به نام خدا

جزوه درسی آموزش PLC

زبان برنامه نویسی LD

مطابق با استاندارد :

IEC6113-3

استاد:

فتح الله نظریان

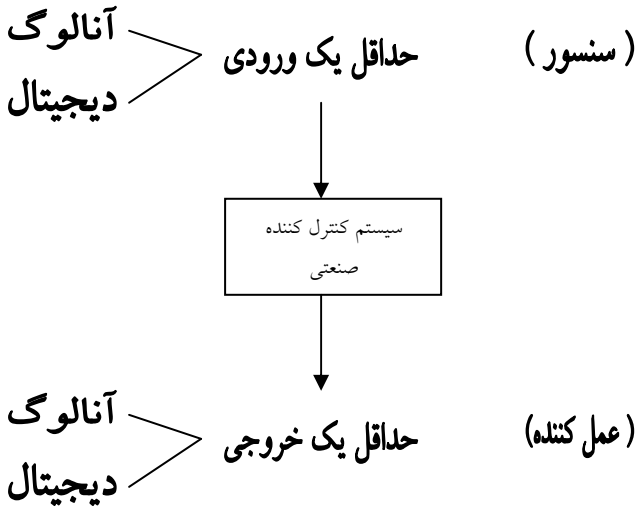
تنظیم :

محمد نحوی

نیمسال دوم ۸۴-۸۵

کنترل کننده های صنعتی :

شمای کلی PLC ها :



در صنعت در عمل هر کمیت فیزیکی که اندازه گیری می شود تبدیل بـه یـک سـیگنال الکتریکـی متناسب با آن کمیت فیزیکی می شود .

به دلایل زیر :

۱- انتقال آن آسان است (با سیم یا بدون سیم)

۲- اندازه گیری آن آسان است .

۳- تبدیل آن به انرژی های دیگر ساده است

۴- امکان ذخیره اطلاعات به صورت یک File کامپیوتری اسان است

استاندارد سیگنال های صنعتی :

** 4—20mA
* 1—5V

در صنعت معمول ترند

0 – 20mA
0 – 5V
1 – 10V
-10 – 10V

0 – 10V

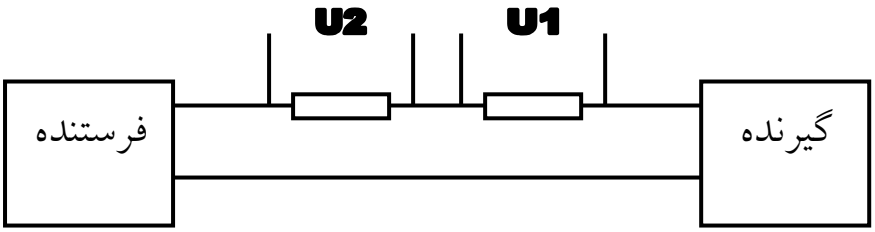
توضیحاتی در مورد قرار داد 20m – 4mA **:**

۱- علت اینکه حد پایین این جریان 4mA است این است که اگر سیم قطع شود جریان صفر می شود و اگر جریان از 0mA باشد نمی توان تشخیص داد که این مقدار صفر ، داده است یا به خاطر قطع سیگنال ایجاد شده است . علت اینکه این مقدار از 4mA شروع می شود و نه از 2mA این است که حداقل جریان مورد نیاز برای بایاس مدارات می باشد .

۲- علت اینکه این جریان در حد میلی آمپر است نه مثلا میکرو آمپر به این خاطر است که در مقابل نویز مقاوم باشد .

۳- علت اینکه از جریان استفاده شده است نه ولتاژ :

- چون اثر نویز بر روی جریان کمتر است
- امپدانس خطوط تاثیر ندارد
- می توان انشعاب های مختلفی از آن گرفت (چون مقاومت تاثیری ندارد)



نکته :

در صنعت به جز سیگنالهای الکتریکی از فشار هوا هم استفاده می شود .

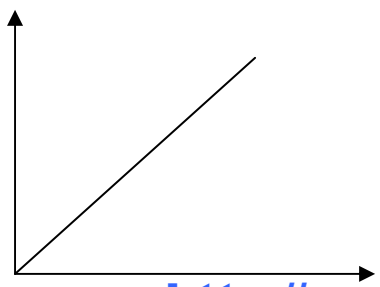
ترانس میتر :

ترانس میتر وسیله ای است که ولتاژ را تبدیل به جریان می کند .

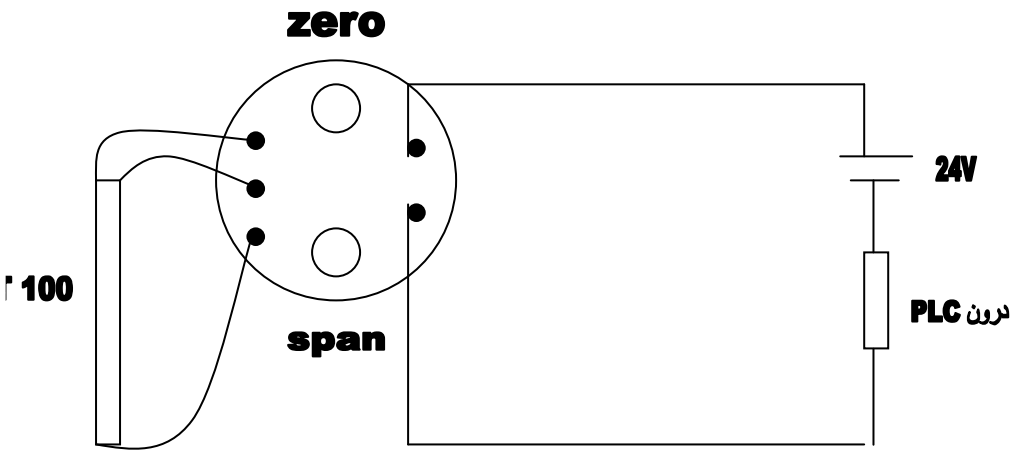
نکته :

معروف ترین سنسور دمایی pt xxx هست . به عنوان مثلا pt 100 که ۱۰۰ به معنی مقدار مقاومت این فلز در دمای صفر درجه سانتی گراد است .

پلاتین فلزی است که تغییرات مقاومت آن نسبت به دما کاملا خطی است :



شکل ظاهری ترانس میتر :



نکات :

۲۴ در واقع همان تغذیه ترانس میتر هست .

برای تعیین گام حرکت بین 4m و 20m باید ابتدا کمترین مقاومتی که در مدار اتفاق می افتد (کمترین دما) را اندازه گرفت ، سپس یک مقاومت برابر مقاومت بدست آمده به جای pt 100 گذاشت بعد باید آنقدر zero را تغییر داد تا جریان به 4m برسد .

به همین ترتیب بیشترین مقاومت را حساب کرده و بعد مقاومتی را به جای pt 100 می گذاریم . آنقدر span را تغییر می دهیم تا جریان به 20m برسد . این کار را باید چند بار تکرار کرد . چون با تغییر span ، تنظیم zero بهم میریزد .

دستگاهی وجود دارد که می تواند pt 100 ، ترموکوپل و ... را شبیه سازی می کند . یعنی خروجی مقاومت دلخواه ما را می دهد .

فرمول بدست آوردن مقاومت pt 100 :

$$R(t) = 100 \times \left(1 + 3.85 \times 10^{-3}\left(x\right)\right)$$

X مقدار دمای محیط می باشد و R مقدار مقاومت بدست آمده به ازای دمای x است .

تفاوت رله های قابل برنامه ریزی با PLC :

مزایای این رله ها :

قیمت ارزان – سبک – کوچک هستند که برای کاربرد های محدود مانند پله برقی ، راه اندازی یک یا چند موتور و موارد مشابه .

در ضمن اکثر در اکثر رله های قابل برنامه ریزی در جلوی پانل آنها یک display کوچک وجود دارد که می توان از طریق آن برنامه مورد نظر را در آن نوشت . (یک صفحه کلید کوچک نیز دارند).

محدودیت های این رله ها :

۱- تعداد ورودی ها و خروجی ها محدود است (بین ۲۰ تا ۳۰)

۲- تعداد تایمر ها و شمارنده های داخل آن محدود است .

۳- تعداد سطرهای برنامه نویسی آن محدود است

۴- تعدا قطعات فرا خوانده شده از حافظه محدود است .

برنامه نویسی PLC به زبان LD (ladder Diagram) طبق استاندارد (IEC 61131-3):

خود PLC نام یک دستگاه است که به علت تکرار زیاد خودش به عنوان یک نام (مثلا میکـرو) جـا افتاده است .

PLC مخفف (programmable logic controller) است .

انواع PLC ها :

نام شرکت سازنده	نام محصول	نام رله های قابل برنامه ریزی این شرکت
Siemens	Simatic	Logo
Omrom	Sysmac	Zen
LG	Glofa	
Allen bradly	PLC	

PLC در واقع یک کامپیوتر صنعتی است .

زبانهای برنامه نویسی در استاندارد IEC 61131-3 :

۱- LD (ladder diagram) ، مناسب برای کسانی که با دیاگرام رله ای آشنایی دارند .

۲- FBD (function black diagram) ، // با دیاگرام مدار منطقی آشنایی دارند .

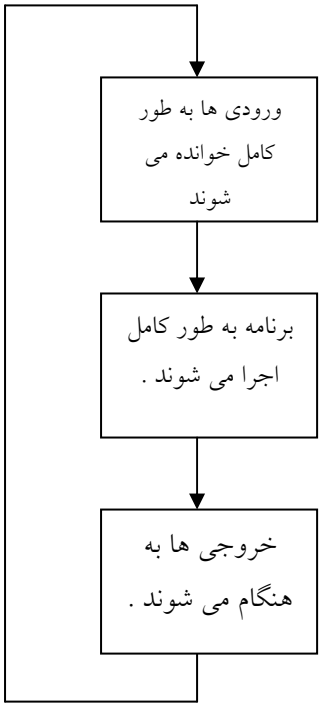
۳- SFC (sequential fanctial chart) ، // با ساختار های الگوریتمی آشنایی دارند .

۴- IL (instruction list) ، // با زبان اسمبلی آشنایی دارند .

۵- ST (structured text) ، // با زبان های سطح بالا آشنایی دارند .

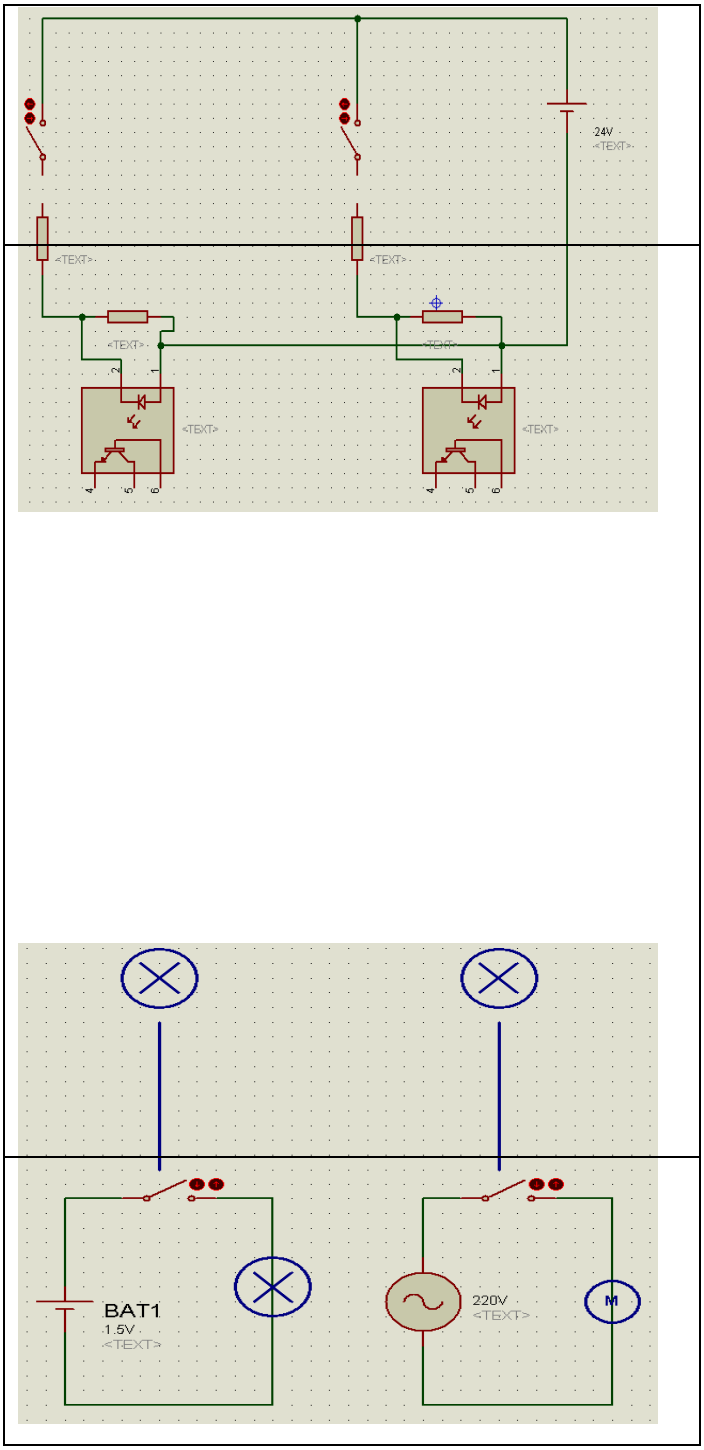
عملکرد PLC :

ساختار کل اجرای برنامه های PLC :



این ساختار به این معنی است که در هنگام اجرای برنامه ، ورودی ها خوانده نمی شوند.

بررسی ساختار ورودی های PLC :



نکته :

در بعضی از PLC ها منبع ۲۴ ولت در داخل PLC است .

نکته :

رله های خروجی انواع PLC ها ۲ آمپر و ۲۵۰ ولت را تحمل مـی کنـد . در عمـل معمـولا مـستقیما از خروجی رله های PLC استفاده نمی شود .

معمولا از رک رله استفاده می شود که رله ها به صورت سوکتی هستند و به راحتی خارج می شـوند و از رله های قوی تر استفاده می شوند .

گاهی خروجی ها به صورت ترانزیستوری هستند . علت استفاده از خروجی های ترانزیستوری این است که این خروجی ها سرعت بالاتری نسبت به خروجی های رله ای دارند .

از معایب این خروجی ها این است که ترانزیستور باید بایاس شود و پلاریته ولتاژ اهمیت دارد .

معمولا برای درایو کردن Stepper موتورها از این خروجی های استفاده می شود .

آدرس های ورودی خروجی در PLC ها : (در استاندارد IEC 61131-3)

		0 63		0 15	
PS	CPU	I	I	O	I
			0 31		0 15

نکات :

% به معنی سیگنال ورودی یا خروجی (از بیرون از PLC) است .

I% به معنی خواندن بیت های ارتباط با خارج است .

O% به معنی نوشتن در بیت های ارتباط با خارج است .

استاندارد تعداد بیت های ورودی برای یک کارت 64 , 32 , 16 , 8

استاندارد تعداد بیت های خروجی برای یک کارت : 64 , 32 , 16 , 8

توضیح دستورات ورودی و خروجی :

% I ☐ ☐ ☐ ☐

مربع اول می تواند شامل

X : به معنی آدرس دهی بیتی .

B : به معنی آدرس دهی بایتی .

W : به معنی آدرس دهی کلمه ای (۲ بایتی) .

M : به معنی دست یابی به ورودی خروجی های مخصوص مونیتورنگ است .

در مربع دوم یک عدد قرار میگیرد . در یک شبکه PLC ها هر کدام از PLC ها شماره گذاری می شوند . به عنوان مثلا base0 یا Base1 . در این مربع شماره PLC مورد نظر نوشته می شود

مربع سوم : شماره اسلاتی که بیت یا بایت مورد نظر ما در آن قرار دارد را مشخص میکند .

