BÀI GIẢNG

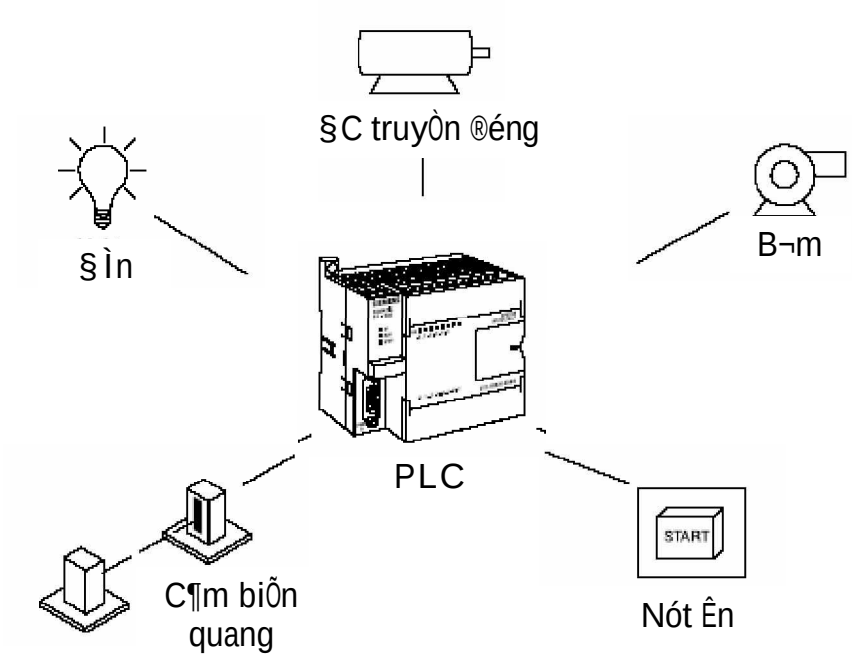
MÔN HỌC : LẬP TRÌNH PLC

Chương1.Giới thiệu tổng quan về PLC.

1.1.Khái niệm hệ thống điều khiển PLC:

PLC viết tắt của Programmable Logic Controller , là thiết bị điều khiển lập trình được (khả trình) cho phép thực hiện linh hoạt các thuật toán điều khiển logic thông qua một ngôn ngữ lập trình. Người sử dụng có thể lập trình để thực hiện một loạt trình tự các sự kiện. Các sự kiện này được kích hoạt bởi tác nhân kích thích (ngõ vào) tác động vào PLC hoặc qua các hoạt động có trễ như thời gian định thì hay các sự kiện được đếm. Một khi sự kiện được kích hoạt thật sự, nó bật ON hay OFF thiết bị điều khiển bên ngoài được gọi là thiết bị vật lý. Một bộ điều khiển lập trình sẽ liên tục "lặp" trong chương trình do "người sử dụng lập ra" chờ tín hiệu ở ngõ vào và xuất tín hiệu ở ngõ ra tại các thời điểm đã lập trình.

PLC (Programable Logic Controler) là một thiết bị điều khiển sử dụng một bộ nhớ có thể lập trình, bộ nhớ này sẽ lu giữ các cấu trúc lệnh (logic, thời gian, bộ đếm, các hàm toán học...) để thực hiện các chức năng điều khiển.



1.2. Cơ sở phát triển của PLC:

Để khắc phục những nhược điểm của bộ điều khiển dùng dây nối (bộ điều khiển bằng Relay) người ta đã chế tạo ra bộ PLC nhằm thỏa mãn các yêu cầu sau :

Lập trình dể dàng , ngôn ngữ lập trình dể học .

Gọn nhẹ, dể dàng bảo quản , sửa chữa.

Dung lượng bộ nhớ lớn để có thể chứa được những chương trình phức tạp .

Hoàn toàn tin cậy trog môi trường công nghiệp .

Giao tiếp được với các thiết bị thông minh khác như : máy tính , nối mạng , các môi Modul mở rộng.

Giá cả cá thể cạnh tranh được.

Sự khác nhau giữa hệ điều khiển bằng Rơle điện và lập trình có nhớ có thể minh hoạ bằng một ví dụ sau:

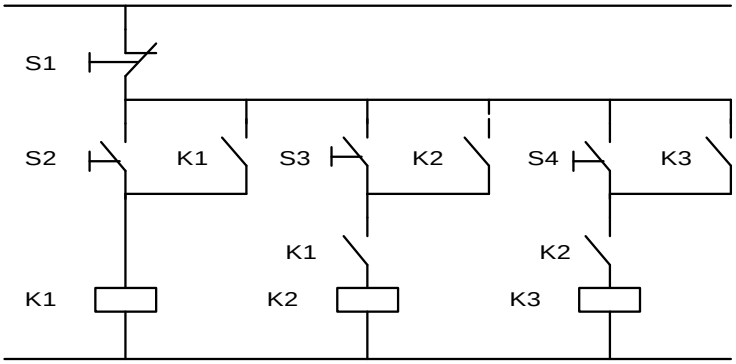
Ví Dụ: ***Điều khiển hệ thống 3 máy bơm nước qua 3 khởi động từ K1, K2, K3. Trình tự điều khiển như sau: Các máy bơm hoạt động tuần tự nghĩa là K1 đóng trước tiếp đến là K2 rồi cuối cùng là K3 đóng.***

Để thực hiện nhiệm vụ theo yêu cầu trên mạch điều khiển ta thiết kế như sau:

Trong đó các nút ấn S1, S2, S3, S4 là các phần tử nhập tín hiệu.

Các tiếp điểm K1, K2, K3 và các mối liên kết là các phần xử lý.

Các khởi động từ K1, K2, K3 là kết quả xử lý.



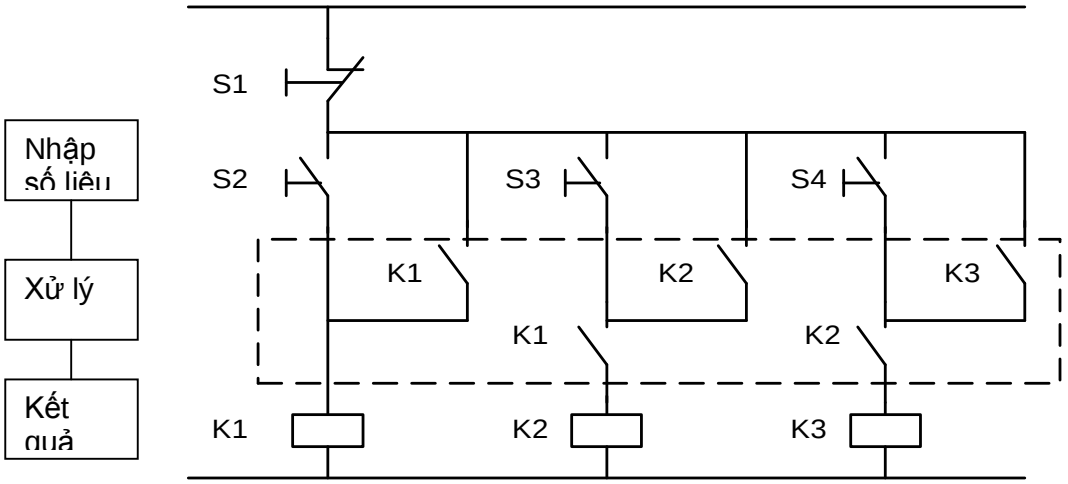
Hình 1-3:Sơ đồ điều khiển

Nếu ta thay bằng thiết bị điều khiển PLC ta có thể mô tả như sau:

-Tín hiệu vào: S1, S2, S3, S4 vẫn giữ nguyên.

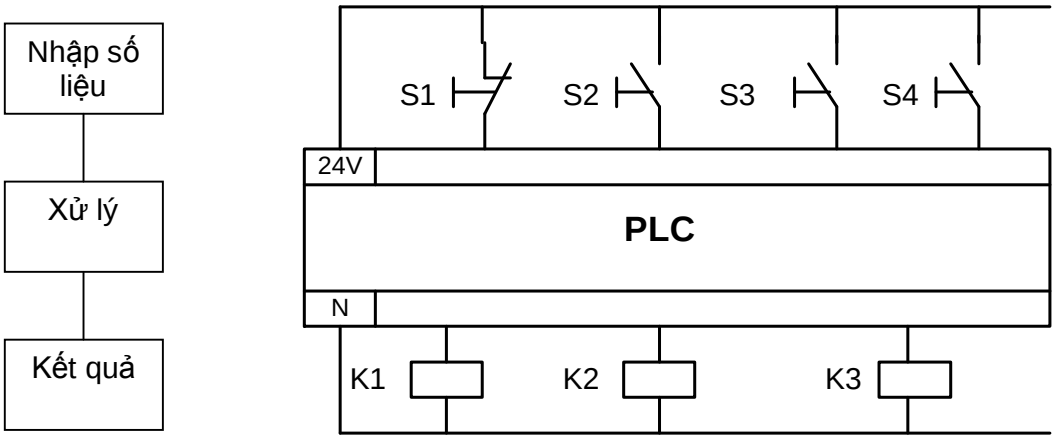
-Tín hiệu ra: K1, K2, K3 là các khởi động từ vẫn giữ nguyên.

-Phần tử xử lý:được thay thế bằng PLC.



Hình 1-4: Sơ đồ nối dây thực hiện bằng

Khi thực hiện bằng chương trình điều khiển có nhớ PLC ta chỉ cần thực hiện nối mạch theo sơ đồ sau:



Hình 1-5:Sơ đồ nối dây thực hiện bằng PLC

Nếu bây giờ nhiệm vụ điều khiển thay đổi ví dụ như các bơm 1,2,3 hoạt động theo nguyên tắc là chỉ một trong số các bơm được hoạt động độc lập. Như vậy đối với mạch điều khiển dùng Rơle ta phải tiến hành lắp giáp lại toàn bộ mạch điều khiển, trong khi đó đối với mạch điều khiển dùng PLC thì ta lại chỉ cần soạn thảo lại chương trình rồi nạp lại vào CPU thì ta sẽ có ngay một sơ đồ điều khiển theo yêu cầu nhiệm vụ mới mà không cần phải nối lại dây trên mạch điều khiển.

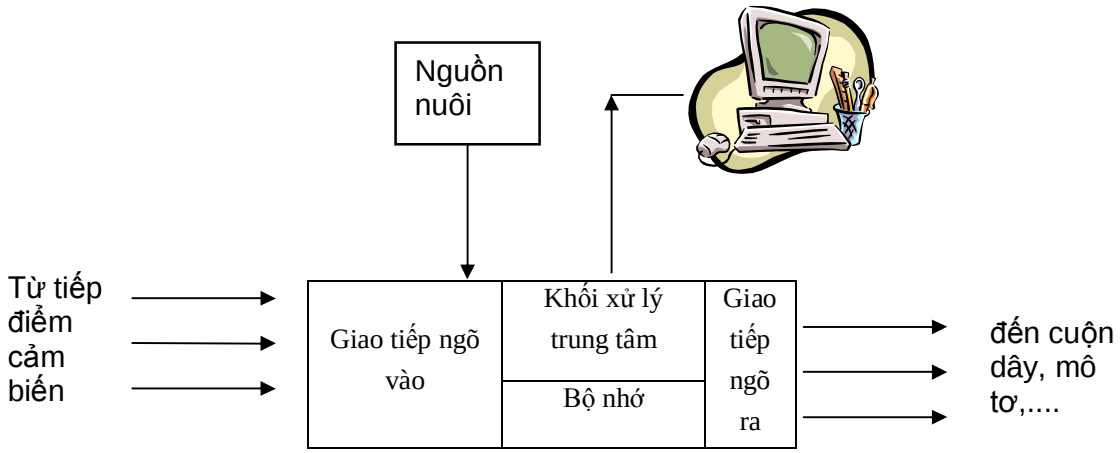
Như vậy một cách tổng quát có thể nói hệ thống điều khiển PLC là tập hợp các thiết bị và linh kiện điện tử. Để đảm bảo tính ổn định, chính xác và an toàn.. trong quá trình sản xuất, các thiết bị này bao gồm nhiều chủng loại, hình dạng khác nhau với công suất từ rất nhỏ đến rất lớn. Do tốc độ phát triển quá nhanh của công nghệ và để đáp ứng được các yêu cầu điều khiển phức tạp nên hệ thống điều khiển phải có hệ thống tự động hoá cao. Yêu cầu này có thể thực hiện được bằng hệ lập trình có nhớ PLC kết hợp với máy tính, ngoài ra còn cần có các thiết bị ngoại vi khác như: Bảng điều khiển, động cơ, cảm biến, tiếp điểm, công tắc tơ,...

Khả năng truyền dữ liệu trong hệ thống rất rộng thích hợp cho hệ thống xử lý và cũng rất linh động trong các hệ thống phân phối .

1.3. Cấu trúc của PLC:

1.3.1. Cấu trúc:

Mỗi một thành phần trong hệ thống điều khiển có một vai trò quan trọng như được trình bày trong hình vẽ sau.



Hình 1-6: Mô hình hệ thống điều khiển PLC

Tất cả các PLC đều có thành phần chính là :

Một bộ nhớ chương trình RAM bên trong (có thể mở rộng thêm một số bộ nhớ ngoài EPROM).

Một bộ vi xử lý có cổng giao tiếp dùng cho việc ghép nối với PLC .

Các Modul vào /ra.

Bên cạnh đó, một bộ PLC hoàn chỉnh còn đi kèm thêm một đơn vị lập trình bằng tay hay bằng máy tính. Hầu hết các đơn vị lập trình đơn giản đều có đủ RAM để chứa đựng chương trình dưới dạng hoàn thiện hay bổ sung . Nếu đơn vị lập trình là đơn vị xách tay , RAM thường là loại CMOS có pin dự phòng, chỉ khi nào chương trình đã được kiểm tra và sẵn sàng sử dụng thì nó mới truyền sang bộ nhớ PLC . Đối với các PLC lớn thường lập trình trên máy tính nhằm hổ trợ cho việc viết, đọc và kiểm tra chương trình . Các đơn vị lập trình nối với PLC qua cổng RS232, RS422, RS458, …

Ví dụ : một modul CPU S7 - 300

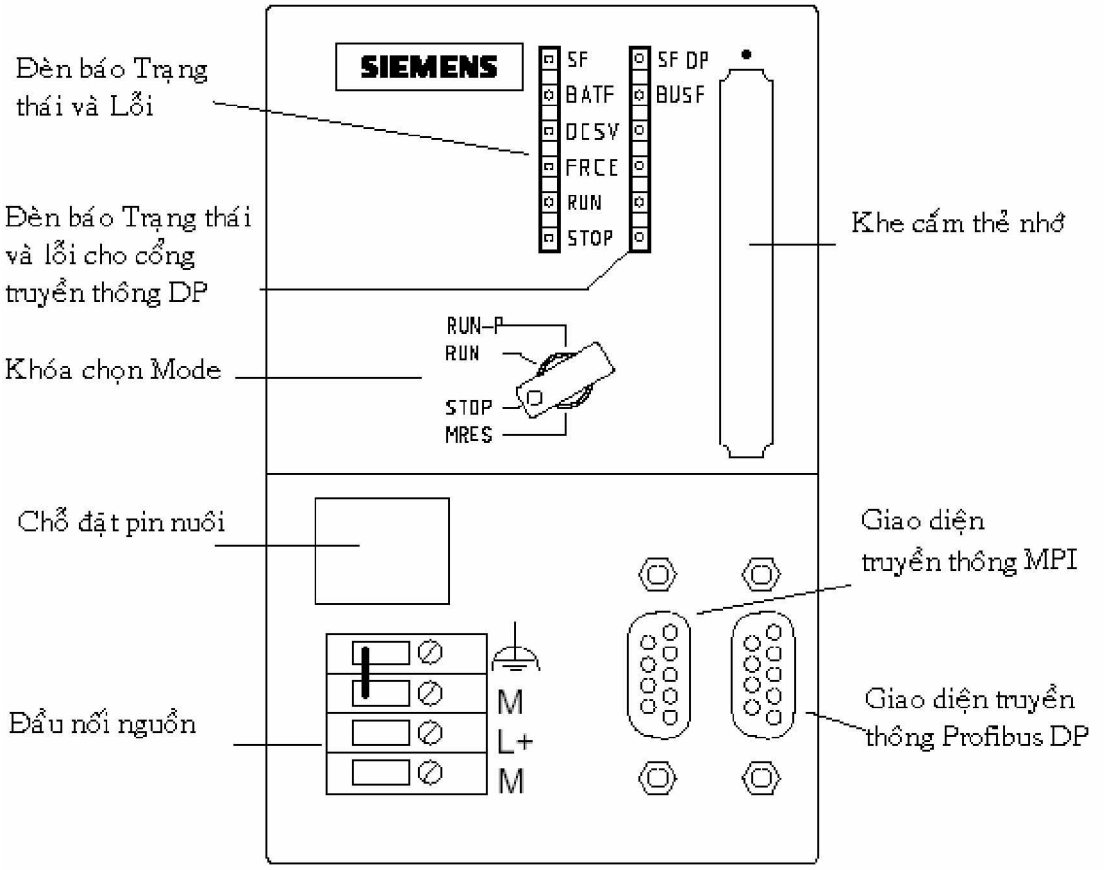
Khóa mode có 4 vị trí:
RUN-P chế độ lập trình và chạy
RUN chế độ chạy chương trình
STOP ngừng chạy chương trình
MRES reset bộ nhớ

Thẻ nhớ có thể có dung lượng từ 16KB đến 4MB, chứa chương trình từ PLC chuyển qua và chuyển chương trình ngược trở lại cho CPU.

Pin nuôi giúp nuôi chương trình và dữ liệu khi bị mất nguồn (tối đa 1 năm), ngoài ra còn nuôi đồng hồ thời gian thực. Với loại CPU không có pin nuôi thi cũng có một phần vùng nhớ được duy trì.

Thông qua cổng truyền thông MPI (MultiPoint Interface) có thể nối : máy tính lập trình, màn hình OP (Operator panel) , các PLC có cổng MPI (S7-300, M7-300, S7-400, M7-

400, C7-6xx), S7-200, vận tốc truyền đến 187.5kbps (12Mbps với CPU 318-2, 10.2 kbps với S7-200) . Cổng Profibus –DP nối các thiết bị trên theo mạng Profibus với vận tốc truyền lên đến 12Mbps.



b. Nguyên lý hoạt động của PLC

v **Đơn vị xử lý trung tâm**

CPU điều khiển các hoạt động bên trong PLC. Bộ xử lý sẽ đọc và kiểm tra chương trình được chứa trong bộ nhớ, sau đó sẽ thực hiện thứ tự từng lệnh trong chương trình , sẽ đóng hay ngắt các đầu ra. Các trạng thái ngõ ra ấy được phát tới các thiết bị liên kết để thực thi. Và toàn bộ các hoạt động thực thi đó đều phụ thuộc vào chương trình điều khiển được giữ trong bộ nhớ.

v **Hệ thống bus**

Hệ thống Bus là tuyến dùng để truyền tín hiệu, hệ thống gồm nhiều đường tín hiệu song song :

Address Bus : Bus địa chỉ dùng để truyền địa chỉ đến các Modul khác nhau.

Data Bus : Bus dùng để truyền dữ liệu.

Control Bus : Bus điều khiển dùng để truyền các tín hiệu định thì và điều khiển đồng bộ các hoạt động trong PLC .

Trong PLC các số liệu được trao đổi giữa bộ vi xử lý và các modul vào ra thông qua Data Bus. Address Bus và Data Bus gồm 8 đường, ở cùng thời điểm cho phép truyền 8 bit của 1 byte một cách đồng thời hay song song.

Nếu môït modul đầu vào nhận được địa chỉ của nó trên Address Bus , nó sẽ chuyển tất cả trạnh thái đầu vào của nó vào Data Bus. Nếu một địa chỉ byte của 8 đầu ra xuất hiện trên Address Bus, modul đầu ra tương ứng sẽ nhận được dữ liệu từ Data bus. Control Bus sẽ chuyển các tín hiệu điều khiển vào theo dõi chu trình hoạt động của PLC .

Các địa chỉ và số liệu được chuyển lên các Bus tương ứng trong một thời gian hạn chế.

Hêï thống Bus sẽ làm nhiệm vụ trao đổi thông tin giữa CPU, bộ nhớ và I/O . Bên cạch đó, CPU được cung cấp một xung Clock có tần số từ 1(8 MHZ. Xung này quyết định tốc độ hoạt động của PLC và cung cấp các yếu tố về định thời, đồng hồ của hệ thống.

- **Bộ nhớ**

PLC thường yêu cầu bộ nhớ trong các trường hợp :

Làm bộ định thời cho các kênh trạng thái I/O.

Làm bộ đệm trạng thái các chức năng trong PLC như định thời, đếm, ghi các Relay.

Mỗi lệnh của chương trình có một vị trí riêng trong bộ nhớ, tất cả mọi vị trí trong bộ nhớ đều được đánh số, những số này chính là địa chỉ trong bộ nhớ .

Địa chỉ của từng ô nhớ sẽ được trỏ đến bởi một bộ đếm địa chỉ ở bên trong bộ vi xử lý. Bộ vi xử lý sẽ giá trị trong bộ đếm này lên một trước khi xử lý lệnh tiếp theo . Với một địa chỉ mới , nội dung của ô nhớ tương ứng sẽ xuất hiện ở đấu ra, quá trình này được gọi là quá trình đọc .

Bộ nhớ bên trong PLC được tạo bởi các vi mạch bán dẫn, mỗi vi mạch này có khả năng chứa 2000 ÷ 16000 dòng lệnh , tùy theo loại vi mạch. Trong PLC các bộ nhớ như RAM, EPROM đều được sử dụng .

RAM (Random Access Memory) có thể nạp chương trình, thay đổi hay xóa bỏ nội dung bất kỳ lúc nào. Nội dung của RAM sẽ bị mất nếu nguồn điện nuôi bị mất . Để tránh tình trạng này các PLC đều được trang bị một pin khô, có khả năng cung cấp năng lượng dự trữ cho RAM từ vài tháng đến vài năm. Trong thực tế RAM được dùng để khởi tạo và kiểm tra chương trình. Khuynh hướng hiện nay dùng CMOSRAM nhờ khả năng tiêu thụ thấp và tuổi thọ lớn .

EPROM (Electrically Programmable Read Only Memory) là bộ nhớ mà người sử dụng bình thường chỉ có thể đọc chứ không ghi nội dung vào được . Nội dung của EPROM không bị mất khi mất nguồn , nó được gắn sẵn trong máy , đã được nhà sản xuất nạp và

chứa hệ điều hành sẵn. Nếu người sử dụng không muốn mở rộng bộ nhớ thì chỉ dùng thêm EPROM gắn bên trong PLC . Trên PG (Programer) có sẵn chổ ghi và xóa EPROM.

Môi trường ghi dữ liệu thứ ba là đĩa cứng hoạc đĩa mềm, được sử dụng trong máy lập trình . Đĩa cứng hoăïc đĩa mềm có dung lượng lớn nên thường được dùng để lưu những chương trình lớn trong một thời gian dài .

Kích thước bộ nhớ :

- Các PLC loại nhỏ có thể chứa từ 300 ÷1000 dòng lệnh tùy vào công nghệ chế tạo .
- Các PLC loại lớn có kích thước từ 1K ÷ 16K, có khả năng chứa từ 2000 ÷16000 dòng lệnh.

Ngoài ra còn cho phép gắn thêm bộ nhớ mở rộng như RAM , EPROM.

v **Các ngỏ vào ra I / O**

Các đường tín hiệu từ bộ cảm biến được nối vào các modul (các đầu vào của PLC) , các cơ cấu chấp hành được nối với các modul ra (các đầu ra của PLC) .

Hầu hết các PLC có điện áp hoạt động bên trong là 5V , tín hiêïu xử lý là 12/24VDC hoặc 100/240VAC.

