
	OV		AND OV=1	1	0.1	0.1	0.08	0.1
Edge-Triggered Instructions								
FP/FN	I/Q	a.b	The positive/negative edge is	2	0.4	0.2	0.16	0.2
	M	a.b	indicated by RLO = 1. The bit	2	0.4	0.2	0.16	0.2
	L	a.b[1)]	addressed in the instruction is	2	0.4	0.2	0.16	0.2
	DBX	a.b	the auxiliary edge bit	2	0.4	0.2	0.16	0.2
			memory.					
	DIX	a.b		2	0.4	0.2	0.16	0.2
	c [d]			2	0.4+	0.2+	0.16+	0.2+
	c [AR1,m]			2	0.4+	0.2+	0.16+	0.2+
	c [AR2,m]			2	0.4+	0.2+	0.16+	0.2+
	[AR1,m]			2	0.4+	0.2+	0.16+	0.2+
	[AR2,m]			2	0.4+	0.2+	0.16+	0.2+
	Parameter			2	0.4+	0.2+	0.16+	0.2+
Plus time required for loading the address of the instruction (see page 20) 1) Unnecessary if the bit being monitored is in the process image (local data of a block are only valid while the block is running).								
Setting/Resetting Bit Addresses								
S R			Set addressed bit to “1” Set addressed bit to “0”					
	I/Q	a.b	Input/output	1[1)]/2	0.3/0.4	0.2	0.16	0.2
	M	a.b	Bit memory	1[2)]/2	0.3/0.4	0.2	0.16	0.2
	L	a.b	Local data bit	2	0.4	0.2	0.16	0.2
	DBX	a.b	Data bit	2	0.4	0.2	0.16	0.2
	DIX	a.b	Instance data bit	2	0.4	0.2	0.16	0.2
	c [d]		Memory-indirect, area-internal	2	0.4+	0.2+	0.16+	0.2+
	c [AR1,m]		Register-indirect, area-internal (AR1)	2	0.4+	0.2+	0.16+	0.2+
	c [AR2,m]		Register-indirect, area-internal (AR2)	2	0.4+	0.2+	0.16+	0.2+
	[AR1,m]		Area-crossing (AR1)	2	0.4+	0.2+	0.16+	0.2+
	[AR2,m]		Area-crossing (AR2)	2	0.4+	0.2+	0.16+	0.2+
	Parameter		Via parameter	2	0.4+	0.2+	0.16+	0.2+

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Setting/Resetting Bit Addresses, continued							
=		Assign RLO					
	I/Q a.b	To input/output	1[1]/2	0.3/0.4	0.2	0.16	0.2
	M a.b	To bit memory	1[2]/2	0.3/0.4	0.2	0.16	0.2
	L a.b	To local data bit	2	0.4	0.2	0.16	0.2
	DBX a.b	To data bit	2	0.4	0.2	0.16	0.2
	DIX a.b	To instance data bit	2	0.4	0.2	0.16	0.2
	c [d]	Memory-indirect, area-internal	2	0.4+	0.2+	0.16+	0.2+
	c [AR1,m]	Register-indirect, area-internal (AR1)	2	0.4+	0.2+	0.16+	0.2+
	c [AR2,m]	Register-indirect, area-internal (AR2)	2	0.4+	0.2+	0.16+	0.2+
	[AR1,m]	Area-crossing (AR1)	2	0.4+	0.2+	0.16+	0.2+
	[AR2,m]	Area-crossing (AR2)	2	0.4+	0.2+	0.16+	0.2+
	Parameter	Via parameter	2	0.4+	0.2+	0.16+	0.2+
Plus time required for loading the address of the instruction (see page 20) With direct instruction addressing; Address area 0 to 127 With direct instruction addressing; Address area 0 to 255							
Instructions Directly Affecting the RLO							
CLR		Set RLO to “0”	1	0.1	0.1	0.08	0.1
SET		Set RLO to “1”	1	0.1	0.1	0.08	0.1
NOT		Negate RLO	1	0.1	0.1	0.08	0.1
SAVE		Save RLO to the BR bit	1	0.1	0.1	0.08	0.1
Timer Instructions							
SP	T f	Start timer as pulse on edge	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	change from “0” to “1”		0.3+0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
SE	T f	Start timer as extended pulse on	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	edge change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
SD	T f	Start timer as ON delay on edge	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
SS	T f	Start timer as retentive ON delay	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	on edge change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
SF	T f	Start timer as OFF delay on edge	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
FR	T f	Enable timer for restarting on	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]	edge change from “0” to “1” (reset edge bit memory for		0.4+	0.2+	0.16+	0.2+
	Timer para.	starting timer)	2	0.4+	0.2+	0.16+	0.2+
R	T f	Reset timer	1[1]/2	0.3/0.4	0.2	0.16	0.2
	T [e]			0.4+	0.2+	0.16+	0.2+
	Timer para.		2	0.4+	0.2+	0.16+	0.2+
Plus time required for loading the address of the instruction (see page 20) With direct instruction addressing Timer No.: 0 to 255							

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Counter Instructions							
S	C f	Presetting of counter on edge	1[0)]/2	0.3/0.4	0.2	0.16	0.2
	C [e]	change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Counter para.		2	0.4+	0.2+	0.16+	0.2+
R	C f	Reset counter to “0”	1[0)]/2	0.3/0.4	0.2	0.16	0.2
	C [e]	when RLO = “1”		0.4+	0.2+	0.16+	0.2+
	Counter para.		2	0.4+	0.2+	0.16+	0.2+
CU	C f	Increment counter by 1 on edge	1[0)]/2	0.3/0.4	0.2	0.16	0.2
	C [e]	change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Counter para.		2	0.4+	0.2+	0.16+	0.2+
CD	C f	Decrement counter by 1 on edge	1[0)]/2	0.3/0.4	0.2	0.16	0.2
	C [e]	change from “0” to “1”		0.4+	0.2+	0.16+	0.2+
	Counter para.		2	0.4+	0.2+	0.16+	0.2+
FR	C f	Enable counter on edge change	1[0)]/2	0.3/0.4	0.2	0.16	0.2
	C [e]	from “0” to “1” (reset edge bit memory for up and down		0.4+	0.2+	0.16+	0.2+
	Counter para.	counting and setting the counter)	2	0.4+	0.2+	0.16+	0.2+
Load Instructions							
L		Load ...					
	IB a	Input byte	1[1)]/2	0.2/0.3	0.1	0.08	0.1
	QB a	Output byte	1[1)]/2	0.2/0.3	0.1	0.08	0.1
	PIB a	Peripheral input byte [2)]	2	0.3	0.1	0.08	0.1
	MB a	Bit memory byte	1[3)]/2	0.2/0.3	0.1	0.08	0.1
	LB a	Local data byte	2	0.3	0.1	0.08	0.1
	DBB a	Data byte	2	0.3	0.1	0.08	0.1
	DIB a	Instance data byte	2	0.3	0.1	0.08	0.1
	g [d]	Memory-indirect, area-internal	2	0.3+	0.1+	0.08+	0.1+
	g [AR1,m]	Register-indirect, area-internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	g [AR2,m]	Register-indirect, area-internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	B[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	B[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
		Load ...					
	IW a	Input word	1[1)]/2	0.2/0.3	0.1	0.08	0.1
	QW	Output word	1[1)]/2	0.2/0.3	0.1	0.08	0.1
	PIW a	Peripheral input word [2)]	2	0.3	0.1	0.08	0.1
	MW a	Bit memory word	1[3)]/2	0.2/0.3	0.1	0.08	0.1
	LW a	Local data word	2	0.3	0.1	0.08	0.1
	DBW a	Data word	2	0.3	0.1	0.08	0.1
	DIW a	Instance data word	2	0.3	0.1	0.08	0.1
		... into ACCU1-L					
	h [d]	Memory-indirect, area-internal	2	0.3+	0.1+	0.08+	0.1+
	h [AR1,m]	Register-indirect, area-internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	h [AR2,m]	Register-indirect, area-internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	W[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	W[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+

\+ Plus time required for loading the address of the instruction (see page 20)

0) With direct instruction addressing Counter No.: 0 to 255

1) With indirect instruction addressing; Address area 0 to 127

2) The following peripheral acknowledgement time must be observed with CPU 414-4H and CPU 417-4H: solo 37 **µs**, redundant 67 **µs**

3) With direct instruction addressing; Address area 0 to 255

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Load Instructions, continued							
L		Load ...					
	IDa	Input double word	1[1]/2	0.3/0.4	0.2	0.16	0.2
	QD a	Output double word	1[1]/2	0.3/0.4	0.2	0.16	0.2
	PID a	Peripheral input double word [2]	2	0.3/0.4	0.2	0.16	0.2
	MD a	Bit memory double word	1[3]/2	0.3/0.4	0.2	0.16	0.2
	LD a	Local data double word	2	0.4	0.2	0.16	0.2
	DBD a	Data double word	2	0.4	0.2	0.16	0.2
	DID a	Instance data double word ... in ACCU1	2	0.4	0.2	0.16	0.2
	i [d]	Memory-indirect, area internal	2	0.4+	0.2+	0.16+	0.2+
	i [AR1,m]	Register-ind., area internal (AR1)	2	0.4+	0.2+	0.16+	0.2+
	i [AR2,m]	Register-ind., area internal (AR2)	2	0.4+	0.2+	0.16+	0.2+
	D[AR1,m]	Area-crossing (AR1)	2	0.4+	0.2+	0.16+	0.2+
	D[AR2,m]	Area-crossing (AR2)	2	0.4+	0.2+	0.16+	0.2+
	Parameter	Via parameter	2	0.4+	0.2+	0.16+	0.2+
		Load ...					
	k8	8-bit constant into ACCU1-LL	2	0.2	0.1	0.08	0.1
	k16	16-bit constant into ACCU1-L	2	0.2	0.1	0.08	0.1
	k32	32-bit constant into ACCU1	3	0.3	0.15	0.12	0.15
	Parameter	Load constant into ACCU1 (addressed via parameter)	2	0.2/0.3+	0.1+	0.08+	0.1+
	2#n	Load 16-bit binary constant into ACCU1-L	2	0.2	0.1	0.08	0.1
		Load 32-bit binary constant into ACCU1	3	0.3	0.15	0.12	0.15
	B#16#p	Load 8-bit-hexadecimal constant into ACCU1-L	1	0.1	0.1	0.08	0.1
	W#16#p	Load 16-bit hexadecimal constant into ACCU1-L	2	0.2	0.1	0.08	0.1
	DW#16#p	Load 32-bit hexadecimal	3	0.3	0.15	0.12	0.15
		Load ...					
	'x'	Load 1 character	2	0.2	0.1	0.08	0.1
	'xx'	Load 2 characters	2	0.2	0.1	0.08	0.1
	'xxx'	Load 3 characters	3	0.3	0.15	0.12	0.15
	'xxxx'	Load 4 characters	3	0.3	0.15	0.12	0.15
	D# time value	Load IEC date	3	0.3	0.15	0.12	0.15
	S5T# time value	Load S7 time constant (16 bits)	2	0.2	0.1	0.08	0.1
	TOD# time value	Load IEC time constant	3	0.3	0.15	0.12	0.15
	T# time value	Load 16-bit time constant	2	0.2	0.1	0.08	0.1
		Load 32-bit time constant	3	0.3	0.15	0.12	0.15
	C# count value	Load counter constant (BCD code)	2	0.2	0.1	0.08	0.1
	B# (b1, b2)	Load constant as byte (b1, b2)	2	0.2	0.1	0.08	0.1
	B# (b1, b2, b3, b4)	Load constant as 4 bytes (b1, b2, b3, b4)	3	0.3	0.15	0.12	0.15
	P# bit pointer	Load bit pointer	3	0.3	0.15	0.12	0.15
	L# integer	Load 32-bit integer constant	3	0.3	0.15	0.12	0.15
	Real number	Load floating-point number	3	0.3	0.15	0.12	0.15