مربع چهارم : شماره ورودی یا خروجی را مشخص می کند .

نکته :

در مربع هایی که اعداد در آنها قرار میگیرد فقط می توان از ۰ تا ۹۹ رو نوشت .

مثال :

%*Ix*0.1.5

این دستور به این معنی است که در PLC صفر (Base 0) از اسلات اول بیت ۵ را بخوان .

مثال :

%*QB*2.3.0

این دستور به این معنی است که از Base 2 اسلات سوم از بیت ۰ تا ۷ (یک بایت) بخوان .

نکته بسیار مهم :

دستورات ورودی و خروجی زمانی که در برنامه نوشته می شوند . به این معنی نیست که در همان لحظه از ورودی و خروجی خوانده می شوند ، بلکه این دستورات در واقع نام متغییر در RAM هستند .

همانطور که در ابتدا جزوه این نکته تذکر داده شد که روند اجرای برنامه ها در PLC با این صورت است که ابتدا تمام ورودی ها خوانده می شوند و سپس برنامه کامل اجرا می شود و در پایان خروجی ها بروز می شوند و این روند ادامه می یابد . به همین دلیل ابتدا تمامی ورودی ها خونده می شوند و در متغییر هایی دقیقا با همان نام سخت افزاری ذخیره می شوند (مثلا %*Ix*0.1.5) . زمانی که شما دستور %*Ix*0.1.5 را می نویسید این بیت در عمل از حافظه RAM خوانده می شود .

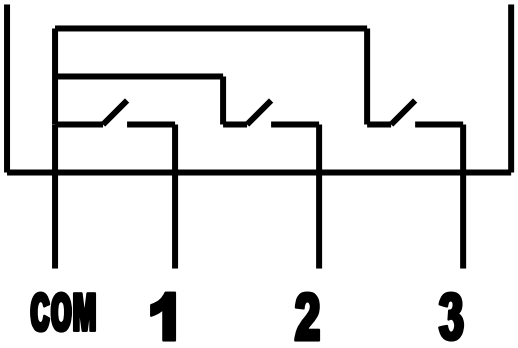
به همین ترتیب زمانی که در برنامه از دستور %*QB*2.3.0 استفاده میکنیم ، در واقع این دستور یک بایت را در آدرسی از حافظه RAM با نام %*QB*2.3.0 قرار می دهد . و در پایان اجرای برنامه تمامی خروجی ها بروز می شوند .

نکته :

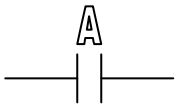
علت اینکه در همان لحظه که از دستورات % استفاده می کنیم ورودی و خروجی ها را چک نمیکند این است که برای هر بار خواندن ورودی ها باید برنامه را رها کند و به ورودی ها و خروجی ها رسیدگی کند که این کار وقت برنامه را تلف می کند . و ممکن است وسط برنامه مقدار ورودی تغییر کند بنابراین نصف برنامه برحسب ورودی مثلا ۰ اجرا شده است و نصف دیگر آن برحسب ورودی ۱ اجرا می شود که این یک حالت نا خواسته است .

نکته :

در عمل به منظور صرفه جویی در تعداد پایه ها از روش زیر استفاده می شود .

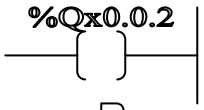


تعریف چند سمبل زبان LD :

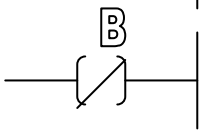


این دستور محتویات حافظه A را می خواند .

محتوایات حافظه %Ix0.0.0 را می خواند و سپس آن را NOT می کند .

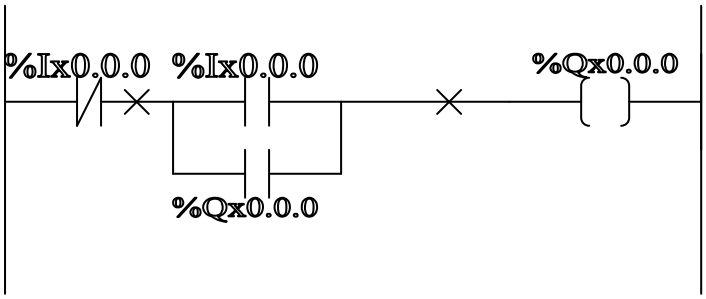


نتیجه عمل منطقی سمت چپ را در متغییر %Qx0.0.2 ذخیره کن .



نتیجه عمل منطقی سمت چپ را ابتدا NOT کن و سپس در متغیر B قرار بده.

یک مثال از برنامه LD :



این برنامه ورودی %*Ix*0.0.0 را می خواند آن را NOT می کند . این عبارت با عبارت ورودی %*Ix*0.0.0 یا خروجی %*Qx*0.0.0 AND شده است . حاصل منطقی آن را در %*Qx*0.0.0 قرار داده است .

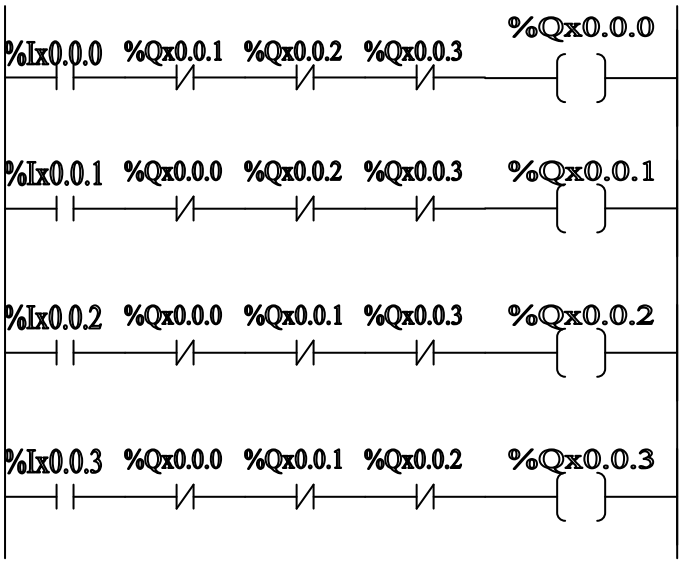
نکته :

تمامی ورودی ها ابتدای برنامه خوانده شده و نام هایی که اینجا به کـار رفتـه در واقـع متغییـر هـایی در RAM می باشند .

مطلب دیگر اینکه تمام خروجی ها در آخر برنامه بروز می شوند و نام های بکار رفته در اینجا نام متغییـر هایی در RAM می باشد .

مثال :

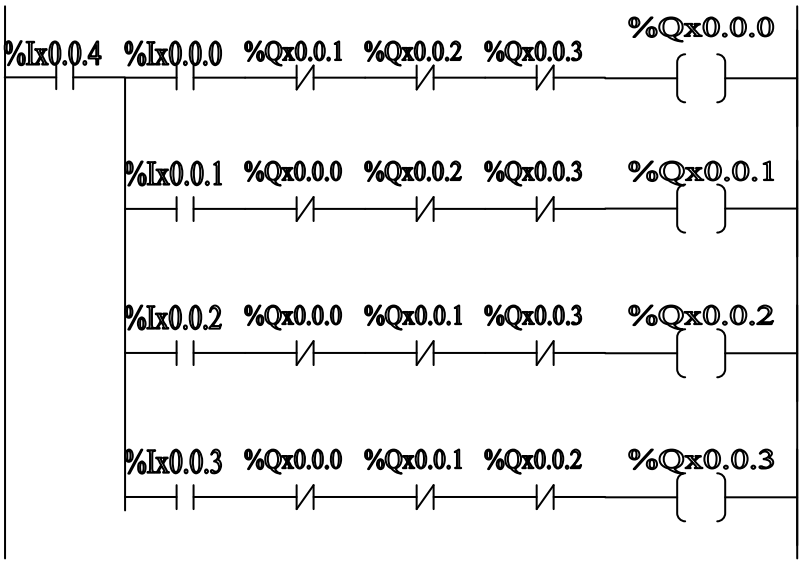
برنامه ای بنویسید که در روی یک میز مسابقه که ۴ نفر شرکت کننده دارد هر کـسی کـه کلیـد را زودتـر فشار داد چراغ او فقط روشن شود و چراغ دیگران خاموش بماند .



در خط اول این برنامه ورودی کلید اول را می خواند اگر ۱ باشد و بقیه چراغ ها روشن نشده باشد چـرا نفر اول را روشن می کند .

مثال :

همان برنامه بالا فقط با در نظر گرفتن این نکته که هر موقع که مجری برنامه اجازه داد چراغ هـا روشـن شوند .



مثال :

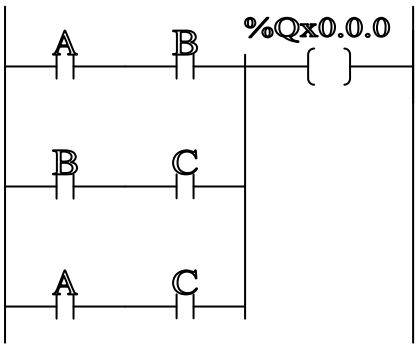
یک میز رای گیری ۳ نفره طراحی کنید .

A	B	C	F
۰	۰	۰	۰
۰	۰	۱	۰
۰	۱	۰	۰
۰	۱	۱	۱
۱	۰	۰	۰
۱	۰	۱	۱
۱	۱	۰	۱
۱	۱	۱	۱

۰	۰	۱	۰
۰	۱	۱	۱

F=AB+BC+AC

در نتیجه برنامه به صورت زیر می باشد .



البته به جای A,B,C شماره بیت ورودی نوشته می شود .

نکته : SSR (salid state rlay)

این قطعه به جای کنتاکتور های پر سروصدا استفاده می شود . این کلید ها تا ۱۰۰۰ آمپر را نیز عبور می دهند . البته مقدار آمپری که روی این SSR ها نوشته شده است برای بار های غیر اهمی هست . اگر بار سلفی خازنی داریم باید جریان کمتری ، از حداکثر جریان نوشته شده روی آن ، کشیده شود .

ساختمان داده ها در PLC :

علت استفاده از این ساختار برای صرفه جویی در حافظه PLC است . به این معنی که با این تکنیک می توان متغییر هایی با تعداد بیت مورد نیاز تعریف کرد .

۱- داده از نوع short integer نحوه نوشتن دستور : SINT

یک بایت فضا می گیرد اعداد + و – را پوشش میدهد . بیت آخر بیت علامت است . اگر بیت آخر ۱ باشد یعنی عدد منفی است .

رنج تغییرات :

از ۱۲۸- تا ۱۲۷+ .

نکته :

اعداد منفی به صورت متمم ۲ ذخیره می شوند . به عنوان مثال :

-5 ➔ 00000101
11111010
+
1

11111011

در عمل این عدد ذخیره می شود برای خواند عدد نیز آن را متمم ۲ ، + ۱ می کنند.

۱	۰	۰	۰	۰	۰	۰	۰
۱	۱	۱	۱	۱	۱	۱	۱

-128

-1

البته اعداد نمایش داده شده در بالا به صورت متمم ۲ می باشد .

۲- داده از نوع unsigned short integer (USINT)

یک بایت است و فقط هم مثبت

رنج تغییرات :

از ۰ تا ۲۵۵

چند نکته :

هر عملی چه منطقی باشد چه محاسباتی فقط بر روی متغییر های هم نوع انجام می شود .

حتی عددی که به صورت بیتی از ورودی خوانده می شود عدد نمی باشد . بلکه از نوع Byte است (بعدا آن را می خوانیم) . در این نوع متغییر بیت ها ارزش مکانی ندارند .

۳- داده از نوع integer (INT)

این نوع داده ۲ بایت از فضای حافظه را به متغییر اختصاص می دهد . این متغییر دارای علامت است .

رنج تغییرات :

از ۳۲۷۶۸- تا ۳۲۷۶۷ .

۴- داده از نوع unsigned integer (UINT)

۲ بایت بی علامت است .

رنج تغییرات :

از ۰ تا ۶۵۵۳۵

۵- داده از نوع double integer (DINT)

این داده ۴ بایت با علامت دارد .

رنج تغییرات :

از -2^{31} تا $2^{31}-1$.

۶- داده از نوع unsigned double integer (UDINT)

بدون علامت و ۴ بایت .

رنج تغییرات :

از ۰ تا $2^{32}-1$

۷- داده long integer (LINT) (البته همه PLC ها از این نوع متغییر حمایت نمی کنند)

این داده ۸ بایت با علامت دارد .

رنج تغییرات :

از -2^{63} تا $2^{63}-1$.

۸- داده unsigned long integer (ULINT)

۸ بیت دارد و بدون علامت .

رنج تغییرات :

از ۰ تا $2^{64}-1$

۹- داده از نوع TIME :

متغییر که داده آن از نوع Time لحاظ شده است می تواند زمان را ذخیره کند .

در LG ۴ بایت به آن اختصاص داده شده است و در زیمنس ۲ بایت .

در واقع یک شمارنده وجود دارد که با یک پاس 1ms می شمارد .

در نهایت به فرض ۴ بایت بودن تا عدد :

4,294,976,295 ms

در واقع ۴۰ روز و ۱۷ ساعت و ۲ دقیقه و ۴۷ ثانیه و ۲۹۵ میلی ثانیه است .

نحوه نمایش آن به صورت زیر است :

T#49D17H2M47S295MS

مثلا ۲ دقیقه و ۵ ثانیه به صورت زیر است .

T#2M5S

۱۰- داده از نوع BOOL :

این نوع داده فقط یک بیت از فضای حافظه را به خود اختصاص می دهد .

۱۱- داده از نوع Byte :

این نوع داده یک بایت از فضای حافظه را به خود اختصاص می دهد .

نکته :

۱- بیت های داده از نوع Byte دارای ارزش مکانی نیستند .

۲- برای اینکه بتوان مقدار عددی ذخیره شده در متغییر بایت را خواند ، باید آن را به داده از نوع USINT تبدیل کرد .

۳- بلعکس اگه بخواهید داده ای را به صورت بیتی AND یا OR کنیم ، حتما باید آنرا به Byte تبدیل کنیم .

۱۲- داده از نوع WORD:

این نوع داده ۲ بایت از فضای حافظه را به خود اختصاص می دهد و بیت ها دارای ارزش مکانی نیستند .

۱۳- داده از نوع DOUBLE WORD (DWORD) :

داین نوع داده ۴ بایت از فضای حافظه را بخود اختصاص می دهد و بیت های داخل آن دارای ارزش مکانی نیستند .

۱۴- داده از نوع LONG WORD (LWORD) :

این نوع داده ۸ بایت از فضای حافظه را به خود اختصاص می دهد و بیت ها ارزش مکانی ندارند.

توجه :

داده های نوع *TIME OF DATE ، DATE ، DATE&TIME ، STRING ، REAL ، LONG* و *REAL* ... را بعدا بررسی خواهیم کرد .

