فصل چهارم

۴-۱- روش برنامه‌نویسی

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

۱- تعریف پروژه و فرآیند تحت کنترل

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

۲- رسم فلوچارت[۱] برنامه

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

۳- تهیهٔ لیستی از ابزار مورد نیاز

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

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

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

۵- در نظر گرفتن شرایط ایمنی و اقدامات حفاظتی

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

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

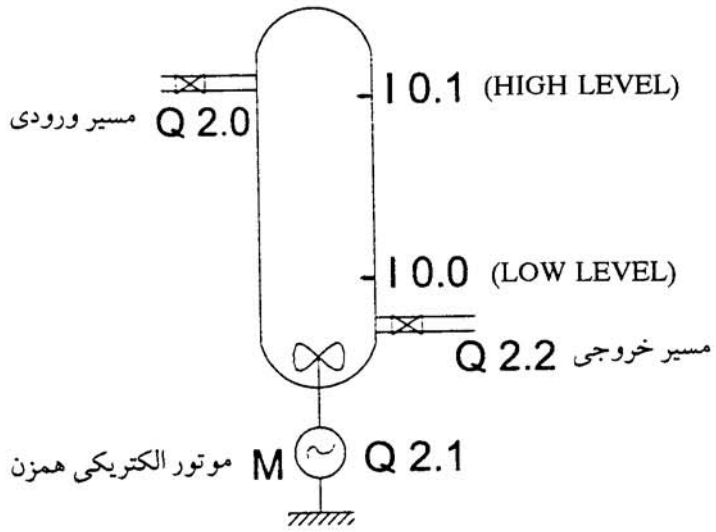
---

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

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

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

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



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

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

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

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

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

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

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

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

نحوهٔ ارائهٔ مطالب در این مثالها به گونه‌ای است که با توضیحات مفصل، تمامی خوانندگان (حتی خوانندگانی که تجربه‌ای در مورد پروژه‌های صنعتی ندارند) بتوانند در موارد بعدی، مراحل برنامه‌نویسی را خود دنبال نمایند. اکثر این مثالها با استفاده از شکلهای مناسب و یا شبیه‌سازهای[1] صنعتی و غیرصنعتی مدل‌سازی شده‌اند و اجرای برنامه را می‌توان به طور کامل و مرحله به مرحله بر روی این شبیه‌سازها دنبال نمود.

مثال ۴-۲ : برنامهٔ چراغ راهنمایی:

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

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

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

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

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

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

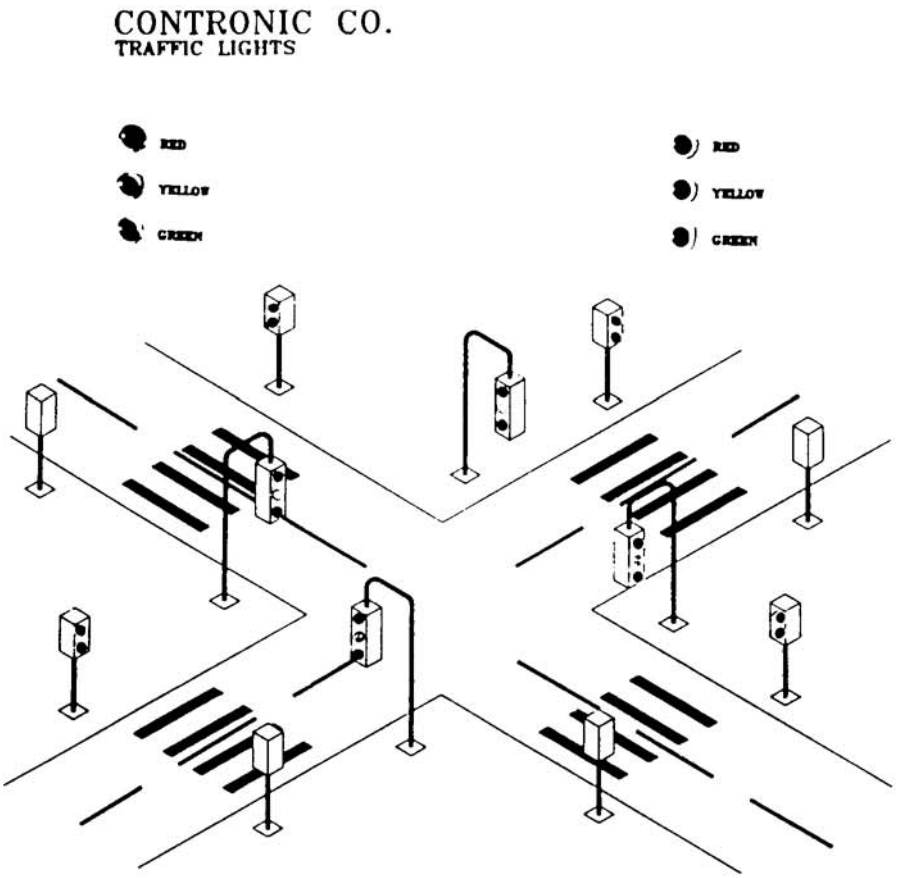
---

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

سمت دیگر چهارراه روشن باشد.