\+ Plus time required for loading the address of the instruction (see page 20)

1) With indirect instruction addressing; Address area 0 to 127

2) The following peripheral acknowledgement time must be observed with CPU 414-4H and CPU 417-4H: solo 37 **µs**, redundant 67 **µs**

3) With direct instruction addressing; Address area 0 to 255

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Load Instructions for Timers and Counters							
L	T f	Load time value	1[1]/2	0.2/0.3	0.1	0.08	0.1
	T (e)		2	0.3+	0.1+	0.08+	0.1+
	Timer para.	Load time value (addressed via parameter)	2	0.3+	0.1+	0.08+	0.1+
	C f	Load count value	1[1]/2	0.2/0.3	0.1	0.08	0.1
	C (e)		2	0.3+	0.1+	0.08+	0.1+
	Counter para.	Load count value (addressed via parameter)	2	0.3+	0.1+	0.08+	0.1+
LC	T f	Load time value in BCD	1[1]/2	0.3	0.3	0.24	0.3
	T (e)		2	0.3+	0.3+	0.24+	0.3+
	Timer para.	Load time value in BCD (addressed via parameter)	2	0.3+	0.3+	0.24+	0.3+
	C f	Load count value in BCD	1[1]/2	0.3	0.3	0.24	0.3
	C (e)		2	0.3+	0.3+	0.24+	0.3+
	Counter para.	Load count value in BCD (addressed via parameter)	2	0.3+	0.3+	0.24+	0.3+
Plus time required for loading the address of the instruction (see page 20)							
1) With direct instruction addressing; Timer/counter No.: 0 to 255							
Transfer Instructions							
T		Transfer contents of ACCU1-LL to ...					
	IB a	input byte	1[1]/2	0.2/0.3	0.1	0.08	0.1
	QB a	output byte	1[1]/2	0.2/0.3	0.1	0.08	0.1
	PQB a	peripheral output byte[2]	2	0.3	0.1	0.08	0.1
	MB a	bit memory byte	1[3]/2	0.2/0.3	0.1	0.08	0.1
	LB a	local data byte	2	0.3	0.1	0.08	0.1
	DBB a	data byte	2	0.3	0.1	0.08	0.1
	DIB a	instance data byte	2	0.3	0.1	0.08	0.1
	g [d]	Memory-indirect, area internal	2	0.3+	0.1+	0.08+	0.1+
	g [AR1,m]	Register-ind., area internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	g [AR2,m]	Register-ind., area internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	B[AR1,m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	B[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
Plus time required for loading the address of the instruction (see page 20) With direct instruction addressing; Address area 0 to 127 The following peripheral acknowledgement time must be observed with CPU 414-4H and CPU 417-4H: solo 29 **µs**, redundant 58 **µs** 3) With direct instruction addressing; Address area 0 to 255							
T		Transfer contents of ACCU1-L to ...					
	IW a	input word	1[1]/2	0.2/0.3	0.1	0.08	0.1
	QW a	output word	1[1]/2	0.2/0.3	0.1	0.08	0.1
	PQW a	peripheral output word[2]	2	0.3	0.1	0.08	0.1
	MW a	bit memory word	1[3]/2	0.2/0.3	0.1	0.08	0.1
	LW a	local data word	2	0.3	0.1	0.08	0.1
	DBW a	data word	2	0.3	0.1	0.08	0.1
	DIW a	instance data word	2	0.3	0.1	0.08	0.1
	h [d]	Memory-indirect, area internal	2	0.3+	0.1+	0.08+	0.1+
	h [AR1.m]	Register-ind., area internal (AR1)	2	0.3+	0.1+	0.08+	0.1+
	h [AR2,m]	Register-ind., area internal (AR2)	2	0.3+	0.1+	0.08+	0.1+
	W[AR1.m]	Area-crossing (AR1)	2	0.3+	0.1+	0.08+	0.1+
	W[AR2,m]	Area-crossing (AR2)	2	0.3+	0.1+	0.08+	0.1+
	Parameter	Via parameter	2	0.3+	0.1+	0.08+	0.1+
Plus time required for loading the address of the instruction (see page 20) With direct instruction addressing; Address area 0 to 127 The following peripheral acknowledgement time must be observed with CPU 414-4H and CPU 417-4H: solo 32 **µs**, redundant 61 **µs** With direct instruction addressing; Address area 0 to 255							

Instr.	Address ID		Description	Length in Words	Execution Time in µs			
					CPU 412	CPU 414	CPU 416	CPU 417
Transfer Instructions, continued								
T			Transfer contents of ACCU1 to ...					
	ID	a	Input double word	1[1]/2	0.3/0.4	0.2	0.16	0.2
	QD	a	Output double word	1[1]/2	0.3/0.4	0.2	0.16	0.2
	PQD	a	periph. output double word [2]	2	0.4	0.2	0.16	0.2
	MD	a	Bit memory double word	1[3]/2	0.3/0.4	0.2	0.16	0.2
	LD	a	Local data double word	2	0.4	0.2	0.16	0.2
	DBD	a	Data double word	2	0.4	0.2	0.16	0.2
	DID	a	Instance data double word	2	0.4	0.2	0.16	0.2
T	i [d]		Memory-indirect, area internal	2	0.4+	0.2+	0.16+	0.2+
	i [AR1,m]		Register-ind., area internal (AR1)	2	0.4+	0.2+	0.16+	0.2+
	i [AR2,m]		Register-ind., area internal (AR2)	2	0.4+	0.2+	0.16+	0.2+
	D[AR1,m]		Area-crossing (AR1)	2	0.4+	0.2+	0.16+	0.2+
	D[AR2,m]		Area-crossing (AR2)	2	0.4+	0.2+	0.16+	0.2+
	Parameter		Via parameter	2	0.4+	0.2+	0.16+	0.2+
Plus time required for loading the address of the instruction (see page 20)								
With direct instruction addressing; Address area 0 to 127								
The following peripheral acknowledgement time must be observed with CPU 414-4H and CPU 417-4H: solo 36µs, redundant 65 µs								
3) With direct instruction addressing; Address area 0 to 255								
Load and Transfer Instructions for Address Registers								
LAR1			Load contents from ...					
	–		ACCU1	1	0.2	0.2	0.16	0.2
	AR2		Address register 2	1	0.2	0.2	0.16	0.2
	DBD	a	Data double word	2	0.4	0.3	0.24	0.3
	DID	a	Instance data double word	2	0.4	0.3	0.24	0.3
	m		32-bit constant as pointer	3	0.3	0.2	0.16	0.2
	LD	a	Local data double word	2	0.4	0.3	0.24	0.3
	MD	a	Bit memory double word	2	0.4	0.3	0.24	0.3
			... into AR1					
LAR2			Load contents from ...					
	–		ACCU1	1	0.2	0.2	0.16	0.2
	DBD	a	Data double word	2	0.4	0.3	0.24	0.3
	DID	a	Instance data double word	2	0.4	0.3	0.24	0.3
	m		32-bit constant as pointer	3	0.3	0.2	0.16	0.2
	LD	a	Local data double word	2	0.4	0.3	0.24	0.3
	MD	a	Bit memory double word	2	0.4	0.3	0.24	0.3
			... into AR2					
TAR1			Transfer contents from AR1 in ...					
	–		ACCU1	1	0.1	0.1	0.08	0.1
	AR2		Address register 2	1	0.2	0.2	0.16	0.2
	DBD	a	Data double word	2	0.4	0.2	0.16	0.2
	DID	a	Instance data double word	2	0.4	0.2	0.16	0.2
	LD	a	Local data double word	2	0.4	0.2	0.16	0.2
	MD	a	Bit memory double word	2	0.4	0.2	0.16	0.2
TAR2			Transfer contents from AR2 in ...					
	–		ACCU1	1	0.1	0.1	0.08	0.1
	DBD	a	Data double word	2	0.4	0.2	0.16	0.2
	DID	a	Instance data double word	2	0.4	0.2	0.16	0.2
	LD	a	Local data double word	2	0.4	0.2	0.16	0.2
	MD	a	Bit memory double word	2	0.4	0.2	0.16	0.2
CAR			Exchange the contents of AR1 and AR2	1	0.4	0.4	0.32	0.4
Load and Transfer Instructions for the Status Word								
L	STW		Load status word into ACCU1		0.1	0.1	0.08	0.1
T	STW		Transfer ACCU1 (bits 0 to 8) to the status word		0.1	0.1	0.08	0.1

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Load Instructions for DB Number and DB Length							
L	DBNO	Load number of data block	1	0.1	0.1	0.08	0.1
L	DINO	Load number of instance data block	1	0.1	0.1	0.08	0.1
L	DBLG	Load length of data block into byte	1	0.1	0.1	0.08	0.1
L	DILG	Load length of instance data block into byte	1	0.1	0.1	0.08	0.1
Integer Math (16 Bits)							
+I		Add 2 integers (16 bits) (ACCU1-L)=(ACCU1-L)+(ACCU2-L)	1	0.1	0.1	0.08	0.1
–I		Subtract 1 integer from another (16 bits) (ACCU1-L)=(ACCU2-L)-(ACCU1-L)	1	0.1	0.1	0.08	0.1
*I		Multiply 1 integer by another (16 bits) (ACCU1)=(ACCU2-L)*(ACCU1-L)	1	0.8	0.8	0.64	0.8
/I		Divide 1 integer by another (16 bits) (ACCU1-L)=(ACCU2-L):(ACCU1-L) The remainder is in ACCU1-H	1	0.8	0.8	0.64	0.8
Integer Math (32 Bits)							
+D		Add 2 integers (32-bit) (ACCU1)=(ACCU2)+(ACCU1)	1	0.1	0.1	0.08	0.1
–D		Subtract 1 integer from another (32 bits) (ACCU1)=(ACCU2)-(ACCU1)	1	0.1	0.1	0.08	0.1
*D		Multiply 1 integer by another (32 bits) (ACCU1)=(ACCU2)*(ACCU1)	1	1.3	1.3	1.04	1.3
/D		Divide 1 integer by another (32 bits) (ACCU1)=(ACCU2):(ACCU1)	1	1.3	1.3	1.04	1.3
MOD		Divide 1 integer by another (32 bits) and load the remainder into ACCU1: (ACCU1)=remainder of [(ACCU2):(ACCU1)]	1	1.3	1.3	1.04	1.3
Floating-Point Math (32 Bits)							
+R		Add 2 real numbers (32 bits) (ACCU1)=(ACCU2)+(ACCU1)	1	0.6	0.6	0.48	0.6
–R		Subtract 1 real number from another (32 bits) (ACCU1)=(ACCU2)-(ACCU1)	1	0.6	0.6	0.48	0.6
*R		Multiply 1 real number by another (32 bits) (ACCU1)=(ACCU2)*(ACCU1)	1	1.4	1.4	1.12	1.4
/R		Divide 1 real number by another (32 bits) (ACCU1)=(ACCU2):(ACCU1)	1	2.1	2.1	1.68	2.1
NEGR		Negate the real number in ACCU1	1	0.1	0.1	0.08	0.1
ABS		Form the absolute value of the real number in ACCU1	1	0.1	0.1	0.08	0.1
Square Root and Square Instructions (32 Bits)							
SQRT		Calculate the square root of a real number in ACCU1	1	72	40	37 - 39	40
SQR		Form the square of the real number in ACCU1	1	1.4	1.4	1.12	1.4