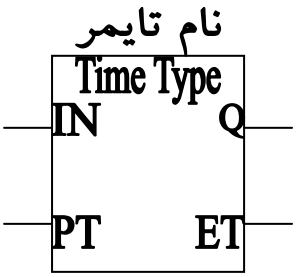
انواع تایمر در PLC (مطابق استاندارد IEC 61131-3):

۱- تایمر ضربه (One Shot) Pulse Timer

۲- تایمر تاخیر در وصل On Delay Timer :

۳- تایمر تاخیر در قطع Off Delay Timer :

قالب کلی تایمر ها :



PT : تنظیم زمان اولیه تایمر Present Time

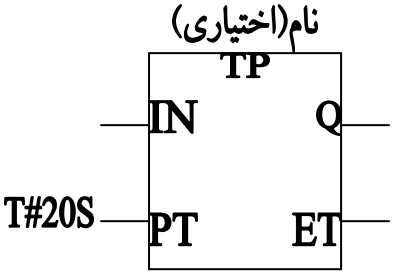
ET : زمان سپری شده Elapsed Timer

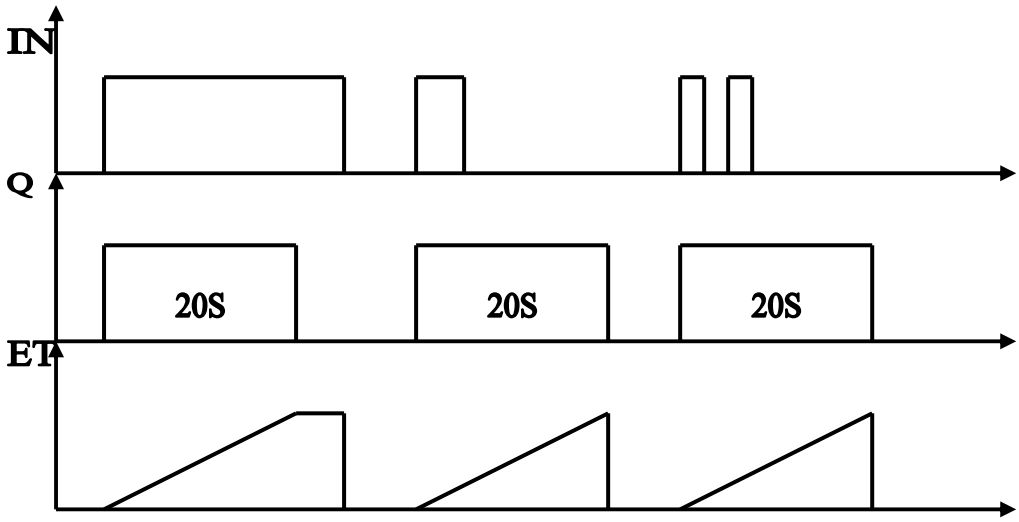
Q : Status Of Timer

IN : Start Input

۱- تایمر نوع اول TP :

جدول زمانبندی :





برای اینکه تایمر فعال شود باید IN یه لبه ۰ به ۱ را حس کند . (حساس به لبه)

چند نکته :

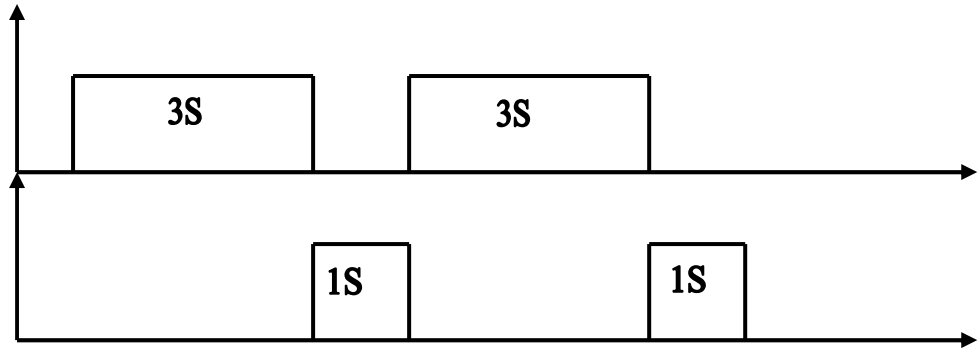
۱- اگر ضربه ای به پایه IN وارد شود حتما تایمر تا زمانی که در PT مشخص شده است می شمارد .

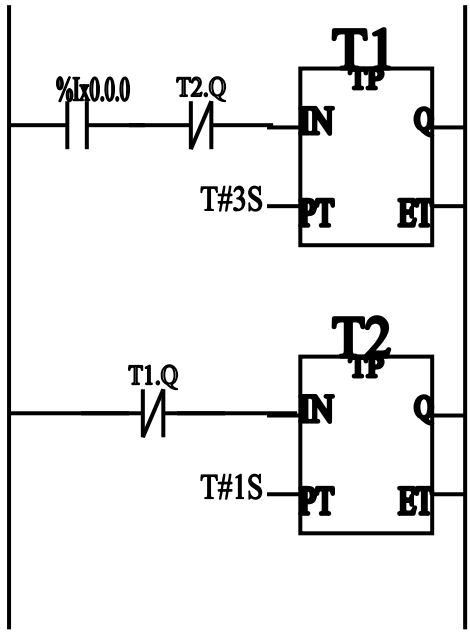
۲- این نوع تایمر قابلیت متوقف شدن در حین شمارش را ندارد .

۳- زمان سپری شده در ET ذخیره می شود .

مثال :

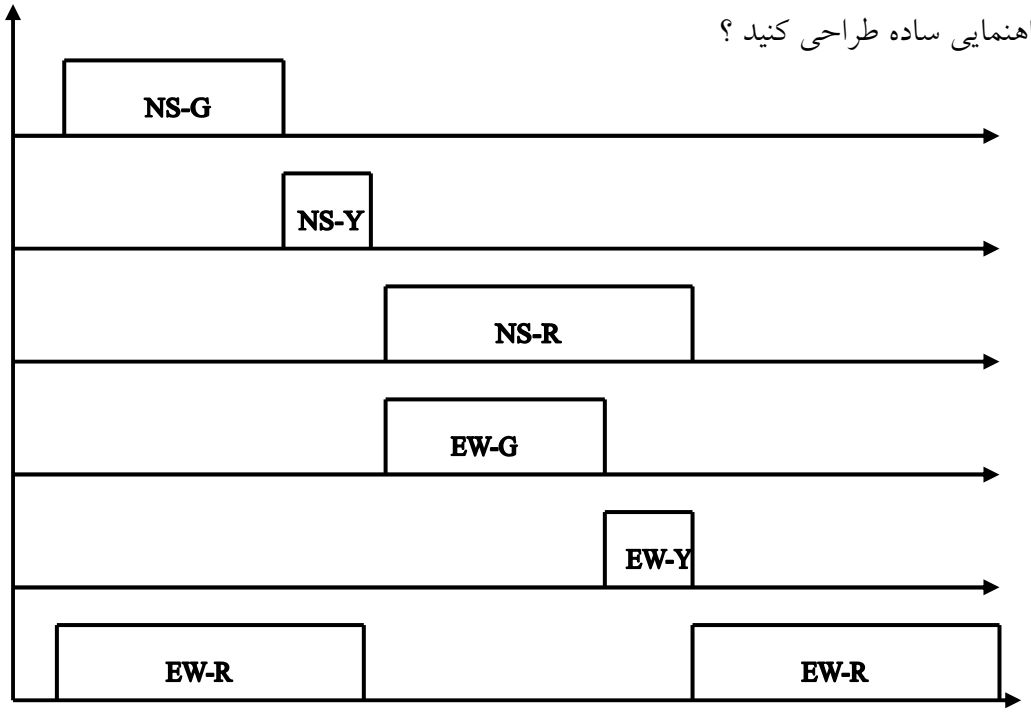
یک مدار چشمک زن با Dute Cycle قابل تنظیم طراحی کنید ؟





مثال :

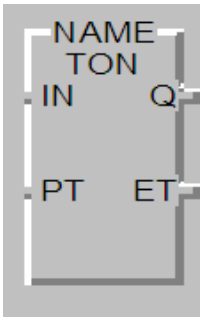
یک چراغ راهنمایی ساده طراحی کنید ؟



توجه :

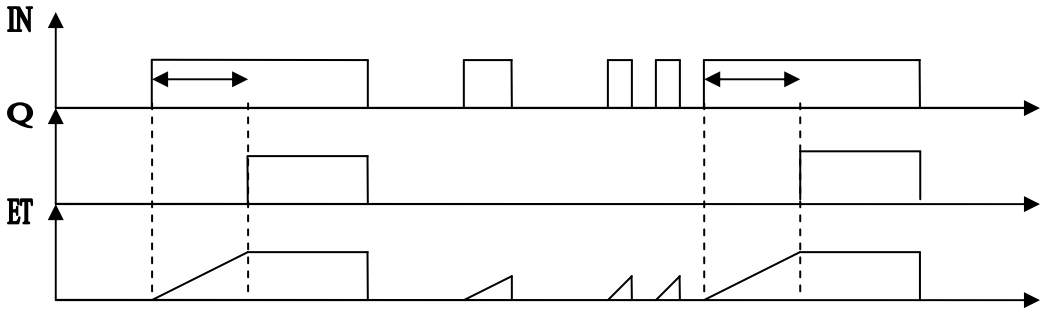
می توان تایمر چراغ های قرمز را حذف کرد چون R=G OR Y .

۲- تایمر تاخیر در وصل : On Delay Timer

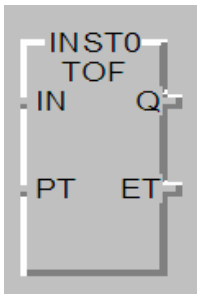


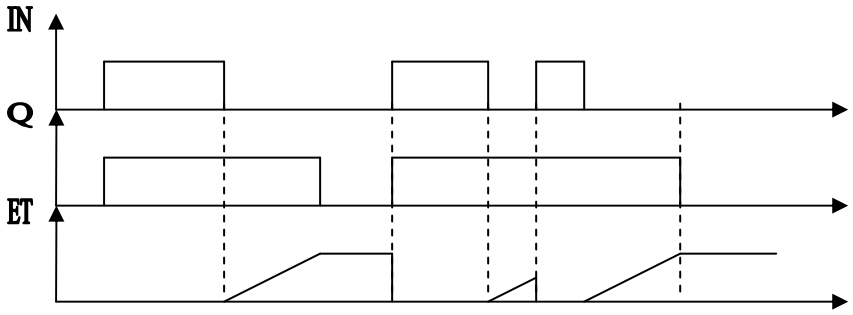
Name.Q : وضعیت خروجی

Name.ET : زمان سپر شده را ذخیره می کند .



۳- تایمر تاخیر در قطع : Off Delay Timer



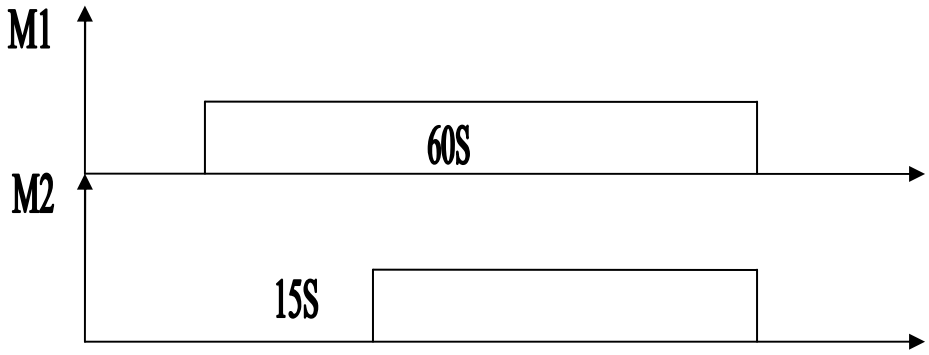


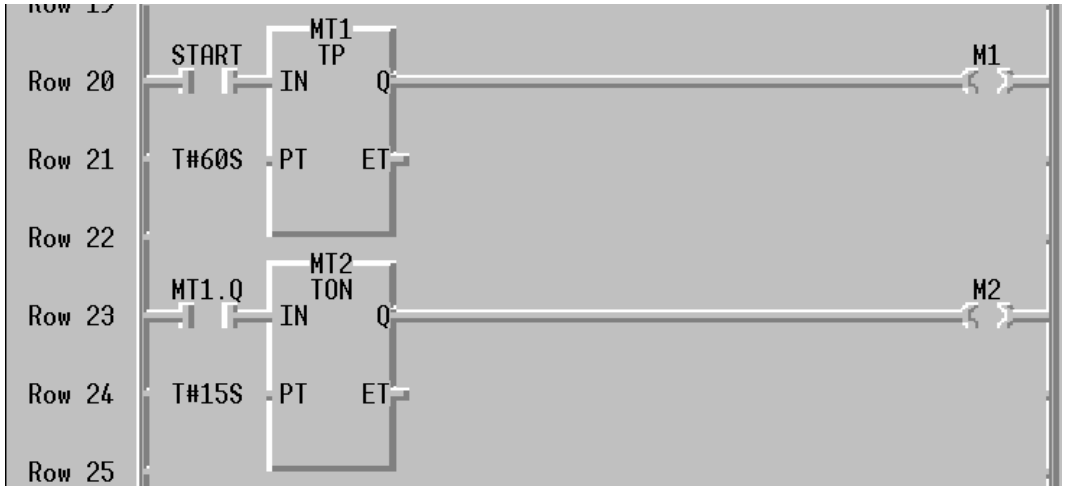
مثال :

می خواهیم ۲ عدد موتور را به شرح زیر راه اندازی کنیم :

۱- با زدن کلید start موتور m1 شروع به کار نموده و به مدت ۶۰ ثانیه روشن و سپس خاموش می شود .

۲- موتور M2 ۱۵ ثانیه پس از موتور M1 روشن شده و همراه با M1 خاموش شود .





مثال :

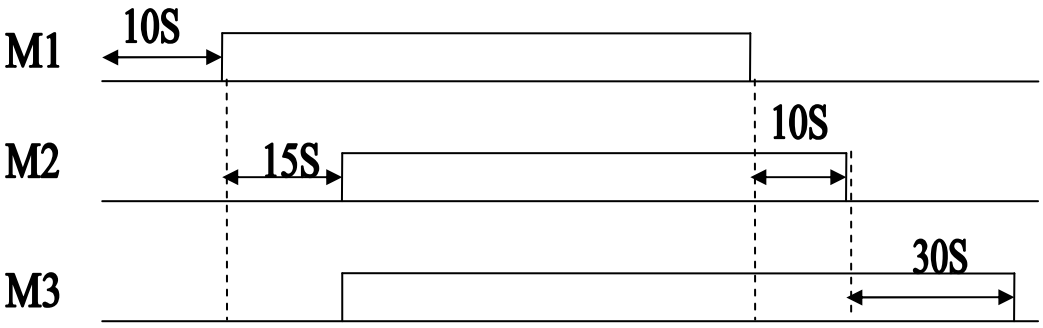
می خواهیم ۳ موتور را به شرح زیر راه اندازی کنیم :

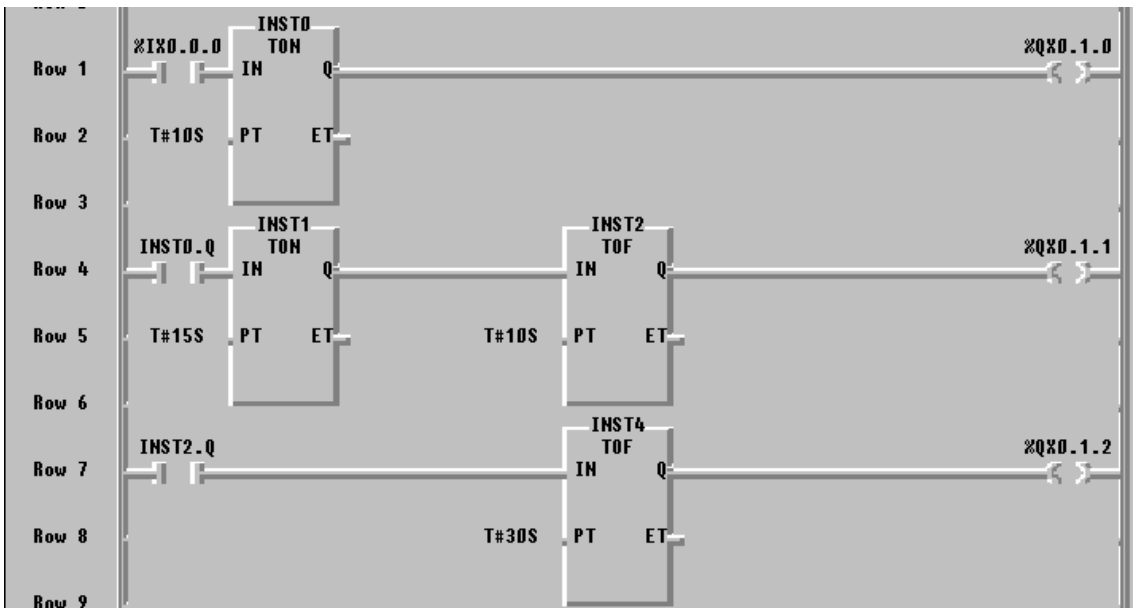
۱- با زدن کلید start موتور M1 با تاخیر ۱۰ ثانیه شروع به کار نموده و به مدت نامعلومی روشن بماند.