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

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



شکل ۴-۲ : شبیه‌ساز برنامهٔ چراغ راهنمایی

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

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

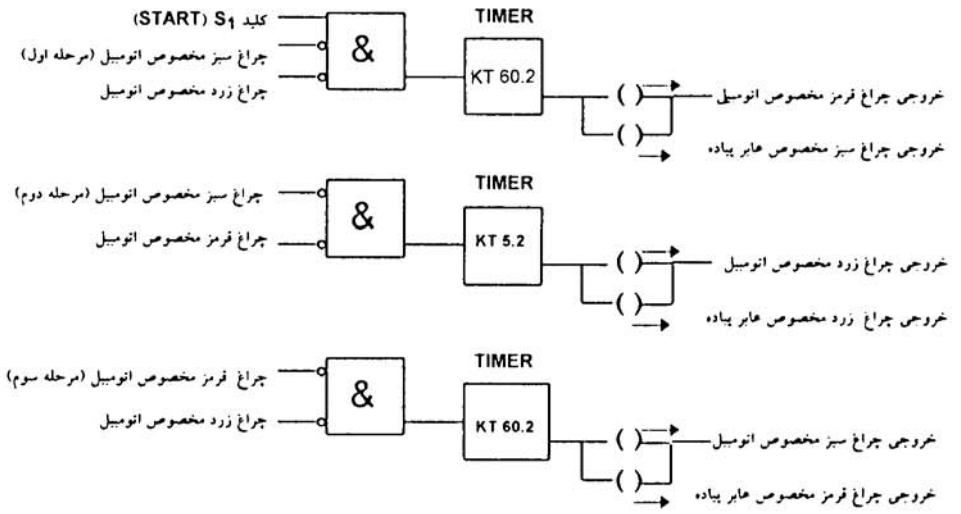
۲- رسم فلوچارت: از آنجایی که این پروژه شامل برنامه‌نویسی برای ۳ حالت می‌باشد برای رسم فلوچارت آن از سه مرحله استفاده می‌کنیم. این مراحل عبارتند از:

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

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

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

در ادامه، فلوچارت هر سه مرحله آورده شده است.

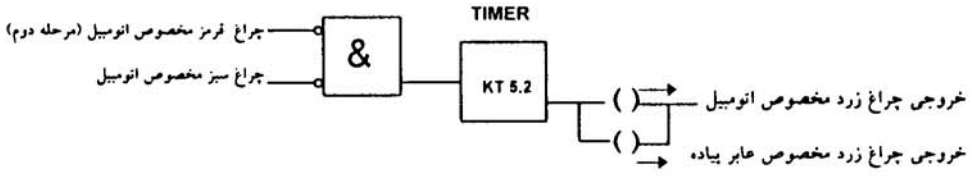


در رسم این فلوچارت، شرایط ایمنی نیز در نظر گرفته شده است. به عنوان مثال در مرحلهٔ اول برای به کار افتادن تایمر باید کلید استارت فعال باشد. این مورد تنها شرط لازم جهت ست نمودن تایمر نیست. همان‌گونه که در فلوچارت مرحلهٔ اول ملاحظه می‌کنید از نقیض خروجی مراحل دیگر یعنی نقیض خروجی چراغ سبز و چراغ زرد مخصوص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

برده شده‌اند.

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

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



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

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

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

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

در این برنامه چون به ورودی R در تایمر و همچنین خروجی‌های BI و DE نیازی نیست از تعریف آنها نیز در فلوچارت خودداری و در برنامه‌نویسی به روش STL (در مرحلهٔ بعدی برنامه‌نویسی) به جای این ورودی و خروجی‌ها از دستور NOP 0 استفاده می‌کنیم.

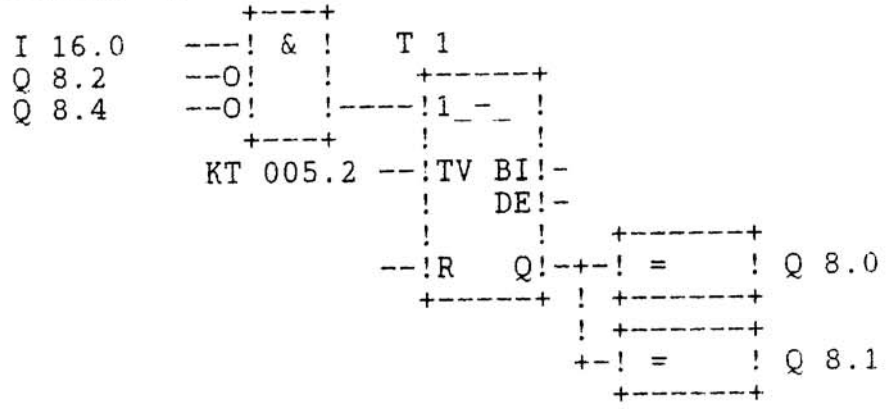
۴- نوشتن برنامه به یکی از سه روش برنامه‌نویسی: در اینجا برای تمرین بیشتر از دو روش STL و CSF جهت برنامه‌نویسی این پروژه استفاده شده است. برنامهٔ مذکور در PB 10 و در سه بخش (Segment) نوشته شده است.

```
 PB 10

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


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

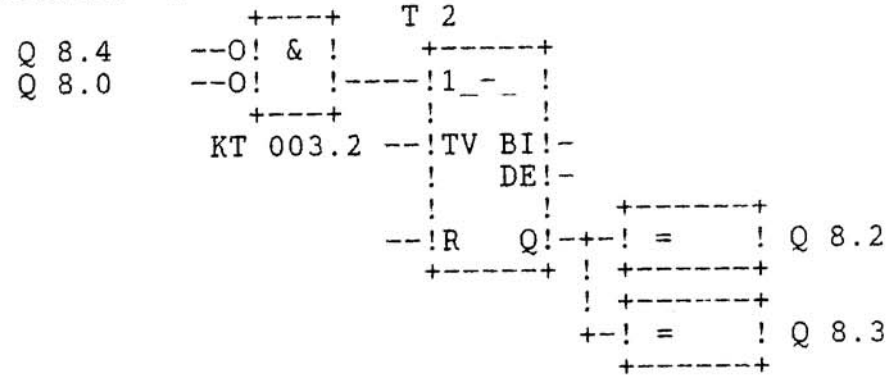
```
SEGMENT  3            0019
0019      :AN  Q     8.0
001A      :AN  Q     8.2
001B      :L   KT  005.2
001D      :SP  T     3
001E      :NOP 0
001F      :NOP 0
0020      :NOP 0
0021      :A   T     3
0022      :=   Q     8.4
0023      :=   Q     8.5
0024      :BE
```

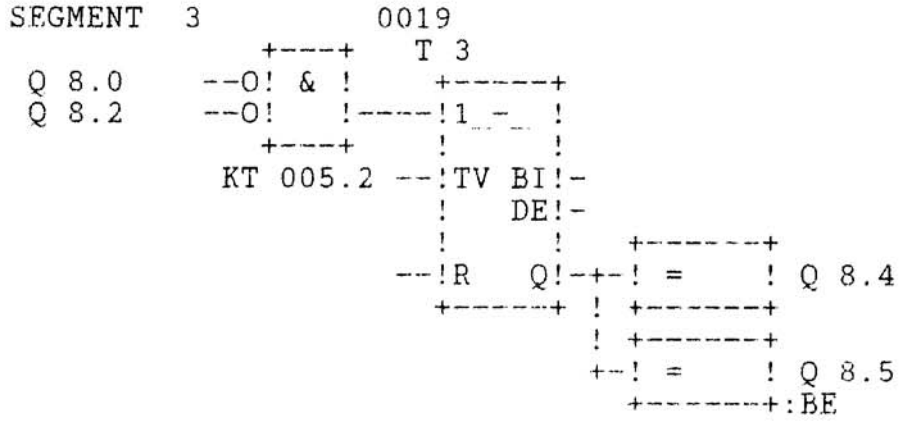
PB 10

SEGMENT 1 0000



SEGMENT 2 000D





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

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

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

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

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

برای حل این مشکل برنامهٔ PB 34 پیشنهاد می‌گردد:

```
 PB 34

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


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

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

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

2.0 Q برای مدت ۵۰ ثانیه و خروجی 2.1 Q برای مدت زمان ۱۰۰ ثانیه فعال خواهند بود.

حال به بررسی یک مشکل دیگر می‌پردازیم. در فصل قبل ذکر شد که حداکثر عددی که می‌توان در ورودی یک شمارنده اعمال نمود ۹۹۹ است. در صورتی که نیاز به شمارش عددی بیش از این مقدار داشته باشیم راه حل مناسبی پیشنهاد کنید.

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

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

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

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

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

۴-۲- بلوک‌های اطلاعاتی (DB)

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

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

می‌شود. این‌گونه پیغامها در DB وجود دارد و PLC به هنگام ایجاد مشکل در سیستم، پیغامهای مناسب را از DB خوانده، به صفحهٔ مونیتور می‌فرستد. در PLC مورد بحث ما می‌توان ۲۵۶ بلوک اطلاعاتی (DB 0 - DB 255) تعریف نمود. هر DB می‌تواند شامل ۲۵۶ سطر باشد که در هر سطر آن ۱۶ بیت (۱ کلمه) وجود دارد. به همین دلیل، هر یک از اطلاعات موجود در سطرهای DB را یک DW (Data Word) می‌گویند.

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

۱- Data (مقادیر و پارامترها)

۲- Text (متن پیامها و ...)

۳- Bit Pattern (این اطلاعات شامل تعدادی بیت‌های "۰" و "۱" است که به صورت بایتی یا کلمه‌ای به‌خروجی ارسال‌شده، عمل‌سیگنالینگ یا فعال وغیرفعال نمودن‌خروجی‌ها را برعهده دارند)

در فصل گذشته دیدیم که در مثال ۳-۱۵ با استفاده از دستورات JU و JC که باعث پرش از یک بلوک به بلوک دیگر می‌شوند می‌توان بلوک‌های PB را اجرا نمود. اما در مورد DBها فراخوانی اطلاعات با دستور دیگری انجام می‌گیرد. برای خواندن اطلاعاتِ هر DB ابتدا باید آن را در طول اجرای برنامه صدا[1] زد.

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

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

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

1 - CALL

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

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

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

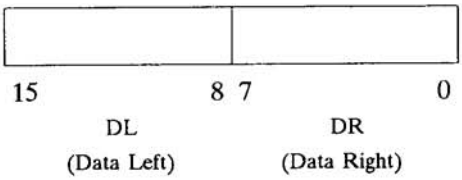
۳) KT : 14 BITS $(001.0 \longrightarrow 999.3)$ برای اعداد ثابت TV

۴) KC : 12 BITS $(000 \longrightarrow 999)$ برای اعداد ثابت شمارنده‌ها

یادآوری: اعداد با فرمت KT و KC در مبنای BCD می‌باشند.

۵) KY : 16 BITS (2 BYTES) در این حالت ۱۶ بیت به دو بایت چپ و راست تقسیم می‌شوند.

این دو بایت کاملاً مجزا بوده، هر یک از آنها می‌تواند مقادیر 000 الی 255 را داشته باشد.



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

۱۶ بیت ۱۶ بیت

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

جهت نمایش اعداد با ممیز اعشاری (Floating Point) و نیز اعداد بسیار بزرگ یا بسیار کوچک از این شکل نمایش استفاده می‌شود. مثلاً عدد ۰۹ ± ۱۵۲۴۶۷۰ را در نظر بگیرید، در PLC زیمنس این عدد ابتدا به 10^{6} تقسیم می‌شود. در مرحلهٔ بعد در صورت مثبت بودن علامت توان، PLC عدد حاصل را در 10^{9} و در صورت منفی بودنِ علامت توان، آن را در 10^{-9} ضرب می‌کند. محدودیتی که در این روش وجود دارد آن است که دو رقم مشخص کنندهٔ توان حداکثر ۳۸ می‌باشد.

۸) KS : TANK LEVEL LOW.

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

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

DB 25:

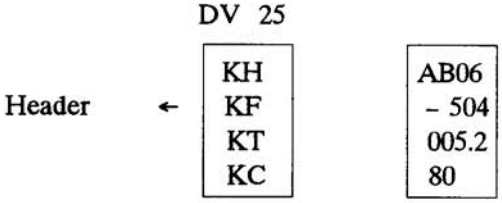
KH = AB06

KF = - 504

KT = 005.2

KC = 080

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



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

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

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

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

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

```
PB 20                                    PB 22

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

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

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

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

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

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

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

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

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

```
 PB 30

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


 DB27

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

روند اجرای برنامه به صورت زیر است:

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

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

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

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

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

برای نوشتن این برنامه از برنامهٔ نوشته شده در مثال ۴-۴ استفاده می‌کنیم. با این تفاوت که در 27 DB، سطر دیگری را که همان KF = 2000 است وارد نموده، با ایجاد تغییراتی در 30 PB برنامه موجب می‌شویم که تنها ۲۰۰۰ سیکل از برنامه اجرا شود.