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Logarithmic Function (32 Bits)							
LN		Form the natural logarithm of a real number in ACCU1	1	63	35	33	35
EXP		Calculate the exponential value of a real number in ACCU1 to the base e (= 2.71828)	1	63	35	32 - 34	35
Trigonometrical Functions (32 Bits)							
SIN		Calculate the sine of a real number	1	56	31	30	31
ASIN		Calculate the arcsine of a real number	1	117 - 133	65 - 74	62 - 70	65 - 74
COS		Calculate the cosine of a real number	1	58	32	30	32
ACOS		Calculate the arccosine of a real number	1	122 - 139	68 - 77	65 - 72	68 - 77
TAN		Calculate the tangent of a real number	1	58 - 63	32 - 35	30 - 33	32 - 35
ATAN		Calculate the arctangent of a real number	1	43 - 58	24 - 32	23 - 30	24 - 32
Adding Constants							
+	i8	Add an 8-bit integer constant	1	0.1	0.1	0.08	0.1
+	i16	Add a 16-bit integer constant	2	0.2	0.1	0.08	0.1
+	i32	Add a 32-bit integer constant	3	0.3	0.15	0.12	0.15
Adding Using Address Registers							
+AR1		Add the contents of ACCU1-L to those of AR1	1	0.2	0.2	0.16	0.2
+AR1	m (0 to 4095)	Add a pointer constant to the contents of AR1	2	0.2	0.2	0.16	0.2
+AR2		Add the contents of ACCU1-L to those of AR2	1	0.2	0.2	0.16	0.2
+AR2	m (0 to 4095)	Add pointer constant to the contents of AR2	2	0.2	0.2	0.16	0.2
Comparison Instructions (16-Bit Integers)							
==I		ACCU2-L=ACCU1-L	1	0.1	0.1	0.08	0.1
<>I		ACCU2-L_ACCU1-L	1	0.1	0.1	0.08	0.1
<I		ACCU2-L<ACCU1-L	1	0.1	0.1	0.08	0.1
<=I		ACCU2-L<=ACCU1-L	1	0.1	0.1	0.08	0.1
>I		ACCU2-L>ACCU1-L	1	0.1	0.1	0.08	0.1
>=I		ACCU2-L>=ACCU1-L	1	0.1	0.1	0.08	0.1

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Comparison Instructions (32-Bit Integers)							
==D		ACCU2=ACCU1	1	0.1	0.1	0.08	0.1
<>D		ACCU2_ACCU1	1	0.1	0.1	0.08	0.1
<D		ACCU2<ACCU1	1	0.1	0.1	0.08	0.1
<=D		ACCU2<=ACCU1	1	0.1	0.1	0.08	0.1
>D		ACCU2>ACCU1	1	0.1	0.1	0.08	0.1
>=D		ACCU2>=ACCU1	1	0.1	0.1	0.08	0.1
==R		ACCU2=ACCU1	1	0.1	0.1	0.08	0.1
<>R		ACCU2_ACCU1	1	0.1	0.1	0.08	0.1
<R		ACCU2<ACCU1	1	0.1	0.1	0.08	0.1
<=R		ACCU2<=ACCU1	1	0.1	0.1	0.08	0.1
>R		ACCU2>ACCU1	1	0.1	0.1	0.08	0.1
>=R		ACCU2>=ACCU1	1	0.1	0.1	0.08	0.1
Shift Instructions							
SLW[1)]		Shift the contents of ACCU1-L to the left. Positions	1	0.1	0.1	0.08	0.1
SLW	0 ... 15	that become free are provided with zeros.					
SLD		Shift the contents of ACCU1 to the left. Positions that	1	0.1	0.1	0.08	0.1
SLD	0 ... 32	become free are provided with zeros.					
SRW[1)]		Shift the contents of ACCU1-L to the right. Positions	1	0.1	0.1	0.08	0.1
SRW	0 ... 15	that become free are provided with zeros.					
SLD		Shift the contents of ACCU1 to the left. Positions that	1	0.1	0.1	0.08	0.1
SLD	0 ... 32	become free are provided with zeros.					
SRW[1)]		Shift the contents of ACCU1-L to the right. Positions	1	0.1	0.1	0.08	0.1
SRW	0 ... 15	that become free are provided with zeros.					
SRD		Shift the contents of ACCU1 to the right. Positions	1	0.1	0.1	0.08	0.1
SRD	0 ... 32	that become free are provided with zeros.					
SSI[1)]		Shift the contents of ACCU1-L with sign to the right.	1	0.1	0.1	0.08	0.1
SSI	0 ... 15	Positions that become free are provided with with the sign (bit 15).					
SSD		Shift the contents of ACCU1 with sign to the right. Positions that become free are provided with with the	1	0.1	0.1	0.08	0.1
1) No. of places shifted: 0 to 16							
Rotate Instructions							
RLD		Rotate the contents of ACCU1 to the left	1	0.1	0.1	0.08	0.1
RLD	0 ... 32						
RRD		Rotate the contents of ACCU1 to the right	1	0.1	0.1	0.08	0.1
RRD	0 ... 32						
RLDA		Rotate the contents of ACCU1 one bit position to the left through	0.1	0.1	0.08	0.1	RLDA

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Rotate Instructions, continued							
RRDA		Rotate the contents of ACCU1 one bit position to the right through condition code bit CC 1	0.1	0.1	0.08	0.1	RRDA
Accumulator Transfer Instructions, Incrementing and Decrementing							
CAW		Reverse the order of the bytes in ACCU1-L.	1	0.1	0.1	0.08	0.1
CAD		Reverse the order of the bytes in ACCU1.	1	0.1	0.1	0.08	0.1
TAK		Swap the contents of ACCU1 and ACCU2	1	0.1	0.1	0.08	0.1
ENT		The contents of ACCU2 and ACCU3 are transferred to ACCU3 and ACCU4.	1	0.1	0.1	0.08	0.1
LEAVE		The contents of ACCU3 and ACCU4 are transferred to ACCU2 and ACCU3.	1	0.1	0.1	0.08	0.1
PUSH		The contents of ACCU1, ACCU2 and ACCU3 are transferred to ACCU2, ACCU3 and ACCU4	1	0.1	0.1	0.08	0.1
POP		The contents of ACCU2, ACCU3 and ACCU4 are transferred to ACCU1, ACCU2 and ACCU3	1	0.1	0.1	0.08	0.1
INC	k8	Increment ACCU1-LL	1	0.1	0.1	0.08	0.1
DEC	k8	Decrement ACCU1-LL	1	0.1	0.1	0.08	0.1
Program Display and Null Operation Instructions							
BLD	k8	Program display instruction: Is treated by the CPU as a null operation instruction	1	0.1	0.1	0.08	0.1
NOP	0 1	Null operation instruction	1	0.1	0.1	0.08	0.1
Data Type Conversion Instructions							
BTI		Convert contents of ACCU1-L from BCD (0 to +/- 999) to integer (16 bits) (**B**CD **T**o **I**nt)	1	0.1	0.1	0.08	0.1
BTD		Convert contents of ACCU1 from BCD (0 to +/-9 999 999) to double integer (32 bits) (**B**CD **T**o **D**oubleint)	1	0.1	0.1	0.08	0.1
DTR		Convert contents of ACCU1 from double integer (32 bits) to real number (32 bits) (**D**oubleint **T**o **R**eal)	1	0.3	0.3	0.24	0.3
ITD		Convert contents of ACCU1 from integer (16 bits) to double integer (32 bits) (**I**nt **T**o **D**oubleint)	1	0.1	0.1	0.08	0.1
ITB		Convert contents of ACCU1-L from integer (16 bits) to BCD from 0 to +/- 999 (**I**nt **T**o **B**CD)	1	0.1	0.1	0.08	0.1
DTB		Convert contents of ACCU1 from double integer (32 bits) to BCD from 0 to +/- 9 999 999 (**D**oubleint **T**o **B**CD)	1	0.2	0.2	0.16	0.2
RND+		Convert a real number into a 32-bit integer. The number is rounded up to the next whole number.	1	0.4	0.4	0.32	0.4
RND		Convert a real number into a 32-bit integer.	1	0.4	0.4	0.32	0.4
RND-		Convert a real number into a 32-bit integer. The number is rounded down to the next whole number.	1	0.4	0.4	0.32	0.4
TRUNC		Convert a real number into a 32-bit integer. The places after the decimal point are truncated.	1	0.4	0.4	0.32	0.4

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Forming the Ones and Twos Complements							
INVI		Form the ones complement of ACCU1-L	1	0.1	0.1	0.08	0.1
INVD		Form the ones complement of ACCU1	1	0.1	0.1	0.08	0.1
NEGI		Form the twos complement of ACCU1-L (integer)	1	0.1	0.1	0.08	0.1
NEGD		Form the twos complement of ACCU1 (double integer)	1	0.1	0.1	0.08	0.1
Block Call Instructions							
CALL	FB q, DB q	Unconditional call of an FB, with parameter transfer	1[1]/2	8.2 [3]	3.2 [3]	2.56 [3]	XX
CALL	SFB q, DB q	Unconditional call of an SFB, with parameter transfer	2	8.2 [3]	3.2 [3]	2.56 [3]	XX
CALL	FC q	Unconditional call of a function, with parameter transfer	1[1]/2	4.6 [3]	1.8 [3]	1.44 [3]	XX
CALL	SFC q	Unconditional call of an SFC, with parameter transfer	2	4.6 [3]	1.8 [3]	1.44 [3]	XX
UC	FB q	Unconditional call of blocks,	1[1]/2	2.1/2.2	1.4	1.12	1.4
	FC q	without parameter transfer		2.1/2.2	1.4	1.12	1.4
	FB [e]	Memory-indirect FB call	2	2.2+	1.4+	1.12+	1.4+
	FC [e]	Memory-indirect FC call	2	2.2+	1.4+	1.12+	1.4+
	Parameter	FB/FC call via parameter	2	2.2+	1.4+	1.12+	1.4+
CC	FB q	Conditional call of blocks, without	1[1]/2	2.3/2.4/0.4 [4]	1.4/0.4 [4]	1.12/0.32 [4]	1.4/04 [4]
	FC q	parameter transfer		2.3/2.4/0.4 [4]	1.4/0.4 [4]	1.12/0.32 [4]	1.4/0.4 [4]
	FB [e]	Memory-indirect FB call	2	2.4+/0.4 [4]	1.4+/0.4 [4]	1.12+/0.32 [4]	1.4+/0.4 [4]
	FC [e]	Memory-indirect FC call	2	2.4+/0.4 [4]	1.4+/0.4 [4]	1.12+/0.32 [4]	1.4+/0.4 [4]
	Parameter	FB/FC call via parameter	2	2.4+/0.4 [4]	1.4+/0.4 [4]	1.12+/0.32 [4]	1.4+/0.4 [4]
OPN		Open:	1[1]/2				
	DB q	Data block		0.6/0.7	0.3	0.24	0.3
	DI q	Instance data block		0.7	0.3	0.24	0.3
	DB [e]	Data block, memory-indirect		0.7+	0.3+	0.24+	0.3+
	DI [e]	Instance DB, memory-indirect		0.7+	0.3+	0.24+	0.3+
	Parameter	Data block using parameters		0.7+	0.3+	0.24+	0.3+