Mỗi đơn vị I / O có duy nhất một địa chỉ, các hiển thị trạng thái của các kênh I / O được cung cấp bởi các đèn LED trên PLC , điều này làm cho việc kiểm tra hoạt động nhập xuất trở nên dể dàng và đơn giản .

Bộ xử lý đọc và xác định các trạng thái đầu vào (ON,OFF) để thực hiện việc đóng hay ngắt mạch ở đầu ra .

1.4.Đặc điểm .ứng dụng của hệ thống điều khiển PLC trong công nghiệp:

1.4.1. Đặc điểm:

Trong giai đoạn đầu của thời kỳ phát triển công nghiệp vào khoảng năm 1960 và 1970, yêu cầu tự động của hệ điều khiển được thực hiện bằng các Rơle điện từ nối nối với nhau bằng dây dẫn điện trong bảng điều khiển, trong nhiều trường hợp bảng điều khiển có kích thước quá lớn đến nỗi không thể gắn toàn bộ lên trên tường và các dây nối cũng không hoàn toàn tốt vì thế rất thường xảy ra trục trặc trong hệ thống. Một điểm quan trong nữa là do thời gian làm việc của các Rơle có giới hạn nên khi cần thay thế cần phải ngừng toàn bộ hệ thống và dây nối cũng phải thay mới cho phù hợp, bảng điều khiển chỉ dùng cho một yêu cầu riêng biệt không thể thay đổi tức thời chức năng khác mà phải lắp giáp lại toàn bộ, và trong trường hợp bảo trì cũng như sửa chữa cần đòi hỏi thợ chuyên môn có tay nghề cao. Tóm lại hệ điều khiển Rơle hoàn toàn không linh động.

**Tóm tắt nhược điểm của hệ thống điều khiển dùng Rơle:*

- Tốn kém rất nhiều dây dẫn .
- Thay thế rất phức tạp.

- Cần công nhân sửa chữa tay nghề cao.
- Công suất tiêu thụ lớn .
- Thời gian sửa chữa lâu.
- Khó cập nhật sơ đồ nên gây khó khăn cho công tác bảo trì cũng như thay thế.

**Ưu điểm của hệ điều khiển PLC:*

Sự ra đời của hệ điều khiển PLC đã làm thay đổi hẳn hệ thống điều khiển cũng như các quan niệm thiết kế về chúng, hệ điều khiển dùng PLC có nhiều ưu điểm như sau:

- Giảm 80% Số lượng dây nối.
- Công suất tiêu thụ của PLC rất thấp .
- Có chức năng tự chuẩn đoán do đó giúp cho công tác sửa chữa được nhanh chóng và dễ dàng.
- Chức năng điều khiển thay đổi dễ dàng bằng thiết bị lập trình (máy tính, màn hình) mà không cần thay đổi phần cứng nếu không có yêu cầu thêm bớt các thiết bị vào, ra.
- Số lượng Rơle và Timer ít hơn nhiều so với hệ điều khiển cổ điển.
- Số lượng tiếp điểm trong chương trình sử dụng không hạn chế.
- Thời gian hoàn thành một chu trình điều khiển rất nhanh (vài mS) dẫn đến tăng cao tốc độ sản xuất .
- Chi phí lắp đặt thấp .
- Độ tin cậy cao.
- Chương trình điều khiển có thể in ra giấy chỉ trong vài phút giúp thuận tiện cho vấn đề bảo trì và sửa chữa hệ thống.

1.4.2. Ứng dụng của hệ thống điều khiển PLC:

Từ các ưu điểm nêu trên, hiện nay PLC đã được ứng dụng trong rất nhiều lĩnh vực khác nhau trong công nghiệp như:

- Hệ thống nâng vận chuyển.
- Dây chuyền đóng gói.
- Các ROBOT lắp giáp sản phẩm .
- Điều khiển bơm.
- Dây chuyền xử lý hoá học.
- Công nghệ sản xuất giấy .
- Dây chuyền sản xuất thuỷ tinh.
- Sản xuất xi măng.
- Công nghệ chế biến thực phẩm.
- Dây chuyền chế tạo linh kiện bán dẫn.
- Dây chuyền lắp giáp Tivi.
- Điều khiển hệ thống đèn giao thông.
- Quản lý tự động bãi đậu xe.

- Hệ thống báo động.
- Dây chuyền may công nghiệp.
- Điều khiển thang máy.
- Dây chuyền sản xuất xe Ôtô.
- Sản xuất vi mạch.
- Kiểm tra quá trình sản xuất .

CHƯƠNG 2: Kiểu dữ liệu và cấu trúc vùng nhớ:

2.1. Phân loại tín hiệu:

Tín hiệu được hiểu là hàm theo thời gian U(t) có giá trị thực, mang thông tin và được truyền tải bởi các đại lượng vật lý. Tín hiệu được gọi là liên tục nếu U(t) là hàm liên tục

Ví dụ: như các modul mở rộng sau.

+ Module vào số có các loại sau:

- SM 321; DI 32 _ 24 VDC
- SM 321; DI 16 _ 24 VDC
- SM 321; DI 16 _ 120 VAC, 4*4 nhóm
- SM 321; DI 8 _ 120/230 VAC, 2*4 nhóm
- SM 321; DI 32 _ 120 VAC 8*4 nhóm

+ **Module ra số:**

- SM 322; DO 32 _ 24 VDC/0.5 A, 8*4 nhóm
- SM 322; DO 16 _ 24 VDC/0.5 A, 8*2 nhóm
- SM 322; DO 8 _ 24 VDC/2 A, 4*2 nhóm
- SM 322; DO 16 _ 120 VAC/1 A, 8*2 nhóm
- SM 322; DO 8 _ 120/230 VAC/2 A, 4*2 nhóm
- SM 322; DO 32_ 120 VAC/1.0 A, 8*4 nhóm

- SM 322; DO 16 _ 120 VAC ReLay, 8*2 nhóm
- SM 322; DO 8 _ 230 VAC Relay, 4*2 nhóm
- SM 322; DO 8 _ 230 VAC/5A Relay,1*8 nhóm

Module vào/ ra

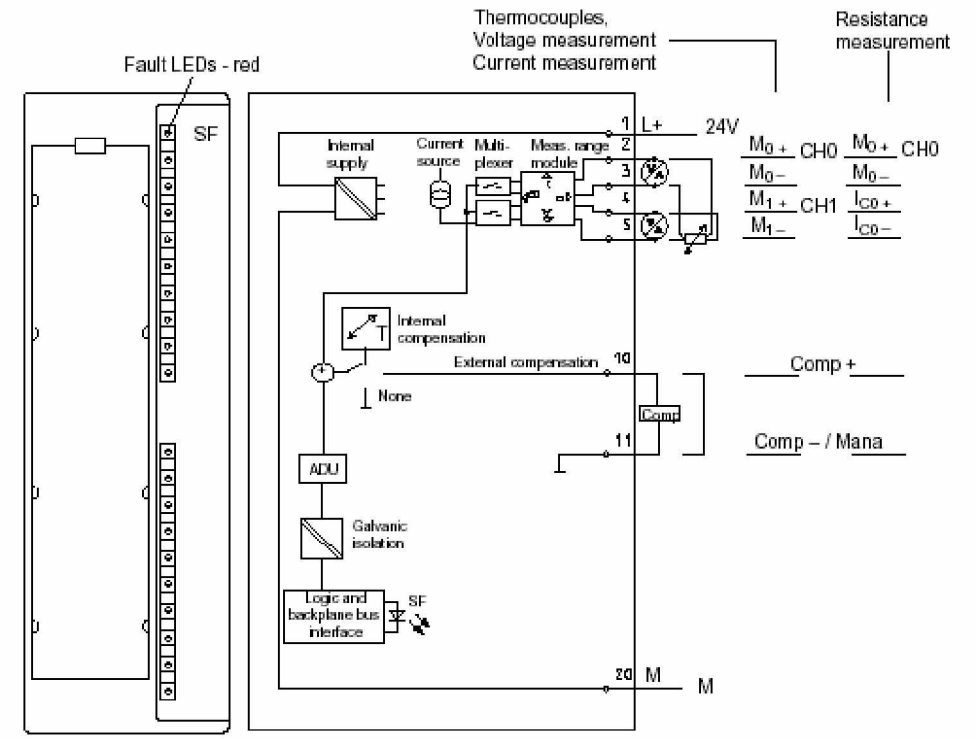
- SM 323; DI 16/DO 16 _ 24 VDC/0.5 A
- SM 323; DI 8/DO 8 _ 24 VDC/0.5 A

Module Analog in

Module analog in có nhiều ngõ vào, dùng để đo điện áp, dòng điện, điện trở ba dây, bốn dây, nhiệt độ. Có nhiều tầm đo, độ phân giải, thời gian chuyển đổi khác nhau. Cài đặt thông số hoạt động cho module bằng phần mềm S7- Simatic 300 Station – Hardware và/hoặc chương trình người dùng sử dụng hàm SFC 55, 56, 57 phù hợp (xem mục) và/hoặc cài đặt nhờ mo(dulle tầm đo (measuring range module) gắn trên module SM. Kết quả chuyển đổi là số nhị phân phụ hai với bit MSB là bit dấu.

- SM331 AI 2*12 : module chuyển đổi hai kênh vi sai áp hoặc dòng, hoặc một kênh điện trở 2/3/4 dây, dùng phương pháp tích phân, thời gian chuyển đổi từ 5ms đến 100ms, độ phân giải 9, 12, 14 bit + dấu, các tầm đo như sau: (80 mV; (250 mV; (500 mV; (1000 mV;

(2.5 V; (5 V;1 .. 5 V; (10 V; (3.2 mA; (10 mA; (20 mA; 0 .. 20 mA; 4 ..20 mA. Điện trở 150 (; 300 (; 600 (; Đo nhiệy độ dùng cặp nhiệt E, N, J, K, L, nhiệt kế điện trở Pt 100, Ni 100. Các thông số mặc định đã được cài sẵn trên module, kết hợp với đặt vị trí của module tầm đo (bốn vị trí A, B, C, D) nếu không cần thay đổi thì có thể sử dụng ngay.



Measuring Range Module Setting	Measuring Method	Measuring Range
A	Voltage	± 1000 mV
B	Voltage	± 10 V
C	Current, 4-wire transducer	4 to 20 mA
D	Current, 2-wire transducer	4 to 20 mA

Measuring Range ± 80 mV	Measuring Range ± 250 mV	Measuring Range ± 500 mV	Measuring Range ± 1 V	Measuring Range ± 2.5 V	Units		Range
					Deci-mal	Hexa-decimal	
> 94.071	> 293.97	> 587.94	> 1.175	> 2.9397	32767	$7FFF_H$	Overflow
94.071 : 80.003	293.97 : 250.01	587.94 : 500.02	1.175 : 1.00004	2.9397 : 2.5001	32511 : 27649	$7EFF_H$: $6C01_H$	Overrange
80.000 60.000 : – 60.000 – 80.000	250.00 187.50 : – 187.50 – 250.00	500.00 375.00 : – 375.00 – 500.00	1.000 0.750 : – 0.750 –1.000	2.500 1.875 : – 1.875 – 2.500	27648 20736 : –20736 –27648	$6C00_H$ 5100_H : $AF00_H$ 9400_H	Nominal range
– 80.003 : – 94.74	– 250.01 : – 293.98	– 500.02 : – 587.96	– 1.00004 : – 1.175	– 2.5001 : – 2.93398	–27649 : –32512	$93FF_H$: 8100_H	Underrange
<– 94.074	<– 293.98	<– 587.96	<– 1.175	<– 2.93398	–32768	8000_H	Underflow

Measuring Range ± 5 V	Measuring Range ± 10 V ± 10 mA	Measuring Range ± 3.2 mA	Measuring Range ± 20 mA	Units		Range
				Deci-mal	Hexa-decimal	
> 5.8794	> 11.7589	> 3.7628	> 23.515	32767	$7FFF_H$	Overflow
5.8794 : 5.0002	11.7589 : 10.0004	3.7628 : 3.2001	23.515 : 20.0007	32511 : 27649	$7EFF_H$: $6C01_H$	Overrange
5.00 3.75 : – 3.75 – 5.00	10.00 7.50 : – 7.50 – 10.00	3.200 2.400 : – 2.400 – 3.200	20.000 14.998 : – 14.998 – 20.000	27648 20736 : –20736 –27648	$6C00_H$ 5100_H : $AF00_H$ 9400_H	Nominal range
– 5.0002 : – 5.8796	– 10.0004 : – 11.759	– 3.2001 : – 3.7629	– 20.0007 : – 23.516	–27649 : –32512	$93FF_H$: 8100_H	Underrange
<– 5.8796	<– 11.759	<– 3.7629	<– 23.516	–32768	8000_H	Underflow

Measuring Range 1 to 5 V	Measuring Range 0 20 mA	Measuring Range 4 to 20 mA	Units		Range
			Deci-mal	Hexa-decimal	
> 5.7036	> 23.515	> 22.810	32767	$7FFF_H$	Overflow
5.7036 : 5.0001	23.515 : 20.0007	22.810 : 20.0005	32511 : 27649	$7EFF_H$: $6C01_H$	Overrange
5.000 4.000 : 1.000	20.000 14.998 : 0.000	20.000 16.000 : 4.000	27648 20736 : 0	$6C00_H$ 5100_H : 0_H	Nominal range
0.9999 : 0.2963	-0.0007 : –3.5185	3.9995 : 1.1852	–1 : –4864	$FFFF_H$: $ED00_H$	Underrange
< 0.2963	<–3.5185	< 1.1852	–32768	8000_H	Underflow

Measuring Range 150 Ω	Measuring Range 300 Ω	Measuring Range 600 Ω	Units		Range
			Deci-mal	Hexa-decimal	
> 176.383	> 352.767	> 705.534	32767	$7FFF_H$	Overflow
176.383 : 150.005	352.767 : 300.011	705.534 : 600.022	32511 : 27649	$7EFF_H$: $6C01_H$	Overrange
150.000 112.500 : 0.000	300.000 225.000 : 0.000	600.000 450.000 : 0.000	27648 20736 : 0	$6C00_H$ 5100_H : 0_H	Nominal range
(negative values physically not possible)			–1 : –4864	$FFFF_H$: $ED00_H$	Underrange
-	-	-	–32768	8000_H	Underflow

- **SM331, AI 8*12 bit , 8 kênh vi sai chia làm hai nhóm, độ phân giải 9 (12, 14) bit + dấu**
- **SM331, AI 8*16 bit , 8 kênh vi sai chia làm 2 nhóm , độ phân giải 15 bit + dấu**

+ Module Analog Out:

Cung cấp áp hay dòng phụ thuộc số nhị phân phụ hai

- SM332 AO 4*12 bit: 4 ngõ ra dòng hay áp độ phân giải 12 bit, thời gian chuyển đổi 0.8 ms .
- SM332 AO 2*12 bit
- SM332 AO 4*16 bit

+ Module Analog In/Out

- SM 334; AI 4/AO 2 * 8 Bit
- SM334; AI 4/AO 2* 12 Bit

Output Range 0 to 10 V	Output Range 1 to 5 V	Output Range ± 10 V	Units		Range
			Decimal	Hexa-decimal	
0	0	0	>32511	>7EFF$_H$	Overflow
11.7589 : 10.0004	5.8794 : 5.0002	11.7589 : 10.0004	32511 : 27649	7EFF$_H$: 6C01$_H$	Overrange
10.0000 : 0	5.0000 : 1.0000	10.0000 : 0	27648 : 0	6C00$_H$: 0$_H$	Nomial range
0	:0.9999 0	:	: – 6912	: E500$_H$	
	0		– 6913 : – 27648	E4FF$_H$: 9400$_H$	
		10.0004 : – 11.7589	– 27649 : – 32512	93FF$_H$: 8100$_H$	Underrange
		0	<– 32512	<8100$_H$	Underflow

Output Range 0 to 20 mA	Output Range 4 to 20 mA	Output Range ± 20 mA	Units		Range
			Decimal	Hexa-decimal	
0	0	0	>32511	>7EFF$_H$	Overflow
23.515 : 20.0007	22.81 : 20.005	23.515 : 20.0007	32511 : 27649	7EFF$_H$: 6C01$_H$	Overrange
20.000 : 0	20.000 : 4.000	20.000 0	27648 : 0	6C00$_H$: 0$_H$	Nominal range
0	3.9995 0	:	: – 6912	: E500$_H$	
	0		– 6913 : – 27648	E4FF$_H$: 9400$_H$	
		: – 23.515	– 27649 : – 32512	93FF$_H$: 8100$_H$	Underrange
		0	<– 32512	<8100$_H$	Underflow

Module chức năng FM

FM350-1 : đếm xung một kênh

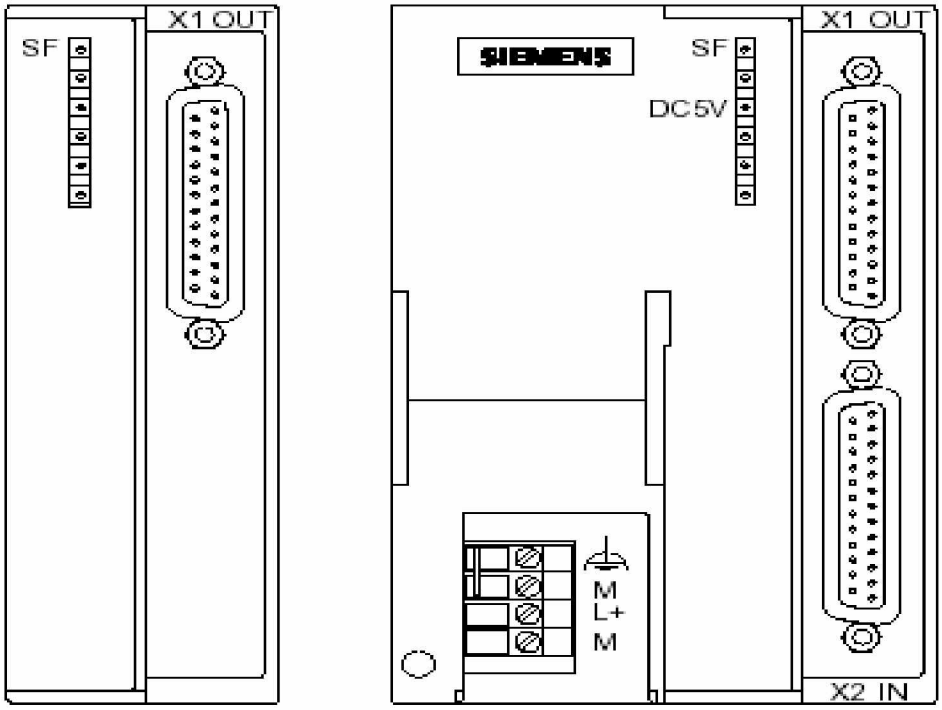
FM350-2 : đếm xung tám kênh

FM351, 353, 354, 357-2 : điều khiển định vị

FM352: bộ điều khiển cam điện tử

FM355: bộ điều khiển hệ kín

+ Module IM



IM360 IM361

Module IM360 gắn ở rack 0 kế CPU dùng để ghép nối với module IM361 đặt ở các rack 1, 2, 3 giúp kết nối các module mở rộng với CPU khi số module lớn hơn 8. Cáp nối giữa hai rack là loại 368.

Trong trường hợp chỉ có hai rack, ta dùng loại IM365.

2.2. Phân loại mã hiệu:

Trong quá trình thực hiện cấu trúc của tín hiệu số được biểu diễn dưới dạng:

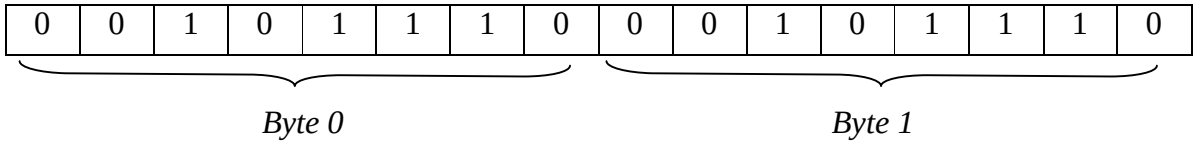
1/ Bit : (ví dụ I0.0) dùng để biểu diễn số nhị phân (có 2 giá trị 1 hoặc 0).

1

2/ Byte : (ví dụ MB0) Một Byte gồm có 8 bits. Ví dụ giá trị của 8 cổng vào (IB0) hoặc 8 cổng ra (QB1),... được gọi là một byte:

0	0	1	1	0	1	0	1

3/ Word: (ví dụ MW0= MB0 + MB1) Một Word gồm có 2 Byte như vậy một Word có độ dài 16 bits.



4/ Doppelword: (ví dụ MD0 = MW0 + MW2): có độ dài 2 từ hoặc 4 Byte tức là 32 bits.

2.3. Kiểu dữ liệu:

Một chương trình trong S7-300 có thể sử dụng các kiểu dữ liệu sau:

1/ BOOL: với dung lượng là 1 bit và có giá trị là 0 hoặc 1 (đúng hoặc sai). Đây là kiểu dữ liệu biến có hai giá trị.

2/ BYTE: gồm 8 bits, thường được dùng để biểu diễn một số nguyên dương trong khoảng từ 0 đến 255 hoặc mã ASCII của một ý tự.

Ví dụ: B#16#14 nghĩa là số nguyên 14 viết theo hệ đếm cơ số 16 có độ dài 1 byte.

3/ WORD: gồm 2 byte, để biểu diễn số nguyên dương từ 0 đến 65535 (216 - 1).

4/DWORD: Là từ kép có giá trị là: 0 đến 232-1.

5/ INT: cũng có dung lượng là 2 bytes, dùng để biểu diễn một số nguyên trong khoảng -32768 đến 32767 hay (2-15...215-1).

6/ DINT: gồm 4 bytes, dùng để biểu diễn số nguyên từ -2147483648 đến 2147483647 hay: (2-31....231-1).

7/ REAL: gồm 4 bytes, dùng để biểu diễn một số thực dấu phảy động có giá trị là: -3,4E38.....3,4E38.

Ví dụ: 1.234567e+13

8/ S5t (hay S5Time): khoảng thời gian, được tính theo giờ/phút/giây: (-2-31+ 231-1 ms).

Ví dụ: S5t#2h_3m_0s_5ms.

Đây là lệnh tạo khoảng thời gian là 2 tiếng ba phút và 5 mili giây.

9/TOD: Biểu diễn giá trị tức thời tính theo Giờ/phút/giây.

Ví dụ: TOD#5:30:00 là lệnh khai báo giá trị thời gian trong ngày là 5 giờ 30 phút.

10/ DATE: Biểu diễn thời gian tính theo năm / ngày / tháng.

Ví dụ: DATE#2003-6-12

Là lệnh khai báo ngày12 tháng 6 năm 2003.

11/ CHAR: biểu diễn một hoặc nhiều ký tự (nhiều nhất là 4 ký tự) (ASCII - code).

Ví dụ: ABCD

2.4. Cấu trúc vùng nhớ và phương pháp truy cập vùng nhớ của CPU S7-300:

+ Được chia ra làm 3 vùng chính:

1) Vùng chứa chương trình ứng dụng: vùng nhớ chương trình được chia làm 3 miền:

   a/ OB: Miền chứa chương trình tổ chức (các chương trình này sẽ được giới thiệu ở mục 1.2.5).

   b/ FC: (Funktion): miền chứa chương trình con được tổ chức thành hàm có biến hình thức để trao đổi dữ liệu với chương trình đã gọi nó.

   c/ FB: (Funktion Block): Miền chứa chương trình con, được tổ chức thành hàm và có khả năng trao đổi dữ liệu với bất cứ một khối chương trình nào khác. Các dữ liệu này phải được xây dựng thành một khối dữ liệu riêng (gọi là DB-Data block).