PB 31

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

```
DB28

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

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

در اینجا به ذکر یک سؤال می‌پردازیم:

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

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

در وضعیت روشن باقی می‌ماند.

– در شمارش تعداد سیکل‌های اجرای برنامه (تعداد ۲۰۰۰ سیکل برنامه) لازم است که شمارش حتماً از عدد صفر آغاز گردد بنابراین لازم است که با فشار دادن یک کلید، شمارش تعداد سیکل‌های برنامه را از صفر آغاز نموده، تا ۲۰۰۰ ادامه یابد. برای این مسأله چه برنامه‌ای پیشنهاد می‌کنید؟

همان‌گونه که در فصل سوم توضیح داده شد بلوک OB 1 ساختار برنامهٔ استفاده کننده را مشخص می‌کند زیرا سیستم عامل در شروع هر سیکل برنامه به بلوک OB 1 رجوع می‌کند. بنابراین کلید معرف آغاز شمارش را در این بلوک تعریف می‌کنیم. برنامهٔ خواسته شده در ادامه آمده است.

```
 OB 1

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


 PB 31

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

 PB 40

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

```
DB28

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

روند اجرای برنامه به صورت زیر است:

در سطر اول بلوک OB 1، بلوک اطلاعاتی ۲۸ یعنی DB 28 فعال می‌گردد تا تمامی اطلاعات از این بلوک خوانده شود. در صورتی که کلید استارت شمارش یا I 0.0 برابر "۱" شود PB 31 که همان برنامهٔ قبلی است اجرا می‌شود. در این برنامه، شمارش از صفر شروع شده، تا ۲۰۰۰ ادامه می‌یابد. ولی اگر کلید استارت شمارش غیرفعال باشد PB 40 اجرا خواهد شد. در این برنامه ابتدا عدد صفر در DW 0 فرستاده می‌شود، در سطر سوم با ری‌ست نمودن بیت خروجی Q 2.0 امکان انجام این کار برای چندین بار عملی شده است.

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

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

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

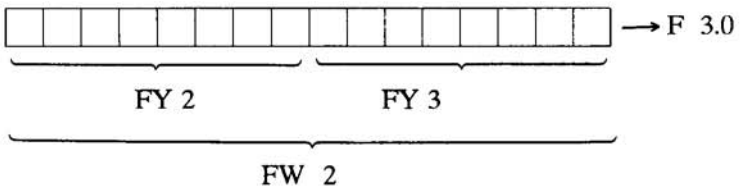
این برنامه به صورت زیر نوشته می‌شود.

```
 OB 1

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

```
 PB 32

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


DB20

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

در زیر اجرای برنامه تشریح شده است.

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



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

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

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

در مورد PLC مورد بحث، عدد به دست آمده در شمارنده، ۳۰۲ می‌باشد که با تقسیم آن بر ۲، عدد ۱۵۱ حاصل می‌شود. این حاصل بدان معنی است که در طول ۱ ثانیه ۱۵۱ سیکل از برنامهٔ PB 32 انجام شده است. بنابراین مدت زمان اجرای یک سیکل از این برنامه $\frac{۱}{۱۵۱}$ ثانیه یا تقریباً ۶ میلی‌ثانیه است. همان‌گونه که قبلاً ذکر شد در مواردی که با فاصله‌های زمانی کوچک سروکار داریم و یا نیاز به دقت در محاسبهٔ زمان باشد از تولرانس‌های ۰ و ۱ استفاده می‌کنیم. بنابراین به جای استفاده از L KT 2.2 در این برنامه می‌توان از دستور L KT 20.1 و یا L KT 200.0 نیز استفاده نمود.

یکی از مواردی که در فرآیندهای صنعتی و خطوط تولید با آن روبرو می‌شویم محاسبهٔ زمان دقیق در مورد انجام هر قسمت از خط یا فرآیند است. می‌توان همین برنامه را در انتهای برنامهٔ کنترلی هر قسمت از خط نوشته، برای هر قسمت سرعت یک سیکل زمانی را اندازه‌گیری نماییم. در این برنامه روش اندازه‌گیری زمان یک سیکل اجرای برنامه تغییر می‌کند و دیگر نیازی به طی ۲۰۰۰ سیکل و سپس تقسیم زمان به دست آمده بر ۲۰۰۰ نیست. همان‌گونه که در این برنامه ملاحظه کردید تعداد سیکل‌های پیموده شده در طی x ثانیه محاسبه می‌شود و سپس زمان اجرای یک سیکل برنامه از رابطهٔ $\frac{x}{\text{تعداد سیکل‌های انجام شده در مدت x ثانیه}}$ به دست می‌آید.

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

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

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

```
 OB 1

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


 PB 25

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

ادامهٔ بلوک PB 25:

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

```
 PB 26

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

```
DB20

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

روند اجرای برنامه به صورت زیر است:

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

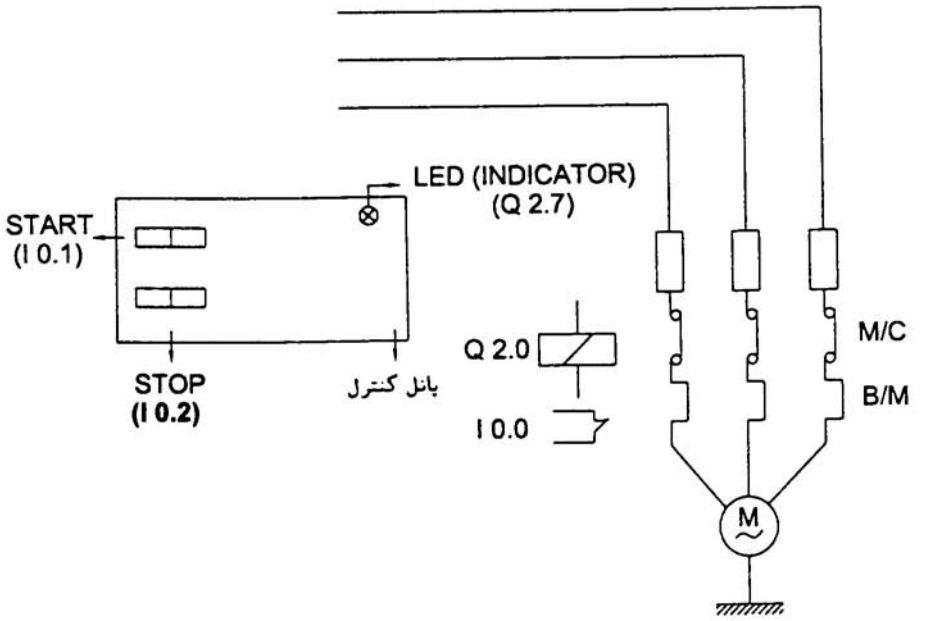
مخصوص اتومبیل و عابر پیاده فعال و سایر چراغها غیرفعال می‌گردند. روند اجرای مرحله سوم یعنی ارسال فرمان به خروجی‌های چراغهای سبز مخصوص اتومبیل و قرمز مخصوص عابر پیاده نیز مانند مراحل قبل است. در حقیقت زمان روشن بودن چراغهای سه مرحله به ترتیب 1500 و 1000 = (2500-1500) و 1500 = (4000-2500) برابر سیکل زمانی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

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

هنگامی که عدد DW 0 به هر دلیلی بیشتر از ۴۰۰۰ باشد پردازنده باید محتوای DW 0 را با عدد ۶۵۵۳۵ ($FFFF_H$) پر نموده، مجدداً صفر شود ولی با استفاده از I 0.0 در هر لحظه امکان پرش به PB 26 میسر است و این کلید کنترل دستی در واقع برای شروع از عدد صفر می‌باشد.

در این برنامه ملاحظه نمودید که به سادگی و بدون استفاده از تایمر، توانستیم ۳ تایمر تعریف کنیم. همانگونه که در فصل قبل نیز ذکر شد با اجرای چنین برنامه‌هایی می‌توان بدون استفاده از تایمر در برنامه، n تایمر را تعریف نمود.

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



برنامهٔ کنترل مورد نظر در ادامه آمده است:

```
PB 41

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

در برنامهٔ فوق در صورتی که کلید START فعال شود خروجی Q 2.0 یعنی فرمان اتصال سه فاز موتور صادر شده، موتور به کار می‌افتد و در صورت فعال شدن کلید STOP موتور خاموش می‌شود. حال اگر کنتاکتور B/M سالم و یا به عبارت دیگر I 0.0 = "۱" باشد، و در این حالت موتور نیز روشن بماند F 10.0 به حالت "۱" ثابت می‌ماند ولی چنانچه کنتاکتور B/M قطع شده، موتور کار کند، فلگ F 21.2 که با فاصلهٔ زمانی خاص فعال و غیرفعال می‌گردد به F 10.1 نسبت داده می‌شود و حاصل ترکیب فصلی دو فلگ F 10.0 و F 10.1 به خروجی Q 2.7 ارسال می‌گردد.

۴-۳- بلوک‌های تابع‌ساز (FB)

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

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

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

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

۲- بدنهٔ بلوک (Block Body) که شامل توابع و دستوراتی است که می‌بایست در FB قرار گیرند. علاوه بر دستورات معمول، گروهی از دستورها که دستورالعمل‌های تکمیلی نامیده می‌شوند در این بلوک مورداستفاده قرار می‌گیرند. به طورکلی می‌توان FB ها را به دو دستهٔ زیر تقسیم نمود:

۱- بلوک تابع‌ساز استاندارد (Standard FB)

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

۲- بلوک تابع‌ساز انتسابی (Assignable FB)

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

در ادامهٔ بحث درمورد برنامه‌نویسی FBها، مثالهایی از توابع استاندارد و انتسابی ارائه خواهدشد.

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

```
 PB 40

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

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

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

شکل ۴-۳ : نمایش شماتیکی بلوک استاندارد FB 242

اکنون به توضیح در مورد پارامترهای این بلوک می‌پردازیم:

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

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

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

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

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

Z32 (QW 14) Z31 (QW 8)

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

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

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

نمایش می‌دهیم.

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

$۲۵_{(۱۰)} = ۱۱۰۰۱_{(۲)}$ = ۰۰۰۰۰۰۰۰۰۰۰۱۱۰۰۱ ⟶ IW 21

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

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

Z3 = ۰ ⟶ Q 10.0

$Z31 = ۰۰۰۰۰۰۰۰۰۱۰۱۱۱۱۰_{(۲)}$ ⟶ QW 8

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

$Z32 = ۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰_{(۲)}$ ⟶ QW 14

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

حال اگر بخواهیم حاصلضرب دو عدد $۵۶۹۸_{(۱۰)}$ و $۳۲۶۵_{(۱۰)}$ را به دست آوریم همان روند قبلی را طی کرده، خواهیم داشت:

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

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

خروجی‌ها به صورت زیر می‌باشند:

Z3 = ۰

$Z31 = ۱۱۰۱۱۱۱۱۱۱۰۰۰۰۱۰_{(۲)}$ ⟶ QW 8

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

نتیجهٔ حاصلضرب = Z32 Z31 = ۰۰۰۰۰۰۰۱۰۰۰۱۱۰۱۱ ۱۱۰۱۱۱۱۱۱۱۰۰۰۰۱۰

$= ۱۸۶۰۳۹۷۰_{(۱۰)}$

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

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

۸ دستهٔ ۴ بیتی

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

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

$$= ۱۸۶۰۳۹۷۰_{(۱۰)}$$

اکنون به ذکر مثالی در مورد FB Assignable می‌پردازیم.