Plus time required for loading the address of the instruction (see page 20)
With direct instruction (DB) addressing; Block No. 0 to 255
Depending on RLO, sets RLO = 1
Plus time required for supplying parameters
4) If call is not executed

Block End Instructions							
BE		End block	1	2.8	2.0	1.60	2.0
BEU		End block unconditionally	1	2.8	2.0	1.60	2.0
BEC		End block conditionally if RLO = "1"	3.0 0.4[1]	2.2 0.4[1]	1.76 0.32[1]	2.2 0.4[1]	

If jump is not executed

Exchanging Shared Data Block and Instance Data Block							
CDB		Exchange shared data block and instance data block	1	0.2	0.2	0.16	

Instr.	Address ID	Description	Length in Words	Execution Time in µs			
				CPU 412	CPU 414	CPU 416	CPU 417
Jump Instructions							
JU	LABEL	Jump unconditionally	1[1]/2	0.5/0.6	0.5	0.4	0.5
JC	LABEL	Jump if RLO = “1”	1[1]/2	0.5/0.6 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JCN	LABEL	Jump if RLO = “0”	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JCB	LABEL	Jump if RLO = “1”. Save the RLO in the BR bit	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JNB	LABEL	Jump if RLO = “0”. Save the RLO in the BR bit	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JBI	LABEL	Jump if BR = “1”	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JNBI	LABEL	Jump if BR = “0”	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JO	LABEL	Jump on stored overflow (OV = “1”)	1[1]/2	0.5/0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JOS	LABEL	Jump on stored overflow (OS = “1”)	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JUO	LABEL	Jump if “unordered math instruction” (CC1=1 and CC0=1)	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JZ	LABEL	Jump if result = 0 (CC1=0 and CC0=0)	1[1]/2	0.5/0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JP	LABEL	Jump if result > 0 (CC1=1 and CC0=0)	1[1]/2	0.5/0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JM	LABEL	Jump if result < 0 (CC1=0 and CC0=1)	1[1]/2	0.5/0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JN	LABEL	Jump if result _ 0 (CC1=1 and CC0=0) or (CC1=0 and CC0=1)	1[1]/2	0.5/0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JMZ	LABEL	Jump if result _ 0 (CC1=0 and CC0=1) or (CC1=0 and CC0=0)	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JPZ	LABEL	Jump if result _ 0 (CC1=1 and CC0=0) or (CC1=0 and CC0=0)	2	0.6/0.2 [2]	0.5/0.2 [2]	0.4/0.16 [2]	0.5/0.2 [2]
JL	LABEL	Jump distributor	2	0.8	0.7	0.56	0.7
LOOP	LABEL	Decrement ACCU1-L and jump if ACCU1-L <> 0	2	0.6/0.2 [1]	0.5/0.2 [1]	0.4/0.08 [1]	0.5/0.2 [1]
1 word long for jump widths between -128 and +127 2) If jump is not executed							
Instructions for the Master Control Relay (MCR)							
MCR(		Open an MCR zone. Save the RLO to the MCR stack.	1	0.1	0.1	0.08	0.1
)MCR		Close an MCR zone. Pop an entry off the MCR stack.	1	0.1	0.1	0.08	0.1
MCRA		Activate the MCR	1	0.1	0.1	0.08	0.1
MCRD		Deactivate the MCR	1	0.1	0.1	0.08	0.1
MCR(		Open an MCR zone. Save the RLO to the MCR stack.	1	0.1	0.1	0.08	0.1
)MCR		Close an MCR zone. Pop an entry off the MCR stack.	1	0.1	0.1	0.08	0.1
MCRA		Activate the MCR	1	0.1	0.1	0.08	0.1

System Functions

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
0	SET_CLK	Set clock	340	249	215	249	289	288
1	READ_CLK	Read clock	40	29	23	29	29	54
2	SET_RTM	Set run-time meter	35	26	20	26	25	26
3	CTRL_RTM	Start and stop run-time meter	30	23	18	23	22	22
4	READ_RTM	Read run-time meter	41	30	23	30	29	58
5	GADR_LGC	Find logical address of a channel	55	39	31	39	38	38
		Rack 0 internal DP	66	46	36	46	46	46
6	RD_SINFO	Read start information of current OB	54	38	30	38	39	39
7	DP_PRAL	Trigger a process interrupt at the DP master First call	294	208	166	208	--	--
		Intermediate call	43	30	24	30	--	--
9	EN_MSG	Enable block-related, symbol-related, and group status messages. First call, REQ = 1	176	122	97	122	128	232
		Last call	61	44	34	44	39	62
10	DIS_MSG	Disable block-related, symbol-related, and group status messages. First call, REQ = 1	176	122	97	122	128	232
		Last call	61	44	34	44	39	63
11	DPSYC_FR	Synchronize groups of DP Slaves First call, internal DP interface, REQ = 1	170	110	90	110	--	--
		Intermediate call, internal DP interface, BUSY = 1 [1]	51 + n* 4	36 + n* 3	28 + n* 2	36 + n* 3	--	--
		Last call, internal DP interface, BUSY=0 [1]	51 + n* 4	36 + n* 3	28 + n* 2	36 + n* 3	--	--
11	DPSYC_FR	First call, external DP interface, REQ=1	94	71	60	71	--	--
		Intermediate call, external DP interface, BUSY = 1 [1]	64 + n* 4	50 + n* 3	39 + n* 2	50 + n* 3	--	--
		Last call, external DP interface, BUSY= 0 [1]	64 + n* 4	50 + n* 3	39 + n* 2	50 + n* 3	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via integrated DP interface, MODE = 0	117	76	61	76	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via integrated DP interface, MODE = 1 First call	269	179	142	179	--	--
		Intermediate call	114	73	59	73	--	--
		Last call	231	167	121	167	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via integrated DP interface, MODE = 2 First call	378	268	202	268	--	--
		Intermediate call	113	72	58	72	--	--
		Last call	119	76	62	76	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via external DP interface, MODE = 0	X	X	X	X	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via external DP interface, MODE = 1	X	X	X	X	--	--
		Intermediate call	X	X	X	X	--	--
		Last call	X	X	X	X	--	--
12	D_ACT_DP	Deactivate and activate DP slaves via external DP interface, MODE = 2 First call	X	X	X	X	--	--
		Intermediate call	X	X	X	X	--	--
13	DP_NRMDG	Read slave diagnostic data First call	300	200	165	200	210	290
		Intermediate call	--	--	--	--	79	79
14	DPRD_DAT	Read consistent user data (n bytes) via integrated DP interface 3 bytes	83	56	45	56	70	96
		via integrated DP interface 32 bytes	94	67	54	67	88	122
		via external DP interface 3 bytes	86	62	50	62	76	99
		via external DP interface 32 bytes	181	156	137	156	152	209

[1] n = number of active jobs with the same logic address

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
15	DPWR_DAT	Write consistent user data (n bytes)	84[1]/91[2]	57[1]/61[2]	45[1]/49[2]	57[1]/61[2]	72[1]/76[2]	94[1]/98[2]
		via integrated DP interface 3 bytes	96[1]/127[2]	67[1]/97[2]	53[1]/78[2]	67[1]/97[2]	88[1]/119[2]	110[1]/142[2]
		via integrated DP interface 32 bytes	88[1]/94[2]	62[1]/67[2]	50[1]/54[2]	62[1]/67[2]	77[1]/83[2]	100[1]/105[2]
		via external DP interface 3 bytes	178[1]/209[2]	150[1]/181[2]	130[1]/154[2]	150[1]/181[2]	171[1]/201[2]	193[1]/224[2]
		via external DP interface 32 bytes						
17	ALARM_SQ	Generate acknowledgeable block-related messages.	440	305	240	305	266	358
		First call, SIG = 0 –> 1						
		Empty call	130	90	72	90	90	156
18	ALARM_S	Generate unacknowledgeable block-related messages.	460	310	250	310	275	365
		First call, SIG = 0 –> 1						
		Empty call	140	90	75	90	97	163
1) without data transmission to the process image 2) with data transmission to the process image								
19	ALARM_SC	Acknowledgment status of the last ALARM_SQ entering state message.	85	60	46	60	56	82
20	BLKMOV	Copy variable	60 + n * 0.3	41 + n * 0.13	32 + n * 0.23	41 + n * 0.13	42 + n * 0.17	42 + n * 0.17
		within the work memory (n = number of bytes to be copied)						
		Source = Load memory	1400 + n* 1.0	1160 + n* 0.7	1100 + n* 0.7	1160 + n * 0.7	1124 + n * 1.0	2065 + n * 1.98
21	FILL	Set array default variables	60 + n * 0.15	44 + n * 0.13	34 + n * 0.1	44 + n * 0.13	45 + n * 0.12	45 + n * 0.12
		within the work memory (n = length of target variables in bytes)						
22	CREAT_DB	Create data block	142	94	72	94	155 + n * 0.1	424 + n * 0.1
		n = DB length [bytes]						
		Occupy last free DB No. from a field of 100 DBs	606	400	320	400	2877	13601
23	DEL_DB	Delete data block	122	81	64	81	179	625
24	TEST_DB	Test data block	47	32	25	32	68	248
25	COMPRESS	Compress user memory	112	78	63	78	93	173
		First call (trigger)						
		Intermediate call (active)	32	23	18	23	22	22
26	UPDAT_PI	Update process image input table (run-time entry for 1 DI 32 in the central rack)	45	35	29	35	67	103
		AI 8* 13Bit	70	59	51	59	155	192
27	UPDAT_PO	Update process image output table (run-time entry for 1 DO 32 in the central rack)	45	35	29	35	54	81
		AO 8* 13Bit	66	55	48	55	122	149
28	SET_TINT	Set time-of-day interrupt	108	75	60	75	74	98
29	CAN_TINT	Cancel time-of-day interrupt	40	29	22	29	34	34
30	ACT_TINT	Activate time-of-day interrupt	73	53	41	53	51	75
31	QRY_TINT	Query time-of-day interrupt	44	34	27	34	33	34
32	SRT_DINT	Start time-delay interrupt	65	46	36	46	44	44
33	CAN_DINT	Cancel time-delay interrupt	41	30	23	30	36	36
34	QRY_DINT	Query time-delay interrupt	43	33	26	33	32	32
35	MP_ ALM	Trigger multicomputing interrupt	240	171	138	171	--	--
36	MSK_FLT	Mask synchronous faults	30	22	17	22	21	21
37	DMSK_FLT	Demask synchronous faults	31	23	18	23	22	23
38	READ_ERR	Read error register	32	23	18	23	23	23
39	DIS_IRT	Discard new events	555	535	580	535	731	732
		Block all events (MODE = 0)						
		Block all events of a priority class (MODE = 1)	70 - 190	50 - 145	40 - 160	50 - 145	42 - 194	42 - 194
		Block one event (MODE = 2)	40 - 50	30 - 37	24 - 28	30 - 37	31 - 36	31 - 37
40	EN_IRT	Stop discarding events	555	535	580	535	736	737
		Enable all events (MODE = 0)						
		Enable all events in a priority class (MODE = 1)	70 - 190	50 - 145	40 - 160	50 - 145	42 - 197	42 - 197
		Enable an event (MODE = 2)	40 - 50	30 - 37	24 - 28	30 - 37	31 - 37	31 - 37
41	DIS_AIRT	Delay interrupt events	248	166	132	166	165	165
		the first time delay is activated[1]						
		if the delay is already activated	26	19	14	19	18	18
1) When activating the delay for the first time, the SFC 41 runtime depends on the priority class in which the SFC 41 is called. The specified runtime refers to the call in OB 1. It decreases while the priority class number increases.								