2) Vùng chứa các tham số của hệ điều hành và chương trình ứng dụng, được phân chia thành 7 miền khác nhau, bao gồm:

a. I (Procees image input): miền bộ đệm các dữ liệu cổng vào số. Trước khi thực hiện chương trình, PLC sẽ đọc giá trị logic của tất cả các đầu vào và cất giữ chúng trong vùng nhớ I. Thông thường chương trình ứng dụng không đọc trực tiếp trạng thái logic của cổng vào số mà chỉ lấy dữ liệu của cổng vào từ bộ đệm I.

b. Q (Process image output): miền bộ đệm các cổng ra số. Kết thúc giai đoạn thực hiện chương trình sẽ chuyển giá trị logic của bộ đệm tới các cổng ra số. Thông thường không trực tiếp gán giá trị tới tận cổng ra mà chỉ chuyển chúng vào bộ đệm Q.

c. M: Miền các biến cờ. Chương trình ứng dụng sử dụng vùng nhớ này để lưu giữ các tham số cần thiết và có thể truy cập nó theo bit (M), byte (MB), từ (MW) hay từ kép (MD).

d. T: Miền nhớ phục vụ bộ thời gian (TIME) bao gồm việc lưu giữ giá trị thời gian đặt trước (PV-preset value), giá trị đếm thời gian tức thời (CV- Curren value) cũng như các giá trị logic đầu ra của bộ thời gian.

e. C: Miền nhớ phục vụ bộ đếm (counter) bao gồm việc lưu giữ giá trị đặt trước (PV), và giá trị đếm tức thời (CV) và giá trị logic đầu ra của bộ đếm.

f. PI: Miền địa chỉ cổng vào của các modul tương tự. Các giá trị tương tự tại cổng vào của modul tương tự sẽ được đọc và chuyển tự động theo những địa chỉ. Chương trình ứng dụng có thể truy nhập miền nhớ PI theo từng byte (PIB), từng từ (PIW) hoặc theo từ kép (PID).

g. PQ: miền địa chỉ cổng ra cho các modul tương tự. Các gía trị theo những địa chỉ này được modul tương tự chuyển tới các cổng ra tương tự. Chương trình ứng dụng có thể truy cập miền nhớ PQ theo từng byte (PQB), từng từ (PQW) hay từng từ kép (PQD)

3) Vùng chứa các khối dữ liệu: được chia làm hai loại:

a. DB (Data block): miền chứa các dữ liệu được tổ chức thành khối. Kích thước cũng như số lượng do người sử dụng qui định, phù hợp với từng bài toán điều khiển.

Chương trình có thể truy cập miền này theo từng bit (DBX), byte (DBB), từ (DBW) hoặc từ kép (DBD).

b. L (Local data block) : miền giữ liệu địa phương, được các khối chương trình OB, FB, FC tổ chức và sử dụng cho các biến nháp tức thời và trao đổi giữ liệu của biến hình thức của chương trình với những khối chương trình đã gọi nó. Nội dung của một số dữ liệu trong miền nhớ này sẽ bị xoá khi kết thúc chương trình tương ứng trong OB, FB, FC. Miền này có thể truy cập từ chương trình theo bit (L), byte (LB), từ (LW) hay từ kép (LD).

+ Các vùng nhớ của PLC

Vùng nhớ chương trình (load memory) chứa chương trình người dùng (không chứa địa chỉ ký hiệu và chú thích) có thể là RAM hay EEPROM trong CPU hay trên trên thẻ nhớ.

Vùng nhớ làm việc (working memory) là RAM, chứa chương trình do vùng nhớ chương trình chuyển qua; chỉ các phần chương trình cần thiết mới được chuyển qua, phần nào không cần ở lại vùng nhớ chương trình , ví dụ block header, data block

Vùng nhớ hệ thống (system memory) phục vụ cho chương trình người dùng, bao gồm timer , counter, vùng nhớ dữ liệu M, bộ nhớ đệm xuất nhập…

Trên CPU 312IFM và 314 IFM vùng nhớ chương trình là RAM và EEPROM; các CPU khác có pin nuôi, vùng nhớ chương trình là RAM và thẻ nhớ. Khi mất nguồn hay ở chế độ MRES (reset bộ nhớ) RAM sẽ bị xóa. Một số vùng nhớ của RAM (timer, counter, vùng nhớ M, khối dữ liệu..) có thể khai báo là lưu giữ (retentive) bằng phần mềm S7 để chuyển các vùng này sang bộ nhớ lưu giữ (NVRAM non volative)ù dù không có pin nuôi, kích thước cụ thể tùy loại CPU.

Bảng sau cho một số thông số chính của các CPU

Thông số	CPU 312IFM	CPU 313	CPU 314	CPU 314IFM
Working memory	6KB	12KB	24KB	32KB
Load memory	20KBRAM 20KBEEPROM	20KBRAM up to 4MB FEPROM (memory card)	40KB up to 4MB FEPROM (memory card)	48KB RAM 48KBEEPROM
Vận tốc	0.7ms/1000 lệnh nhị phân	0.7ms/1000 lệnh nhị phân	0.3ms/1000 lệnh nhị phân	0.3ms/1000 lệnh nhị phân
Data Memory	1KB Retentivity adjustable MB0..MB71	2KB Retentivity adjustable MB0..MB71	2KB Retentivity adjustable MB0..MB255	2KB Retentivity adjustable MB0..MB143

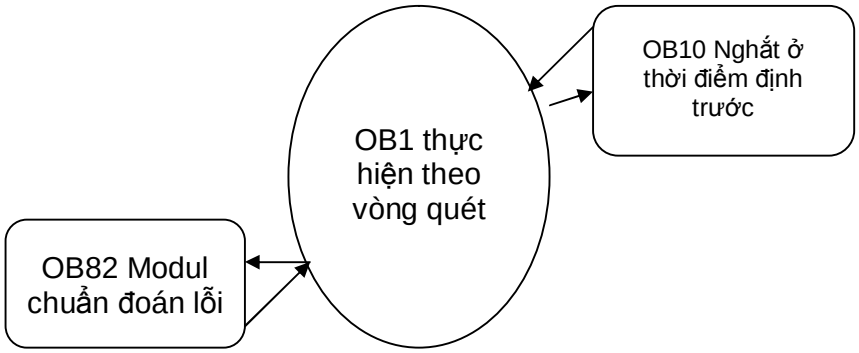
	Preset MB0..MB15	Preset MB0..MB15	Preset MB0..MB15	Preset MB0..MB15
Counter	adjustable Retentivity C0..C31 Preset C0..C7	adjustable Retentivity C0..C63 Preset C0..C7	adjustable Retentivity C0..C63 Preset C0..C7	Adjustable Retentivity C0..C63 Preset C0..C7
Timer	T0..T63 no retentivity	T0..T127 Adjustable Retentivity T0..T31 Preset: no	T0..T127 Adjustable Retentivity T0..T127 Preset: no	T0..T127 Adjustable Retentivity T0..T71 Preset: no
Digital inputs	10 integrated + 128	128	512	496 + 20 integrated
Digital outputs	6 integrated + 128	128	512	496 +16 integrated
Analog inputs	32	32	64	64 + 4 integrated
Analog outputs	32	32	64	64 + 1 integrated
Process image input	I0.0.. I127.7	I0.0.. I127.7	I0.0.. I127.7	
Process image output	Q0.0 ..Q127.7	Q0.0 ..Q127.7	Q0.0 ..Q127.7	

2.5. Cấu trúc chương trình:

2.5.1.Lập trình tuyến tính:

Phần bộ nhớ của CPU dành cho chương trình ứng dụng có tên gọi là logic Block. Như vậy logic block là tên chung để gọi tất cả các khối bao gồm những khối chương trình tổ chức OB, khối chương trình FC, khối hàm FB. Trong các loại khối chương trình đó thì chỉ có khối duy nhất khối OB1 được thực hiện trực tiếp theo vòng quét. Nó được hệ điều hành gọi theo chu kỳ lặp với khảng thời gian không cách đều nhau mà phụ thuộc vào độ dài của chương trình. Các loại khối chương trình khác không tham gia vào vòng quét.

Với tổ chức chương trình như vậy thì phần chương trình trong khối OB1 có đầy đủ điều kiện của một chương trình điều khiển thời gian thực và toàn bộ chương trình ứng dụng có thể chỉ cần viết trong OB1 là đủ như hình vẽ sau. Cách tổ chức chương trình với chỉ một khối OB1 duy nhất như vậy được gọi là lập trình tuyến tính.



Hình 2-1: Sơ đồ khối kiểu lập trình tuyến tính

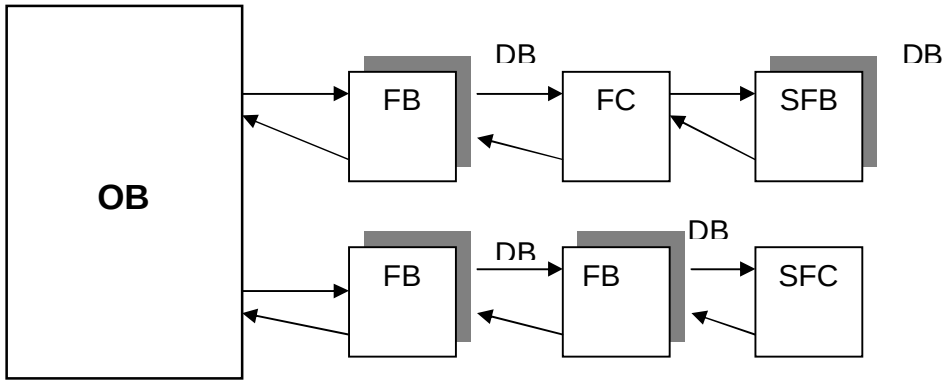
2.5.2.Lập trình có cấu trúc:

Khối OB1 được hệ thống gọi xoay vòng liên tục theo vòng quét.

Các khối OB khác không tham gia vào vòng quét được gọi bằng những tín hiệu báo ngắt. S7-300 có nhiều tín hiệu báo ngắt như tín hiệu báo ngắt khi có sự cố nguồn nuôi, có sự cố chập mạch ở các modul mở rộng, tín hiệu báo ngắt theo chu kỳ thời gian, và mỗi loại tín hiệu báo ngắt như vậy cũng chỉ có khả năng gọi một khối OB nhất định. Ví dụ tín hiệu báo ngắt sự cố nguồn nuôi chỉ gọi khối OB81, tín hiệu báo ngắt truyền thông chỉ gọi khối OB87.

Mỗi khi xuất hiện tín hiệu báo ngắt hệ thống sẽ dừng công việc đang thực hiện lại, chẳng hạn như tạm dừng việc thực hiện chương trình trong OB1, và chuyển sang thực hiện chương trình xử lý ngắt tong các khối OB tương ứng. Ví dụ khi đang thực hiện chương trình trong khối OB1 mà xuất hiện ngắt báo sự cố truyền thông, hệ thống sẽ tạm dừng việc thực hiện chương trình trong OB1 lại để gọi chương trình trong khối truyền thông OB87. Chỉ sau khi đã thực hiện xong chương trình trong khối OB87 thì hệ thống mới quay trở về hực hiện tiếp tục phần chương trình còn lại trong OB1.

Với kiểu lập trình có cấu trúc thì khác vì toàn bộ chương trình điều khiển được chia nhỏ thành các khối FC và FB mang một nhiệm vụ cụ thể riêng và được quản lý chung bởi những khối OB. Kiểu lập trình này rất phù hợp cho những bài toán phức tạp, nhiều nhiệm vụ và lại rất thuận lợi cho việc sửa chữa sau này.



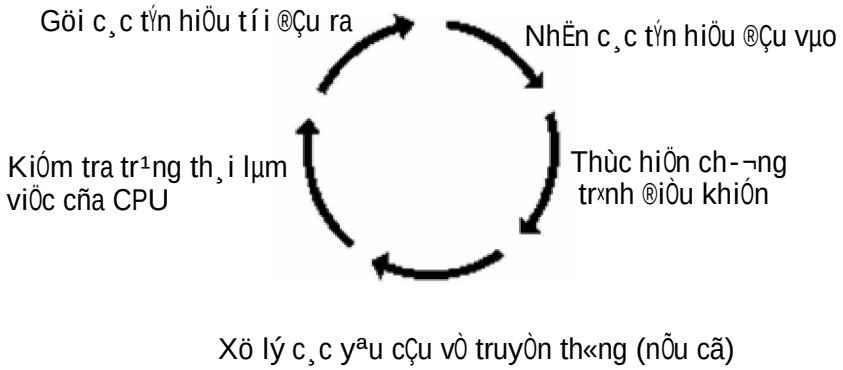
Hình 2-2: Sơ đồ kiểu lập trình có cấu trúc.

OB: Organization Block
FB: Function Block
FC: Function
SFB: System Function block
SFC: System function
SDB: System Data Block
DB: Data block
Chú ý: Bao giờ FB cũng sử dụng chung với DB.

2.6. Vòng quét của chương trình:

SPS (PLC) thực hiện các công việc (bao gồm cả chương trình điều khiển) theo chu trình lặp. Mỗi vòng lặp được gọi là một vòng quét (scancycle). Mỗi vòng quét được bắt đàu bằng việc chuyển dữ liệu từ các cổng vào số tới vùng bộ đệm ảo I, tiếp theo là giai đoạn thực hiện chương trình. Trong từng vòng quét , chương trình được thực hiện từ lệnh đầu tiên đến lệnh kết thúc của khối OB1. Sau giai đoạn thực hiện chương trình là giai đoạn chuyển các nội dung của bộ đệm ảo Q tới các cổng ra số. Vòng quét được kết thúc bằng giai đoạn xử lý các yêu cầu truyền thông (nếu có) và kiểm tra trạng thái của CPU. Mỗi vòng quét có thể mô tả như sau:

Vòng quét (Cycle scan):



Hình1-8: Quá trình hoạt động của một vòng quét.

<u>Chú ý</u> : Bộ đệm I và Q không liên quan tới các cổng vào/ra tương tự nên các lệnh truy nhập cổng tương tự được thực hiện trực tiếp với cổng vật lý chứ không thông qua bộ đệm.

Thời gian cần thiết để cho PLC thực hiện được một vòng quét được gọi là thời gian vòng quét (Scan time). Thời gian vòng quét không cố định, tức là không phải vòng quét nào cũng được thực hiện trong một khoảng thời gian như nhau. Có vòng quét được thực hiện lâu, có vòng quét được thực hiện nhanh tuỳ thuộc vào số lệnh trong chương trình được thực hiện, vào khối lượng dữ liệu truyền thông. Trong vòng quét đó .

Như vậy giữa việc đọc dữ liệu từ đối tượng để xử lý, tính toán và việc gửi tín hiệu điều khiển đến đối tượng có một khoảng thời gian trễ đúng bằng thời gian vòng quét. Nói cách khác, thời gian vòng quét quyết định tính thời gian thực của chương trình điều khiển trong PLC. Thời gian vòng quét càng ngắn, tính thời gian thực của chương trình càng cao.

Nếu sử dụng các khối chương trình đặc biệt có chế độ ngắt, ví dụ khối OB40, OB80,... Chương trình của các khối đó sẽ được thực hiện trong vòng quét khi xuất hiện tín hiệu báo ngắt cùng chủng loại. Các khối chương trình này có thể thực hiện tại mọi vòng quét chứ không phải bị gò ép là phải ở trong giai đoạn thực hiện chương trình. Chẳng hạn một tín hiệu báo ngắt xuất hiện khi PLC đang ở giai đoạn truyền thông và kiểm tra nội bộ, PLC sẽ tạm dừng công việc truyền thông, kiểm tra, để thực hiện ngắt như vậy, thời gian vòng quét sẽ càng lớn khi càng có nhiều tín hiệu ngắt xuất hiện trong vòng quét. Do đó để nâng cao tính thời gian thực cho chương trình điều khiển, tuyệt đối không nên viết chương trình xử lý ngắt quá dài hoặc quá lạm dụng việc sử dụng chế độ ngắt trong chương trình điều khiển.

Tại thời điểm thực hiện lệnh vào/ra, thông thường lệnh không làm việc trực tiếp với cổng vào/ra mà chỉ thông qua bộ nhớ đệm của cổng trong vùng nhớ tham số. Việc truyền thông giữa bộ đêm ảo với ngoại vi trong giai đoạn 1 và 3 do hệ điều hành CPU quản lý. ở một số modul CPU, khi gặp lệnh vào/ra ngay lập tức hệ thống sẽ cho dừng mọi công việc khác, ngay cả chương trình xử lý ngắt, để thực hiện với cổng vào/ra.

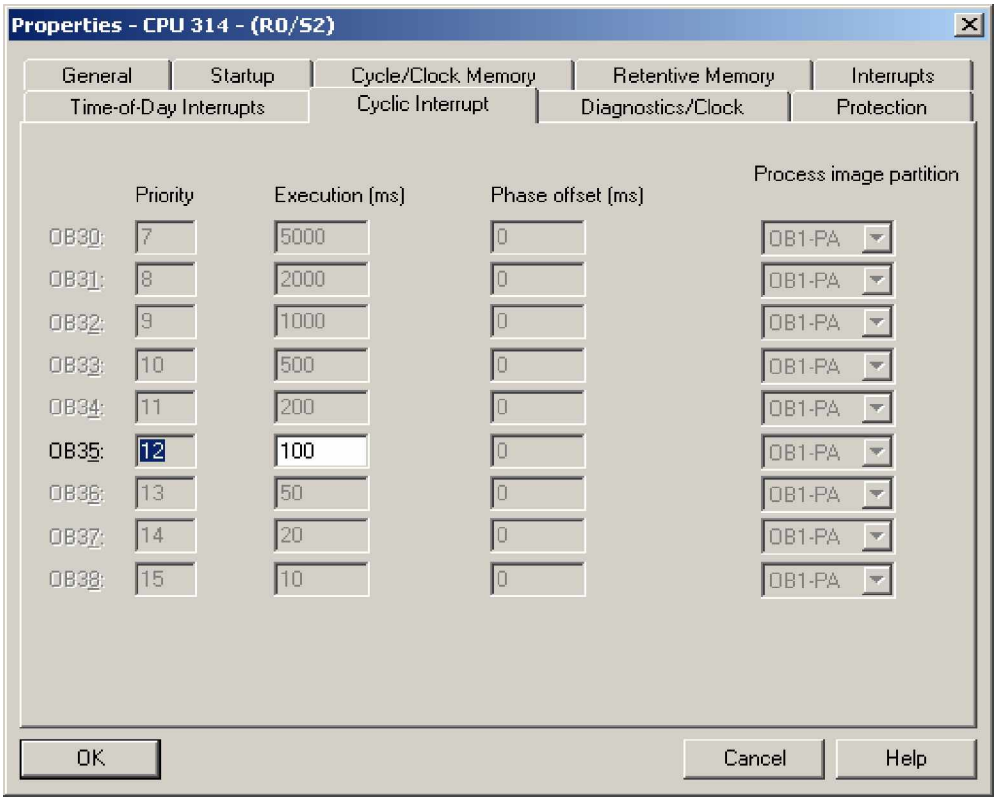
2.7. Những khối OB đặc biệt:

Khối OB1 có chức năng quản lý chính trong toàn bộ chương trình, có nghĩa là nó sẽ thực hiện một cách đều đặn ở từng vòng quét trong khi thực hiện chương trình. Ngoài ra Step7 còn có rất nhiều các khối OB đặc biệt khác và mỗi khối OB đó có một nhiệm vụ khác nhau, ví dụ các khối OB chứa các chương trình ngắt của các chương trình báo lỗi ,....Tuỳ thuộc vào từng loại CPU khác nhau mà có các khối OB khác nhau. Ví dụ các khối OB đặc biệt.

1. OB10: (Time of Day Interrupt): Chương trình trong khối OB10 sẽ được thực hiện khi giá trị của đồng hồ thời gian thực nằm trong một khoảng thời gian đã qui định. OB10 có thể được gọi một lần, nhiều lần cách đều nhau từng phút, từng giờ, từng ngày,....Việc qui định thời gian hay số lần gọi OB10 được thực hiện bằng chương trình hệ thống SFC28 hoặc trong bảng tham số modul CPU nhờ phần mềm Step7.
2. OB20: (Time Delay Interrupt): chương trình trong khối OB20 sẽ được thực hiện sau một khoảng thời gian trễ đặt trước kể từ khi gọi chương trình hệ thống SFC32 để đăt thời gian trễ.

3. OB35: (Cyclic Interrupt): Chương trình OB35 sẽ được thực hiện cách đều nhau một khoảng thời gian cố định. Mặc định khoảng thời gian này là 100ms, xong ta có thể thay đổi trong bảng đặt tham số cho CPU nhờ phần mềm Step7.
4. OB40 (Hardware Interrupt): Chương trình trong khối OB40 sẽ được thực hiện khi xuất hiện một tín hiệu báo ngắt từ ngoại vi đưa vào CPU thông qua các cổng vào/ra số onboard đặc biệt, hoặc thông qua các modul SM, CP, FM.
5. OB80: (cycle Time Fault): Chương trình sẽ được thực hiện khi thời gian vòng quét (scan time) vượt qua khoảng thời gian cực đại đã qui định hoặc khi có một tín hiệu ngắt gọi một khối OB nào đó mà khối OB này chưa kết thúc ở lần gọi trước. Mặc định, scan time cực đại là 150ms, nhưng có thể thay đổi tham số nhờ phần mềm Step7.
6. OB81(Power Supply Fault): nếu có lỗi về phần nguồn cung cấp thì sẽ gọi chương trình trong khối OB81.
7. OB82: (Diagnostic Interrupt) chương trình trong khối này sẽ được gọi khi CPU phát hiện có lỗi từ các modul vào/ra mở rộng. Với điều kiện các modul vào/ra này phải có chức năng tự kiểm tra mình.
8. OB85 (Not Load Fault): CPU sẽ gọi khối OB85 khi phát hiện chương trình ứng dụng có sử dụng chế độ ngắt nhưng chương trình xử lý tín hiệu ngắt lại không có trong khối OB tương ứng.
9. OB87 (Communication Fault): Chương trình trong khối này sẽ được gọi khi CPU phát hiện thấy lỗi trong truyền thông.
10. OB100 (Start Up Information): Khối này sẽ được thực hiện một lần khi CPU chuyển trạng thái từ STOP sang trạng thái RUN.
11. OB121: (Synchronouns error): Khối này sẽ được gọi khi CPU phát hiện thấy lỗi logic trong chương trình như đổi sai kiểu dữ liệu hoặc lỗi truy nhập khối DB, FC, FB không có trong bộ nhớ của CPU.
12. OB122 (Synchronouns error): Khối này sẽ được thực hiện khi CPU phát hiện thấy lỗi truy nhập Modul trong chương trình, ví dụ trong chương trình có lệnh truy nhập modul mở rộng nhưng lại không có modul này.