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

در صورتی که بخواهیم حاصل تابع منطقی XOR دو ورودی I 0.1 و I 0.0 را در Q 2.0 قرار دهیم برنامهٔ PB 10 را خواهیم داشت:

```
 PB 10

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

 PB 10

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

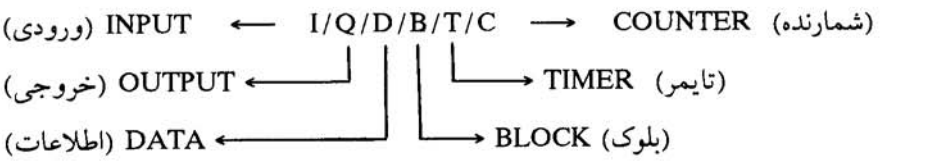
مثال ۴-۱۰ : با استفاده از تعریف تابع منطقی XOR در بلوک تابع‌ساز، برنامه‌ای بنویسید که هرگاه تنها یکی از دو موتور با شماره‌های ۱ و ۲ روشن باشد LED موجود بر روی پانل کنترل روشن شود. برای نوشتن این برنامه به دو بلوک FB و PB نیاز داریم، که باید در بلوک FB تابع XOR را به همراه پارامترهای موردنظر تعریف کنیم.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

```
 PB 3

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


 FB 20

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

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

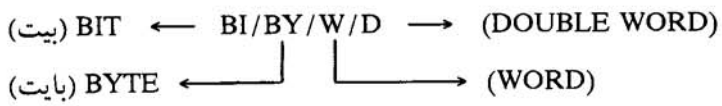
روند وارد نمودن اطلاعات به شرح زیر است:

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

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



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



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

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

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

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

عملوندهای مورد نظر نسبت دهیم.

```
 PB 3

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

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

```
 OB 1

 SEGMENT 1 :

            : JU   PB   3

            : BE
```

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

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

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

```
 PB 6

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

۴-۴- دستورات تکمیلی (Supplementary)

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

۴-۴-۱- دستور AW[1]

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

---

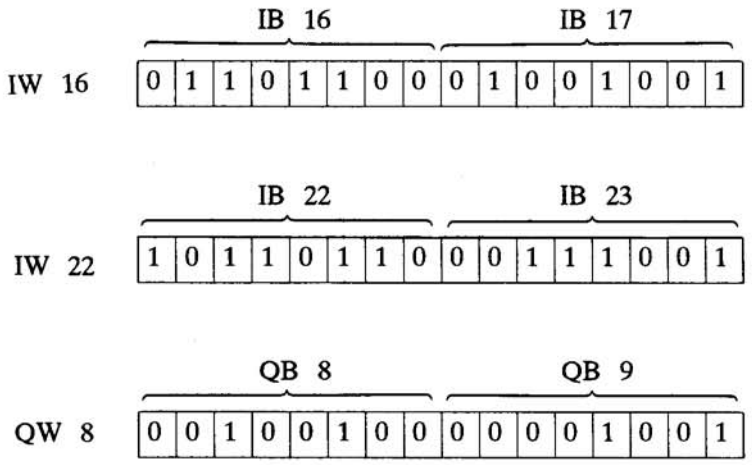
1 - AND WORD

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

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

```
 FB 10

SEGMENT  1              0000
NAME :AND

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

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



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

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

```
 PB 4

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

۴-۴-۲- کاربرد عملی دستور AW

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

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

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

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

```
 FB 11

SEGMENT  1              0000
NAME :AND1

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

KH AAAA	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
IW 14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
QW 2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

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

---

1 - Mask

۴-۴-۳- دستور OW[1]

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

```
 FB 12

SEGMENT  1           0000
NAME :OR

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

با استفاده از دستور OW می‌توانیم ترکیبی از ورودی‌ها را در خروجی ایجاد نمائیم، که به این عمل ترکیب[2] ورودی‌ها نیز گویند.

۴-۴-۳- دستور XOW[3]

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

```
 FB 13

SEGMENT  1           0000
NAME :XR

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

1 - OR WORD 2 - Combination 3 - XOR WORD

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

۴-۴-۵- دستور CFW[1]

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

همان‌گونه که می‌دانید مکمل "۰"، "۱" و مکمل "۱"، "۰" می‌باشد، بنابراین برای به دست آوردن مکمل ۱ یک کلمه ورودی کافی است ارزش بیت‌های آن را عکس نمود. در ادامه، یک برنامه با استفاده از این دستور نوشته شده است.

```
 FB 14

SEGMENT  1              0000
NAME :FIRS

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

IW 16	1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	1
QW 8	0	1	0	0	1	1	1	0	1	0	1	1	0	0	1	0

در این برنامه مکمل ۱ بیت‌های کلمهٔ ورودی ۱۶ در کلمهٔ خروجی ۸ قرار گرفته‌اند یا به عبارت دیگر ارزش بیت‌های ورودی عکس شده است.

---

1 - 1st Complement Word

۴-۴-۶- دستور CSW[1]

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

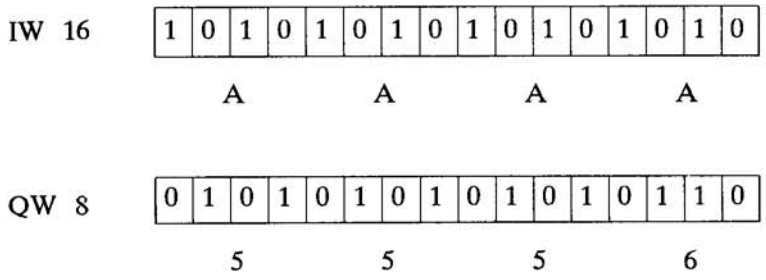
در زیر برنامه‌ای شامل این دستور آورده شده است:

```
 FB 15

SEGMENT  1              0000
NAME :TWOS

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

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

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

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



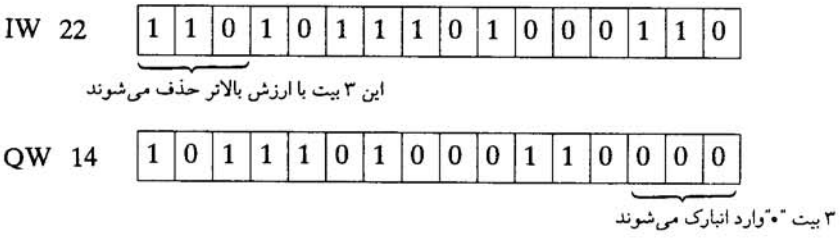
1 - 2^{s} Complement Word

۴-۴-۷- دستور SLW[1]

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