۲- موتور M2 ۱۵ ثانیه پس از روشن شدن موتور M1 روشن شده و به مدت نامعلومی روشن بماند .

۳- موتور M3 پس از روشن شدن موتور M2 روشن شده و به مدت نامعلومی روشن بماند .

۴- اگر به هر دلیلی موتور M1 خاموش شد موتور M2 ۱۰ ثانیه پس از خاموش شدن M1 خاموش شود . و موتور M3 ۳۰ ثانیه بعد از M2 خاموش شود .





فانکشن ها و فانکش بلاک ها در استاندارد IEC 61131-3 :

فانکشن ها :

فانکش در برنامه نوسی PLC مانند یک سابروتین در سایر زبان های برنامه نویسی است .

در حقیقت یک فانکش یک برنامه کوچک منطقی است که با اجرای آن ، فانکشن یک کار را انجام می دهد .

ویژگی های فانکش ها :

۱. فانکشن ها فاقد حافظه هستند . باید برای پایه خروجی آنها متغیری تعریف کرد تا داده در آن ذخیره شود . (برعکس تایمر ها ، تایمر ها فانکشن بلاک هستند)
۲. نوع داده ورودی و خروجی در فانکش ها باید یکی باشد .
۳. هر فانکش می تواند چند ورودی داشته باشد ولی همواره دارای یک خروجی هست .
۴. هر فانکش دارای یک پایه توان کننده ورودی (EN) است (توجه کنید که این ورودی حساس به لبه نیست بلکه حساس به سطح است) . هر گاه EN=1 باشد فانکش اجرامی شود . و هر فانکشن دارای یک فعال ساز خروجی نیز هست . هر گاه EN=1 شود ENo=1 می شود مگر اینکه در فانکش خطایی رخ داده باشد . در این صورت علی رغم EN=1 مقدار ENo=0 باقی می ماند .

ویژگی های فانکشن بلاک ها:

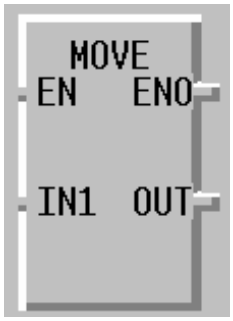
۱. فانکشن بلاک ها دارای حافظه هستند .
۲. نوع داده ورودی و خروجی در فانکشن بلاک ها می تواند متفاوت باشد .
۳. یک فانکش بلاک میتوان شامل تعدادی ورودی و تعدادی خروجی باشد .
۴. هر فانکشن بلاک دارای یک پایه توان کننده ورودی است . هر گاه مقدار این پایه از ۰ به ۱ تغییر کند برنامه فانکش بلاک اجرا می شود .

بررسی برخی فانکشن ها :

فانکشن MOVE :

عملکرد :

مقدار ورودی را در خروجی کپی می کند . باید نوع متغییر ورودی و خروجی یکسان باشند .



عملگر های ریاضی :

نام فانکشن :

ADD : جمع ، تعداد ورودی ها از ۲ تا ۸ .

DIV : تقسیم ، تعداد ورودی ها ۲ .

MUL : ضرب ، تعداد ورودی ها ۲ تا ۸ .

SUB : تفریق ، تعداد ورودی ها ۲ .

ABS : قدرمطلق ، تعداد ورودی ۱ .

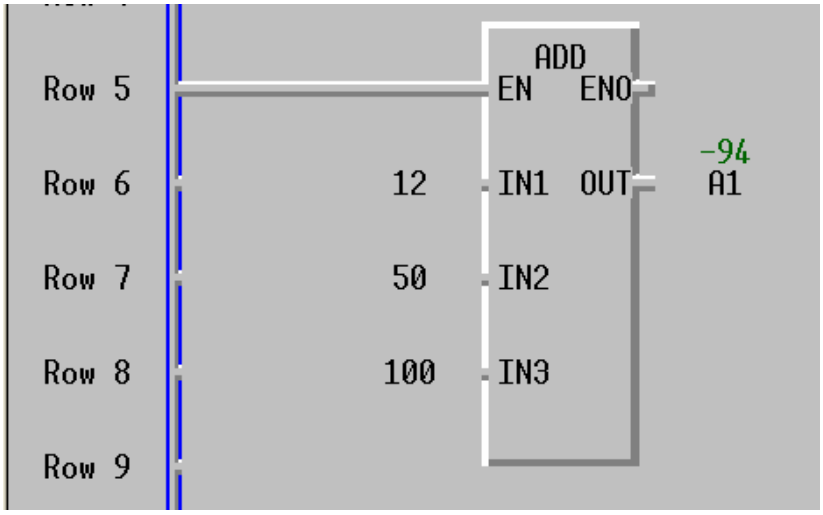
MOD : باقیمانده تقسیم ، تعداد ورودی ها ۲ .

نکته :

برای انجام عملیات های ریاضی پیچیده تر باید به قابلیت های CPU توجه داشت . بسته به توانایی CPU می توان عملیات های پیچیده تر مانند Sin و ... را نیز انجام داد .

به عنوان مثال :

توجه کنید که این برنامه ۳ مقدار ۱۲ ، ۵۰ و ۱۰۰ را با هم جمع می کند . البته باید خروجی ان ۱۶۲ باشد ولی همانطور که مشاهده می کنید خروجی آن ۹۴- است . علت آن این است که متغییر خروجی تعریف شده (A1) از نوع SINT تعریف شده است ، در این متغییر فقط میتـوان اعـداد بـین ۱۲۸- تـا ۱۲۷ را ذخیره کرد .



نکته :

تمامی عملگر های ریاضی فانکشن هستند .

عملگر های مقایسه کننده :

در زبان LD و یا سایر زبان ها و در همه نرم افزار ها عملگر مقایسه به شرح زیر داریم :

GT : بزرگتر از .

GE : بزرگتر یا مساوی از .

LT : کوچکتر از .

LE : کوچکتر یا مساوی از .

EQ : مساوی .

NE : نا مساوی .

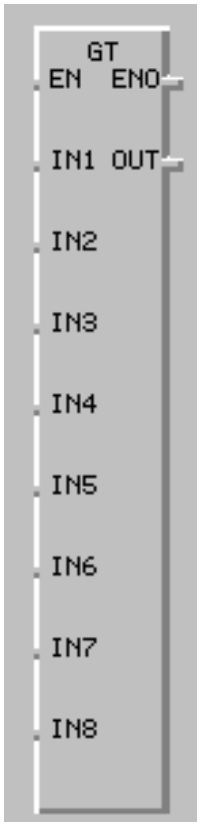
تمامی توابع بالا نفسا فانکش هستند .

مثال GT :

اگر

$IN1 > IN2 > IN3 > IN4 > IN5 > IN6 > IN7 > IN8$

و EN=1 آنگاه خروجی out برابر ۱ می شود . در غیر این صورت برابر ۰ است .



تذکر :

۱- در عملگر های مقایسه کننده تمام ورودی ها باید از یک نوع باشند.

۲- تمامی داده های مجاز می توانند با یکدیگر مقایسه شوند .

نکات:

در مقایسه string ها به ترتیبی که حروف در دیگشنری آمده اند توجه می کند . به عنوان مثلا اگر ۲ رشته ali و mohamad با هم مقایسه شوند چون علی با a شروع می شود و a در دیگشنری زودتر از حرف m است پس رشته ali از mohamad بزرگتر است . نکته دیگر اینکه خروجی مقایسه ۲ رشته همسان یکی با حروف بزرگ و یکی با حروف کوچک برای رشته ۱ می شود که کد اسکی آن رشته بزرگتر از رشته دیگر باشه . مثلا کد اسکی a از A بزرگتر است پس رشته a از A بزرگتر است . در مورد متغیر Byte نیز باید گفت که آن را تبدیل به USINT می کند بعد عمل مقایسه را انجام می دهد .

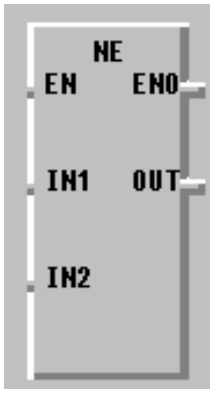
تمامی فانکش های بالا مانند GT هستند فقط علائم کوچکتری یا بزرگتری آن تغییر می کند .

به جز NE که فقط ۲ ورودی دارد .

NE :

اگر EN=1 و IN1~=IN2 آنگاه OUT=1

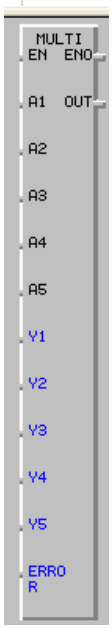
در غیر این صورت OUT=0 .

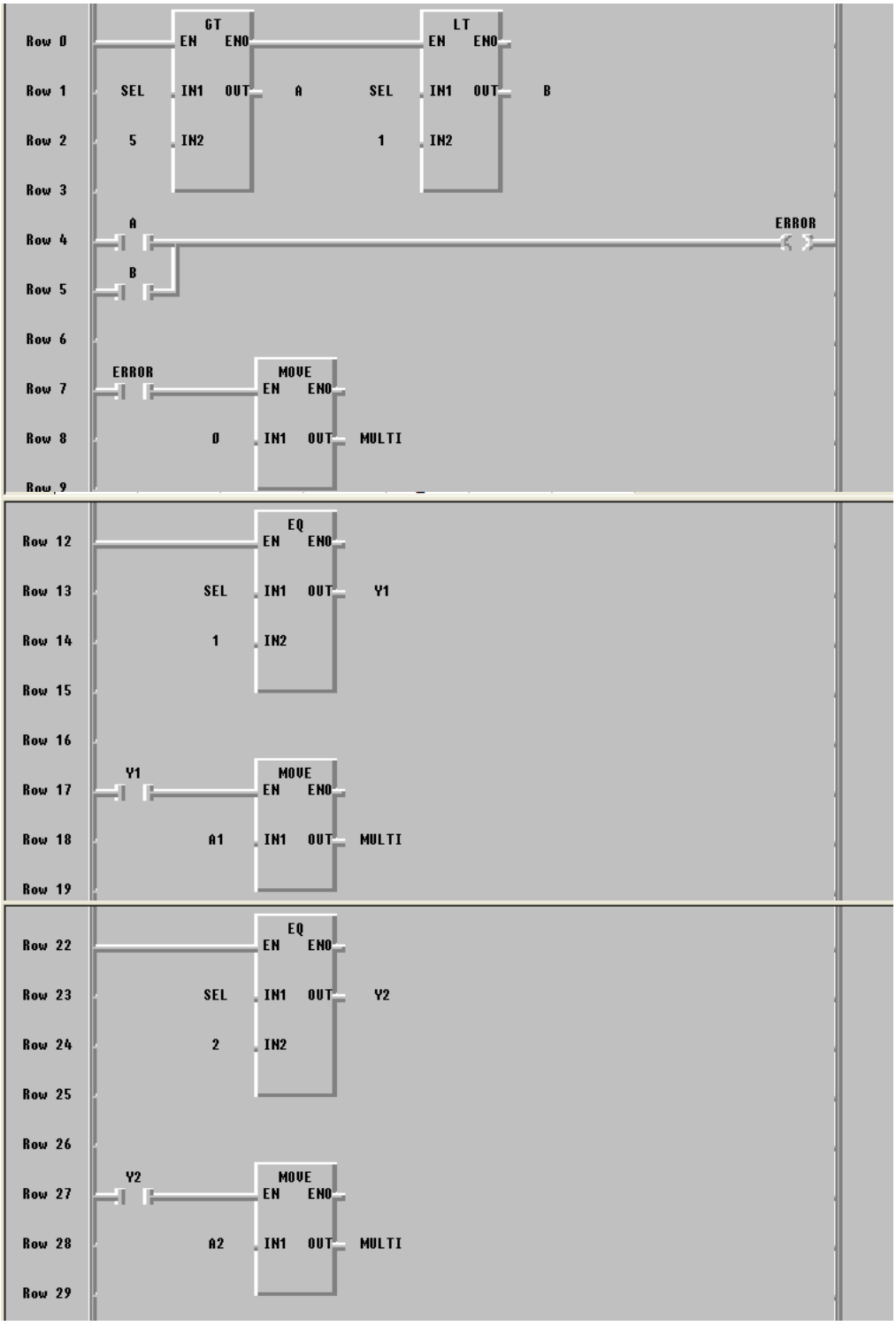


مثال :

یک مالتی پلکسر ۵ ورودی طراحی کنید :

برای این منظور احتیاج به ۵ ورودی ، یک خروجی که در هر لحظه یکی از ورودی ها رو آن است و یک سلکتور برای چرخش بین ورودی ها . به ۵ خروجی بیتی احتیاج داریم که نمایانگر این است که کدام ورودی به خروجی متصل شده است . یک خروجی به نام ERROR که اگر سلکتور عددی خارج از محدوده ۱ تا ۵ بود خطا را نشان دهد .

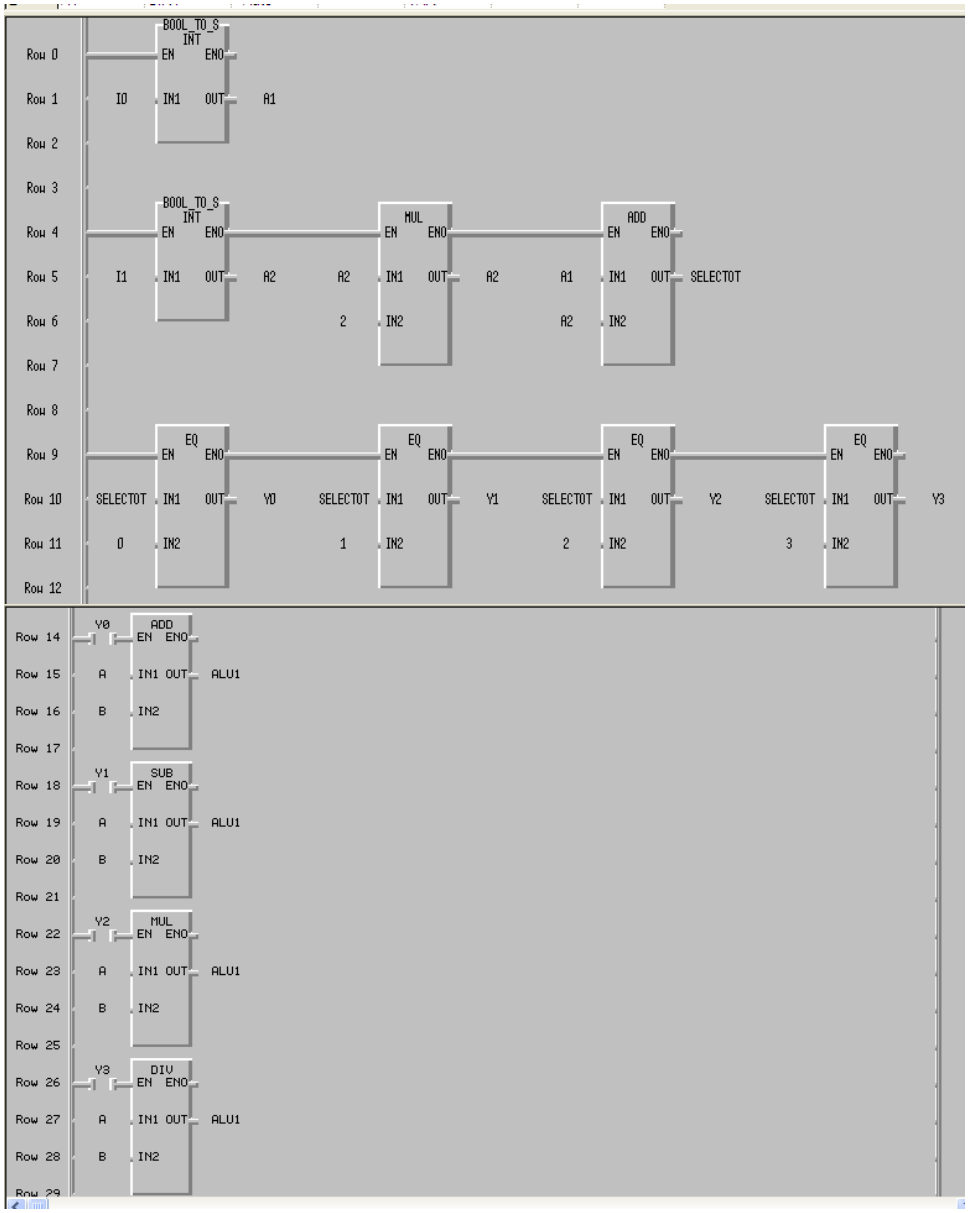




و به همین ترتیب برای بقیه ورودی و خروجی ها .