Để thực hiện thay đổi các chức năng của các khối OB trong CPU ta chỉ cần kích đúp chuột trái vào vị tí CPU trong bảng cấu hình cứng của Project khi đó trên màn hình sẽ xuất hiện một cửa sổ như sau:



Hình 1-9: Bảng thay đổi chức năng cho OB

Chú ý : K*hông phải tất cả các CPU đều có các khối OBs như đã giới thiệu. Số lượng và chủng loại khối OB tuỳ thuộc vào từng loại CPU.*

Chương 3: Thiết bị phần cứng của PLC:

3.1. Các modul:

3.1.1. Giới thiệu chung:

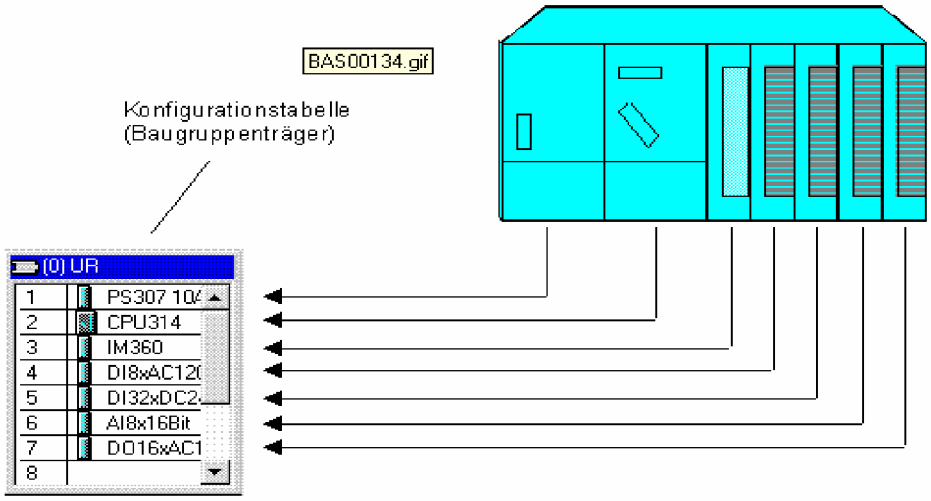
Muốn xây dựng một chương trình điều khiển sử dụng phần mềm Step7 cần thực hiện các thủ tục như sau:

- Khai báo cấu hình cứng cho một trạm PLC thuộc họ Simatic S7-300/400.
- Xây dựng cấu hình mạng gồm nhiều trạm PLC S7-300/400 cũng như thủ tục truyền thông giữa chúng.
- Soạn thảo và cài đặt chương trình điều khiển cho 1 hoặc nhiều trạm.
- Giám sát việc thực hiện chương trình điều khiển trong một trạm PLC và gỡ rối chương trình.

Ngoài ra Step 7 còn có cả một thư viện đầy đủ với các hàm chuẩn hữu ích, phần trợ giúp Online rất mạnh có khả năng trả lời mọi câu hỏi của người sử dụng về cách sử dụng Step 7, về cú pháp lệnh trong lập trình, về xây dựng cấu hình cứng của một trạm cũng như của một mạng gồm nhiều trạm PLC.

3.1.2. Các modul:

1/ PS(Power supply): modul nguồn nuôi. Có 3 loại 2A ,5A và 10A.



Hình 1-7:Sơ đồ bố trí một trạm PLC(S7-300).

2/ SM: Modul mở rộng cổng rín hiệu vào ra , bao gồm:

a) DI(Digital input): Modul mở rộng cổng vào số. Số các cổng vào của modul này có thể là 8, 16, 32 tuỳ thuộc vào từng loại modul.

b) DO(Digital output) Modul mở rộng cổng ra số. Số các cổng ra của modul này có thể là 8, 16, 32 tuỳ thuộc vào từng loại modul.

c) DI/DO: (Digital input/ Digital output): modul mở rổng các cổng vào/ra số số các cổng vào/ra có thể là 8 vào/8 ra hoặc 16 vào/16 ra tuỳ thuộc vào từng loại modul.

d) AI(Analog Input): Modul mở rổng các cổng vào tương tự. Về bản chất chúng chính là những bộ chuyển đổi tương tự-số (AD), tức là mỗi tín hiệu tương tự được chuyển thành một tín hiệu số (nguyên) có độ dài 12 bít, số các cổng vào có thể là 2, 4 hoặc 8 tuỳ thuộc vào từng loại Modul.

e) AO(Analog ouput): Modul mở rộng các cổng ra tín hiệu tương tự. Chúng chính là các bộ chuyển đổi số - tương tự (DA). Số các cổng ra tương tự có thể là 2 hoặc 4 tuỳ thuộc từng loại modul.

f) AI/AO (Analog input/Analog output): Modul mở rộng các cổng vào ra tương tự. Số các cổng có thể là 4 vào/2 ra hoặc 4 vào/4 ra tuỳ thuộc vào tùng loại modul.

3/ IM (Interface module): Modul ghép nối. Đây là loại modul chuyên dụng có nhiệm vụ nối từng nhóm các modul mở rộng lại với nhau thành một khối và được quản lý chung bởi một modul CPU. Thông thường các modul mở rộng được gá liền với nhau trên một thanh đỡ gọi là Rack. Trên mỗi một Rack chỉ có thể gá được nhiều nhất 8 modul mở rộng (không kể modul CPU, Modul nguồn nuôi). Một modul PU S7-300 có thể làm việc trực tiếp được với nhiều nhất 4 Racks và các Racks này phải được nối với nhau bằng modul IM.

4/ FM (Function modul): modul có chức năng điều khiển riêng , ví dụ Modul chức năng điều khiển động cơ bước , modul điều khiển động cơ Servo, modul PID, modul điều khiển vòng kín.

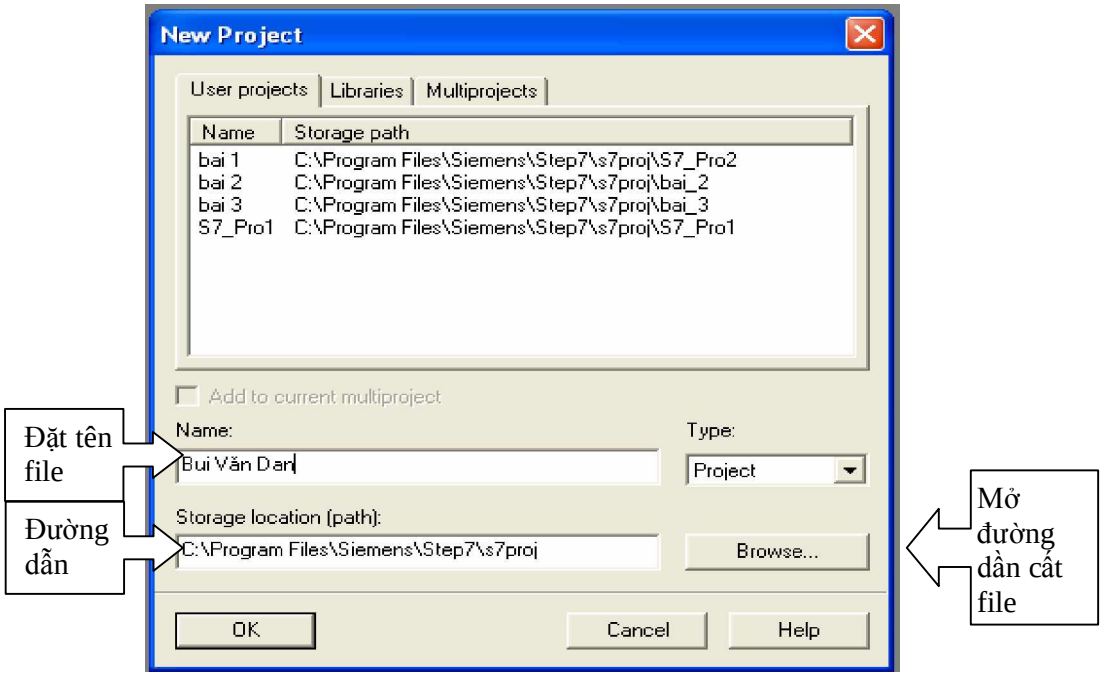
5/ CP (communication modul): Modul phục vụ truyền thông trong mạng giữa các PLC với nhau hoặc giữa PLC với máy tính.

3.2. Xây dựng cấu hình cứng cho trạm PLC.

3.2.1. Xây dựng cấu hình cứng:

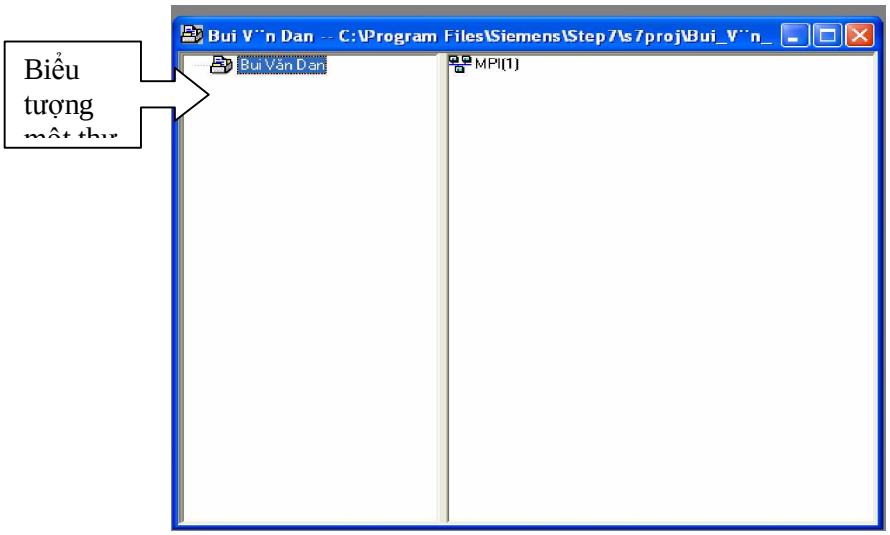
Bước1. Khai báo và mở một Project mới:

Mở một file mới : Vào File chọn New xuất hiện hộp thoại.(Hình 1.4)



Hình 1.4: Khai báo và mở một Project mới

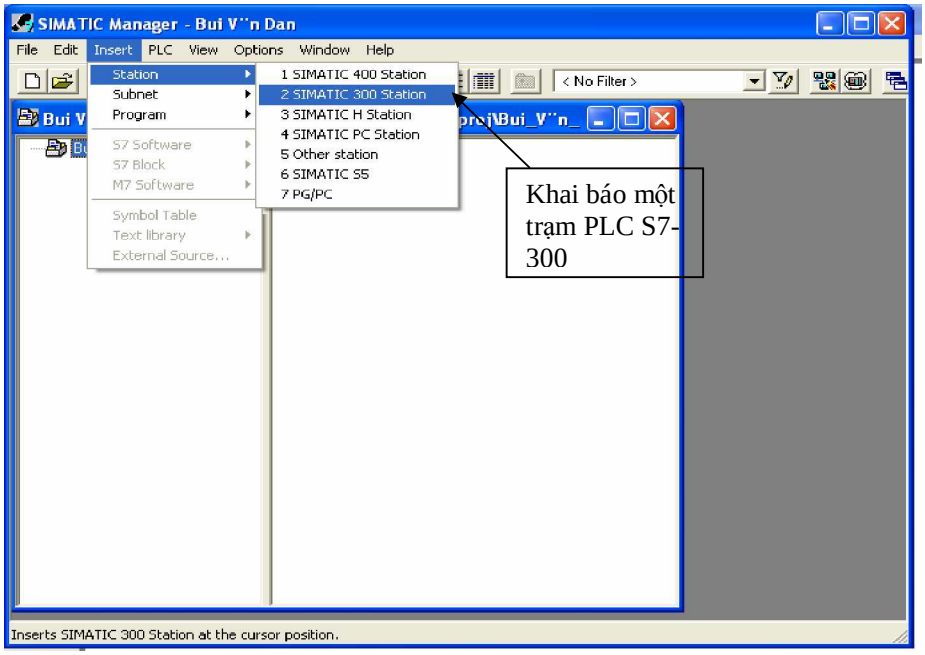
Sau khi khai báo xong một Project mới, trên màn hình sẽ xuất hiện Project đó nhưng ở dạng rỗng (chưa có gì trong project), điều này ta nhận biết được qua biểu tượng thư mục bên cạnh tên Project giống như một thư mục rỗng của Window.(Hình 1.5)



Hình 1.5 : Biểu tượng một Project mới.

Bước 2: Khai báo cấu hình cứng cho một trạm PLC :

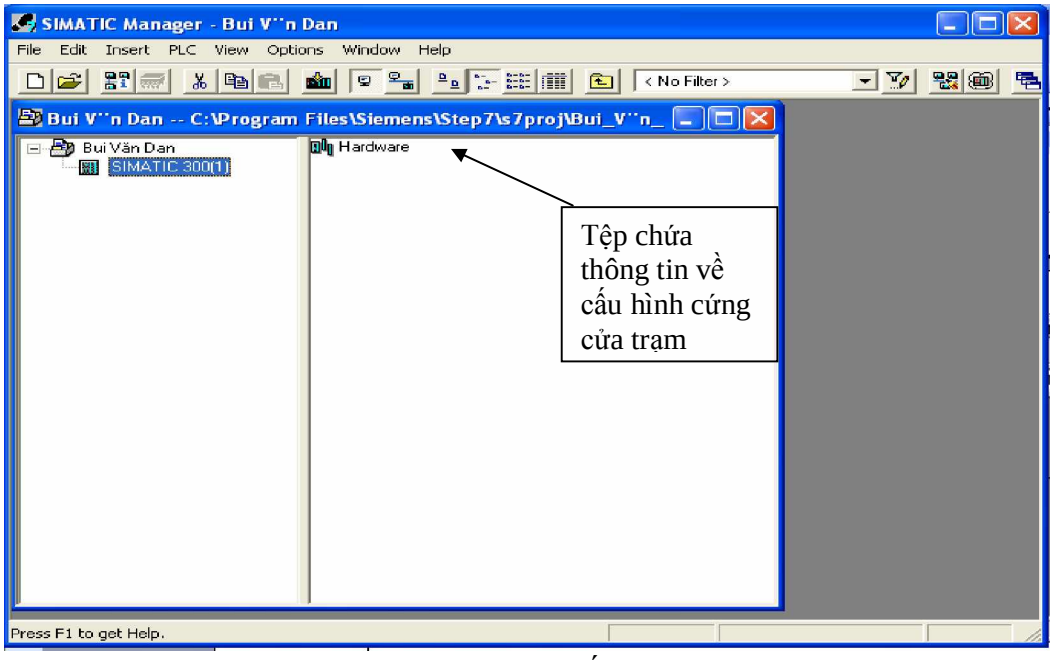
Với simatic S7-300 bằng cách vào: Insert -> Station ->Simatic 300- Station:(Hình 1.6)



Hình 1.6: Khai báo cấu hình cứng cho

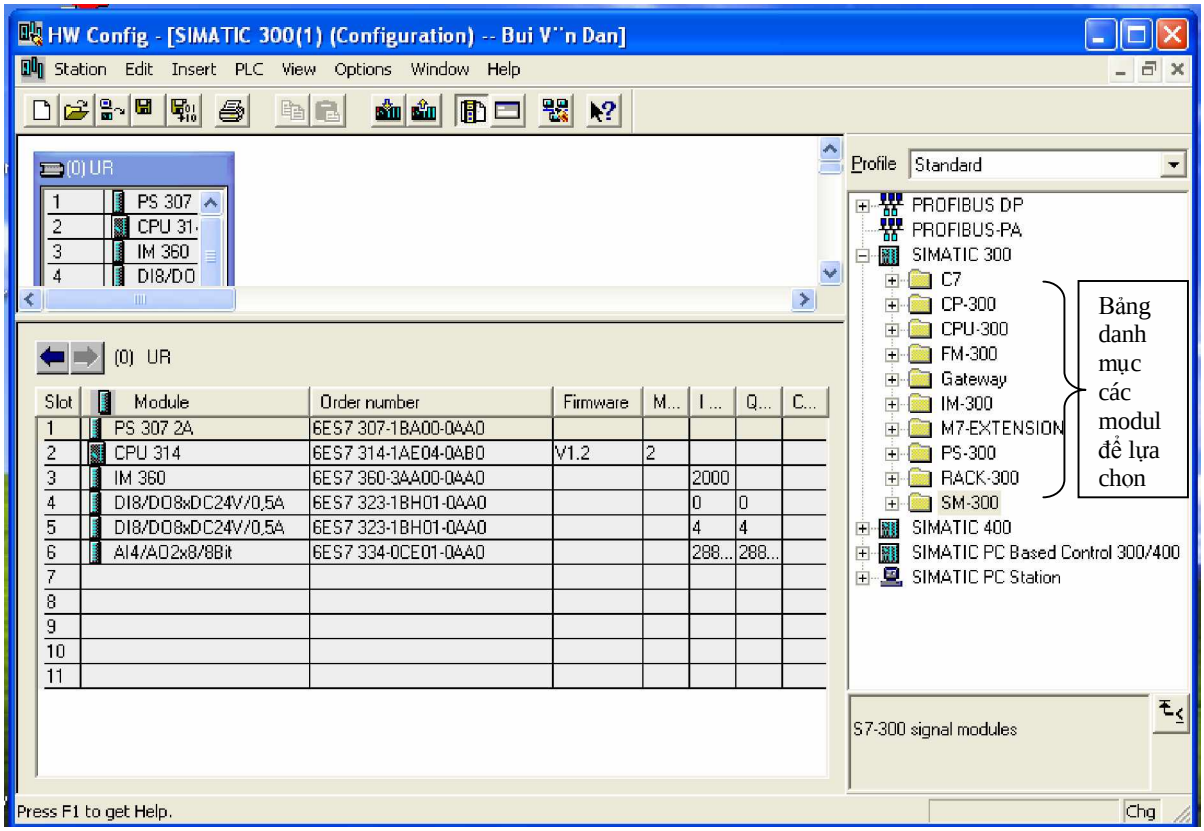
Trong trường hợp không muốn khai báo cấu hình cứng mà đi ngay vào chương trình ứng dụng ta có thể chọn thẳng. Động tác này sẽ hữu ích cho những trường hợp một trạm PLC có nhiều phiên bản ứng dụng khác nhau.

Sau khi đã khai báo một trạm (chèn một Station), thư mục Project chuyển sang dạng không rỗng với thư mục con trong nó tên mặc định là Simatic300(1) chứa tệp thông tin về cấu hình cứng của trạm.(Hình 1.7)



Hình 1.7:Màn hình khai báo cấu hình cứng cho tạm PLC

Để vào màn hình khai báo cấu hình cứng, ta nháy chuột tại biểu tượng Hardware. Trong hộp thoại hiện ra ta khai báo thanh Ray (Rack) và các module có trên thanh Ray đó.
Ví dụ:(Hình 1.8)



Hình 1.8: Thư viện để lấy các Modul

3.2.2. Các mã modul PLC :

Khi khai báo phải đúng mã hiệu cho từng module(*mã hiệu này được ghi trực tiếp trên các module*) thực hiện khai báo đúng trình tự sau:

1. *Khai báo nguồn nuôi cho CPU :* **SPS 307 2A – 6ES7 307_1BA00_ 0AA0**
2. *Khai báo loại CPU:* **CPU314 – 6ES7 314_1AE04_ 0AA0**
3. *Khai báo module IM đây là module ghép nối giữ các module mở rộng thành một khối và được quản lý chung bởi một CPU (nếu cần sử dụng thì khai báo):* **IFM 360 – 6ES7 360_3AA00_ 0AA0**

Chú ý: Từ vị trí số 4 – 11 đây là các vị trí để khai báo các modul mở rộng:

4. *Khai báo modul mở rộng DI/DO.* **: DI8/DO8xDC24V/0,5V – 6ES7 323 _ 1BH01_ 0AA0**
5. *Khai báo modul mở rộng DI/DO.:* **DI8/DO8xDC24V/0,5V – 6ES7 323 _ 1BH01_ 0AA0**
6. *Khai báo modul mở rộng AI/AO.:* **AI4/AO2x8/8bit – 6ES7 334 _ 0CE01_ 0AA0**
7. .....

Step7 giúp việc khai báo cấu hình cứng được đơn giản nhờ bảng danh mục các module của nó. Muốn đưa module nào vào bảng cấu hình ta chỉ cần đánh dấu vị trí nơi module sẽ được đưa vào rồi nháy kép chuột trái tại tên của module đó trong bảng danh mục các module kèm theo.

Chương 4: Ngôn ngữ lập trình

4.1..Qui trình thiết kế chương trình điều khiển dùng PLC:

Qui trình thiết kế hệ thống điều khiển dùng PLC bao gồm các bước sau:

1.Xác định qui trình điều khiển:

Điều đầu tiên cần biết là đối tượng điều khiển của hệ thống, mục đích cính của PLC là phải điều khiển được các thiết bị ngoại vi. Các chuyển động của đối tượng điều khiển được kiểm tra thường xuyên bởi các thiết bị vào, các thiết bị nạy gửi tín hiệu đến PLC và tiếp theo đó PLC sẽ đưa tín hiêu điều khiển đến các thiết bị để điều khiển chuyển động của đối tượng. Để đơn giản, qui trình điều khiển có thể mô tả theo lưu đồ (hình vẽ 2-3).

2.xác định tín hiệu vào ra:

Bước thứ hai là phải xác định vị trí kết nối giữa các thiết bị vào ra với PLC. Thiết bị vào có thể là tiếp điểm, cảm biến, Thiết bị ra có thể là Rơle điện từ, Motor, đèn, Mỗi vị trí kết nối được đánh số tương tự ứng với PLC sử dụng.

3.Soạn thảo chương trình:

Chương trình điều khiển được soạn thảo dưới dạng lưu đồ hình thang như đã trình bày ở bước 1.

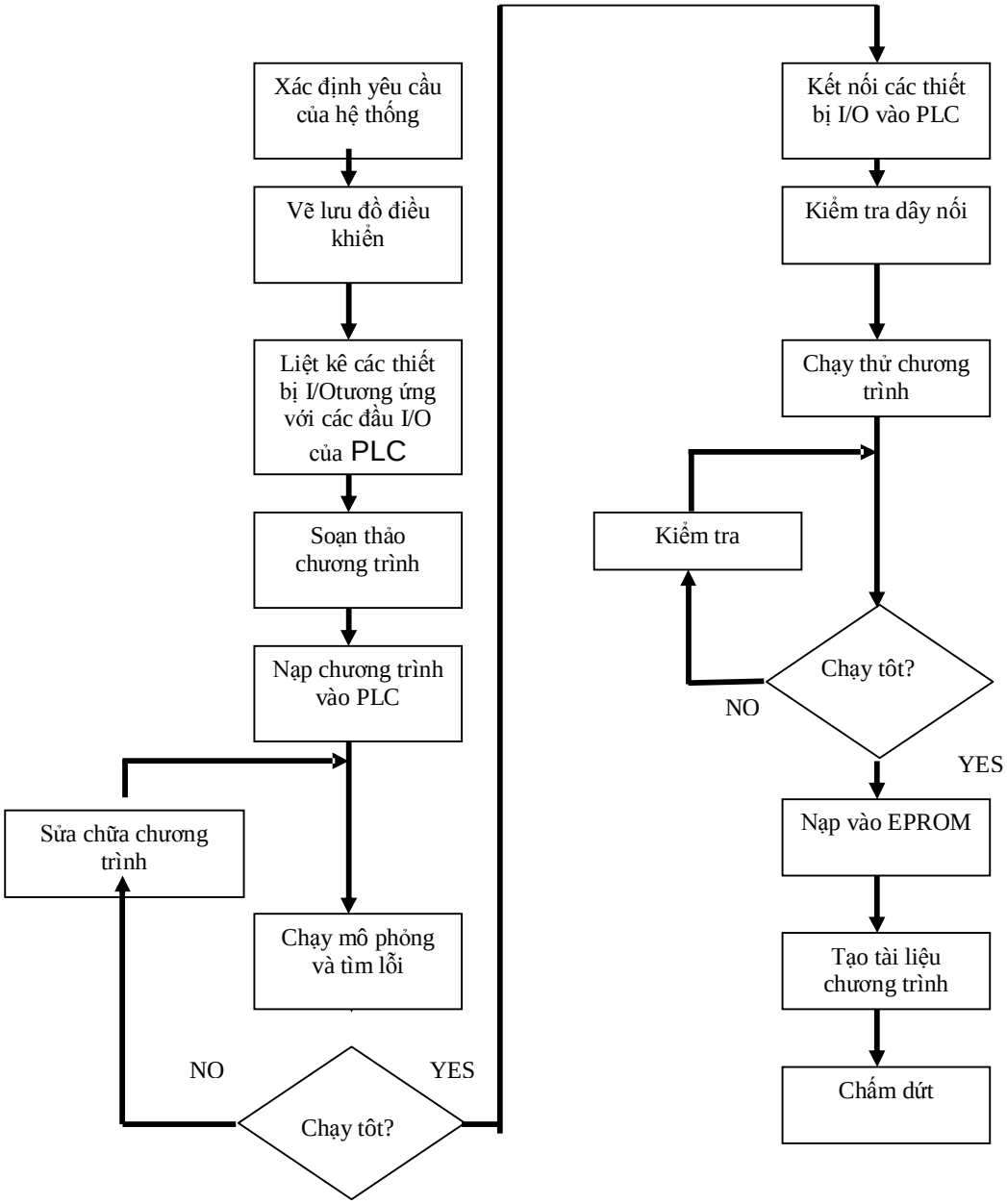
4.Nạp chương trình vào bộ nhớ:

Cấp nguồn cho PLC, cài đặt cấu hình khối giao tiếp I/O nếu cần (Phụ thuộc vào từng loại PLC). Sau đó nạp chương trình soạn thảo trên màn hình vào bộ nhớ của PLC. Sau khi hoàn tất nên kiểm tra lỗi bằng chức năng tự chuẩn đoán và nếu có thể thì chạy chương trình mô phỏng hoạt động của hệ thống (Ví dụ chương trình S7-SIM, S7- VISU,...).