```
 FB 16

SEGMENT  1              0000
NAME :SHIFT

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

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



1 - Shift Left Word

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

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

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

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

IW 22	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	$۱۰_{(۲)} = ۲$
SLW 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	$۱۰۰_{(۲)} = ۴$
SLW 2	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	$۱۰۰۰_{(۲)} = ۸$
SLW 3	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	$۱۰۰۰۰_{(۲)} = ۱۶$

همان‌گونه که دیده می‌شود با اجرای دستور SLW 1 بر روی IW 22 که حاوی عدد $۲_{(۱۰)}$ می‌باشد، عدد $۴_{(۱۰)}$ یعنی ۲ برابر مقدار اولیه به‌دست می‌آید. با اجرای دستور SLW 2 ، حاصل $۸_{(۱۰)}$ یعنی $۲^۲$ برابر عدد اول یعنی محتویات اولیهٔ IW 22 به‌دست می‌آید و در دستور SLW 3 نتیجهٔ حاصل شده $۲^۳$ برابر عدد اولیه خواهد بود. پس نتیجه می‌گیریم که با هر بار حرکت بیت‌ها به سمت چپ، عدد به‌دست آمده ۲ برابر می‌شود. بنابراین دستور SLW 5 عدد اولیهٔ موجود در IW 22 را در ۳۲ ($۲^۵$) ضرب می‌کند. اما جواب این حاصلضرب تنها هنگامی صحیح خواهد بود که عدد اول موجود در IW 22 به هنگام شیفت به سمت چپ، بیت‌های با ارزش بالای خود که حاوی بیت "۱" می‌باشد را از دست ندهد. پس می‌توان گفت که برای ضرب اعداد در توانهایی از ۲ به عنوان مثال ۲، ۴، ۸، ۱۶ و ... از دستورات SLW 1 ، SLW 2 ، SLW 3 ، SLW 4 و ... استفاده می‌شود به شرطی که پس از شیفت، بیت‌های با ارزش بالا با محتوای "۱" حذف نگردند.

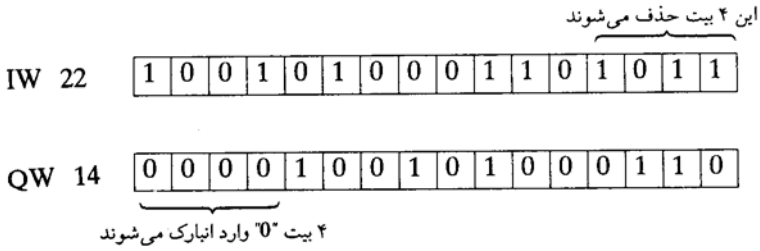
۴-۴-۸- دستور SRW[1]

عملکرد این دستور دقیقاً بر عکس دستور SLW است به این صورت که بیت‌های کلمهٔ ورودی را به سمت راست حرکت می‌دهد. تعداد انتقال بیت‌ها از ۰ تا ۱۵ قابل تغییر است. بلوک FB 17 را در نظر بگیرید:

```
 FB 17

SEGMENT  1            0000
NAME :RIGHT

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

در برنامهٔ فوق پس از بارگذاری کلمهٔ ورودی ۲۲ در انبارک، تک‌تک بیت‌ها به اندازهٔ ۴ بیت به سمت راست حرکت داده می‌شوند. در سمت چپ یا به عبارت دیگر بیت‌های با ارزش بالاتر ۴ بیت "۰" وارد انبارک شده، ۴ بیت از بیت‌های با ارزش پائین‌تر نیز حذف می‌گردند.

در این حالت نیز چرخش بیت‌ها وجود ندارد. در ادامه محتویات اولیهٔ IW 22 و همچنین محتویات QW 14 پس از انجام عمل شیفت نشان داده شده است.



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

---

1 - Shift Right Word

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

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

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

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

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

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

با مقایسهٔ اعداد به‌دست آمده در هر قسمت با عدد اول یعنی $۱۹۲_{(۱۰)}$ ملاحظه می‌شود که با اجرای هر بار دستور SRW نصف عدد قبلی به دست می‌آید. با اجرای دستور SRW 1 عدد اول بر $۲^۱$ تقسیم شده است. با اجرای دستور SRW 2 عدد به‌دست آمده برابر $\frac{۱}{۴}$ $(\frac{۱}{۲^۲})$ عدد اول است، با اجرای دستور SRW 3 به $\frac{۱}{۸}$ $(\frac{۱}{۲^۳})$ عدد اول دست خواهیم یافت.

مثال **۴-۱۴ :** عدد $۲۱۵_{(۱۰)}$ را در نظر گرفته، دستور SRW 1 را در مورد آن اعمال نمائید. پس از مقایسهٔ عدد به‌دست آمده با عدد اول چه نتیجه‌ای می‌گیرید؟

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

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

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

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

$\neq \frac{۲۱۵}{۲}$

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

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

۴-۴-۹- دستور I[1]

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

```
FB 18

SEGMENT  1              0000
NAME :PLUS

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

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

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

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

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

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

```
FB 19

SEGMENT  1              0000
NAME :PLUS1

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

1 - Increament

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

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

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

۴-۴-۱۰- دستور D[1]

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

```
 FB 20

SEGMENT  1          0000
NAME :MINUS

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

در اجرای برنامهٔ این بلوک از محتویات کلمهٔ ورودی ۲۲، ۱۲ واحد کسر می‌گردد.