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
42	EN_AIRT	Stop delaying interrupt events when canceling the last delay [2]	26	19	14	19	18	18
		if other delays are present	435	320	272	320	343	343
43	RE_TRIGR	Retrigger watchdog monitoring	155	104	84	104	118	307
44	REPL_VAL	Transfer substitute value to ACCU1	30	21	16	21	20	20
46	STP	Force CPU into STOP mode cannot be measured	—	—	—	—	—	—
47	WAIT	Delay program execution in addition to waiting time	13 - 18	7 - 15	4 - 11	7 - 15	6 - 13	6 - 13
48	SNC_RTCB	Synchronize slave clocks	25	19	14	19	18	41
49	LGC_GADR	Find slot with logical address	55	40	31	40	41	41
50	RD_LGADR	Find all logical addresses of a block (run-time entry for 1 DI 32 in the central rack)	146	101	80	101	104	104
When cancelling the last delay, the SFC 42 runtime depends on the priority class in which the SFC 42 is called. The specified runtime refers to the call in OB 1. It decreases while the priority class number increases.								
51	RDSYSST	List of all system status list information (0000)	618	493	395	493	477	477
		List of all system status list information (0F00)	140	97	77	97	97	97
51	RDSYSST	”Module Identification” partial list	224	170	135	170	169	168
		Display all data records (0011)						
		Display one data record (0111)	175	125	100	125	123	122
		Display header information (0F11)	145	100	80	100	99	99
51	RDSYSST	“CPU Characteristics” partial list	317	235	187	235	233	232
		Display all data records (0012)						
		Display one data record (0112)	190 - 215	135 - 155	108 - 123	135 - 155	135 - 155	135 - 154
		Display header information (0F12)	145	100	80	100	99	98
51	RDSYSST	“Save” partial list	185	134	105	134	134	133
		Display all data records (0013)						
		Display one data record (0113)	185	134	105	134	134	133
		Display header information (0F13)	145	100	80	100	100	99
51	RDSYSST	“System Areas” partial list	220	145	120	145	145	144
		Display all data records (0014)						
		Display one data record (0114)	170	117	93	117	117	117
		Display header information (0F14)	745	480	480	99	99	99
51	RDSYSST	“Block Types” partial list	196	425	425	145	145	144
		Display all data records (0015)						
		Dispaly one data record (0115)	165 - 185	118 - 128	94 - 102	118 - 128	119 - 128	118 - 127
		Display header information (0F15)	142	100	78	100	98	98
51	RDSYSST	”Priority Classes” partial list	858	740	765	740	947	947
		Display all data records (0016)						
		Display one data record (0116)	196 - 347	110 - 250	110 - 135	110 - 250	137 - 291	137 - 290
		Display header information (0F16)	153	106	85	106	107	106
51	RDSYSST	“Status of Module LEDs” partial list	322	216	175	216	225	--
		Display status of all LEDs (0019)						
		Display status of one LED (0119)	225	150	120	150	151	--
		Display header information (0F19)	206	136	110	136	136	--
51	RDSYSST	“Interrupt/Error Assignment” partial list	1225	1010	1055	1010	1298	1297
		Display all data records (0021)						
		Display all data records of one interrupt class (0121)	210 - 590	145 - 410	115 - 330	145 - 410	145 - 365	144 - 364
51	RDSYSST	Display one data record (0221)	195 - 215	135 - 150	110 - 120	135 - 150	135 - 152	135 - 151
		Display all assigned interrupts of one class (0921)	225 - 640	155 - 440	125 - 390	155 - 440	155 - 485	155 - 485
		Alternative: n= number of loaded OBs (0921)	(225/ 375)+ n*34	(155/ 260)+ n*23	(125/ 245)+ n*18	(155/ 260)+ n*23	(155/ 305)+ n*23	(155/ 305)+ n*23
51	RDSYSST	“Interrupt/Error Assignment” partial list	930 - 1510	795 - 1285	835 - 1390	795 - 1285	1037 - 1697	1037 - 1697
		Display all assigned interrupts (0A21)						
		Display header information (0F21)	155	107	85	107	108	107

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
51	RDSYSST	"Interrupt Status" partial list	225 - 660	160 - 490	125 - 390	160 - 490	157 - 432	160 - 450
		Display all data records of one interrupt class (0122)						
		Display one data record (0222)	210 - 225	148 - 158	118 - 128	148 - 158	148 - 155	148 - 158
		Display all assigned interrupts of one class (0822)	235 - 720	165 - 515	130 - 470	165 - 515	165 - 560	165 - 560
		Alternative: n = number of loaded OBs	(235/ 375)+ n*45	(165/ 260)+ n*35	(130/ 245)+ n*25	(165/ 260)+ n*35	(165/ 305)+ n*35	(165/ 305)+ n*35
		Display header information (0F22)	158	110	87	110	44	108
51	RDSYSST	"Status of Priority Classes" partial list	210	147	117	147	147	147
		Display one data record (0123)						
		All priority classes in process (0223) (n= number of priority classes)	535 + n*52	450 + n*35	443 + n*28	450 + n*35	540 + n*36	540 + n*36
		Display header information (0F23)	145	100	80	100	101	100
51	RDSYSST	"Operating Mode" partial list	200	140	111	140	139	138
		Display the last operating mode transition (0124)						
		Display the current operating mode	175	125	100	125	125	125
51	RDSYSST	"Status Information Communication" partial list	205	150	120	150	157	181
		Display status information of a communication unit (0132) INDEX = 5						
51	RDSYSST	"Status Information Communication" partial list	–	–	–	–	235	425
		Display status information of a communication unit (0232) INDEX = 4						
51	RDSYSST	"Start Information List" partial list	190 - 225	128 - 155	102 - 135	128 - 155	127 - 168	127 - 167
		All synchronization error start information of one priority class (0281)						
		All start information of one priority class (0381)	210 - 395	128 - 305	102 - 255	128 - 305	128 - 318	127 - 317
		All synchronization error start information of one priority class before processing (0681)	190 - 225	128 - 155	102 - 135	128 - 155	127 - 168	127 - 167
		All start information of one priority class before processing (0781)	190 - 390	145 - 295	115 - 235	145 - 295	142 - 293	141 - 293
		All synchronization error start information of one priority class in process (0A81)	190 - 225	130 - 160	102 - 135	130 - 160	129 - 170	128 - 169
		All start information of one priority class in process (0B81)	190 - 240	130 - 170	102 - 145	130 - 170	129 - 179	129 - 179
		Display one header information (0F81)	160	112	90	112	112	111
51	RDSYSST	"Start Information List" partial list	190 - 220	128 - 150	102 - 125	128 - 150	128 - 156	128 - 155
		All synchronization error start events of one priority class (0282)						
		All start events of one priority class (0382)	210 - 305	128 - 225	102 - 185	128 - 225	129 - 230	128 - 229
		All synchronization error start events of one priority class before processing (0682)	190 - 220	128 - 150	102 - 125	128 - 150	128 - 156	128 - 155
		All start events of one priority class before processing (0782)	210 - 310	145 - 225	115 - 180	145 - 225	143 - 225	142 - 223
		All synchronization error start events of one priority class in process (0A82)	190 - 220	130 - 150	102 - 125	130 - 150	130 - 158	129 - 161
		All start events of one priority class in process (0B82)	190 - 225	130 - 155	102 - 130	130 - 155	130 - 162	130 - 161
		Display one header information (0F82)	160	112	90	112	113	112

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
51	RDSYSST	"Module Status Information" partial list Display the status information of all inserted modules (n = number of the data records) (0091)	660 + n* 22	508 + n* 19	408 + n* 16	508 + n* 19	--	--
		Display the status information: of all modules/racks with incorrect type ID (0191)	570 + n* 70	427 + n* 60	365 + n* 40	405 + n* 24	--	--
		of all faulty modules (0291)	580 + n* 138	428 + n* 22	344 + n* 18	428 + n* 22	--	--
		of all unavailable modules (0391)	585 + n* 72	430 + n* 60	370 + n * 40	430 + n* 60	--	--
		Display the status information: of all submodules of the host module in the specified rack (0991)	354 + n* 30	250 + n* 26	200 + n* 21	250 + n* 26	--	--
		centralized of one module with logical base address (0C91)	200 - 315	180	145	180	177	242
		distributed of one module with logical base address (0C91)	315	225	180	225	224	289
51	RDSYSST	"Module Status Information" partial list of a module (distributed) with a logical base address (4C91) first call	200 - 315	145 - 240	130 - 190	145 - 240	242	305
		"Module Status Information" partial list of a module (distributed) with a logical base address (4C91) intermediate call	--	--	--	--	148	148
		"Module Status Information" partial list of a module (distributed) with a logical base address (4C91) last call	--	--	--	--	167	167
		local of all modules in the specified rack (n = number of the DS) (0D91)	377 + n* 13	275 + n* 16	240 + n* 10	275 + n * 16	260 + n * 20	405 + n * 23
		distributed of all modules in the specified distributed I/O station (0D91)	330 - 390	250 - 300	200 - 240	250 - 300	305	408 - 420
51	RDSYSST	Display one header information (0F91) "Rack/Station Status Information" partial list, local, Display the setpoint status of the racks 0 (0092)	560 180	435 127	350 100	435 127	-- 130	-- 154
		distributed, Display the setpoint status of distributed I/O system 1 (0092)	900	725	585	725	712	743
		local, Display the actual status of the racks 0 (0292)	180	127	103	127	131	155
		distributed, Display the actual status of distributed I/O system 1 (0292)	940	745	600	745	725	757
51	RDSYSST	Display header information (0F92) "Diagnostic Buffer" partial list Display all available current operating mode event information (max. 23) (00A0) Display the n newest entries (n = 1-23) (01A0) Display the standard OB start information (04A0). Max value of -04A0 is calculated	160 195 - 525 195 + n* 14.5 195 - 1270	113 138 - 410 138 + n* 12 138 - 1530	90 110 - 330 110 + n* 9.5 110 - 1095	113 138 - 410 138 + n* 12 138 - 1530	113 140 - 412 140 + n* 12 140 - 1540	113 140 - 412 140 + n* 12 140 - 1540

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
51		Display all communication information (05A0) Display all object management system information (06A0) Display all test and startup information (07A0) Display all operating mode information (08A0) Display all asynchronous error start information (09A0) Display all synchronous error start information (0AA0) Display all STOP/cancel/operating mode transition information (0BA0) Display all fault-tolerant/failsafe information (0CA0) Display all diagnostic information (0DA0) Display all user information (0EA0)	195 - 1270	138 - 1530	110 - 1095	138 - 1530	140 - 1540	140 - 1540
		Display header information (0FA0)	167	--	90	--	114	114
		Display header information (0FA0)	167	--	90	--	114	114
51	RDSYSST	“Diagnostic Data DS 0”, partial list Display via logical address (00B1) local	406	286	233	286	300	360
		distributed First call	392	270	217	270	278	356
		distributed Intermediate call, REQ = 0	215	150	120	150	153	152
		distributed Last call	405	165	132	165	170	169
51	RDSYSST	”Diagnostic Data DS 1” partial list Display via graphical address (00B2) Display a 16-byte long DS 1	408	300	250	300	313	375
51	RDSYSST	”Diagnostic Data DS 1” partial list Display via logical address (00B3) Display a 16-byte long DS 1 local	447	324	268	324	340	402
		distributed First call	395	270	218	270	272	356
		distributed Intermediate call	218	150	120	150	153	153
		distributed Last call	257	178	142	178	182	182
51	RDSYSST	”Diagnostic Data DP Slave” partial list Display via configured diagnostic address (00B4) First call	385	266	213	266	272	351
		Intermediate call, REQ = 0	--	--	115	--	149	148
		Last call (6 - 240 bytes)	246	170	135	170	174	173
52	WR_USMSG	Write user entry in diagnostic buffer write with message	186	128	102	128	75	100
		without message	107	75	60	75	74	98
54	RD_DPA-RAM	Read dynamic parameters local AI 8*13 bits	180	125	95	125	126	153
		distributed AI 8*12 bits (DS1 = 14 bytes)	200	135	105	135	121	121
55	WR_PARM	Write dynamic parameters local AI 8*13 bits	485	345	280	345	360	418
		distributed First call AI 8*12 bits (14 - 240 bytes)	370	260	210	260	268	347
		distributed Intermediate/last call, REQ = 0	175	115	90	115	122	122