5.Chạy chương trình:

Trước khi khởi động hệ thống cần phải chắc chắn dây nối từ PLC đến các thiết bị ngoại vi là đúng, trong quá trình chạy kiểm tra có thể cần thiết phải thực hiện các bước tinh chỉnh hệ thống nhằm đảm bảo an toàn khi đưa vào hoạt động thực tế.

Qui trình thiết kế hệ thống điều khiển bằng PLC:



Hình 2-3: Qui trình thiết kế một hệ thống điều khiển tự động.

4.2.Các ngôn ngữ lập trình:

a) Các ngôn ngữ lập trình

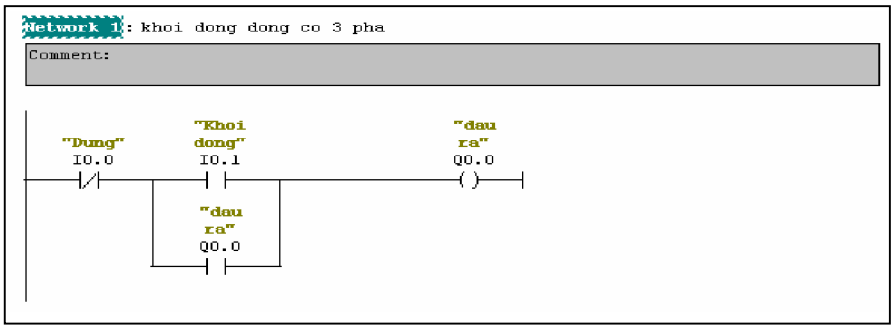
Đối với PLC S7-300 có thể sử dụng 6 ngôn ngữ để lập trình.

1/ Ngôn ngữ lập trình LAD:

Với loại ngôn ngữ này rất thích hợp với người quen thiết kế mạch điều khiển logic

chương trình được viết dưới dạng liên kết giữa các công tắc:

ví dụ:



Hình 2-4: ví dụ kiểu lập trình LAD.

2/ Ngôn ngữ lập trình FBD :

Loại ngôn ngữ này thích hợp cho những người quen sử dụng và thiết kế mạch điều khiển số.

Chương trình được viết dưới dạng liên kết của các hàm logic kỹ thuật số:

Ví dụ:

Hình 2-5: Ví dụ kiểu lập trình FBD.

3/ Ngôn ngữ lập trình STL

Đây là ngôn ngữ lập trình thông thường của máy tính. Một chương trình được ghép bởi nhiều lệnh theo một thuật toán nhất định, mỗi lệnh chiếm một hàng và đều có cấu trúc chung là : "tên lệnh" + "toán hạng".

Ví dụ:

Hình 2-6: Ví dụ kiểu lập trình STL.

4/ Ngôn ngữ lập trình SCL (Structured Control Language):

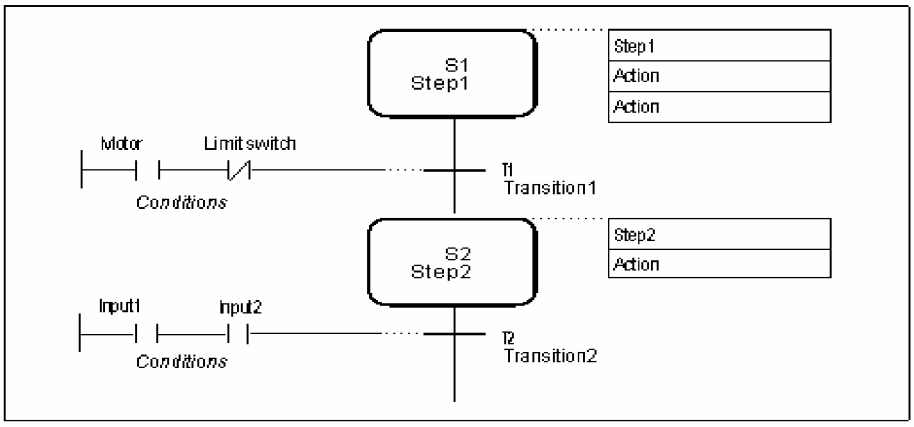
Kiểu viết chương trình này sử dụng ngôn ngữ PASCAL. Rất phù hợp cho những người đã viết các chương trình bằng ngôn ngữ máy tính.