مثال۶ :

برنامه ی یک alu رو بنویسید با قابلیت های جمع و ضرب و تفریق و تقسیم.با ورودی سلکتور باینری ؟



توضیح برنامه :

ورودی های I1 و I0 ۲ بیت هستند . برای بدست آوردن عدد قرار داده شده در این ۲ بیت بیت دوم را در عدد ۲ ضرب می کنیم و با بیت اول جمع می کنیم . حاصل جمع عدد بین ۰ تا ۳ خواهد بود .

نحوه ساختن فانکشن بلاک ها :

بار دیگر خواص فانکش بلاک ها را مرور می کنیم :

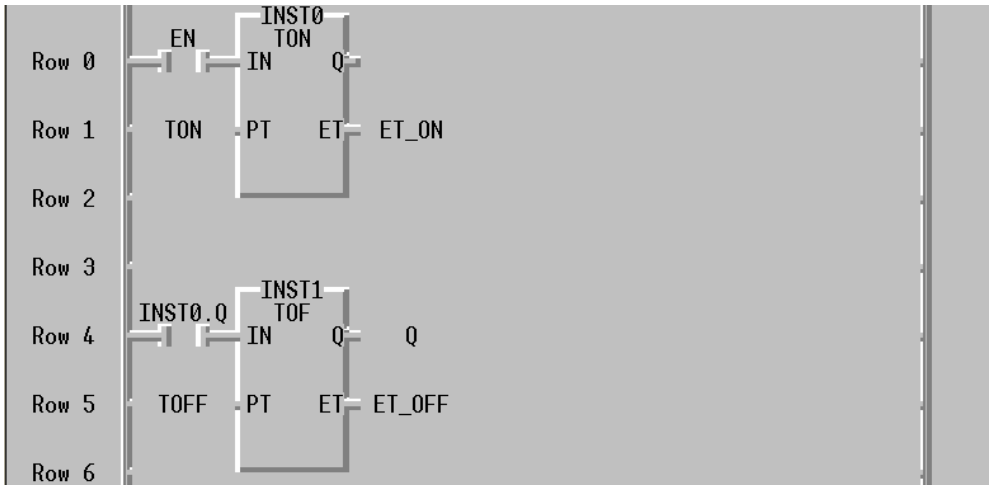
- دارای حافظه هستند .
- نوع داده ورودی و خروجی می تواند متفاوت باشد .
- یک فانکشن بلاک می تواند تعدادی ورودی و خروجی داشته باشد .

نکته بسیار مهم :

توجه داشته باشید که در هر فانکش بلاک حتما باید اولین ورودی و اولین خروجی ای کـه تعریـف مـی شوند از نوع bool باشند .

مثال ۷ :

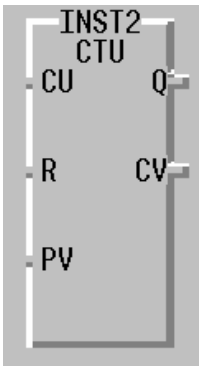
تایمری طراحی کنید که هم تاخیر در وصل باشد و هم تاخیر در قطع :



شمارنده ها :

همه PLC ها دارند
- شمارنده صعودی up counter
- شمارنده نزولی down counter
- شمارنده صعودی / نزولی up/down counter
- شمارنده حلقوی ring counter

شمارنده صعودی (یک فانکشن بلاک):



CU ➔ counter up input pulse (BOOL)
R➔ reset input (BOOL)
PV ➔ preset value (INT)
Q ➔ up counter output (BOOL)
CV ➔ current value (INT)

اگر R=1 آنگاه CV=0 .

اگر CV>=PV آنگاه Q=1

نکته :

CV می تواند منفی باشد .

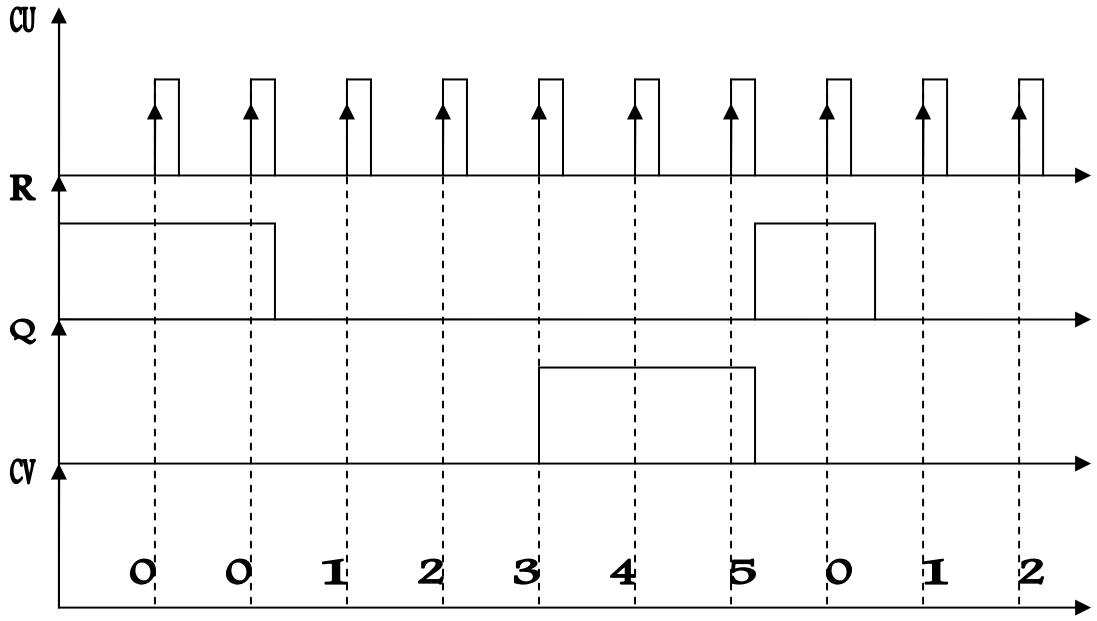
اگر PV منفی باشد همواره خروجی Q=1 است . چون همیشه شرط برقرار است .

بوسیله این شمارنده های نرم افزاری تعبیه شده در PLC حداکثر می توان تا ۲۰۰ بار در ثانیه شمرد .

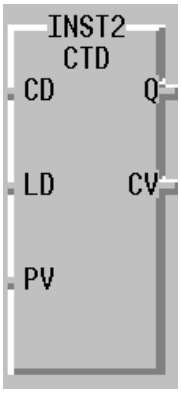
برای شمارش بیشتر باید از شمارنده های سخت افزاری استفاده نمود مانند HSC .

دیاگرام تایمینگ :

با فرض PV=3



شمارنده نزولی Down Counter **(** CTD **):**

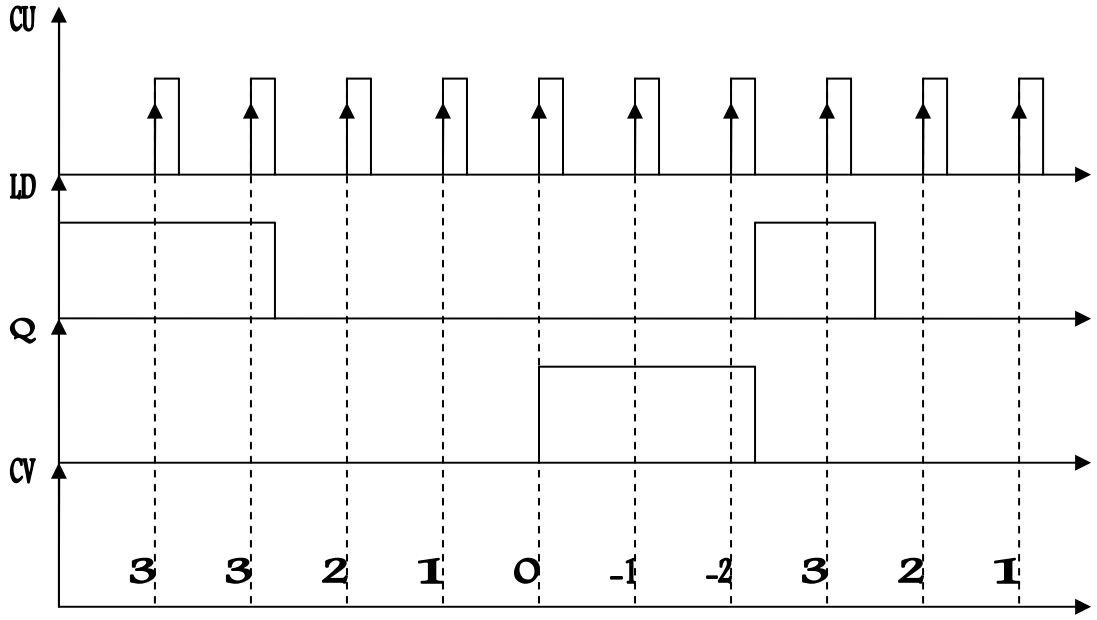


CD ➔ counter Down input pulse (BOOL)
LD➔ Load Perset Value (BOOL)
PV ➔ preset value (INT)
Q ➔ up counter output (BOOL)
CV ➔ current value (INT)
اگر LD=1 باشد آنگاه CV=PV

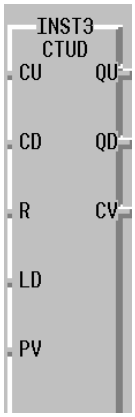
اگر $CV \leq 0$ آنگاه Q=1 .

دیاگرام تایمنیگ :

با فرض PV=3 .



شمارنده صعودی نزولی (CTUD) :



اگر R=1 باشد آنگاه CV=0 .

اگر LD=1 باشد آنگاه CV=PV .

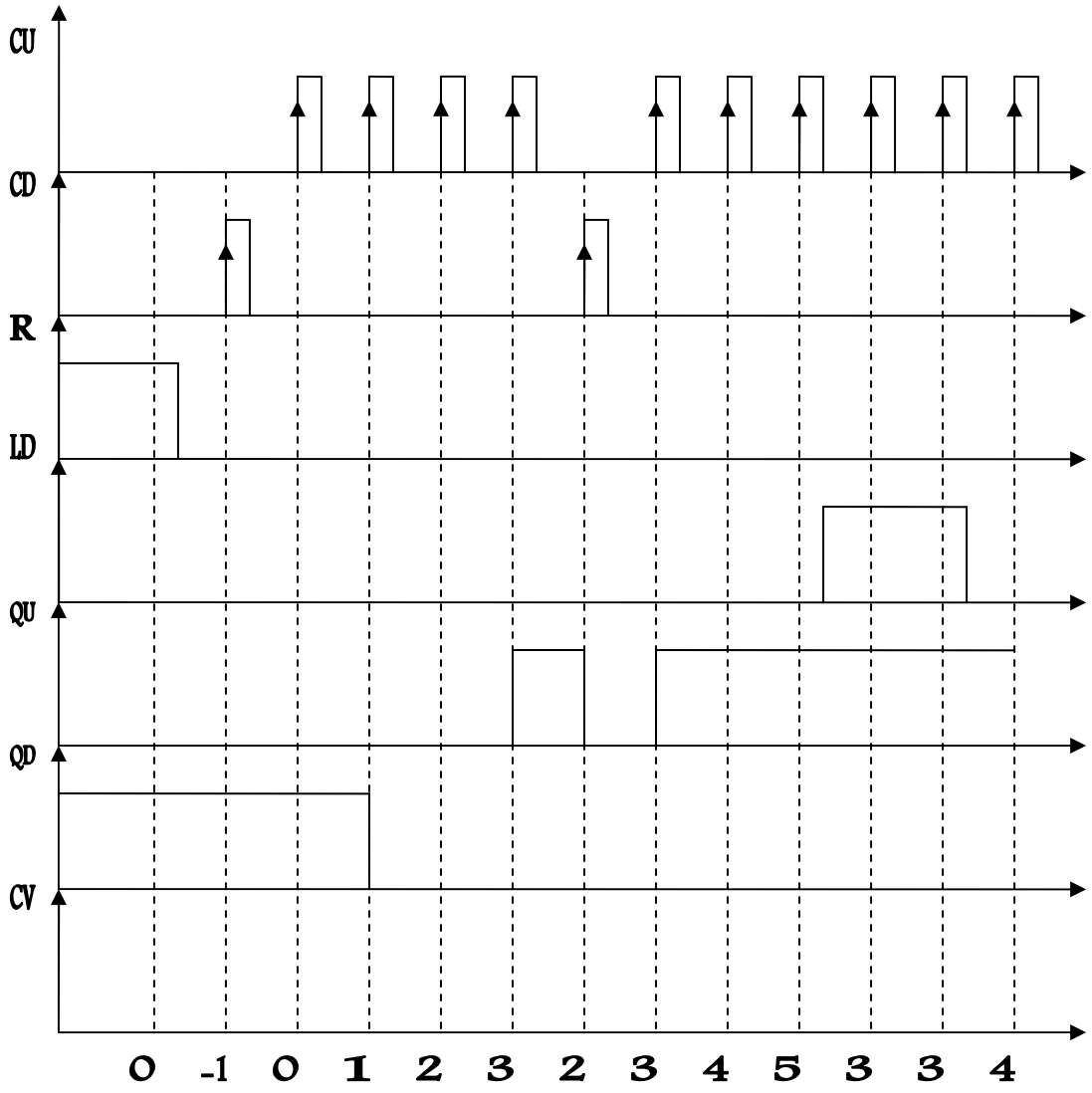
اگر $CV \geq PV$ آنگاه $Q_u = 1$.

اگر $CV \leq 0$ آنگاه $Q_D = 1$.

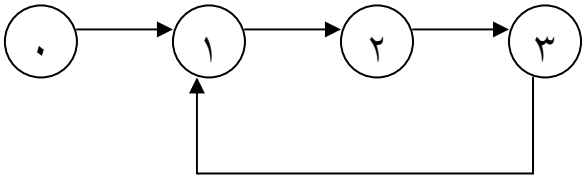
به ترتیب نوشته شده دارای اولویت هستند . اگر R و LD با هم یک شوند ، اول R پذیرفته می شود بعد LD .

دیاگرام تایمنیگ :

با فرض PV=3 .