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
56	WR_DPARM	Write predefined dynamic parameters	445	336	280	336	353	411
		AI 8*13 bits						
		local						
		distributed	300	205	165	205	217	296
		First call AI 8*12 bits (2 - 240 bytes)						
		Intermediate/last call	145	100	80	100	106	106
57	PARM_MOD	Assign module parameters	770	580	490	580	609	695
		local						
		Module/DS number/DS lengths in bytes						
		AI 8*13 bits						
		distributed	300	205	165	205	215	295
		AO 8*12 bits						
		First call (16 - 240 bytes)						
		distributed	145	100	80	100	104	104
		Intermediate/last call						
58	WR_REC	Write parameter data record	390 + n*	267 + n*	217 + n*	267 + n*	282 + n*	311 + n*
		local (n = number of bytes)	2.87	2.71	2.52	2.71	2.68	2.71
		First call, integrated DP interface module	334 + n*	228 + n*	182 + n*	228 + n*	222 + n*	276 + n*
		(n = number of bytes)	0.42	0.35	0.30	0.35	0.39	0.32
		Intermediate call, REQ = 0	138	90	70	90	94	94
		integrated DP interface module						
		Last call, integrated DP interface module	138	90	70	90	95	94
		First call, external DP interface module	332 + n*	215 + n*	171 + n*	215 + n*	208 + n*	208 + n*
		(n = number of bytes)	0.32	0.26	0.23	0.26	0.26	0.29
		Intermediate call, REQ = 0	139	90	72	90	95	94
		external DP interface module						
		Last call, external DP interface module	140	91	72	91	95	95
59	RD_REC	Read data record	390 + n*	267 + n*	218 + n*	267 + n*	282 + n*	342 + n*
		local (n = number of bytes)	3.13	2.90	2.71	2.90	2.97	3.13
		First call, integrated DP interface module	322	217	172	217	212	264
		Intermediate call, REQ = 0	138	90	70	90	95	94
		integrated DP interface module						
		Last call, integrated DP interface module	198 + n*	132 + n*	106 + n*	132 + n*	138 + n*	138 + n*
		(n = number of bytes)	0.35	0.33	0.27	0.33	0.33	0.33
		First call, external DP interface module	304	204	163	204	198	197
		Intermediate call, REQ = 0	139	91	72	91	95	94
		external DP interface module						
		Last call, external DP interface module	200 + n*	132 + n*	105 + n*	132 + n*	136 + n*	136 + n*
		(n = number of bytes)	0.33	0.2	0.2	0.2	0.33	0.27
60	GD_SND	Send GD packet	295	215	175	215	--	--
		1 byte						
		32 bytes	910	640	515	640	--	--
61	GD_RCV	Receive GD package	145	105	85	105	--	--
		(1 - 32 Byte)						
62	CONTROL	Check status of the connection belonging	116	87	69	87	107	136
		to a local communication-SFB-instance						
64	TIME_TCK	Display millisecond timer	24	19	15	19	19	47
65	X_SEND	Transmit data to external partner	860 -	710 -	765 -	710 -	--	--
		First call, establish a connection	910	740	795	740		
		(1 - 76 bytes) REQ = 1						
		First call, connection present	590 -	400 -	320 -	400 -	--	--
		(1-76 bytes)	635	430	345	430		
		Intermediate call	180	130	100	130	--	--
		(1-76 bytes)						
		Last call, BUSY = 0	285	195	155	195	--	--
66	X_RCV	Receive data from external partner	92	65	55	65	--	--
		Test reception (1-76 bytes)						
		Read data	275 -	190 -	150 -	190 -	--	--
		(1-76 bytes)	315	220	175	220		

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
67	X_GET	Read data from external partner First call, establish a connection (1-76 bytes) REQ = 1	760	645	715	645	--	--
		First call, connection present (1-76 bytes)	490	335	265	335	--	--
		Intermediate call (1-76 bytes)	195	135	110	135	--	--
		Last call BUSY = 0	450 - 490	310 - 340	245 - 270	310 - 340	--	--
68	X_PUT	Write data to external partner First call, establish a connection (1-76 bytes) REQ = 1	880 - 925	725 - 755	780 - 810	725 - 755	--	--
		First call, connection present (1-76 bytes)	610 - 655	415 - 445	330 - 360	415 - 445	--	--
		Intermediate call (1-76 bytes)	195	135	110	135	--	--
		Last call, BUSY = 0	300	205	162	205	--	--
69	X_ABORT	Abort connection to external partner First call, REQ = 1	220	160	125	160	--	--
		Intermediate call	125	90	70	90	--	--
		Last call, BUSY = 0	365	375	75 - 500	375	--	--
72	I_GET	Read data from internal partner First call, establish a connection (1-76 bytes) REQ = 1	815	680	745	680	--	--
		First call, connection present (1-76 bytes)	505	345	275	345	--	--
		Intermediate call (1-76 bytes)	205	145	115	145	--	--
		Last call, BUSY = 0	460 - 505	315 - 345	250 - 275	315 - 345	--	--
73	I_PUT	Write data to internal partner First call, establish a connection (1-76 bytes) REQ = 1	690 - 980	430 - 800	340 - 840	430 - 800	--	--
		First call, connection present (1-76 bytes)	625 - 665	425 - 455	340 - 365	425 - 455	--	--
		Intermediate call (1-76 bytes)	205	145	115	145	--	--
		Last call, BUSY = 0	310	215	170	215	--	--
74	I_ABORT	Abort connection to internal partner First call, REQ = 1	225	160	125	160	--	--
		Intermediate call	125	90	75	90	--	--
		Last call, without / with connection BUSY = 0	365	380	70 / 503	380	--	--
79	SET [1]	Set bit array in I/O area n = number of bits to set at 1	43 + n * 0.39	28 + n * 0.32	23 + n * 0.26	28 + n * 0.32	53 + n * 1.35	80 + n * 1.32
80	RSET [1]	Delete bit array in I/O area n = number of bits to set at 0	43 + n * 0.39	28 + n * 0.32	23 + n * 0.26	28 + n * 0.32	53 + n * 1.35	80 + n * 1.32
81	UBLKMOV	Copy variable without interruption n = number of bytes to copy	62 + n* 0.30	44 + n* 0.17	33 + n* 0.17	44 + n* 0.17	43 + n* 0.17	42 + n* 0.17
90	H_CTRL	Influence processes involving fault-tolerant systems	--	--	--	--	19 - 21	19 - 21
100	SET_CLKS	Set time-of-day and clock status MODE = 1	370	263	227	263	439	1169
		MODE = 2	125	84	67	84	192	403
		MODE = 3	375	266	232	266	442	1167

[1] Measured with I/O modules of the type “Binary Simulator C79459-A1002-A1, Release 1” in the central rack

SFC NO.	SFC Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
105	READ_SI	Read dynamically assigned system resources MODE = 0	176 - 1807 [0]	117 - 3574 [0]	94 - 2859 [0]	117 - 3574 [0]	117 - 3205 [0]	117 - 3206 [0]
		MODE = 1	204 - 2098 [1]	138 - 4128 [1]	110 - 3302 [1]	138 - 4128 [1]	136 - 3802 [1]	303 - 3971 [1]
		MODE = 2	205 - 1478 [1]	140 - 2868 [1]	111 - 2294 [1]	140 - 2868 [1]	137 - 2901 [1]	304 - 3069 [1]
		MODE = 3	206 - 2152 [2]	140 - 4129 [2]	111 - 3303 [2]	140 - 4129 [2]	137 - 3802 [2]	305 - 3970 [2]
106	DEL_SI	Enable dynamically assigned system resources MODE = 1	176 - 1289 [1]	125 - 2666 [1]	97 - 2131 [1]	125 - 2666 [1]	145 - 6954 [1]	507 - 23875 [1]
		MODE = 2	180 - 1272 [1]	127 - 2580 [1]	99 - 2061 [1]	127 - 2580 [1]	147 - 2668 [1]	510 - 3033 [1]
		MODE = 3	177 - 1350 [2]	125 - 2705 [2]	98 - 2162 [2]	125 - 2705 [2]	145 - 6974 [2]	507 - 23906 [2]

[0] Depending on the size of the SYS_INST target area and on the number of the system resources to be read

[1] Depending on the number of active messages (assigned system resources)

[2] Depending on the number of active messages (assigned system resources) and on the number of assigned instances with the desired CMP_ID.

107	ALARM_DQ	Acknowledgeable block-related messages create first call, SIG = 0 -> 1	497	336	267	336	349	566
		Call (without message)	145	98	78	98	101	157
108	ALARM_D	Not achnowledgeable block-related messages create first call, SIG = 0 -> 1	499	337	266	337	350	548
		Call (without message)	146	98	78	98	101	156

System Function Blocks

SFB NO.	SFB Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
0	CTU	Count up	26	16	13	16	17	16
1	CTD	Count down	25	17	13	17	17	17
2	CTUD	Count up and down	29	19	15	19	19	19
3	TP	Generate pulse	34	23	18	23	24	52
4	TON	Generate on-delay	34	23	18	23	24	52
5	TOF	Generate off-delay	36	24	19	24	20	53
8	USEND	Send data without coordination	473 -	318 -	253 -	317 -	330 -	425 -
		(one send parameter supplied)	737	509	407	509	436	542
		JOB activated (1 - 440 bytes)						
		JOB checked	159	107	86	108	115	145
		JOB finished (DONE = 1)	152	103	82	104	107	137
9	URCV	Receive data without coordination	137	93	74	94	100	130
		(one receive parameter supplied)						
		JOB activated						
		JOB checked	137	93	74	94	100	130
		JOB finished	345 -	232 -	186 -	233 -	243 -	314 -
		(NDR = 1; 1 - 440 bytes)	610	421	337	421	363	435
12	BSEND	Send data block by block	386	258	207	258	264	323
		JOB activated (1 - 3000 bytes)						
		JOB checked	171	115	92	116	122	152
		JOB finished (DONE = 1)	165	110	88	111	115	145
13	BRCV	Receive data block by block	203	138	110	139	145	175
		JOB activated (1 - 3000 bytes)						
		JOB checked	161	110	88	111	117	147
		JOB finished	162	109	87	110	113	143
14	GET	Read data from remote CPU	336	227	183	228	227	297
		(one area specified)						
		JOB activated						
		JOB checked	161	109	87	110	116	146
		JOB finished (NDR = 1; 1 - 450 bytes)	344 -	231 -	185 -	232 -	243 -	314 -
			626	431	345	432	369	441
15	PUT	Write data to remote CPU	498 -	337 -	269 -	337 -	349 -	443 -
		JOB activated (1 - 404 bytes)	748	513	410	515	458	552
		JOB checked	161	108	87	109	116	146
		JOB finished (DONE = 1)	154	104	83	105	108	138
16	PRINT	Send data to a printer	513 -	338 -	271 -	339 -	354 -	449 -
		JOB activated, REQ = 1	757	516	414	518	462	545
		JOB checked	160	107	86	108	115	145
		JOB finished, DONE = 1	153	103	82	104	107	137
19	START	Start remote device	497	333	265	333	339	408
		JOB activated, REQ = 1						
		JOB checked	169	114	91	115	121	151
		JOB finished, DONE = 1	164	110	88	111	115	146
20	STOP	Stop remote device	472	314	251	314	322	384
		JOB activated, REQ = 1						
		JOB checked	169	114	91	115	121	151
		JOB finished, DONE = 1	164	110	88	111	115	146