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

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

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

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

```
 FB 21

SEGMENT  1          0000
NAME :MINU1
0005       :L   IW  22
0006       :D      100
0007       :T   QW   8
0008       :BE
```

1 - Decreament

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

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

QW 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | $۱۰۱۱۰۱۰۰۱_{(۲)} = ۳۶۱_{(۱۰)}$

۴-۴-۱۱- دستور ADD

در مورد دستورات I و D خاطر نشان کردیم که حداکثر مقداری که می‌توان به کلمهٔ ورودی اضافه و یا از آن کسر نمود ۲۵۵ واحد می‌باشد. حال در صورتی که در اجرای برخی از برنامه‌ها به جمع و یا تفریق نمودن یک مقدار ورودی با مقادیری بیش از مقادیر ذکر شده نیاز داشته باشیم با مشکل مواجه خواهیم شد. برای رفع این مشکل از دستور ADD استفاده می‌کنیم. به کمک این دستور می‌توان اعدادی را که در فاصلهٔ [۳۲۷۶۸+ و ۳۲۷۶۸-] قرار دارند با عدد مورد نظر جمع نمود. در هنگام استفاده از این دستور عدد اضافه شونده می‌تواند مثبت یا منفی باشد و با فرمت KF در برنامه استفاده شود.

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

```
 FB 54

SEGMENT  1            0000
NAME :ADD

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

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

```
 FB 55

SEGMENT  1            0000
NAME :MINUS

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

۴-۴-۱۲- دستور JZ[1]

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

```
 FB 50

SEGMENT  1            0000
NAME :JPZERO

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

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

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

1 - Jump if Zero

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

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

```
 FB 14

SEGMENT  1            0000
NAME :LABEL

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

۴-۴-۱۳- دستور JN[1]

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

---

1 - Jump if Not Zero

```
 FB 51

SEGMENT  1          0000
NAME :JPNZERO

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

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

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

```
 FB 45

SEGMENT  1          0000
NAME :NOT

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

۴-۴-۱۴- دستور JP[1]

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

```
 FB 52

SEGMENT  1              0000
NAME :JPPOSI

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

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

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

---

1 - Jump if Positive

```
 FB 46

SEGMENT  1          0000
NAME :JMINU

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

۴-۴-۱۵- دستور JM[1]

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

```
 FB 53

SEGMENT  1          0000
NAME :JMINUS

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

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

1 - Jump if Minus

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

```
 FB 47

SEGMENT  1            0000
NAME :JMINUS1

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

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

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

```
 FB 10

SEGMENT  1            0000
NAME :CYCLET

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

ادامهٔ بلوک FB 10 :

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

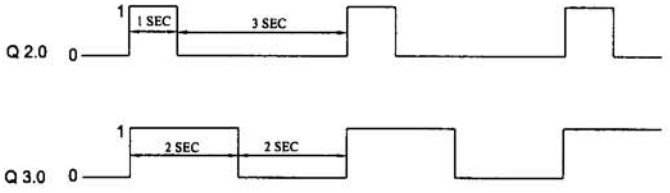
روند اجرای برنامه به ترتیب زیر است:

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

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

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

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



برنامهٔ مطلوب در PB 40 نوشته شده است.

```
PB 40

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

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

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

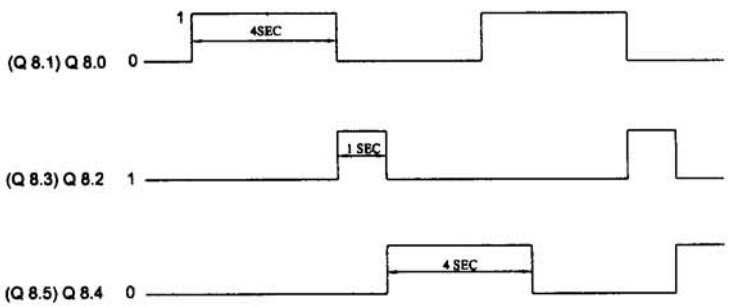
در صورتی که تایمر به 90 برسد خروجی Q 2.0 را ری‌ست می‌کند.

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

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

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

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



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

در اکثر پروسه‌ها و فرآیندهای صنعتی گاه لازم است که پیغام هشدار دهنده‌ای مثلاً روشن شدن یک چراغ (LED) چشمک زن به کاربران اعلام شود. سرعت چشمک زدن این چراغ نیز باید به گونه‌ای باشد که کاربران قدرت تشخیص آن را از فواصل دور و در شرایط کاری مختلف داشته باشند. برنامهٔ چراغ چشمک زن در بیشتر موارد با استفاده از تایمر نوشته می‌شود، در این مثال سعی شده تا این برنامه در یک بلوک FB نوشته شود. در این قسمت به ذکر یک مثال در این مورد می‌پردازیم.

مثال ۴-۲۰: برنامه‌ای برای یک چراغ چشمک زن[1] بنویسید که به هنگام صدا زدن FB مربوطه از کاربر دو سؤال بپرسد:

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

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

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

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

```
FB 100
SEGMENT  1              0000
NAME  :BLINK
DECL  :TIM          I/Q/D/B/T/C: T
DECL  :DATA         I/Q/D/B/T/C: D   KM/KH/KY/KS/KF/
                                     KT/KC/KG: KT
000B        :AN   =TIM                نام تایمر
000C        :LW   =DATA               زمان KT
000D        :SEC  =TIM                نوع تایمر
000E        :L    =TIM          خروجی باینری تایمر
000F        :T    FW   20     و ارسال به FW 20
0010        :BE
```

1 - Blinking LED

بلوک OB 1 نیز به صورت زیر برنامه‌نویسی می‌گردد.

```
 OB 1

SEGMENT  1            0000
0000       :JU   FB 100
0001 NAME  :BLINK
0002 TIM   :     T     8
0003 DATA  :     KT 050.1
0004       :A    F    21.2
0005       :=    Q     8.0
0006       :A    F    21.3
0007       :=    Q     9.0
0008       :BE
```

انتخاب شمارهٔ تایمر

زمان موردنظر در تایمر به همراه تولرانس آن

سرعت خاموش و روشن به فاصله زمانی هر ۰/۴ ثانیه یک بار ارسال فرمان روشن و خاموش شدن به خروجی Q 8.0

سرعت خاموش و روشن به فاصله زمانی هر ۰/۸ ثانیه یک بار ارسال فرمان روشن و خاموش شدن به خروجی Q 9.0