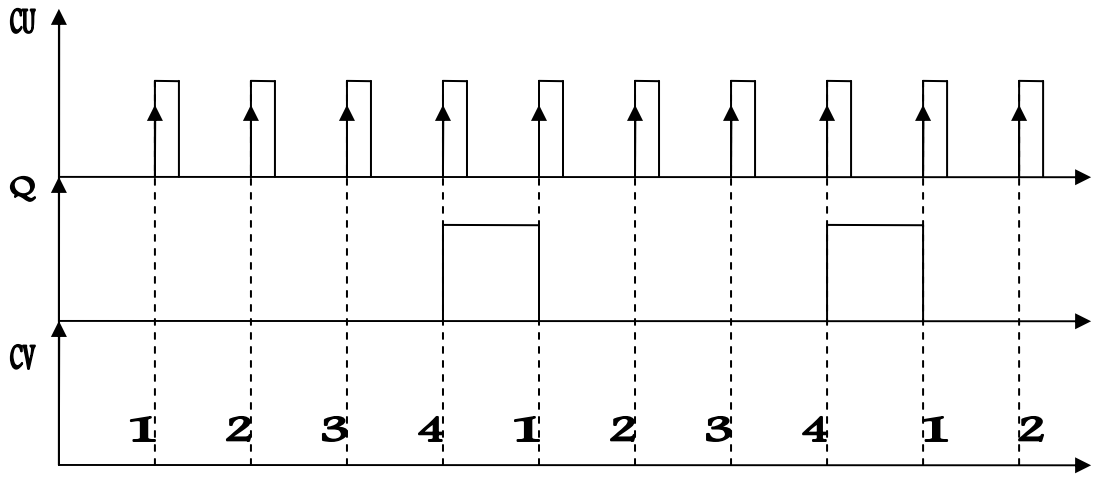


شمارنده حلقوی یا Ring Counter **:**



دیاگرام تایمینگ :

با فرض PV=3.



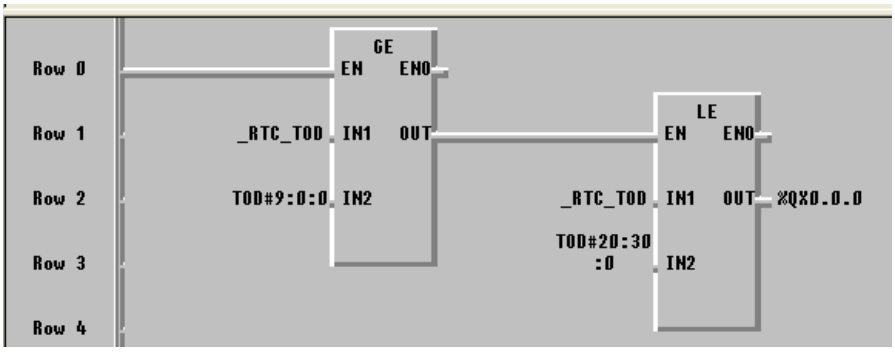
(Real Time Clock) : RTC

یک آرایه است که زمان دقیق را در خود در هر لحظه حفظ می کند .

یکی از خانه های آن با نام Time Of Day **به صورت** _RTC_TOD **قابل آدرسی دهی می باشد .**

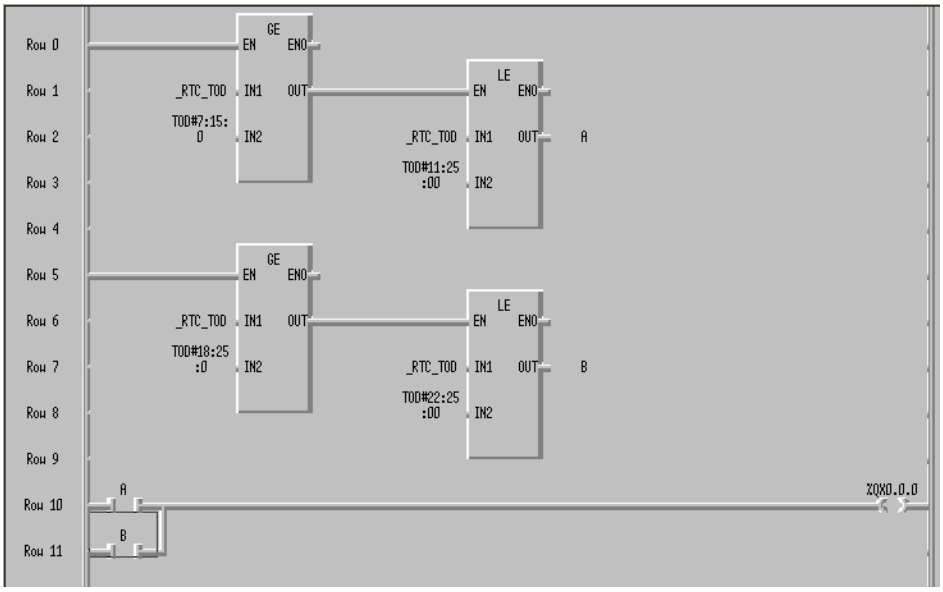
مثال ۸ :

برنامه ای بنویسید که همه روز از ساعت ۹ الی ۲۰:۳۰ دقیقه شب موتوری را روشن کند .



مثال ۹ :

برنامه ایکه از ساعت ۷:۱۵ تا ۱۱:۱۰ و از ساعت ۱۸:۲۵ تا ۲۲:۲۵ دقیقه موتوری را روشن کند ؟



نکته :

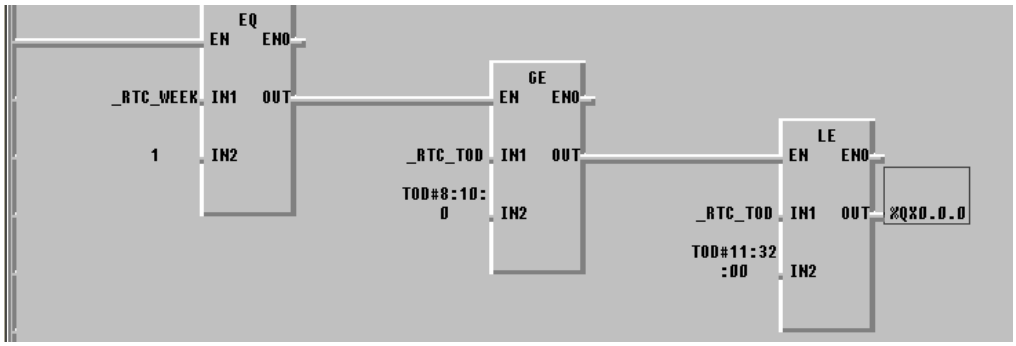
اگر بخواهیم با این روش از ساعت مثلا ۲۳ تا ساعت ۲ شب را جدا کنیم باید ۲ مرحله ای نوشته شـود .
یکی بزرگتر از ۲۳ و دیگری از ۰ تا ۲ شب .

متغییری که روزهای هفته را در خود نگه می دارد و نام آن RTC_WEEK_ است .

روزهای هفته	شناسه عددی
دوشنبه	۰
سه شنبه	۱
چهارشنبه	۲
پنجشنبه	۳
جمعه	۴
شنبه	۵
یکشنبه	۶

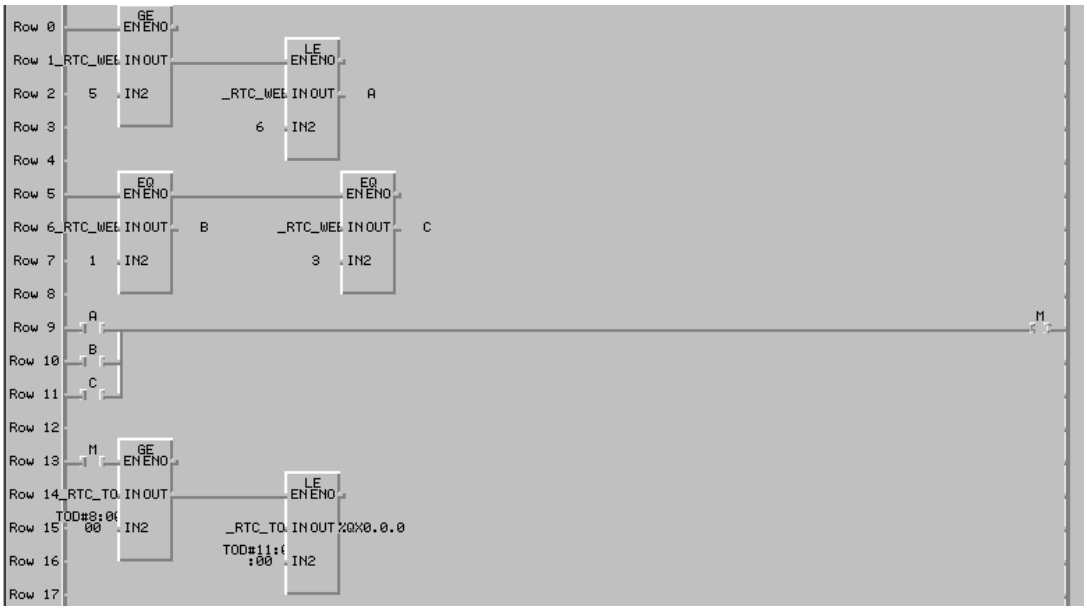
مثال ۱۰:

برنامه ای که فقط روز های ۳ شنبه از ساعت ۸:۱۰ تا ساعت ۱۱:۳۲ دقیقه موتوری را روشن کند ؟



مثال ۱۱:

برنامه ای بنویسید که موتوری را از ساعت ۸ تا ۱۱ در روز های شنبه یکشنبه و سه شنبه و پنجشنبه روشن کند ؟

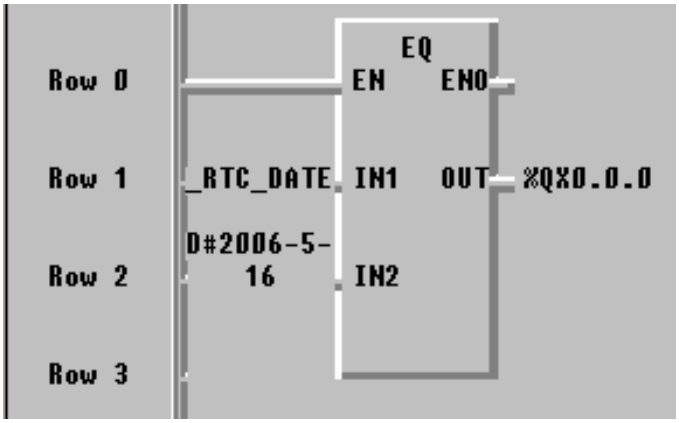


تاریخ :

متغییر RTC_DATE_ تاریخ (سال و ماه و روز) را در خود نگه میدارد .

مثال ۱۲:

برنامه ای بنویسید که فقط در یک روز خاص موتوری را ۱ کند .

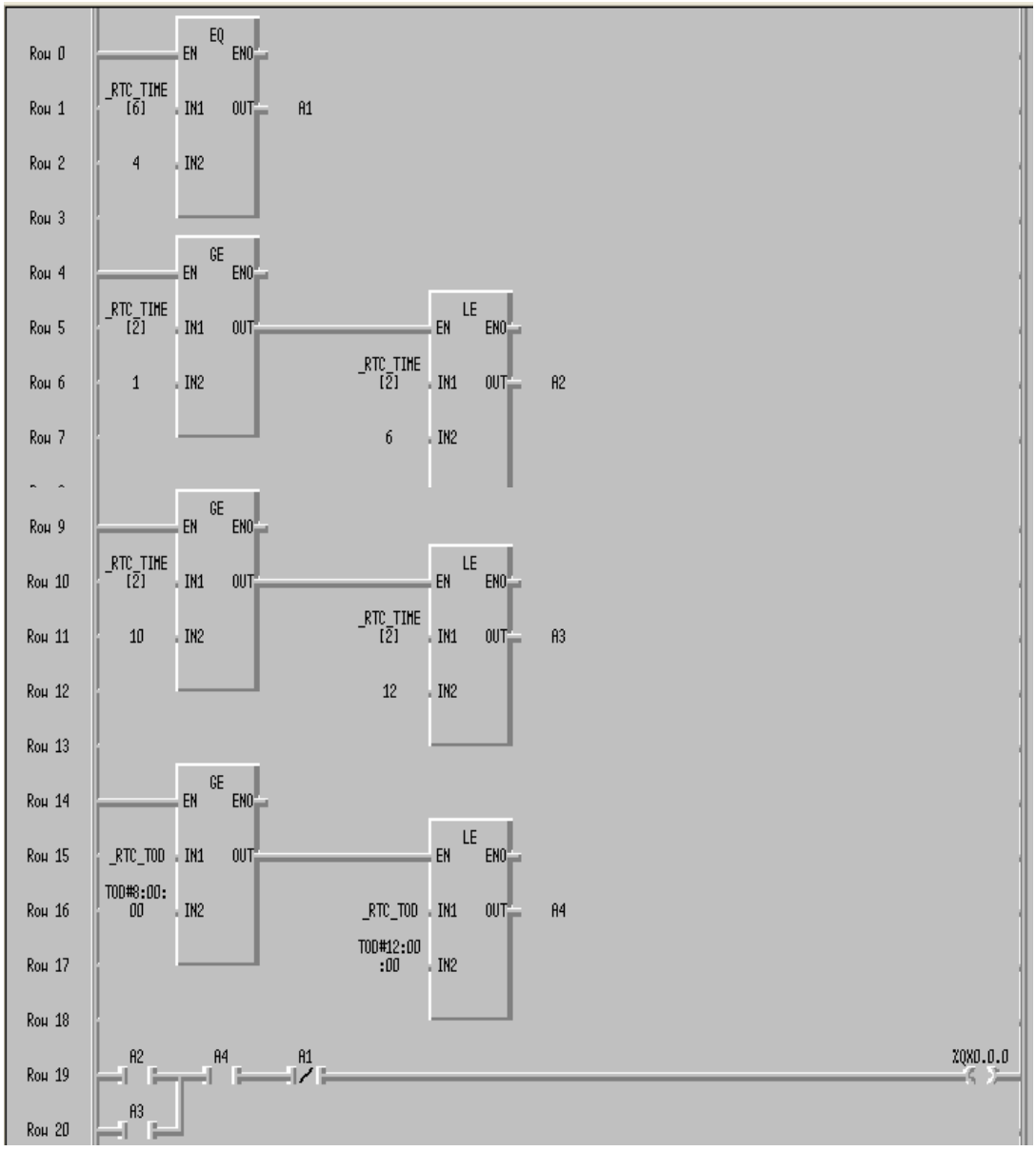


تمامی متغییر های بالا در آرایه ای به نام RTC_TIME[ ]_ از صفر تا ۷ وجود دارد .

_RTC_TIME[ ]	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
توضیحات	سال از ۰۰ تا ۹۹	ماه از ۱ تا ۱۲	روزهای ماه از ۱ تا ۳۱	ساعت از ۰۰ تا ۲۳	دقیقه از ۰۰ تا ۵۹	ثانیه از ۰۰ تا ۵۹	روزهای هفته از ۰ تا ۶	قرن از ۲۰ تا ۲۱

مثال ۱۳ :

برنامه ای بنویسید که روزهای اول تا ۶ هر ماه و ۱۰ و ۱۲ هر ماه یک موتور از ساعت ۸ تا ۱۲ کـار کنـد اگر در روز های یاد شده جمعه وجود داشت ، جمعه کار نکند ؟



یاد آوری :

۲ نوع متغییر جدید که در این بخش به آنها اشاره شد متغییر های زمان و تاریخ بودند :

TOD و D .

به عنوان مثال :

TOD#12:31:20 به معنی ساعت ۱۲ و ۳۱ دقیقه و ۲۰ ثانیه می باشد .

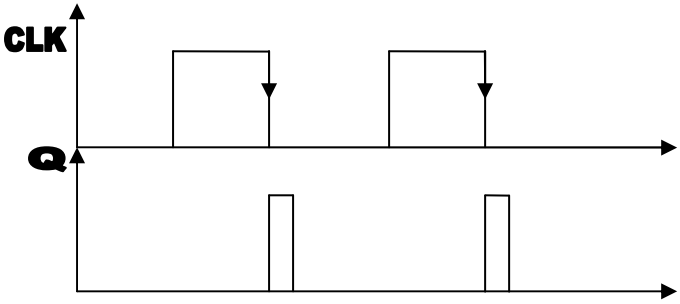
و D#2006-5-16 به روز ۱۶ و ماه ۵ و سال ۲۰۰۶ می باشد .