SFB NO.	SFB Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
21	RESUME	Restart remote device JOB activated, REQ = 1	496	334	265	332	339	399
		JOB checked	169	114	91	115	121	151
		JOB finished, DONE = 1	164	110	88	111	115	145
22	STATUS	Query status of remote partner JOB activated, REQ = 1	268	183	146	184	188	258
		JOB checked	161	108	87	109	116	146
		JOB finished, NDR = 1	604	404	323	404	415	486
23	USTATUS	Receive status of remote device without coordination JOB activated, NDR = 1	137	93	74	94	100	131
		JOB checked	137	93	74	94	100	130
		JOB finished	604	404	323	404	415	486
32	DRUM	Implement sequencer	52	33	26	33	35	62
33	ALARM	Generate block-related message with acknowledgment JOB activated, SIG = 0–> 1 (1 - 420 bytes)	581 - 843	386 - 587	307 - 470	385 - 589	392 - 518	527 - 652
		JOB checked	205	136	109	137	141	171
		JOB finished, DONE = 1	207	137	110	138	136	166
34	ALARM_8	Generate block-related message without accompanying values for 8 signals JOB activated, SIG = 0–> 1 (1 - 420 bytes)	416	278	222	279	278	372
		JOB checked	203	135	108	136	140	170
		JOB finished, DONE = 1	206	137	109	138	135	166
35	ALARM_8P	Generate block-related message with accompanying values for 8 signals JOB activated, SIG = 0–> 1 (1 - 420 bytes)	580 - 842	384 - 587	308 - 469	385 - 597	392 - 517	526 - 651
		JOB checked	204	136	108	137	140	170
		JOB finished, DONE = 1	207	137	110	138	135	166
36	NOTIFY	Generate block-related message without acknowledgment JOB activated, SIG = 0–> 1 (1 - 420 bytes)	561 - 823	373 - 580	301 - 462	379 - 578	384 - 510	519 - 644
		JOB checked	186	125	100	126	133	163
		JOB finished, DONE = 1	191	128	102	129	130	160
37	AR_SEND	Send archive data JOB activated, REQ = 1 (1 - 3000 bytes)	388	258	208	258	265	328
		JOB checked	173	116	92	116	123	155
		JOB finished, DONE = 1	167	111	88	112	115	147
52	RDREC	Read data record from a DP slave via integrated DP interface, First call (2-16 bytes)	341	221	177	221	228	269
		Intermediate call	173	111	89	111	117	114
		Last call	236	157	127	157	164	161
52	RDREC	Read data record from a DP slave via external DP interface, First call (4-16 bytes)	323	211	170	211	213	210
		Intermediate call	174	112	90	112	117	114
		Last call	238	154	124	154	161	158

SFB NO.	SFB Name	Function	Execution Time in µs					
			CPU 412	CPU 414	CPU 416	CPU 417	CPU 414-H 417-H (SOLO)	CPU 414-H 417-H (Redun)
53	WRREC	Write data record in a DP slalve via integrated DP interface, First call (1-10 bytes)	354	234	187	234	241	281
		Intermediate call	170	110	88	110	116	112
		Last call	171	110	89	110	116	113
53	WRREC	Write data record in a DP slave via external DP interface, First call (2-14 bytes)	339	224	180	224	226	223
		Intermediate call	170	110	89	110	116	113
		Last call	172	111	89	111	117	113
54	RALRM	Receive interrupt from a DP slave Runtime measurement for non-I/O-dependent OBs, MODE = 1, OB 1	133	81	70	81	83	83
54	RALRM	Receive interrupt from a DP slave Runtime measurement at integrated DP interface, MODE = 1, OB 40, OB 83, OB 86	250	164	135	164	245	245
		OB 55 to OB 57, OB 82	257	171	140	171	251	251
		OB 70	--	--	--	--	242	242
54	RALRM	Receive interrupt from a DP slave Runtime measurement at external DP interface, MODE = 1, OB 40, OB 83, OB 86	429	290	234	290	458	458
		OB 55 to OB 57, OB 82	704	499	413	499	747	747
		OB 70	--	--	--	--	460	460
54	RALRM	Receive interrupt from a DP slave Runtime measurement at central I/O, MODE = 1, OB 40, OB 82, OB 83, OB 86	215	138	111	138	143	143
		OB 55 to OB 57	619	472	414	472	567	567

ضمیمه ۵

لیست بلاک های سیستم

Sysytem Blocks

List of SFCs

No.	Short Name	Function
SFC 0	SET_CLK	Set System Clock
SFC 1	READ_CLK	Read System Clock
SFC 2	SET_RTM	Set Run-time Meter
SFC 3	CTRL_RTM	Start/Stop Run-time Meter
SFC 4	READ_RTM	Read Run-time Meter
SFC 5	GADR_LGC	Query Logical Address of a Channel
SFC 6	RD_SINFO	Read OB Start Information
SFC 7	DP_PRAL	Trigger a Hardware Interrupt on the DP Master
SFC 9	EN_MSG	Enable Block-Related, Symbol-Related and Group Status Messages
SFC 10	DIS_MSG	Disable Block-Related, Symbol-Related and Group Status Messages
SFC 11	DPSYC_FR	Synchronize Groups of DP Slaves
SFC 12	D_ACT_DP	Deactivation and activation of DP slaves
SFC 13	DPNRM_DG	Read Diagnostic Data of a DP Slave (Slave Diagnostics)
SFC 14	DPRD_DAT	Read Consistent Data of a Standard DP Slave
SFC 15	DPWR_DAT	Write Consistent Data to a DP Standard Slave
SFC 17	ALARM_SQ	Generate Acknowledgeable Block-Related Messages
SFC 18	ALARM_S	Generate Permanently Acknowledged Block-Related Messages
SFC 19	ALARM_SC	Query the Acknowledgment Status of the last ALARM_SQ Entering State Message
SFC 20	BLKMOV	Copy Variables
SFC 21	FILL	Initialize a Memory Area
SFC 22	CREAT_DB	Create Data Block
SFC 23	DEL_DB	Delete Data Block
SFC 24	TEST_DB	Test Data Block
SFC 25	COMPRESS	Compress the User Memory
SFC 26	UPDAT_PI	Update the Process Image Update Table
SFC 27	UPDAT_PO	Update the Process Image Output Table
SFC 28	SET_TINT	Set Time-of-Day Interrupt
SFC 29	CAN_TINT	Cancel Time-of-Day Interrupt
SFC 30	ACT_TINT	Activate Time-of-Day Interrupt
SFC 31	QRY_TINT	Query Time-of-Day Interrupt
SFC 32	SRT_DINT	Start Time-Delay Interrupt
SFC 33	CAN_DINT	Cancel Time-Delay Interrupt
SFC 34	QRY_DINT	Query Time-Delay Interrupt
SFC 35	MP_ALM	Trigger Multicomputing Interrupt
SFC 36	MSK_FLT	Mask Synchronous Errors
SFC 37	DMSK_FLT	Unmask Synchronous Errors
SFC 38	READ_ERR	Read Error Register
SFC 39	DIS_IRT	Disable New Interrupts and Asynchronous Errors
SFC 40	EN_IRT	Enable New Interrupts and Asynchronous Errors
SFC 41	DIS_AIRT	Delay Higher Priority Interrupts and Asynchronous Errors
SFC 42	EN_AIRT	Enable Higher Priority Interrupts and Asynchronous Errors
SFC 43	RE_TRIGR	Re-trigger Cycle Time Monitoring

No.	Short Name	Function
SFC 44	REPL_VAL	Transfer Substitute Value to Accumulator 1
SFC 46	STP	Change the CPU to STOP
SFC 47	WAIT	Delay Execution of the User Program
SFC 48	SNC_RTCB	Synchronize Slave Clocks
SFC 49	LGC_GADR	Query the Module Slot Belonging to a Logical Address
SFC 50	RD_LGADR	Query all Logical Addresses of a Module
SFC 51	RDSYSST	Read a System Status List or Partial List
SFC 52	WR_USMSG	Write a User-Defined Diagnostic Event to the Diagnostic Buffer
SFC 54	RD_PARM	Read Defined Parameters
SFC 55	WR_PARM	Write Dynamic Parameters
SFC 56	WR_DPARM	Write Default Parameters
SFC 57	PARM_MOD	Assign Parameters to a Module
SFC 58	WR_REC	Write a Data Record
SFC 59	RD_REC	Read a Data Record
SFC 60	GD_SND	Send a GD Packet
SFC 61	GD_RCV	Fetch a Received GD Packet
SFC 62	CONTROL	Query the Status of a Connection Belonging to a Communication SFB Instance
SFC 63 *	AB_CALL	Assembly Code Block
SFC 64	TIME_TCK	Read the System Time
SFC 65	X_SEND	Send Data to a Communication Partner outside the Local S7 Station
SFC 66	X_RCV	Receive Data from a Communication Partner outside the Local S7 Station
SFC 67	X_GET	Read Data from a Communication Partner outside the Local S7 Station
SFC 68	X_PUT	Write Data to a Communication Partner outside the Local S7 Station
SFC 69	X_ABORT	Abort an Existing Connection to a Communication Partner outside the Local S7 Station
SFC 72	I_GET	Read Data from a Communication Partner within the Local S7 Station
SFC 73	I_PUT	Write Data to a Communication Partner within the Local S7 Station
SFC 74	I_ABORT	Abort an Existing Connection to a Communication Partner within the Local S7 Station
SFC 78	OB_RT	Determine OB program runtime
SFC 79	SET	Set a Range of Outputs
SFC 80	RSET	Reset a Range of Outputs
SFC 81	UBLKMOV	Uninterruptible Block Move
SFC 82	CREA_DBL	Generating a Data Block in the Load Memory
SFC 83	READ_DBL	Reading from a Data Block in Load Memory
SFC 84	WRIT_DBL	Writing from a Data Block in Load Memory
SFC 87	C_DIAG	Diagnosis of the Actual Connection Status
SFC 90	H_CTRL	Control Operation in H Systems
SFC 100	SET_CLKS	Setting the Time-of-Day and the TOD Status
SFC 101	RTM	Handling runtime meters
SFC 102	RD_DPARA	Redefined Parameters
SFC 103	DP_TOPOL	Identifying the bus topology in a DP master system
SFC 104	CiR	Controlling CiR
SFC 105	READ_SI	Reading Dynamic System Resources
SFC 106	DEL_SI	Deleting Dynamic System Resources
SFC 107	ALARM_DQ	Generating Always Acknowledgeable and Block-Related Messages
SFC 108	ALARM_D	Generating Always Acknowledgeable and Block-Related Messages
SFC 126	SYNC_PI	Update process image partition input table in synchronous cycle
SFC 127	SYNC_PO	Update process image partition output table in synchronous cycle
* SFC 63 "AB_CALL" only exists for CPU 614. For a detailed description, refer to the corresponding Manual		