ví dụ:

```
FUNCTION_BLOCK FB 20
VAR_INPUT
ENDWERT : INT;
END_VAR
VAR_IN_OUT
IQ1 : REAL;
END_VAR
VAR
INDEX : INT;
END_VAR
BEGIN
CONTROL := FALSE;
FOR INDEX := 1 TO ENDWERT DO
IQ1 := IQ1 * 2;
IF IQ1 > 10000 THEN
CONTROL = TRUE
END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

5/ Ngôn ngữ lập trình : S7-Graph.

Ví dụ:

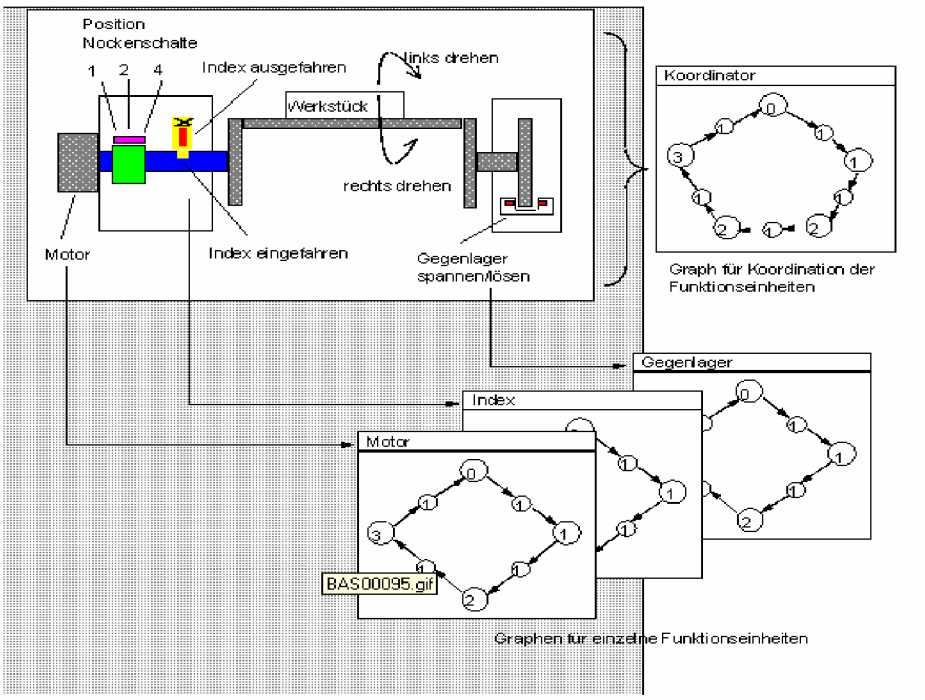


Hình2-7: Sơ đồ khối lập trình kiểu S7-Graph.

6/ Ngôn ngữ lập trình : S7-HiGraph.

Đây là một loại ngôn ngữ viết chương trình rất phù hợp cho các bài toán làm việc có tính tuần tự. Tại mỗi thời điểm chỉ có một bước được thực hiện. Với kiểu lập trình này người lập trình phải sử dụng phương pháp lập trình có cấu trúc.

Ví dụ:



Hình 2-8 : Sơ đồ lập trình bằng ngôn ngữ S7-HiGraph.

Trong cuốn tài liệu này sẽ giới thiệu 4 loại ngôn ngữ dùng để lập trình (FBD, STL, LAD và S7GRAPH) trong phần bài tập mẫu.

b) .Nạp chương trình và giám sát việc thực hiện chương trình.

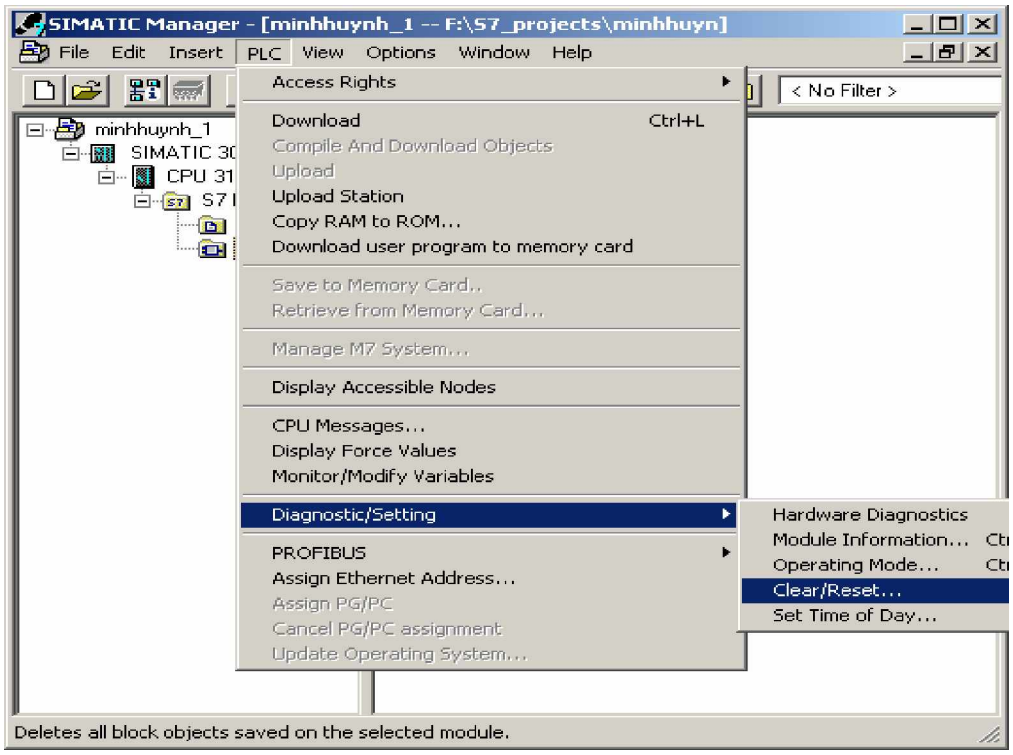
+ . Nạp chương trình soạn thảo từ PC xuống CPU:

Chương trình sau khi đã soạn thảo cần được truyền xuống CPU. Để làm được điều này, ta nhấn chuột trái vào biểu tượng này  trên thanh công cụ và trả lời đầy đủ các câu hỏi. Chú ý khi nạp chương trình cần phải đặt CPU ở trạng thái Stop hoặc đặt CPU ở trạng thái RUN-P.

+ .Xoá chương trình đã có trong CPU:

Để thực hiện việc nạp chương trình mới từ PC xuống CPU ta cần thực hiện công việc xoá chương trình đã có sẵn trong CPU. Điều này ta thực hiện các bước như sau:

- Đưa trạng thái của CPU về STOP : Từ màn hình chính của Step7 ta chọn lệnh:

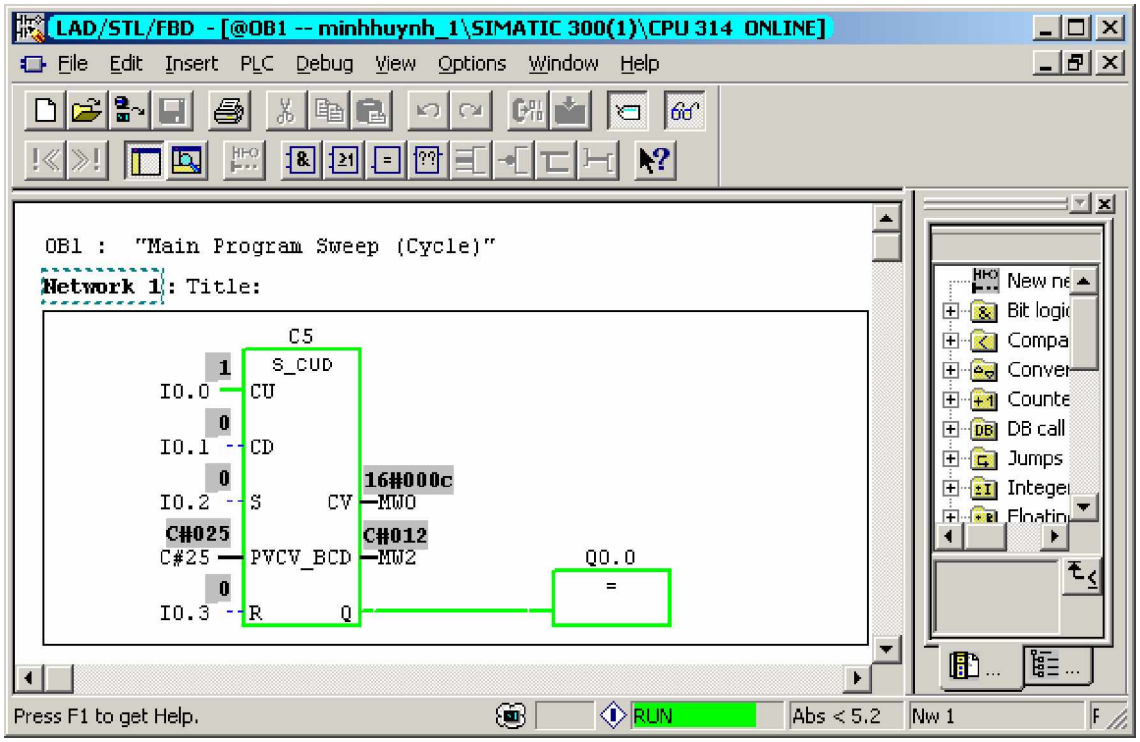


Hình 3-29

+ .Quan sát việc thực hiên chương trình:

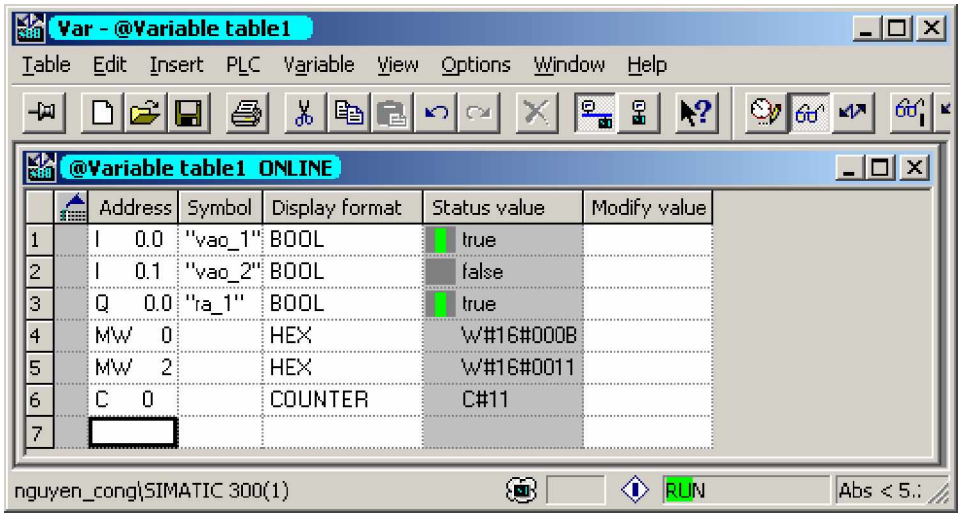
Sau khi đã nạp chương trình soạn thảo xuống CPU lúc này chương trình đã được ghi vào bộ nhớ của CPU. Khi đó ta có thể tách rời PC và CPU của S7 mà chương trình vẫn hoạt động bình thường. Để thực hiện việc quan sát quá trình hoạt động của chương trình và CPU ta sử dụng chức năng giám sát chương trình bằng cách nhấn vào biểu tượng này  trên thanh công cụ. Sau khi chọn chức năng giám sát chương trình này thì trên màn hình sẽ xuất hiện một cửa sổ sau:

Tuỳ theo kiểu viết chương trình mà ta nhận được sự khác nhau về kiểu hiển thị trên màn hình (Dưới đây sử dụng kiểu viết chương trình FBD).



Hình 3-30: Quan sát quá trình hoạt động.

Ngoài ra ta còn có thể quan sát được nội dung của ô nhớ. Những ô nhớ muốn quan sát cần phải khai báo trong bảng Variable.



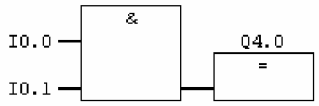
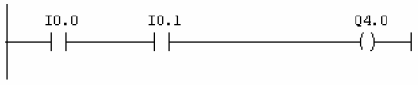
Hình 3-31: Quan sát nội dung của ô nhớ.

Sau khi khai báo tất cả các biến cần quan sát ta kích vào phím quan sát trên màn hình xuất hiện cửa sổ như hình trên. Tuỳ theo yêu cầu mà ta kích vào phím quan sát tương ứng

trên màn hình sẽ hiển thị nội dung của ô nhớ tại thời điểm hiện tại hay liên tục quan sát theo từng thời điểm.

4.3.Nhóm hàm Logic tiếp điểm:

1/ Hàm AND : Toán hạng là kiểu dữ liệu BOOL hay địa chỉ bit I,Q, M, T, C, D, L

FBD	LAD	STL
		A I 0.0 A I 0.1 = Q 4.0

Hình 4-1: Cách khai báo hàm AND

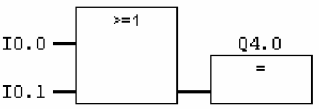
Tín hiệu ra Q4.0 sẽ bằng 1 khi đồng thời tín hiệu I0.0=1 và I0.1=1.

Dữ liệu vào và ra :

Vào: I0.0, I0.1: BOOL

Ra : Q4.0 : BOOL

2/ Hàm OR : Toán hạng là kiểu dữ liệu BOOL hay địa chỉ bit I,Q, M, T, C, D, L.

FBD	LAD	STL
		0 I 0.0 0 I 0.1 = Q 4.0

Hình 4-1: Cách khai báo hàm AND

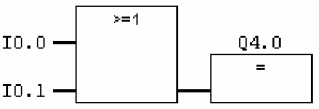
Tín hiệu ra Q4.0 sẽ bằng 1 khi đồng thời tín hiệu I0.0=1 và I0.1=1.

Dữ liệu vào và ra :

Vào: I0.0, I0.1: BOOL

Ra : Q4.0 : BOOL

2/ Hàm OR : Toán hạng là kiểu dữ liệu BOOL hay địa chỉ bit I,Q, M, T, C, D, L.

FBD	LAD	STL
		0 I 0.0 0 I 0.1 = Q 4.0

Hình 4-2: Khai báo hàm OR

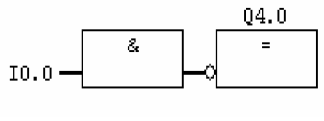
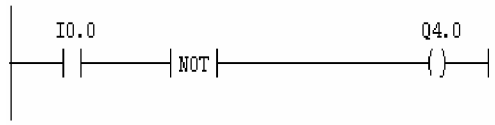
Tín hiệu ra sẽ bằng 1 khi ít nhất có một tín hiệu vào bằng 1.

Dữ liệu vào và ra:

Vào : I0.0, I0.1: BOOL

Ra : Q4.0: BOOL

3/ Hàm NOT:

FBD	LAD	STL
		U E 0.0 NOT = A 4.0

Hình 4-3: Khai báo hàm thực hiện chức năng phủ định.

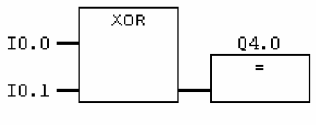
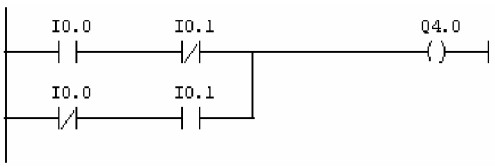
Tín hiệu ra sẽ là nghich đảo của tín hiệu vào.

Dữ liệu vào và ra:

Vào : I0.0 : BOOL

Ra : Q4.0 : BOOL

4/ Hàm XOR: Toán hạng là kiểu dữ liệu BOOL hay địa chỉ bit I, Q, M, T, C, D, L.

FBD	LAD	STL
		X I 0.0 X I 0.1 = Q 4.0

Hình 4-4: Khối thực hiện chức năng XOR.

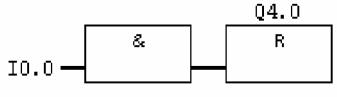
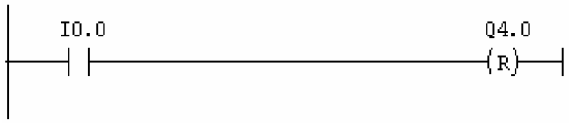
Tín hiệu ra Q4.0= 1 khi I0.0 khác I0.2

Dữ liệu vào và ra:

Vào: I0.0, I0.1 : BOOL

Ra : Q4.0 : BOOL

5/ Lệnh xoá RESET: Toán hạng là địa chỉ bit I, Q, M, T, C, D, L.

FBD	LAD	STL
		A I 0.0 R Q 4.0

Hình 4-5: Khối thực hiện chức năng RESET

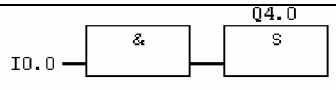
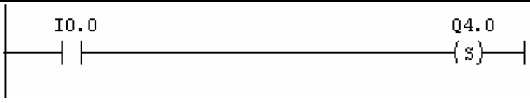
Tín hiệu ra Q4.0 = 0 (Q4.0 sẽ được xoá) khi I0.0 =1 .

Dữ liệu vào và ra:

Vào: I0.0 : BOOL

Ra : Q4.0 : BOOL

6/ Lệnh SET: Toán hạng là địa chỉ bit I, Q, M, T, C, D, L.

FBD	LAD	STL
		A I 0.0 S Q 4.0

Hình 4-6: Khối thực hiện chức năng SET.

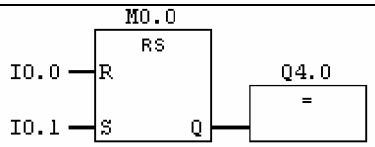
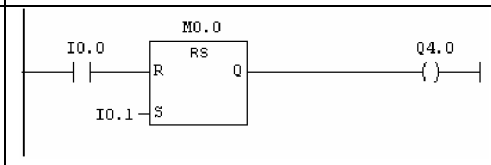
Tín hiệu ra Q4.0 = 1 (Q4.0 sẽ được thiết lập) khi I0.0 =1.

Dữ liệu vào và ra:

Vào I0.0 : BOOL

Ra Q4.0 : BOOL

7/Bộ nhớ RS: Toán hạng là địa chỉ bit I, Q, M, D, L.

FBD	LAD	STL
		A I 0.0 R M 0.0 A I 0.1 S M 0.0 A M 0.0 = Q 4.0

Hình 4-7: Khối thực hiện chức năng RS.

Khi I0.0 = 1 và I0.1 =0 Merker M0.0 bị Reset và đầu ra Q4.0 là "0". Nếu I0.0 = 0 và I0.1 = 1 thì Set cho M0.0 và đầu ra Q4.0 là "1".

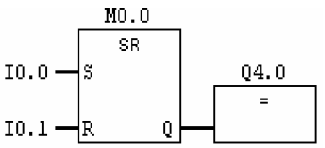
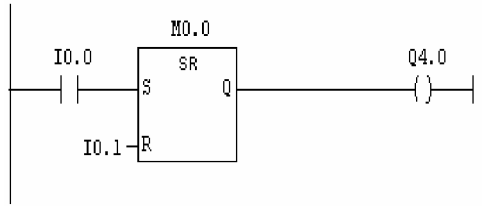
Khi cả hai đầu vào Set va Reset cùng đồng thời =1 thì M0.0 và Q4.0 có giá trị là "1".

Dữ liệu vào và ra:

Vào I0.0, I0.1 : BOOL

Ra Q4.0 : BOOL

8/ Bộ nhớ SR: Toán hạng là địa chỉ bit I, Q, M, D, L

FBD	LAD	STL
		A I 0.0 S M 0.0 A I 0.1 R M 0.0 A M 0.0 = Q 4.0

Hình 4-8: Khối thực hiện chức năng SR

Khi I0.0 = 1 và I0.1 =0 thì Set cho Merker M0.0 và đầu ra Q4.0 là "1". Nếu I0.0 = 0 và I0.0 = 1 thì M0.0 bị Reset và đầu ra Q4.0 là "0".

Khi cả hai đầu vào Set va Reset cùng đồng thời =1 thì M0.0 và Q4.0 có giá trị là "0".

Dữ liệu vào và ra:

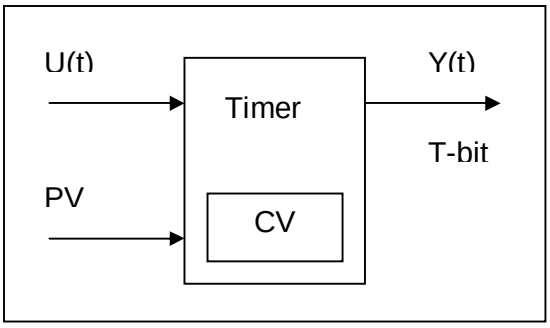
Vào	I0.0, I0.1	: BOOL
Ra	Q4.0	: BOOL

Chú ý: Trong kỹ thuật số trạng thái của trigơ RS sẽ bị cấm khi R=1 và S=1. Nên ở đây có hai loại bộ nhớ RS và SR là loại Trigơ ưu tiên R hay ưu tiên S

4.4.Bộ thời gian:

4.4.1 Nguyên lý làm việc chung của bộ Timer.

Bộ thời gian Timer là bộ tạo thời gian trễ T mong muốn giữa tín hiệu logic đầu vào X(t) và đầu ra Y(t)



Hình 4-34: Sơ đồ khối bộ thời gian.

S7-300 có 5 bộ thời gian Timer khác nhau. Tất cả 5 loại Timer này cùng bắt đầu tạo thời gian trễ tín hiệu kể từ thời điểm có sườn lên của tín hiệu đầu vào , tức là khi có tín hiệu đầu vào U(t) chuyển trạng thái từ logic "0" lên logic"1", được gọi là thời điểm Timer được kích.

Thời gian trễ T mong muốn được khai báo với Timer bằng giá trị 16 bits bao gồm hai thành phần :

- Độ phân giải với đơn vị là mS. Timer của S7 có 4 loại phân giải khác nhau là 10ms, 100ms, 1s và 10s.
- Một số nguyên BCD trong khoảng từ 0 đến 999 được gọi là PV (Preset Value- giá trị đặt trước).

Như vậy thời gian trễ T mong muốn sẽ được tính như sau :

T= Độ phân giải x PV.

Ngay tại thời điểm kích Timer, giá trị PV được chuyển vào thanh ghi 16 bits của Timer T-Word (gọi là thanh ghi CV- Curren value- giá trị tức thời). Timer sẽ ghi nhớ khoảng thời gian trôi qua kể từ khi kích bằng cách giảm dần một cách tương ứng nội dung thanh ghi CV. Nếu nội dung thanh ghi CV trở về bằng 0 thì Timer đã đạt được thời gian mong muốn T và điều này được báo ra ngoài bằng cách thay đổi trạng thái tín hiệu đầu ra Y(t). Việc thông báo ra ngoài bằng cách đổi trạng thái tín hiệu dầu ra Y(t) như thế nào còn phụ thuộc vào loại Timer được sử dụng.

Bên cạnh sườn lên của tín hiệu đầu vào U(t), Timer còn có thể kích bằng sườn lên của tín hiệu kích chủ động có tên là tín hiệu ENABLE nếu như tại thời điểm có sườn lên của tín hiệu ENABLE, tín hiệu đầu vào U(t) có gic là "1".

Từng loại Timer được đánh số từ 0 đến 255 (tuỳ thuộc vào từng loại CPU). Một Timer được đặt tên là Tx, trong đó x là số hiệu của Timer (0<=x<=255). Ký hiệu Tx cũng đồng thời là tín hiệu hình thức của thanh ghi CV (T-Word) và đầu ra T-bits của Timer đó . Tuy

chúng có cùng địa chỉ hình thức , nhưng T-Word và T-bits vẫn được phân biệt với nhau nhờ kiểu lệnh sử dụng toán hạng Tx. Khi dùng làm việc với từ Tx được hiểu là T-Word còn khi làm việc với điểm thi Tx được hiểu là T-bit.

Để xóa tức thời trạng thái của T-word và T-bit người ta sử dụng một tín hiệu reset Timer . Tại thời điểm sườn lên của tín hiệu này giá trị T-Word và T-bit đồng thời có giá trị bằng 0 tức là thanh ghi tức thời CV được đặt về 0 và tín hiệu đầu ra cũng có trạng thái Logic là "0". Trong thời gian tín hiệu Reset có giá trị logic là "1" Timer sẽ không làm việc.

4.4.2. Khai báo sử dụng:

Các tín hiệu điều khiển cho một bộ Timer phải được khai báo bao gồm các bước sau:

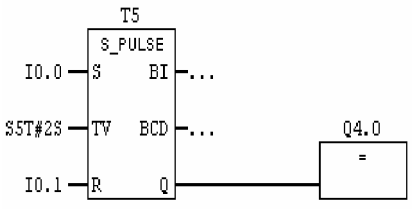
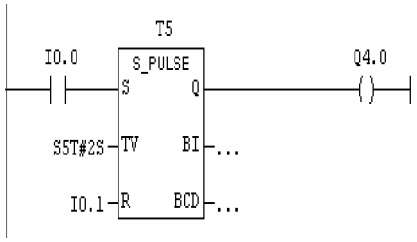
- Khai báo tín hiệu ENABLE nếu muốn sử dụng tín hiệu chủ động kích.
- Khai báo tín hiệu đầu vào U(t).
- Khai báo thời gian trễ mong muốn TW.
- Khai báo loại Timer được sử dụng (SP, SE, SD, SS, SF).
- Khai báo tín hiệu xoá Timer nếu muốn sử dụng chế độ Reset chủ động.

  Trong các bước trên thì bước 1 và 5 có thể bỏ qua .
- Dạng dữ liệu vào / ra của bộ Timer:

S : BOOL	BI (DUAL): WORD
TW: S5TIME	BCD (DEZ) : WORD
R : BOOL	Q : BOOL

1. *Bộ thời gian SP:*

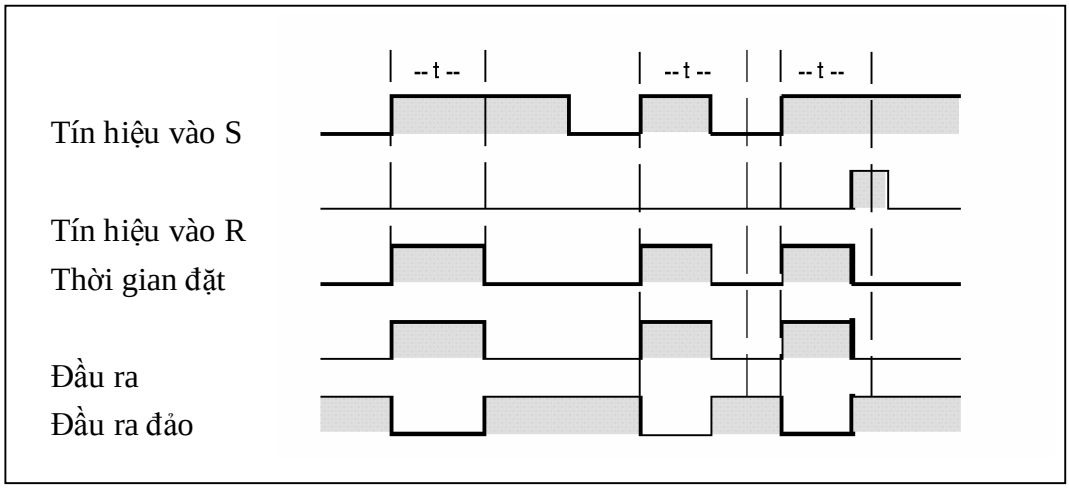
-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 L S5T#2S SP T 5 A I 0.1 R T 5 NOP 0 NOP 0 A T 5 = Q 4.0

Hình 4-35: Bộ thời gian SP.

-Nguyên lý làm việc:

Tại thời điểm sườn lên của tín hiệu vào SET thời gian sẽ đươc tính đồng thời giá trị Logic ở đầu ra là "1". Khi thời gian đặt kết thúc giá trị đầu ra cũng trở về 0.



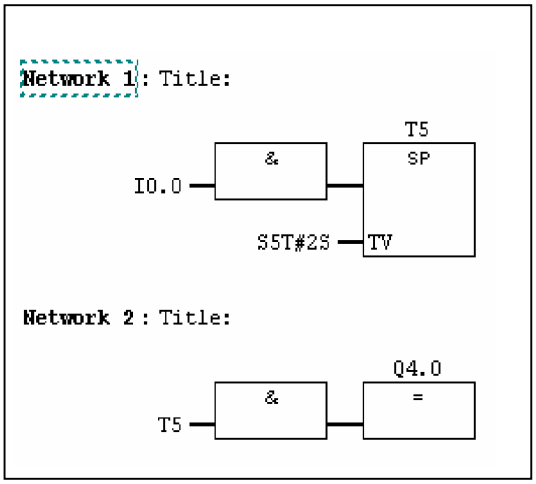
Hình 4-36: Giản đồ thời gian của bộ tạo trễ kiểu SP.

Khi có tín hiệu RESET (R) thời gian tính lập tức trở về 0 và tín hiệu đầu ra cũng giá trị là "0".

-Trường hợp không sử dụng các tín hiệu đầu vào SET(S), RESET (R), BI và BCD ta sử dụng khối Timer SI sau:

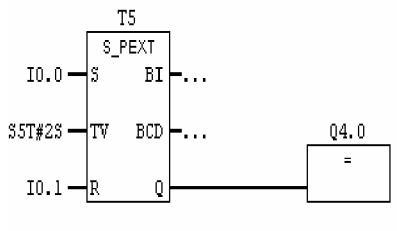
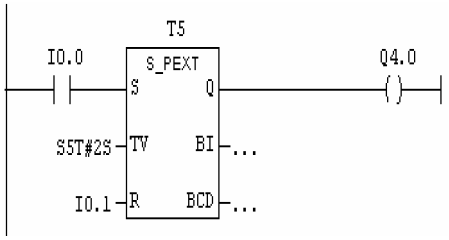
Tín hiệu đầu vào I0.0 chính là tín hiệu kích.
S5T#2s là thời gian đặt 2s