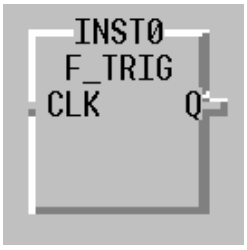
چند فانکشن بلاک پر کاربرد:

F_TRIG که آشکار ساز لبه پایین رونده است .

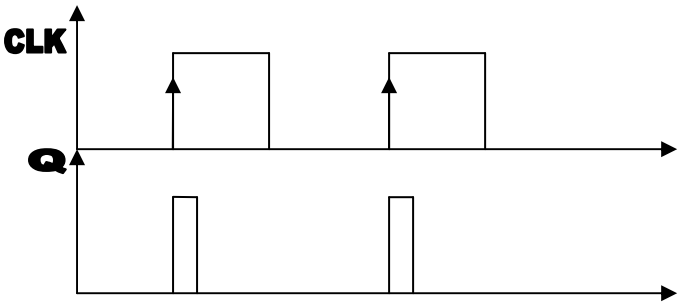
R_TRIG که آشکار ساز لبه بالارونده است .

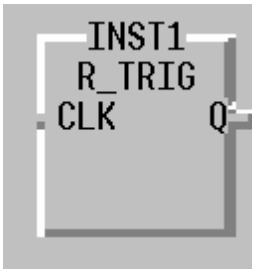
F_TRIG :





R_TRIG :



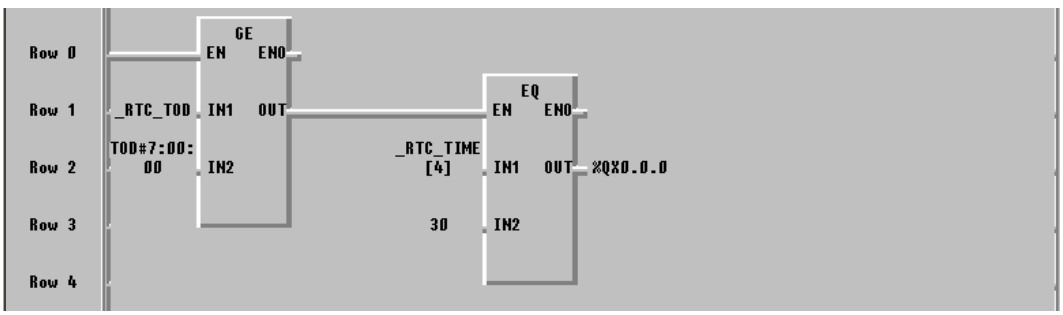


توجه :

عرض هر کدام از خروجی های (Q) برابر با زمان یک دور کاملِ اجرای برنامه است .

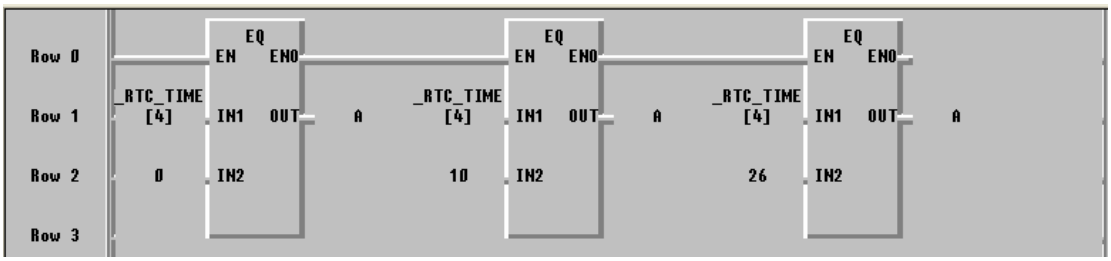
مثال ۱۴ :

برنامه ای بنویسید که هر دقیقه در تمام ساعات مثلا ۷:۳۰ و ۸:۳۰ و ... بـه جـز از سـاعت ۱۲ شـب تـا ۷ صبح موتوری را روشن کند؟



مثال ۱۵ :

برنامه ای بنویسید که هر ۲۶ دقیقه ، ۱۰ دقیقه و هر ۰ دقیقه پالسی بدهد .



نکته :

_RTC_TIME در هر محلی از برنامه که باشد به صورت مستقیم از حافظه خوانده می شود .

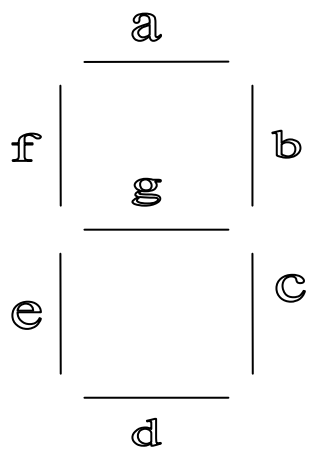
نکته :

بعضی از مواقع می توان ورودی ها را در جا خواند .

درایو سون سگمنت :

برای این منظور حتما باید عددی که قرار است نمایش داده شود را در یک متغییر از نوع WORD قرار داد .

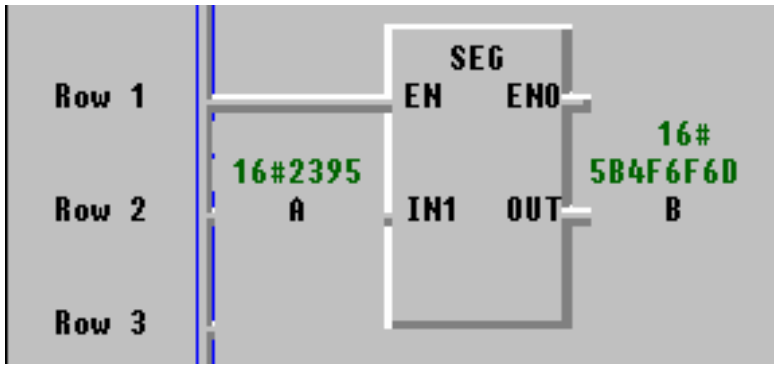
سپس با استفاده از فانکشن بلاک SEG می توان کد اعداد BCD دخیره شده در متغییر WORD را بدست آورد .



مثال A=16#1385 (#۱۶ به معنای مبنای ۱۶ است) در این حالت که A از نوع WORD می باشد و نحوه ذخیره آن به صورت زیر است .

۰۰۱۰	۰۰۱۱	۱۰۰۱	۰۱۰۱

با استفاده از فانکشن بلاک SEG تبدیل به متغییر از نوع DWORD به صورت زیر می شود :



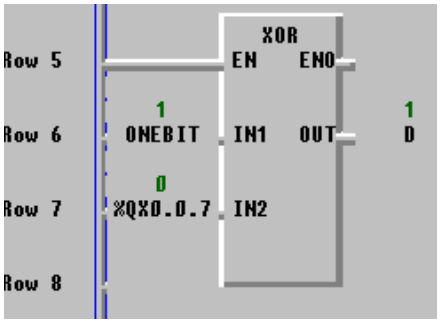
B برابر خواهد شد با : (16#5B4F6F6D) که از نوع DWORD است .

01011011	01001111	01101111	0gfedcba 01101101

نکته :

همانطور که در بالا مشاهده می کنید یک سون سگمنت از ۷ قسمت تشکیل شده است بنابراین در هر بایت استفاده شده ۱ بیت بلااستفاده خواهد ماند . برای استفاده از این بیت می توان از فانشکن XOR استفاده کرد . در منطق خروجی XOR اگر یکی از ورودی ها صفر باشد خروجی برابر با ورودی دوم خواهد بود اگر برابر صفر بود خروجی صفر و اگر برابر با ۱ بود خروجی یک می شود .

به عنوان مثال فرض کنید %Qx0.0.7 بیت بلااستفاده باشد و می خواهیم بیت onebit را به جای قرار دهیم . از روش زیر استفاده می کنیم .

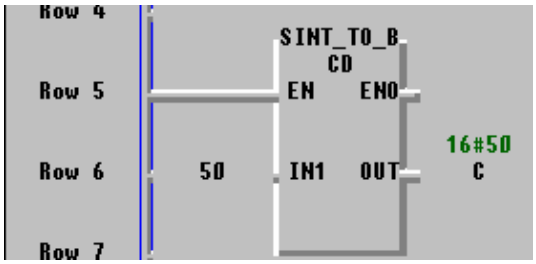


می توان به جای خروجی D مستقیما %Qx0.0.7 را قرار داد .

یاد آوری چند فانکش پرکاربرد که عملکری مشابه عملکرد فانکشن بالا دارند :

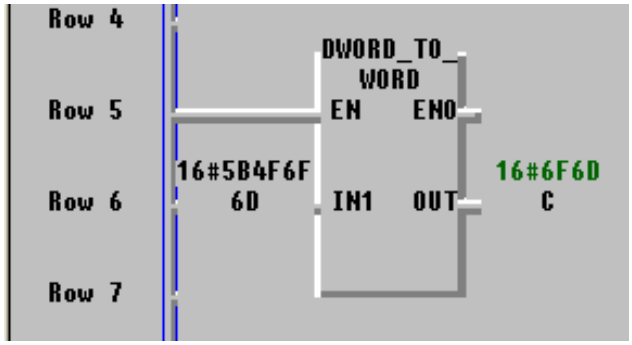
SINT_TO_BCD :

برای تبدیل SINT به BCD .



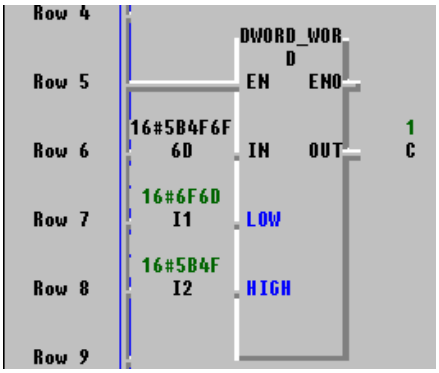
DWORD_TO_WORD :

۱۶ بیت کم ارزش DWORD را در متغییر WORD سمت راست قرار می دهد .



DWORD_WORD :

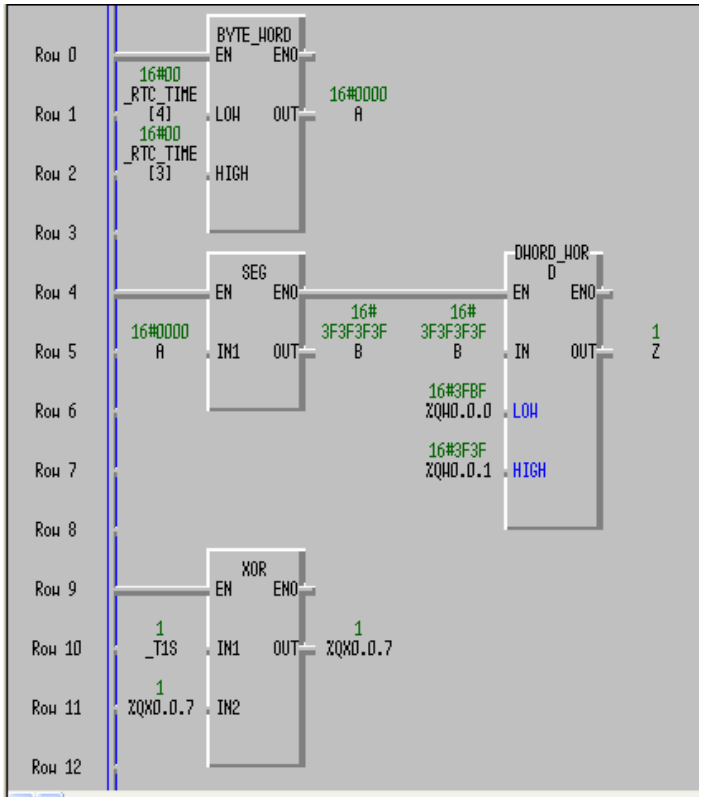
DWORD را به ۲ قسمت تقسیم کرده و هر قسمت را در یک WORD قرار می دهد .



فانکش هایی مشابه عملکرد فانشکن های بالا با نام های BYTE_BIT و WORD_BYTE نیز وجود دارند .

مثال ۱۶ :

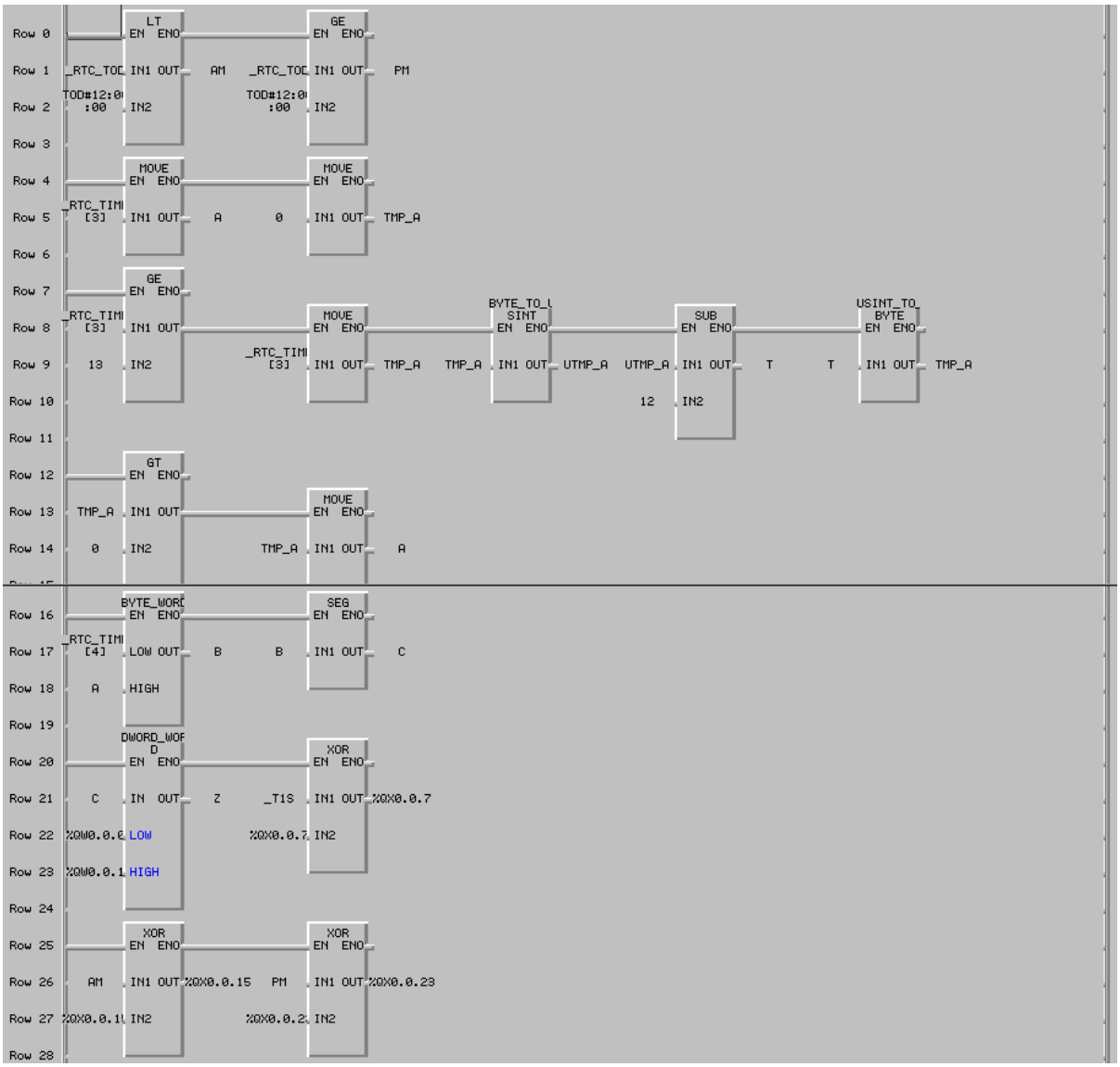
برنامه ای بنویسید که ساعت و دقیقه را برروی ۴ سون سگمنت نشان دهد .