List of SFBs

No.	Short Name	Function
SFB 0	CTU	Count Up
SFB 1	CTD	Count Down
SFB 2	CTUD	Count Up/Down
SFB 3	TP	Generate a Pulse
SFB 4	TON	Generate an On Delay
SFB 5	TOF	Generate an Off Delay
SFB 8	USEND	Uncoordinated Sending of Data
SFB 9	URCV	Uncoordinated Receiving of Data
SFB 12	BSEND	Sending Segmented Data
SFB 13	BRCV	Receiving Segmented Data
SFB 14	GET	Read Data from a Remote CPU
SFB 15	PUT	Write Data to a Remote CPU
SFB 16	PRINT	Send Data to Printer
SFB 19	START	Initiate a Warm or Cold Restart on a Remote Device
SFB 20	STOP	Changing a Remote Device to the STOP State
SFB 21	RESUME	Initiate a Hot Restart on a Remote Device
SFB 22	STATUS	Query the Status of a Remote Partner
SFB 23	USTATUS	Receive the Status of a Remote Device
SFB 29 *	HS_COUNT	Counter (high-speed counter, integrated function)
SFB 30 *	FREQ_MES	Frequency Meter (frequency meter, integrated function
SFB 31	NOTIFY_8P	Generating block related messages without acknowledgement indication
SFB 32	DRUM	Implement a Sequencer
SFB 33	ALARM	Generate Block-Related Messages with Acknowledgment Display
SFB 34	ALARM_8	Generate Block-Related Messages without Values for 8 Signals
SFB 35	ALARM_8P	Generate Block-Related Messages with Values for 8 Signals
SFB 36	NOTIFY	Generate Block-Related Messages without Acknowledgment Display
SFB 37	AR_SEND	Send Archive Data
SFB 38 *	HSC_A_B	Counter A/B (integrated function)
SFB 39 *	POS	Position (integrated function)
SFB 41	CONT_C [1)]	Continuous Control
SFB 42	CONT_S [1)]	Step Control
SFB 43	PULSEGEN [1)]	Pulse Generation
SFB 44	ANALOG [2)]	Positioning with Analog Output
SFB 46	DIGITAL [2)]	Positioning with Digital Output
SFB 47	COUNT [2)]	Controlling the Counter
SFB 48	FREQUENC [2)]	Controlling the Frequency Measurement
SFB 49	PULSE [2)]	Controlling Pulse Width Modulation
SFB 52	RDREC	Reading a Data Record from a DP Slave
SFB 53	WRREC	Writing a Data Record in a DP Slave
SFB 54	RALRM	Receiving an Interrupt from a DP Slave
SFB 60	SEND_PTP [2)]	Sending Data (ASCII, 3964(R))
SFB 61	RCV_PTP [2)]	Receiving Data (ASCII, 3964(R))
SFB 62	RES_RECV [2)]	Deleting the Receive Buffer (ASCII, 3964(R))
SFB 63	SEND_RK [2)]	Sending Data (RK 512)
SFB 64	FETCH_RK [2)]	Fetching Data (RK 512)
SFB 65	SERVE_RK [2)]	Receiving and Providing Data (RK 512)
SFB 75	SALRM	Send interrupt to DP master

SFB 29 "HS_COUNT" and SFB 30 "FREQ_MES" only exist on the CPU 312 IFM and CPU 314 IFM. SFBs 38 "HSC_A_B" and 39 "POS" only exist on the CPU 314 IFM

1) SFBs 41 "CONT_C," 42 "CONT_S" and 43 "PULSEGEN" only exist on the CPU 314 IFM

2) SFBs 44 to 49 and 60 to 65 only exist on the S7-300C CPUs

ضمیمه ۶

مقایسه دستورات برنامه نویسی و فرمت دیتا ها
بین S7 و S5

مقایسه فرمت دیتاها در S7 و S5

Data Class	Data Types in S5	Data Types in S7
Elementary data types	BOOL, BYTE, WORD, DWORD, Integer, Double integer, Floating point, Time value, - ASCII character	BOOL, BYTE, WORD, DWORD, INT, DINT, REAL, S5TIME, TIME, DATE; TIME_OF_DAY, CHAR
Complex data types	-	DATE_AND_TIME, STRING, ARRAY, STRUCT
Parameter types	Timers, Counters, Blocks – –	TIMER, COUNTER, BLOCK_FC, BLOCK_FB, BLOCK_DB, BLOCK_SDB, POINTER, ANY

مقایسه فرمت ثوابت در S7 و S5

Formats in S5	Example	Formats in S7	Example
KB	L KB 10	K8	L B#16# A
KF	L KF 10	K16	L 10
KH	L KH FFFF	16#	L 16# FFFF
KM	L KM 1111111111111111	2#	L 2# 11111111_11111111
KY	L KY 10,12	B#	L B# (10,12)
KT	L KT 10.0	S5TIME# (S5T#)	L S5TIME# 100ms
KC	L KC 30	C#	L C#30
DH	L DH FFFF FFFF	16#	L DW#16# FFFF_FFFF
KS	L KS WW	‘xx’	L ’ WW ’
KG	L KG +234 +09	Floating Point	L +2.34 E+08

مقایسه OB های S5 و S7

Function		S5	S7
Main program	Free cycle	OB1	OB1
Interrupts	Time-delay (delayed) interrupt	OB6	OB20 to OB23
	Time-of-day (clock-controlled) interrupt	OB9	OB10 to OB17
	Hardware interrupts	OB2 to OB5	OB40 to OB47
	Process interrupts	OB2 to OB9	Replaced by hardware interrupts
	Cyclic (timed) interrupts	OB10 to OB18	OB30 to OB38
	Multicomputing interrupt	-	OB60
Startup	Manual complete (cold) restart OB100	OB21 (S5-115U) OB20 (S5-135U)	OB100
	Manual (warm) restart	OB21 (from S5-135U)	OB101
	Automatic (warm) restart	OB22	OB101
Errors	Error	OB19 to OB35	OB121, OB122, OB80 to OB87
Other	Processing in STOP mode	OB39	Omitted
	Background processing	-	OB90

مقایسه دستورات S7 و S5

Instruction Type	S5	S7
Accumulator instructions	TAK, ENT, I, D, ADDBN, ADDKF, ADDDH	TAK, ENT, INC, DEC, +, New in S7: CAW, CAD, PUSH,POP, LEAVE
Address register instructions / Register instructions	MA1, MBR, ABR,MAS, MAB, MSB,MSA, MBA, MBS; TSG, LRB, LRW,LRD, TRB, TRW,TRD	New in S7: LAR1, LAR2, TAR1, TAR2, +AR1, +AR2, CAR
Bit logic instructions	A, AN, O, ON, A(, O(,), O, S, R, RB,RD= TB, TBN, SU, RU	A, AN, O, ON, A(, O(,), O, S, R, = SET; A, SET; AN,SET; S, SET; R New in S7: X, XN, X(, XN(,FP, FN, NOT, SET,CLR, SAVE
Timer instructions	SP, SE, SD,SS/SSU, SF/SFD, FR, SEC	SP, SE, SD, SS, SF, FR, S T
Counter instructions	CU/SSU, CD/SFD,FR, SEC	CU, CD, FR, S C
Load and transfer instructions	L, LD, LW, LDW, TL PB,L QB L PW, L QW, T PB, T QB, T PW, T QW	L, LC, T L PIB, L PIW, T PQB, T PQW
	LY GB / GW / GD /CB / CW / CD,LW GW / GD / CW / CD,TY GB / GW / GD / CB / CW / CD,TW GW / GD / CW /CD	-
Integer math instructions	+F, –F, xF, :F, +D,–D	+I, –I, *I, /I, +D, –D, *D, /D New in S7: MOD
Floating-point math instructions	+G, –G, xG, :G	+R, –R, *R, /R
Comparison instructions	!=F, ><F, >F, <F, >=F, <=F, !=D, ><D, D, <D, >=D, <=D, !=G, ><G, >G, <G, >=G, <=G	==I, <>I, >I, <I; >=I, <=I, ==D, <>D, >D, <D, >=D, <=D, ==R, <>R, >R, <R, >=R, <=R
Conversion instructions	CFW, CSW, CSD,DEF, DED, DUF, DUD, GFD, FDG	INVI, NEGI, NEGD, BTI, BTD, DTB, ITB, RND, DTR New in S7: ITD, RND+, RND–, TRUNC, INVD, NEGR

ادامه مقایسه دستورات S7 و S5

Instruction Type	S5	S7
Word logic instructions	AW, OW, XOW	AW, OW, XOW New in S7: AD, OD, XOD
Shift and rotate instructions	SLW, SLD, SRW, SRD, SVW, SVD, RLD, RRD	SLW, SLD, SRW, SRD, SSI, SSD, RLD, RRD New in S7: RLDA, RRDA
Data block instructions	G, CX	OPN
	G, GX	SFC22
		New in S7: CDB L DBLG, L DBNO,L DILG, L DINO
Logic control instructions, jump	JU, JC, JN, JZ, JP, JM, JO, JOS, JUR	JU, JC, JN, JZ, JP, JM, JO, JOS New in S7: JCN, JCB, JNB,JBI, JNBI, JMZ, JPZ, JUO, LOOP, JL
Block instructions	JU, JC, DOU, DOC, BE, BEU,BEC	CALL, BE, BEU, BEC
Command output instructions/ Master control relay instructions	BAS, BAF	New in S7: MCRA, MCRD, MCR(,)MCR
Stop commands	STP, STS, STW	SFC46
Processing functions	DO <Formal parameter>	-
	DO FW, DO DW	Memory-indirect addressing
	DO RS	Area-crossing register-indirect addressing
Absolute memory addressing	LIR, TIR, LDI, TDI	–
Block transfers	TNB, TNW, TXB, TXW	SFC20
Interrupt commands	LIM, SIM, IAE, RAE, IA, RA	SFC39 to 42
Page commands	ACR, TSC, TSG	-
Math functions	-	ABS, COS, SIN, TAN, ACOS, ASIN, ATAN, EXP, LN
Null instructions	BLD xxx NOP 0, NOP 1	BLD xxx NOP 0, NOP 1

فهرست برخی منابع و مراجع استفاده شده

- ***Working with Step7*** — ***Siemens***
- ***Programming with Step7*** — ***Siemens***
- ***Configuring Hardware with Step7*** — ***Siemens***
- ***From S5 to S7*** — ***Siemens***
- ***Statement List For S7-300,S7-400*** — ***Siemens***
- ***Ladder Logic for S7-300 and S7-400*** — ***Siemens***
- ***Function Block Diagram for S7-300 and S7-400*** — ***Siemens***
- ***System and Standard Functions for S7-300 , S7-400*** — ***Siemens***
- ***S7-300 Hardware and Installation*** — ***Siemens***
- ***S7-300 Module Specification*** — ***Siemens***
- ***S7-400 Hardware and Installation*** — ***Siemens***
- ***S7-400 Module Specification*** — ***Siemens***
- ***S7-400 Instruction List*** — ***Siemens***
- ***Simatic S7 Supplement to Manual*** — ***Siemens***
- ***S7-300 Quick Start*** — ***Siemens***
- ***FAQ*** — ***www.ad.siemens.de***
- ***PLC History*** — ***www.plcs.net***
- ***General Introduction into IEC 61131 all parts*** — ***www.plcopen.org***
- ***خودکاری با PLC – سید حجت سبز پوشان***
- ***برنامه نویسی و کار با Step7– محمدرضا ماهر***