Tín hiệu ra của bộ thời gian tác động tới đầu ra Q4.0



Hình 4-37: Ví dụ khai báo một bộ thời gian SP

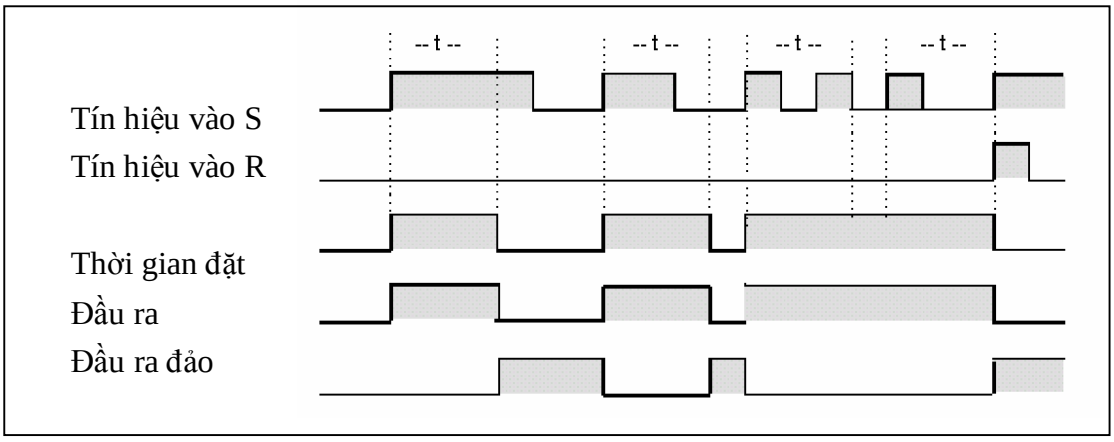
2. *<u>Bộ thời gian SE.</u>*

FBD	LAD	STL
		A I 0.0 L S5T#2S SE T 5 A I 0.1 R T 5 NOP 0 NOP 0 A T 5 = Q 4.0

Hình 4-38: Khối hàm thời gian SE

-Nguyên lý làm việc:

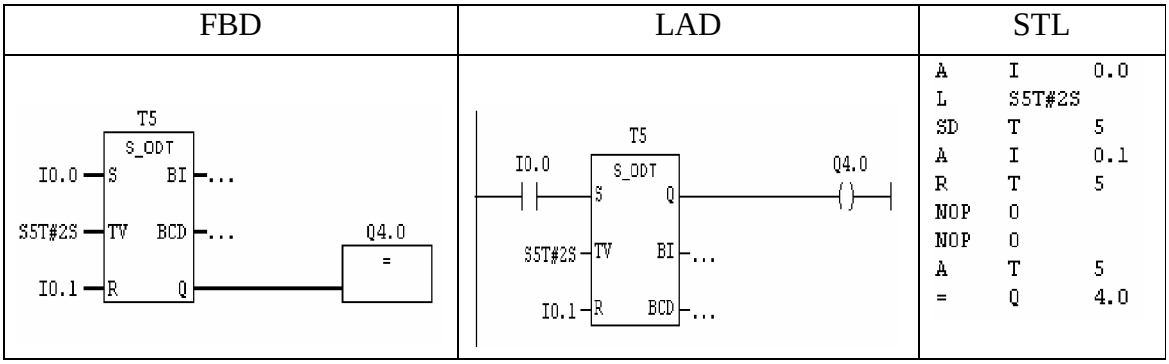
Tại thời điểm sườn lên của tín hiệu vào SET cuối cùng bộ thời gian đươc thiết lập và thời gian sẽ đươc tính đồng thời giá trị Logic ở đầu ra là "1". Kết thúc thời gian đặt tín hiệu đầu ra sẽ trở về 0.



Hình 4-39: Giản đồ thời gian khối SE

Khi có tín hiệu RESET (R) thời gian tính lập tức trở về 0 và tín hiệu đầu ra cũng giá trị là "0".

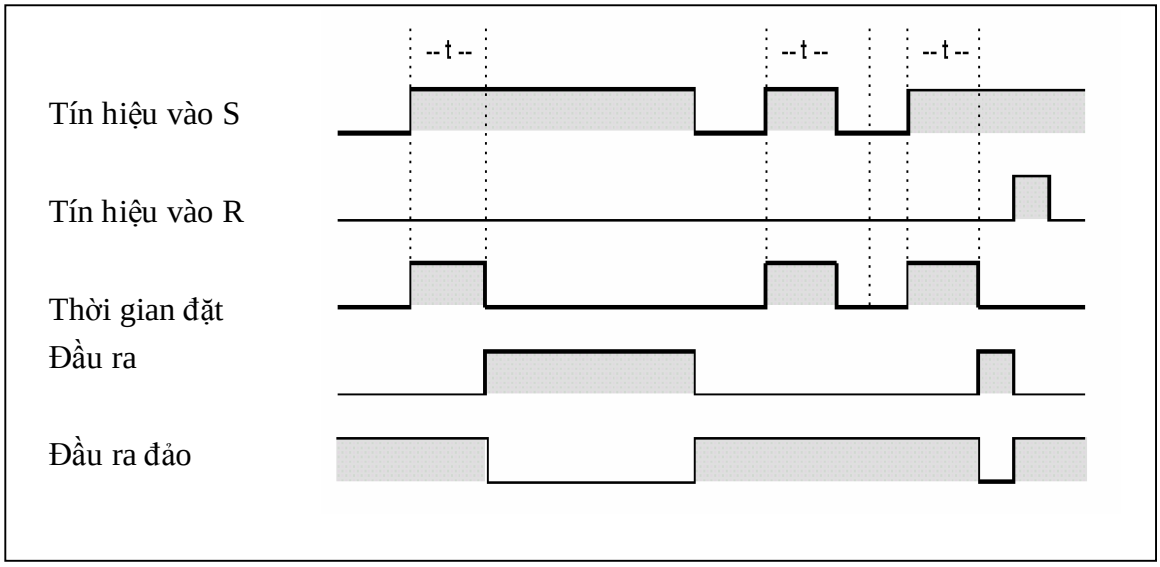
3. <u>Bộ thời gian SD.</u>



Hình 4-40: Sơ đồ khối hàm SD.

-Nguyên lý làm việc:

Tại thời điểm sườn lên của tín hiệu vào SET bộ thời gian đươc thiết lập và thời gian sẽ đươc tính. Kết thúc thời gian đặt tín hiệu đầu ra sẽ có giá trị là "1". Khi tín hiệu đầu vào kích S là "0" đầu ra cũng lập tức trở về "0" nghĩa là tín hiệu đầu ra sẽ không được duy trì hi tín hiệu kích có giá trị là "0".



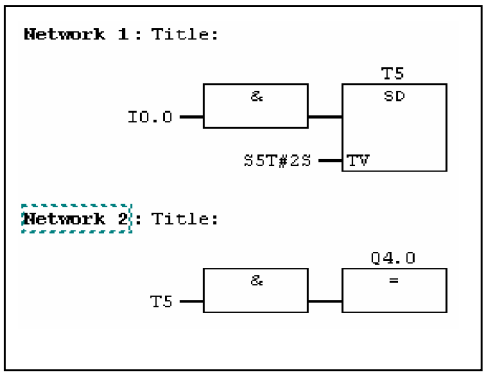
Hình 4-41: Giản đồ thời gian SD.

Khi có tín hiệu RESET (R) thời gian tính lập tức trở về "0" và tín hiệu đầu ra cũng giá trị là "0".

-Trường hợp không sử dụng các tín hiệu đầu vào SET(S), RESET (R), BI và BCD ta sử dụng khối Timer SE sau:

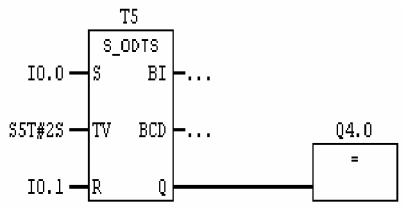
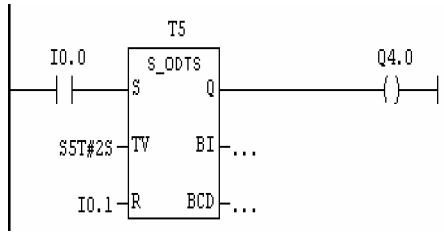
Tín hiệu đầu vào I0.0 chính là tín hiệu kích.
S5T#2s là thời gian đặt 2s

Tín hiệu ra của bộ thời gian tác động tới đầu ra Q4.0.

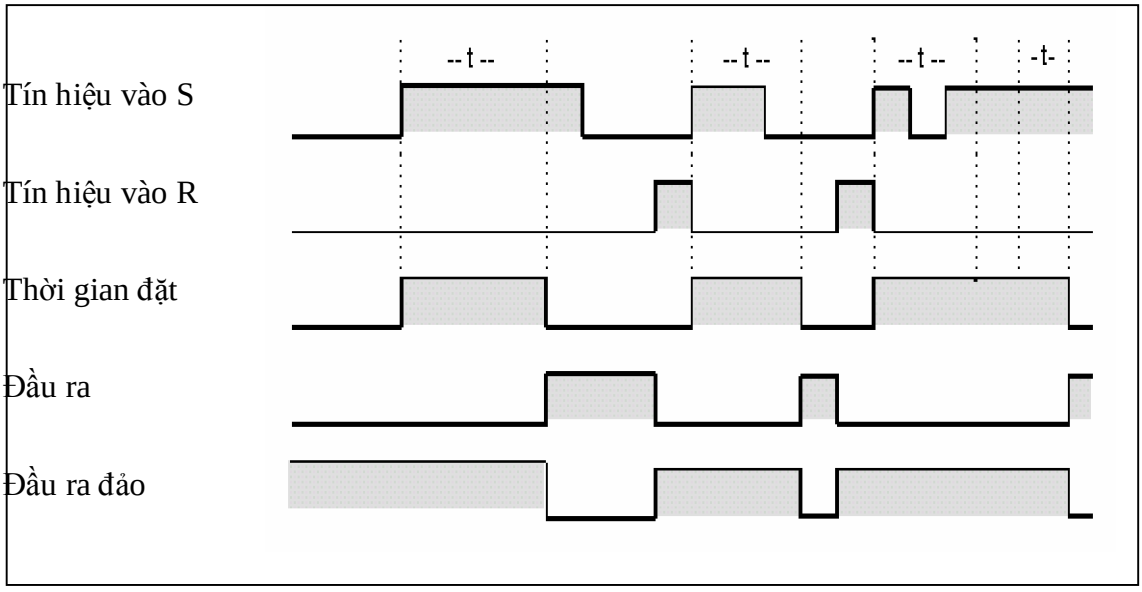


Hình 4-42: ví dụ sử dụng khối hàm SD.

4. *Bộ thời gian SS:*

FBD	LAD	STL
		A I 0.0 L S5T#2S SS T 5 A I 0.1 R T 5 NOP 0 NOP 0 A T 5 = Q 4.0

Hình 4-43: Khai báo bộ thời gian SS .



Hình 4-44: Giản đồ thời gian hàm SS.

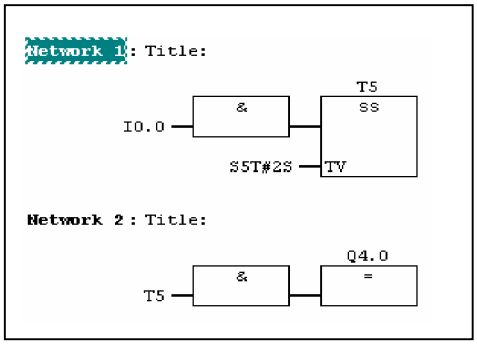
-Nguyên lý làm việc:

Tại thời điểm sườn lên của tín hiệu vào SET bộ thời gian đực thiết lập và thời gian sẽ đực tính. Kết thúc thời gian đặt tín hiệu đầu ra sẽ có giá trị 1 giá trị này vẫn duy trì ngay cả khi tín hiệu đầu vào kích S có giá trị là 0. Khi có tín hiệu RESET (R) thời gian tính lập tức trở về 0 và tín hiệu đầu ra cũng giá trị là "0".

-Trường hợp không sử dụng các tín hiệu đầu vào SET(S), RESET (R), BI và BCD ta sử dụng khối Timer SS sau:

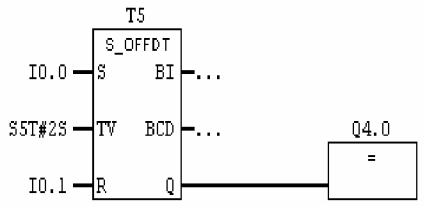
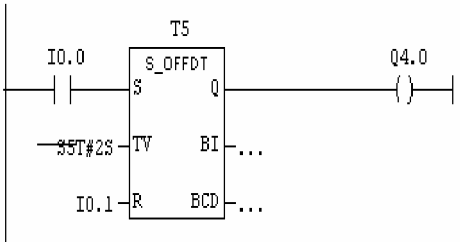
Tín hiệu đầu vào I0.0 chính là tín hiệu kích.
S5T#2s là thời gian đặt 2s

Tín hiệu ra của bộ thời gian tác động tới đầu ra Q4.0



Hình 4-45: Ví dụ sử dụng khối hàm SS

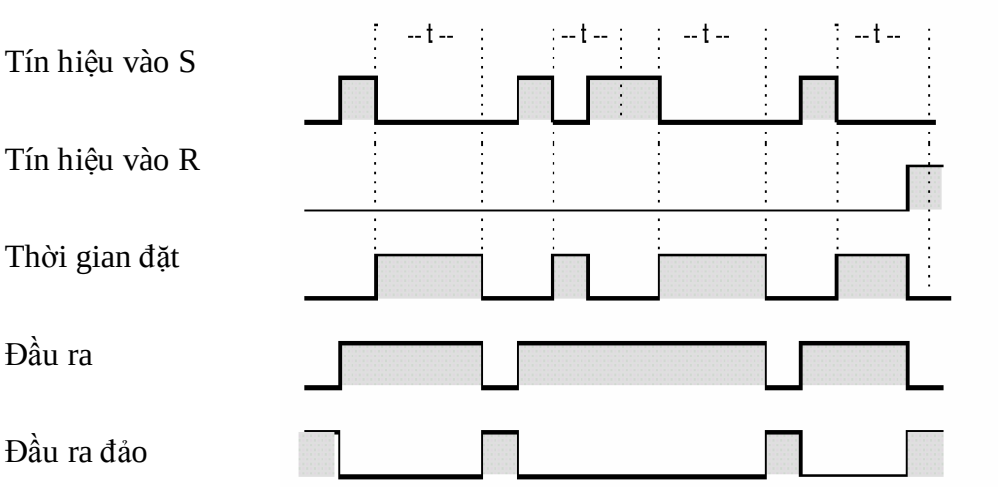
5. *Bộ thời gian SA:*

FBD	LAD	STL
		A I 0.0 L S5T#2S SF T 5 A I 0.1 R T 5 NOP 0 NOP 0 A T 5 = Q 4.0

Hình 4-46: Sơ đồ khối.

-Nguyên lý làm việc:

Tại thời điểm sườn lên của tín hiệu vào SET bộ thời gian đươc thiết lập. Tín hiệu đầu ra có giá trị là 1. Nhưng thời gian sẽ đươc tính ở thời điểm sườn xuống cuối cùng của tín hiệu đầu vao SET(S). Kết thúc thời gian đặt tín hiệu đầu ra sẽ trở về 0.



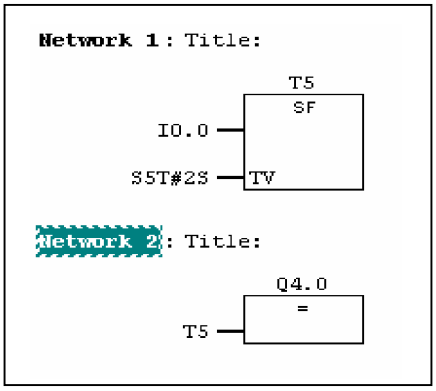
Hình 4-47: Giản đồ thời gian.

Khi có tín hiệu RESET (R) thời gian tính lập tức trở về 0 và tín hiệu đầu ra cũng giá trị là "0".

-Trường hợp không sử dụng các tín hiệu đầu vào SET(S), RESET (R), BI và BCD ta sử dụng khối Timer SF sau:

Tín hiệu I0.0 là tín hiệu kích
hời gian đặt S5T#2s là 2

àm thời gian sẽ tác động tới đầu ra Q4.0



Hình 4-48: Sử dụng hàm SF.

4.5. Bộ đếm COUNTER:

4.5.1.Nguyên lý làm việc:

Counter thực hiện chức năng đếm tại các sườn lên của các xung đầu vào. S7-300 có tối đa là 256 bộ đếm phụ thuộc vào từng loại CPU, ký hiệu bởi Cx. Trong đó x là số nguyên trong khoảng từ 0 đến 255. Trong S7-300 có 3 loại bộ đếm thường sử dụng nhất đó là : Bộ đếm tiến lùi (CUD), bộ đếm tiến (CU)và bộ đếm lùi (CD).

Một bộ đếm tổng quát có thể được mô tả như sau:

trong đó:

CU : BOOL là tín hiệu đếm tiến

CD : BOOL là tín hiệu đếm lùi

S : BOOL là tín hiệu đặt

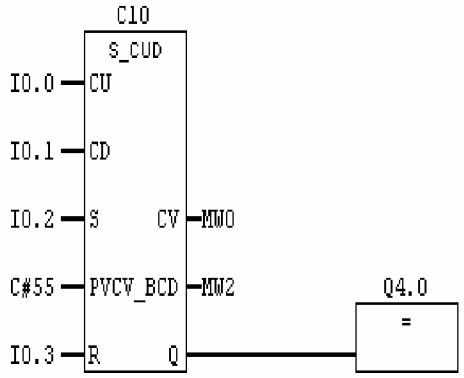
PV : WORD là giá trị đặt trước

R : BOOL là tín hiệu xoá

CV : WORD Là giá trị đếm ở hệ đếm 16

CV_BCD: WORD là giá trị đếm ở hệ đếm BCD

Q : BOOL Là tín hiệu ra .



Hình 4-49: sơ đồ khối bộ đếm Counter

Quá trình làm việc của bộ đếm được mô tả như sau:

Số sườn xung đếm được, được ghi vào thanh ghi 2 Byte của bộ đếm, gọi là thanh ghi C-Word. Nội dung của thanh ghi C-Word được gọi là giá trị đếm tức thời của bộ đếm và ký hiệu bằng CV và CV_BCD. Bộ đếm báo trạng thái của C-Word ra ngoài C-bit qua chân Q của nó. Nếu CV <> 0 , C-bit có giá trị "1". Ngược lại khi CV = 0, C- bit nhận giá trị 0. CV luôn là giá trị không âm. Bộ đếm sẽ không đếm lùi khi CV = 0.

Đối với Counter, giá trị đặt trước PV chỉ được chuyển vào C-Word tại thời điểm xuất hiện sườn lên của tín hiệu đặt tới chân S.

Bộ đếm sẽ được xoá tức thời bằng tín hiệu xoá R (Reset). Khi bộ đếm được xóa cả C-Word và C- bit đều nhận giá trị 0.

4.5.2.Khai báo sử dụng:

Việc khai báo sử dụng một Counter bao gồm các bước sau:

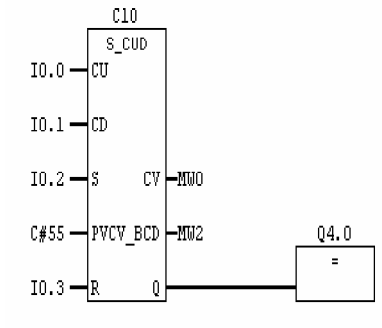
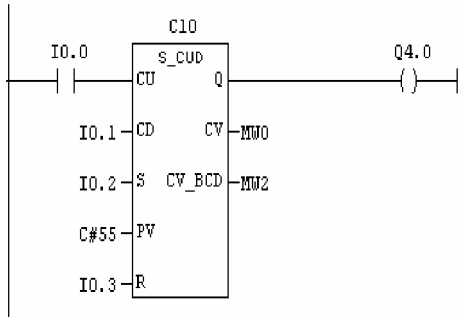
- Khai báo tín hiệu Enable nếu muốn sử dụng tín hiệu chủ động kích đếm (S): dạng dữ liệu BOOL
- Khai báo tín hiệu đầu vào đếm tiến CU : dạng dữ liệu BOOL
- Khai báo tín hiệu đầu vào đếm lùi CD : dạng dữ liệu BOOL
- Khai báo giá trị đặt trước PV: dạng dữ liệu WORD
- Khai báo tín hiệu xoá: dạng dữ liệu BOOL

- Khai báo tín hiệu ra CV nếu muốn lấy giá trị đếm tức thời ở hệ 16. dạng dữ liệu WORD
- Khai báo tín hiệu ra CV-BCD nếu muốn lấy giá trị đếm tức thời ở hệ BCD dạng dữ liệu WORD
- Khai báo đầu ra Q nếu muốn lấy tín hiệu tác động của bộ đếm. dạng dữ liệu BOOL

Trong đó cần chú ý các tín hiệu sau bắt buộc phải khai báo: Tên của bộ đếm cần sử dụng, tín hiệu kích đếm CU hoặc CD.

1. *Bộ đếm tiến lùi:*

-Sơ đồ khối :

FBD	LAD	STL
		A I 0.0 CU C 10 A I 0.1 CD C 10 A I 0.2 L C#55 S C 10 A I 0.3 R C 10 L C 10 T MW 0 LC C 10 T MW 2 A C 10 = Q 4.0

Hình 4-50: Sơ đồ khối bộ đếm tiến lùi.

-Nguyên lý hoạt động:

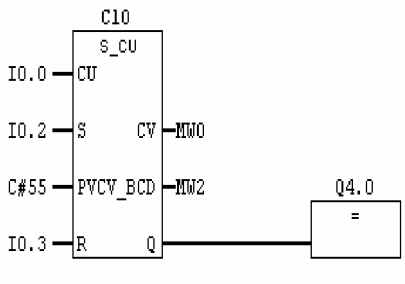
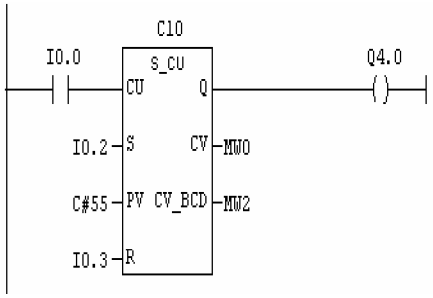
Khi tín hiệu I0.2 chuyển từ 0 lên 1bộ đếm được đặt giá trị là 55. Giá trị đầu ra Q4.0 =1 .

Bộ đếm sẽ thực hiên đếm tiến tại các sườn lên của tín hiệu tại chân CU khi tín hiệu I0.0 chuyển giá trị từ "0" lên "1"

Bộ đếm sẽ đếm lùi tại các sườn lên của tín hiệu tại chân I0.1 khi tín hiệu chuyển từ "0" lên "1"

Giá trị của bộ đếm sẽ trở về 0 khi có tín hiệu tai sườn lên của chân R (I0.3)

2. <u>Bộ đếm tiến : CU</u>

FBD	LAD	STL
		A I 0.0 CU C 10 BLD 101 A I 0.2 L C#55 S C 10 A I 0.3 R C 10 L C 10 T MW 0 LC C 10 T MW 2 A C 10 = Q 4.0

Hình 4-51: sơ đồ khối bộ đếm tiến.

-Nguyên lý hoạt động:

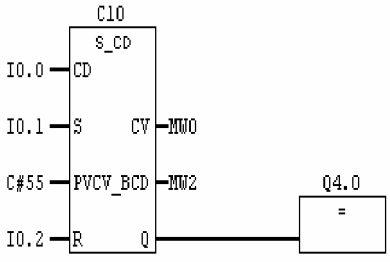
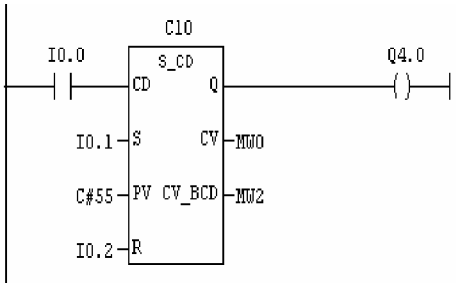
Khi tín hiệu I0.2 chuyển từ "0" lên "1" bộ đếm được đặt giá trị là 55. Giá trị đầu ra Q4.0 =1 .

Bộ đếm sẽ thực hiên đếm tiến tại các sườn lên của tín hiệu tại chân CU khi tín hiệu I0.0 chuyển giá trị từ "0" lên "1"

Giá trị của bộ đếm sẽ trở về 0 khi có tín hiệu tai sườn lên của chân R (I0.3)

Bộ đếm sẽ chỉ đếm đến giá trị <= 999.

3. <u>Bộ đếm lùi: CD</u>

FBD	LAD	STL
		A I 0.0 CD C 10 BLD 101 A I 0.1 L C#55 S C 10 A I 0.2 R C 10 L C 10 T MW 0 LC C 10 T MW 2 A C 10 = Q 4.0

Hình 4-52: Sơ đồ khối bộ đếm lùi.

-Nguyên lý hoạt động:

Khi tín hiệu I0.2 chuyển từ "0" lên "1" bộ đếm được đặt giá trị là 55. Giá trị đầu ra Q4.0 =1 .

Bộ đếm sẽ thực hiên đếm lùi tại các sườn lên của tín hiệu tại chân CD khi tín hiệu I0.0 chuyển giá trị từ "0" lên"1"

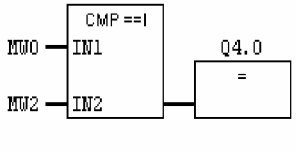
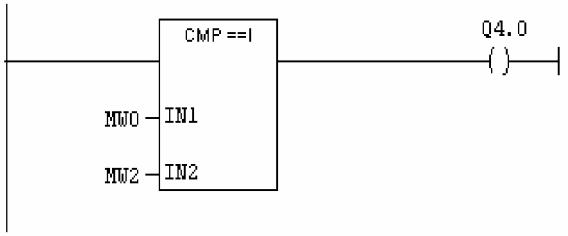
Giá trị của bộ đếm sẽ trở về 0 khi có tín hiệu tai sườn lên của chân R (I0.3).

Bộ đếm sẽ chỉ đếm đến giá trị >= 0.

4.6. Nhóm hàm so sánh và chuyển dữ liệu:

4.6.1. Nhóm hàm so sánh:

4.6.1.1.Nhóm hàm so sánh số nguyên 16 bit:

FBD	LAD	STL
		L MW 0 L MW 2 ==I = Q 4.0

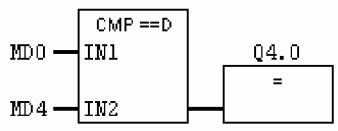
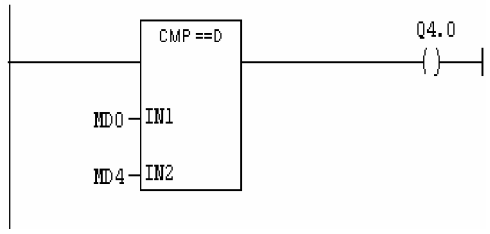
Hình 4-9: Khối thực hiện chức năng so sánh bằng nhau

Có các dạng so sánh hai số nguyên 16 bits như sau :

- Hàm so sánh bằng nhau giữa hai số nguyên 16 bits: ==
- Hàm so sánh khác nhau giữa hai số nguyên 16 bits: <>
- Hàm so sánh lớn hơn giữa hai số nguyên 16 bits: >
- Hàm so sánh nhỏ hơn giữa hai số nguyên 16 bits: <
- Hàm so sánh lớn hơn hoặc bằng nhau giữa hai số nguyên 16 bits: >=
- Hàm so sánh nhỏ hơn hoặc bằng nhau giữa hai số nguyên 16 bits: <=

Trong ví dụ trên đầu ra Q4.0 sẽ là "1" khi MW0 = MW1.

4.6.1.2. Nhóm hàm so sánh hai số nguyên 32 bits:

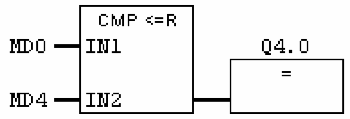
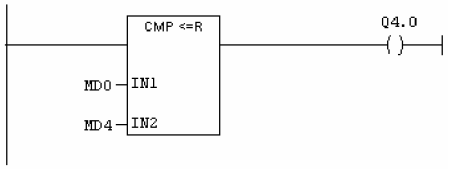
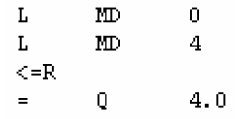
FBD	LAD	STL
		L MD 0 L MD 4 ==D = Q 4.0

Hình 4-10: Khối thực hiện chức năng so sánh

Trong ví dụ trên đầu ra Q4.0 sẽ là "1" khi MD0 = MD4.

- Hàm so sánh bằng nhau giữa hai số nguyên 32 bits: ==
- Hàm so sánh khác nhau giữa hai số nguyên 32 bits: <>
- Hàm so sánh lớn hơn giữa hai số nguyên 32 bits: >
- Hàm so sánh nhỏ hơn giữa hai số nguyên 32 bits: <
- Hàm so sánh lớn hơn hoặc bằng nhau giữa hai số nguyên 32 bits: >=
- Hàm so sánh nhỏ hơn hoặc bằng nhau giữa hai số nguyên 32 bits: <=

4.6.1.3.Nhóm hàm so sánh các số thực 32 bits

FBD	LAD	STL
		

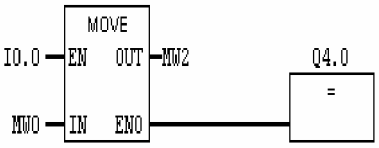
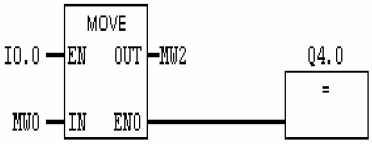
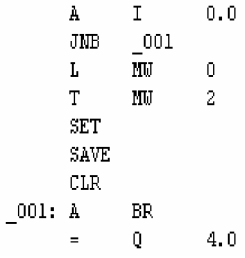
Hình 4-11: Khối thực hiện chức năng so sánh hai số thực.

Trong ví dụ trên đầu ra Q4.0 sẽ là "1" khi MD0 < MD1 .

Các dạng so sánh hai số thực 32 bits như sau :

- Hàm so sánh bằng nhau giữa hai số thực 32 bits: ==
- Hàm so sánh khác nhau giữa hai số thực 32 bits: <>
- Hàm so sánh lớn hơn giữa hai số thực 32 bits: >
- Hàm so sánh nhỏ hơn giữa hai số thực 32 bits: <
- Hàm so sánh lớn hơn hoặc bằng nhau giữa hai số thực 32 bits: >=
- Hàm so sánh nhỏ hơn hoặc bằng nhau giữa hai số thực 32bits: <=

4.6.2 . Khối chuyển dữ liệu:

FBD	LAD	STL
		

Hình 4-53: Sơ đồ khối MOV

-Nguyên lý hoạt động:

Khi có tín hệu kích I0.0 khối Copy được thiết lập , tín hiệu đầu ra ENO là Q4.0 =1. Đồng thời số liệu ở đầu vào IN là MW0 được Copy sang đầu ra OUT là MW2.

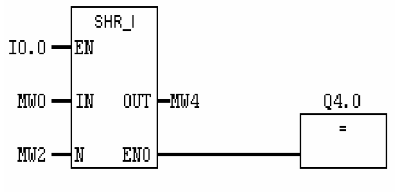
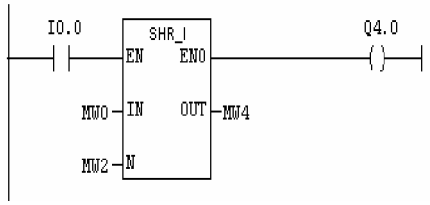
Khi tín hiệu kích I0.0 = 0 tín hiệu đầu ra Q4.0 = 0.

Trong trường hợp muốn thay đổi số liệu trong bộ nhớ (tức là thay đổi giá trị trong MW2) ta có thể không cần sử dụng tín hiệu kích I0.0.