در مثال بالا از آدرس ۰۰ تا آدرس ۱۵ مربوط به دقیقه و از ۱۶ تا ۳۱ نیز مربوط به ساعت می باشد همچنین بیت ۷ نیز به عنوان چشمک زن ثانیه به کار رفته است .

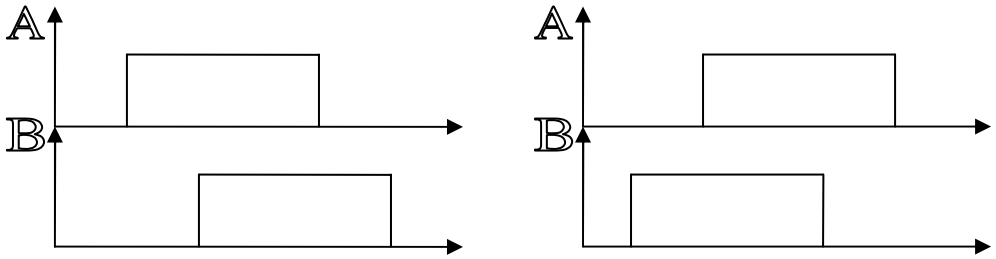
مثال ۱۷ :

برنامه ای بنویسید که ساعت و دقیقه را برروی ۴ سون سگمنت نشان دهد . و همچین ساعت را همـواره از ۰۰ تا ۱۲ نشان دهد و چراغ هایی برای AM و PM نیز داشته باشد .



شمارنده صعودی نزولی : Rotary-encoder

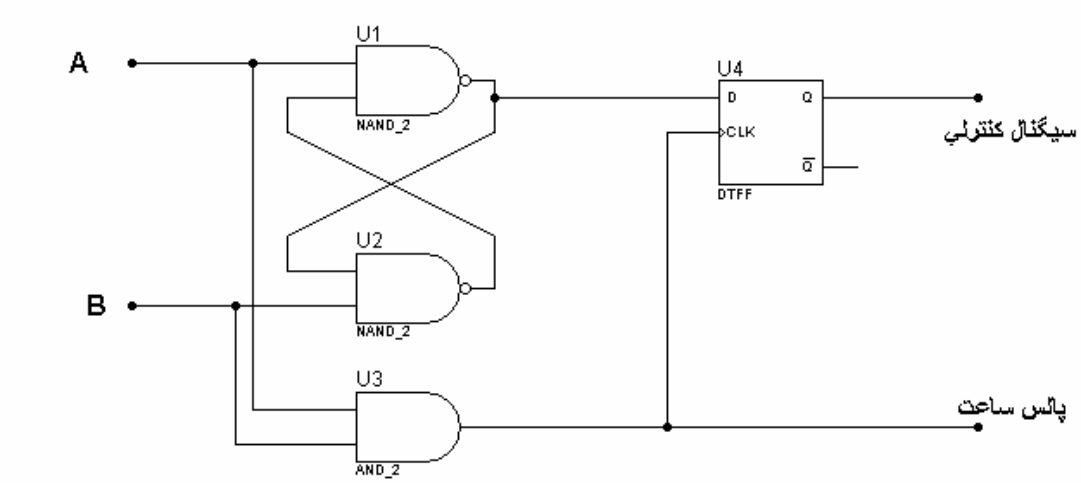
وسیله ای است که جهت حرکت را نشان می دهد. توسط ۲ بیت A و B می توان تعیین کرد که حرکت به سمت چپ است یا به سمت راست:



اول B بعد A جهت دیگر

اول A بعد B یک جهت

مدار داخلی :

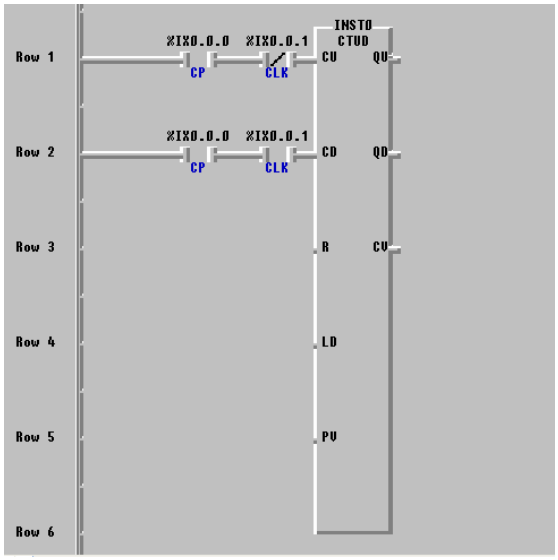


با فرض اولیه A=0 و B=0 به راحتی می توان مدار بالا را تحلیل کرد.

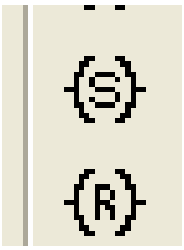
در نهایت با توجه به سیگنال کنترلی و پالس ساعت برای حرکت از AB مقدار صفر و برای حرکت از BA مقدار یک در خروجی سیگنال کنترلی ظاهر می شود .

مثال ۱۸:

برنامه ای بنویسید که با داشتن CP و CLK شمارش را انجام دهد ؟

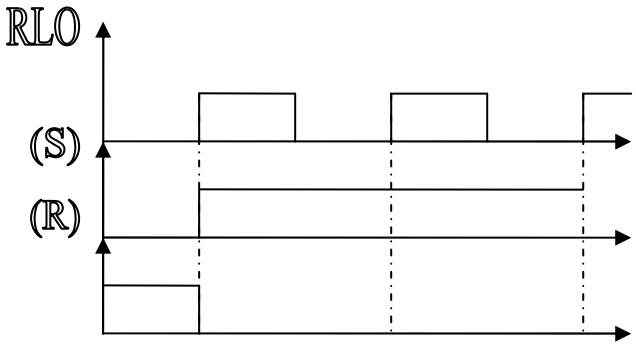


توضیح عملکرد Set **و** Reset **.**



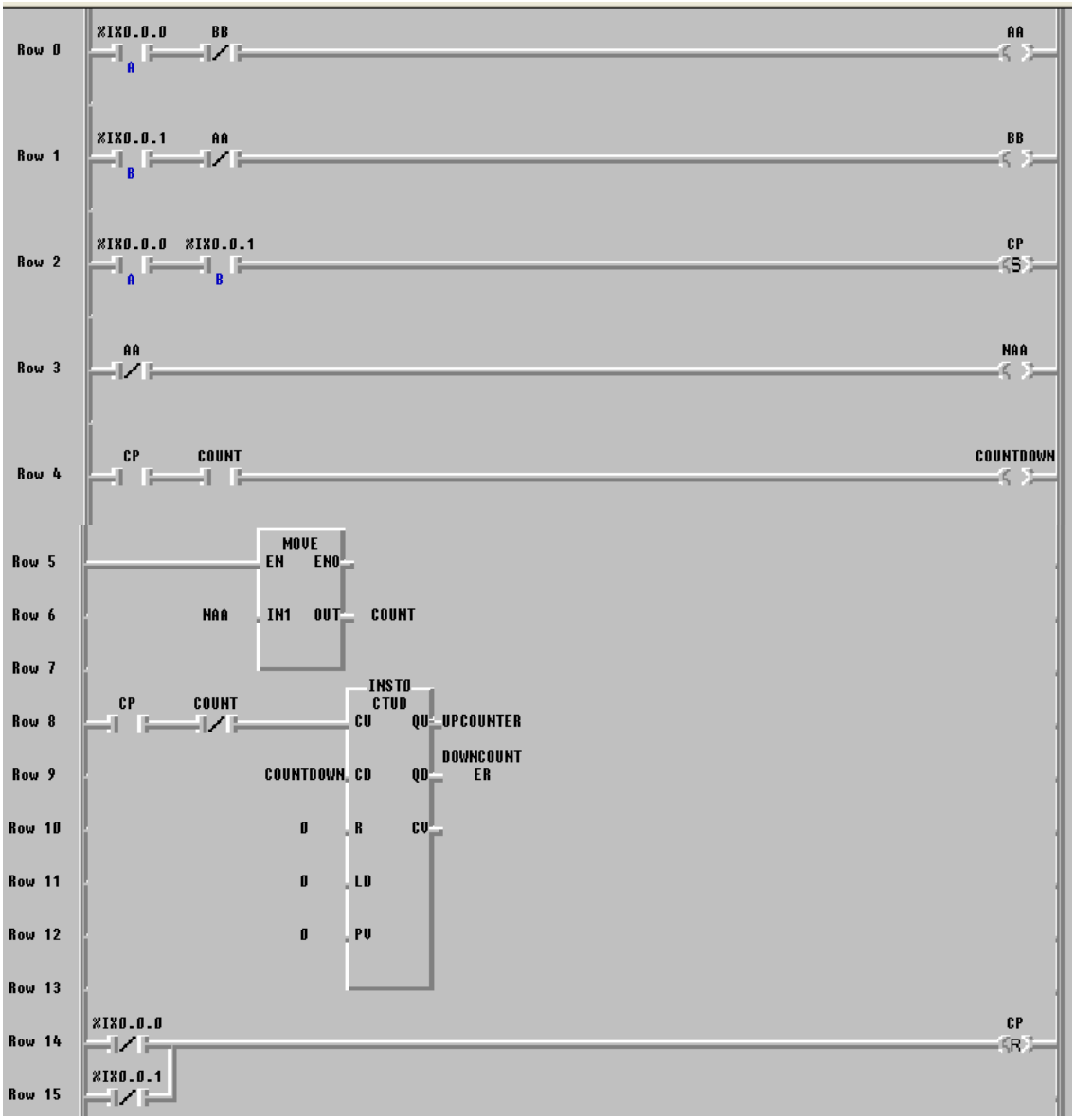
در میله ابزار می توان با کلیک بر روی اشکال روبرو به Set و Reset دسترسی داشت .

RLO : نتیجه عمل منطقی سمت چپ .



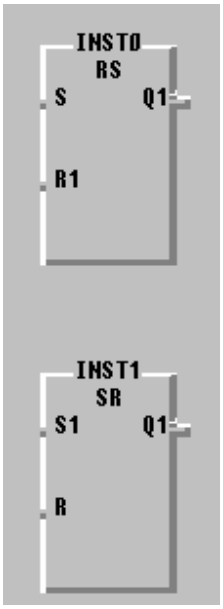
مثال ۱۸َ:

برنامه ای بنویسید که با داشتن A و B سیگنال های CP و CLK را تولید کند؟



فلیپ فلاپ RS و SR (فانکش بلاک):

عملکرد آنها مانند فلیپ فلاپ های معمولی است فقط با این تفاوت که برای حالت های ورودی ۱ و ۱ نیز مقادیری در نظر گرفته شده است .



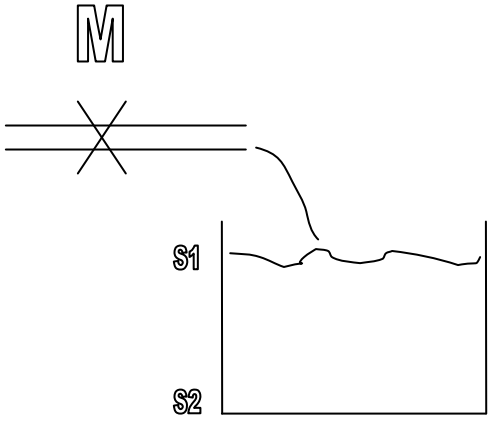
S	R	Q	
1	0	1	
0	0	1	حفظ حالت قبل
0	1	0	
0	0	0	حفظ حالت قبل
1	1	0	

R	S	Q	
1	0	1	
0	0	1	حفظ حالت قبل
0	1	0	
0	0	0	حفظ حالت قبل
1	1	1	

مثال :

اگر برای کنترل سطح آب نیاز بخواهیم از PLC استفاده کنیم به روش زیر عمل می کنیم :

سنسور های S1 و S2 می توانند از نوع سنسورهای مغناطیسی باشند که با فعال شدن آنها موتور روشن یا خاموش می شود .



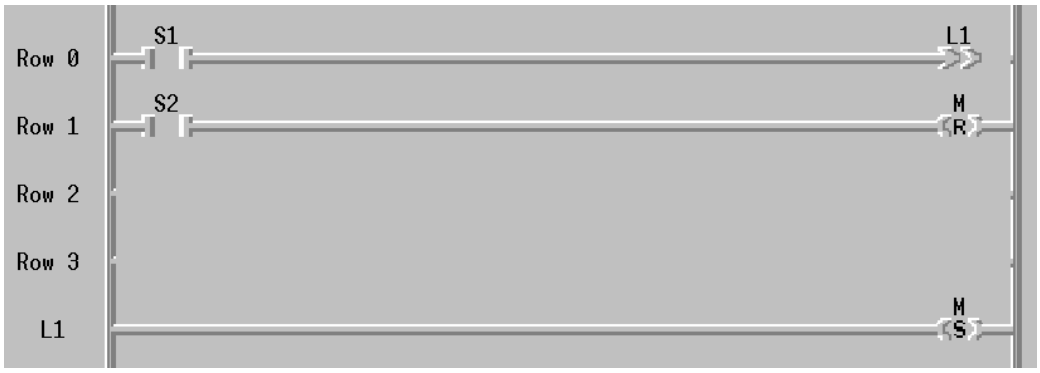
چند نکته در مورد برنامه نویسی PLC:

دستورات jump , subroutine call , return .

۱– دستور jump :

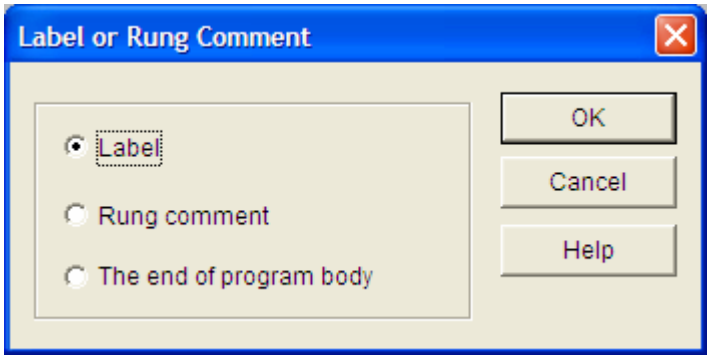
یک پرش بدون قید و شرط است اگر RLO (نتیجه عمل منطقی سمت چپ) برابر یک باشد به برچسب مشخص شده پرش می کند و خطوط برنامه بین سطر فعلی تا برچسب را اجرا نمی کند.

مثال :



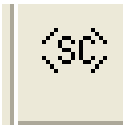
نکته:

برای قرار دادن یک برچست در برنامه باید به سطر مورد نظر رفته و بر روی شماره سطر ۲ بار کلیک کنیم . بعد از پنجره باز شده گزینه اول را انتخاب می کنیم .



برنامه بالا اگر S1 برابر یک باشد ، خطوط از ۱ تا ۳ را اجرا نخواهد کرد .

۲– Subrotine Call :



برنامه ای که جزو برنامه اصلی نیست و مانند تابع فراخوانی می شود .

برای قرار دادن یک Subrotine Call در برنامه باید مراحل زیر را طی نمود :

۱– تعیین پایان برنامه که با دوبار کلیک بر روی عنوان سطر آخر و انتخاب گزینه آخر از پنجره باز شده می باشد .

۲- قرار دادن یک برچسب در خط اول Subrotine مورد نظر .

۳- در پایان Subrotine نیز باید return قرار گیرد .

مثال ۱۹:

برنامه ای بنویسید که ۲ عدد را گرفته و زمانی که یک کلید فشرده شد ، با فراخـوانی یـک Subrotine آنها رو با هم جمع کند Call .