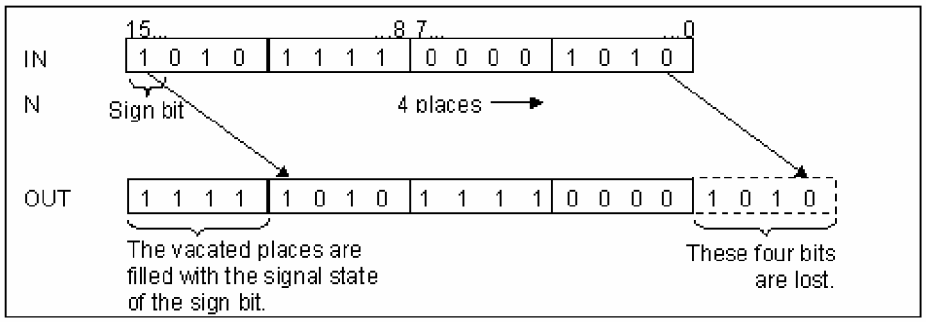
4.7.Các bộ ghi dịch và quay số liệu trên thanh ghi:

1. ***Dịch phải số nguyên 16 bits:***

FBD	LAD	STL

		`A I 0.0`
		`JNB _001`
		`L MW 2`
		`L MW 0`
		`SSI`
		`T MW 4`
		`SET`
		`SAVE`
		`CLR`
		`_001: A BR`
		`= Q 4.0`

Hình 4-54: Sơ đồ khối dịch phải.



Hình 4-55: Nguyên lý hoạt động.

Khi tín hiệu kích I0.0 = 1 Khối sẽ thực hiện chức năng dich chuyển sang phải số liệu trong thanh ghi. Đồng thời tín hiệu ra tại ENO là Q4.0 có giá trị là 1.

Số liệu đưa vào tại IN là MW0

Số bit sẽ dich chuyển là MW2 (tại chân N).

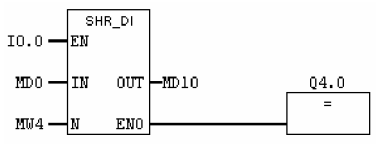
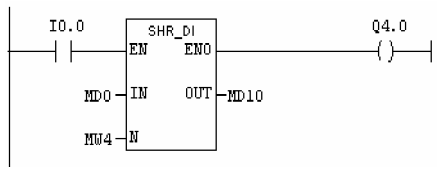
Kết quả sau khi dịch được cất vào MW4.

Trên sơ đồ cho ta thấy kết quả của bộ dịch phải 4 bit.

2. Dịch phải số nguyên 32 bits:

-Sơ đồ khối:

FBD	LAD	STL

		`A I 0.0`
		`JNB _001`
		`L MW 4`
		`L MD 0`
		`SSD`
		`T MD 10`
		`SET`
		`SAVE`
		`CLR`
		`_001: A BR`
		`= Q 4.0`

Hình 4-56: Khối dịch phải.

Khi tín hiệu kích I0.0 = 1. Khối sẽ thực hiện chức năng dich chuyển sang phải số liệu trong thanh ghi. Đồng thời tín hiệu ra tại ENO là Q4.0 có giá trị là 1.

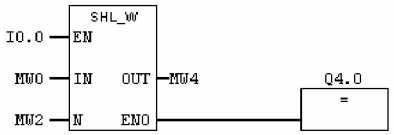
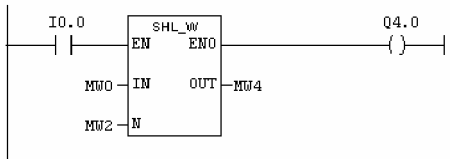
Số liệu đưa vào tại IN là MD0

Số bit sẽ dịch chuyển là MW2 (tại chân N). Kết quả sau khi dịch được cất vào MW4.

Trên sơ đồ cho ta thấy kết quả của bộ dịch phải 4 bit.

3. ***Dich trái 16 bit:***

-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 2 L MW 0 SLW T MW 4 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-57: Khối dịch trái.

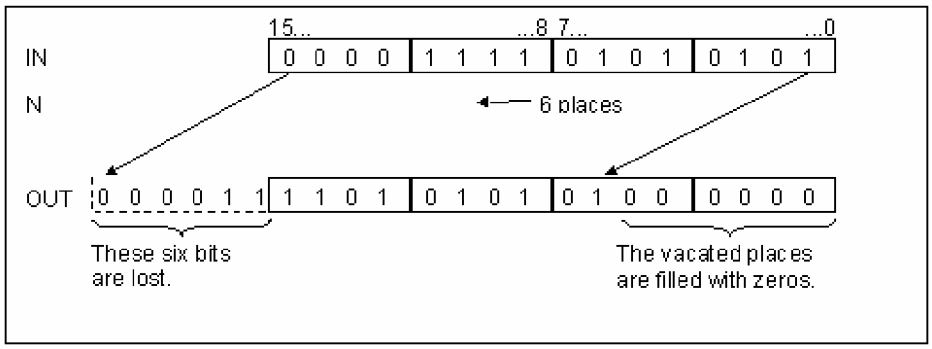
-Nguyên lý hoạt động:

Khi có tín hiệu kích I0.0 = 1 tín hiệu ra Q4.0 được thiết lập và có giá trị 1.

Dữ liệu ở đầu vào MW0 được dịch sang trái với số bit được đặt tại chân N (MW2).

Kết quả sau khi dịch được ghi vào MW4.

-Giản đồ thời gian:

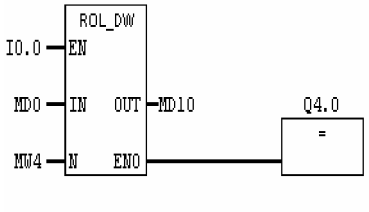
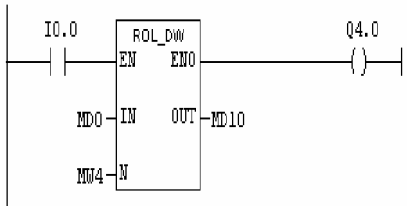
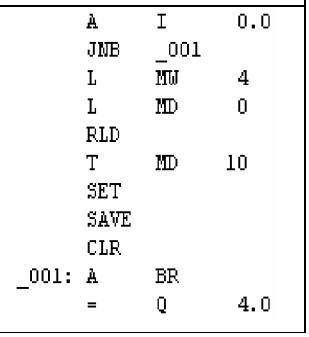


Hình 4-58: Giản đồ thời gian bộ dịch trái 6 vị trí.

Chú ý: Trong trường hợp cần dich trái một số 32 bits ta chỉ cần khai báo dữ liệu ở đầu vào IN dưới dạng MD ví dụ: MD0 và kết quả đầu ra cũng sẽ được lưu giữ ở MD Ví dụ: MD4

4. Quay trái số 32 bits:

-Sơ đồ khối:

FBD	LAD	STL
		

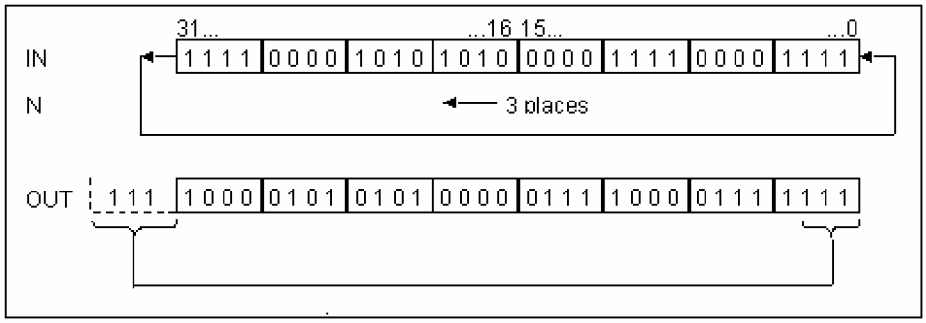
Hình 4-59: Sơ đồ khối quay trái.

-Nguyên lý hoạt động:

Khi có tín hiệu kích I0.0 = 1 tín hiệu ra Q4.0 được thiết lập và có giá trị 1.

Dữ liệu ở đầu vào MD0 được quay sang tráI với số bit được đặt tại chân N (MW4).

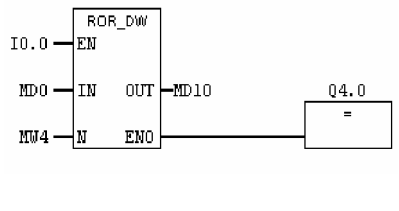
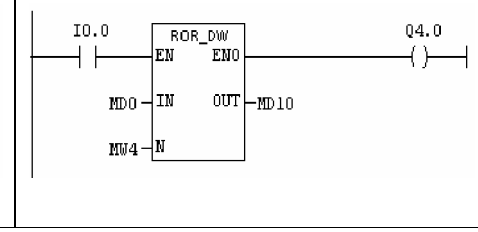
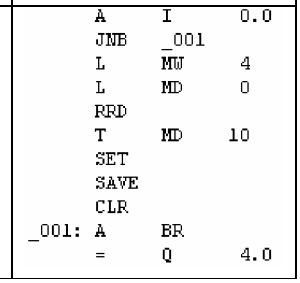
Kết quả sau khi dịch được ghi vào MD10.



Hình 4-60: Giản đồ thời gian.

5. Quay phải số 32 bits:

-Sơ đồ khối:

FBD	LAD	STL
		

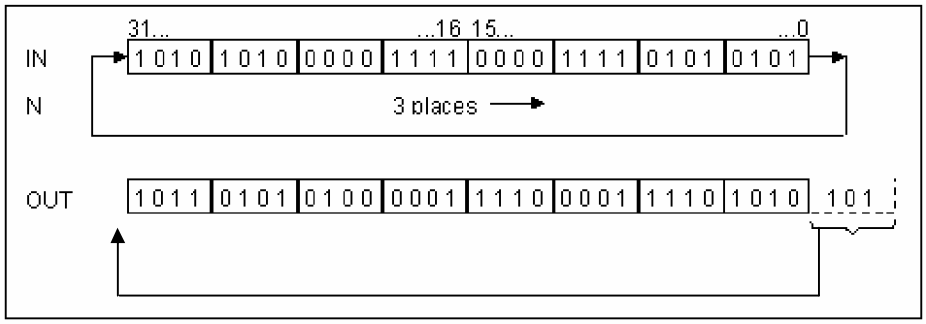
Hình 4-61: sơ đồ khối bộ quay phải.

-Nguyên lý hoạt động:

Khi có tín hiệu kích I0.0 = 1 tín hiệu ra Q4.0 được thiết lập và có giá trị 1.

Dữ liệu ở đầu vào MD0 được quay phải với số bit được đặt tại chân N (MW4).

Kết quả sau khi dịch được ghi vào MD10.



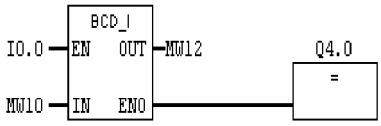
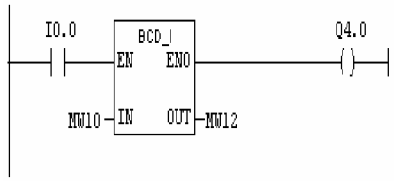
Hình 4-62: Giản đồ thời gian của bộ dịch phải 3 vị trí số 32 bits.

4.8. Nhóm hàm đổi kiểu dữ liệu :

Trong ngôn ngữ lập trình của S7-300 có một số kiểu dữ liệu khác nhau như:
- Số nguyên 16 bits (Interger)
- Số nguyên 32 bits (DI)
- Số nguyên dạng BCD.
- Số thực REAL
- và một số dạng dữ liệu khác .

Khi làm việc với nhiều dạng dữ liệu khác nhau cho ta vấn đề cần phải chuyển đổi chúng. Ví dụ khi đọc tín hiệu từ cổng vào tương tự ta nhận được số liệu dạng nguyên 16 bits mang giá trị tín hiệu tương tự chứ không phải bản thân giá trị đó, bởi vậy để xử lý tiếp thì cần thiết phải chuyển đổi số nguyên đó thành đúng giá trị thực, dấu phảy động của tín hiệu tương tự ở cổng. Ta có một số hàm chuyển đổi các dạng dữ liệu như sau:

4.8.1.Hàm chuyển số BCD thành số số nguyên 16 bits:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 10 BTI T MW 12 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-26: Chuyển đổi số BCD sang dạng số nguyên 16 bits.

Dữ liệu vào và ra:

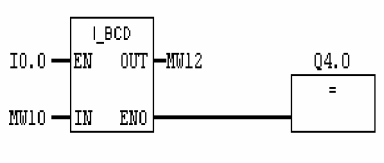
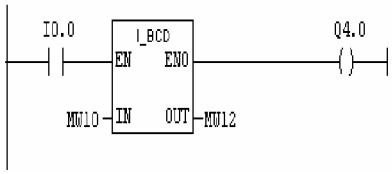
EN: BOOL IN: WORD

OUT: INT ENO: BOOL

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm thực hiên chức năng chuyển số BCD (MW10) sang số nguyên rồi cất vào MW12.

Khi tin hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm không thực hiên chức năng chuyển đổi.

4.8.2. Hàm chuyển đổi số nguyên 16 bits sang dạng BCD.

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 10 ITB T MW 12 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-27: Chuyển đổi số nguyên sang số BCD.

Dữ liệu vào và ra:

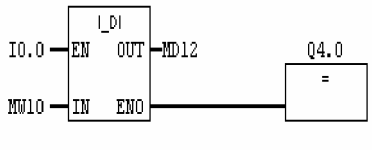
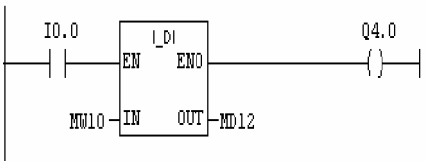
EN: BOOL IN: INT

OUT: BCD ENO: BOOL

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm thực hiên chức năng chuyển số nguyên 16 bits (MW10) sang số BCD rồi cất vào MW12.

Khi tin hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm không thực hiên chức năng chuyển đổi.

4.8.3. Hàm chuyển đổi số nguyên 16 bits sang số nguyên 32 bits:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 10 ITD T MD 12 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-28: Chuyển đổi số nguyên 16 bits sang số nhuyên 32 bits.

Dữ liệu vào và ra:

EN: BOOL IN: INT

OUT: DINT ENO: BOOL

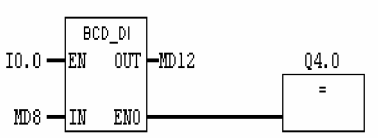
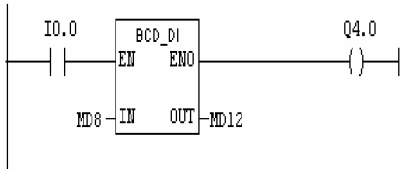
Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm thực hiên chức năng chuyển số nguyên 16 bits (MW10) sang số nguyên 32 bits rồi cất vào MW12.

Khi tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm không thực hiên chức năng chuyển đổi.

4.8.4.Chuyển đổi số BCD sang số nguyên 32 bits:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm thực hiên chức năng chuyển số BCD (MW10) sang số nguyên 32 bits rồi cất vào MW12.

Khi tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm không thực hiện chức năng chuyển đổi.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 8 BTD T MD 12 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-29: Chuyển số BCD sang số nguyên 32 bits

-Kiểu dữ liệu vào/ra:

EN: BOOL IN: DWORD

OUT: DINT ENO: BOOL.

4.8.5.Hàm đảo giá trị các bits .

1/Với số nguyên có độ dài 16 bits:

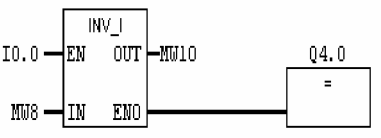
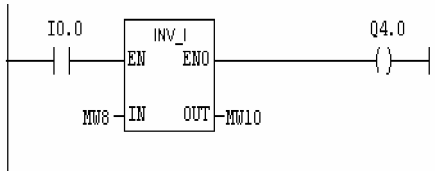
-Nguyên lý hoạt động: Hàm sẽ thực hiện chức năng chuyển đổi giá trị các bits trong MW8 rồi cất vào MW10 khi tín hiệu I0.0 =1. Đồng thời giá trị Q4.0 = 1.

Khi I0.0 = 0, giá trị Q4.0 = 0

-Kiểu dữ liệu vào/ra:

EN: BOOL IN: INT

OUT: INT ENO: BOOL

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 8 INVI T MW 10 SET SAVE CLR _001: A BR = Q 4.0

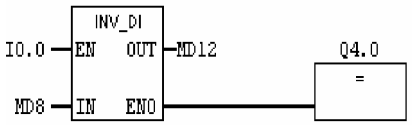
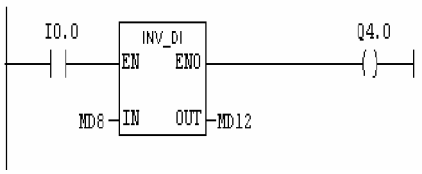
Hình 4-30: Hàm thực hiện chức năng đảo giá trị các bits

-Ví dụ:

Trước khi thực hiện

Sau khi thực hiện

2/ Với số nguyên có độ dài 32 bits.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 8 INVD T MD 12 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-31: Hàm thực hiện chức năng đảo giá trị các bits.

-Nguyên lý hoạt động: Hàm sẽ thực hiện chức năng chuyển đổi giá trị các bits trong MD8 rồi cất vào MD12 khi tín hiệu I0.0 =1. Đồng thời giá trị Q4.0 = 1 .

Khi I0.0 = 0, giá trị Q4.0 = 0

-Kiểu dữ liệu vào/ra:

EN: BOOL IN: DINT

OUT: DINT ENO: BOOL

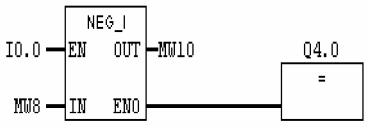
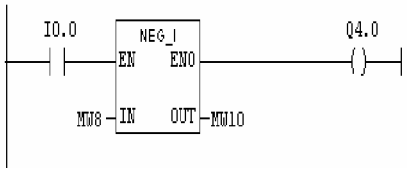
- Ví dụ:

Trước khi thực hiện: MD8 = F0FF FFF0

Sau khi thực hiện : MD12 = 0F00 000F

4.8.6.Các hàm đổi dấu :

Hàm sẽ thực hiện chức năng đổi dấu dữ liệu vào . Các hàm đổi dấu như đổi dấu số thực độ dài 16bits (I), 32 bits (DI) hay số nguyên (R).

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 8 NEGI T MW 10 AN OV SAVE CLR _001: A BR = Q 4.0

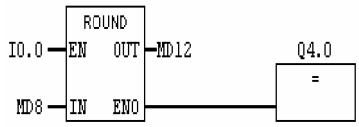
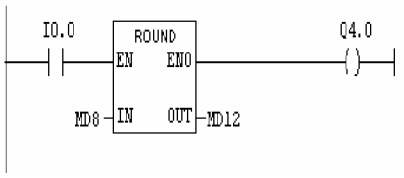
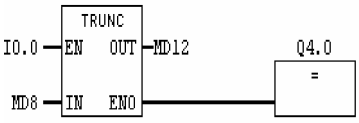
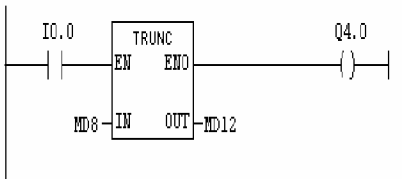
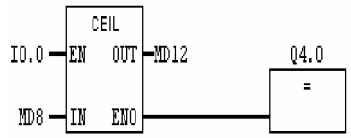
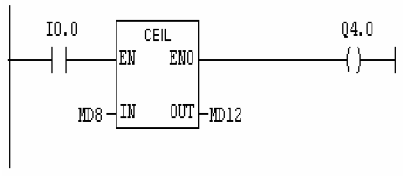
Hình 4-32:

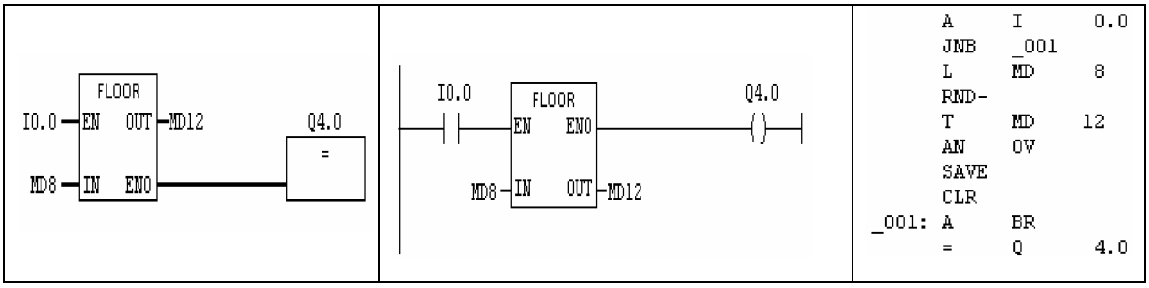
Dạng dữ liệu vào:

	NEG_I	NEG_DI	NEG_R
EN	BOOL	BOOL	BOOL
IN	INT	DI	REAL
OUT	INT	DI	REAL
ENO	BOOL	BOOL	BOOL

Ví dụ: Trước khi thực hiện MW8 = +10, sau khi thực hiện MW10 = -10.

4.8.7.Các hàm thực hiện chức năng làm tròn

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 8 RND T MD 12 AN OV SAVE CLR _001: A BR = Q 4.0
		A I 0.0 JNB _001 L MD 8 TRUNC T MD 12 AN OV SAVE CLR _001: A BR = Q 4.0
		A I 0.0 JNB _001 L MD 8 RND+ T MD 12 AN OV SAVE CLR _001: A BR = Q 4.0



Hình 4-33:

-Hàm ROUND : (chuyển số thực thành số nguyên gần nhất) thực hiện làm tròn như sau: nếu phần lẻ < 0,5 thì làm tròn xuống. Nếu phần lẻ > 0,5 thì làm tròn lên.

Ví dụ: 1,2 -> 1 ; 1,6 -> 2.

-1,2 -> -1 ; -1,6 -> -2.

-Hàm TRUNC: (lấy phần nguyên cắt bỏ phần lẻ) thực hiện làm tròn xuống giá trị tròn nhỏ ví dụ: dữ liệu vào từ 1,1 đến 1,9 -> 1.

-Hàm CEIL: thực hiện làm tròn lên.

ví dụ: dữ liệu vào từ 1,1 đến 1,9 -> 2.

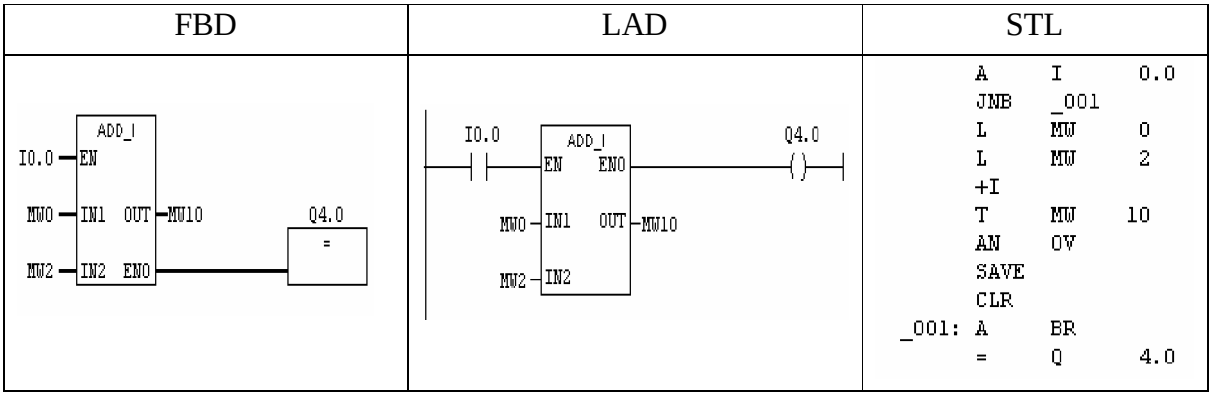
-Hàm FLOOR: thực hiện làm tròn xuống.

ví dụ: +1,7 -> 1 ; - 1,7 -> -2

4.9.Các hàm toán học:

4.9.1. Nhóm hàm làm việc với số nguyên 16 bits:

1/ Cộng hai số nguyên 16 bits:



Hình 3-12: Khối thực hiện chức năng cộng hai số nguyên 16 bits.

Dữ liệu vào và ra:

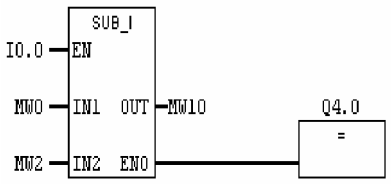
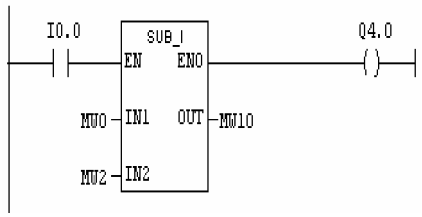
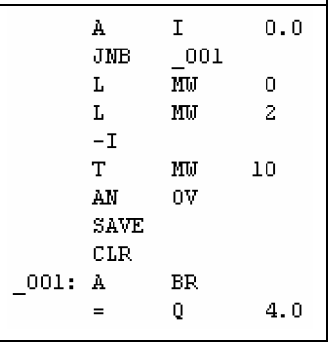
EN: BOOL IN1: INT

IN2: INT OUT: INT ENO: BOOL

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện cộng hai số nguyên 16 bits MW0 với MW2. Kết quả được cất vào MW10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

2/ Trừ hai số nguyên 16 bits:

FBD	LAD	STL
		

Hình 4-13: Khối thực hiện chức năng trừ hai số nguyên 16 bits

Dữ liệu vào và ra:

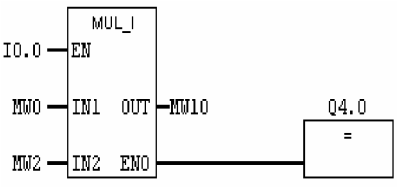
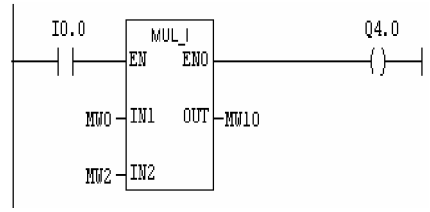
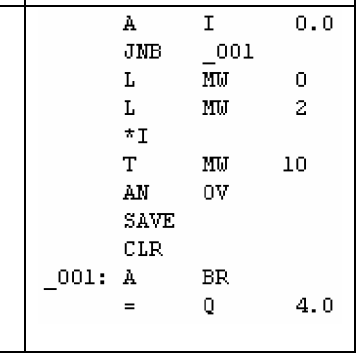
EN : BOOL IN1: INT

IN2: INT OUT: INT ENO: BOOL

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện trừ hai số nguyên 16 bits MW0 với MW2. Kết quả được cất vào MW10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

3/ Nhân hai số nguyên 16 bits:

FBD	LAD	STL
		

Hình 4-14: Khối thực hiện chức năng nhân hai số 16 bits.

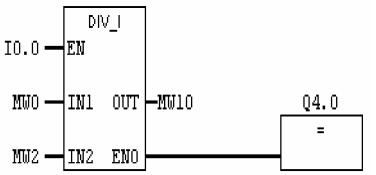
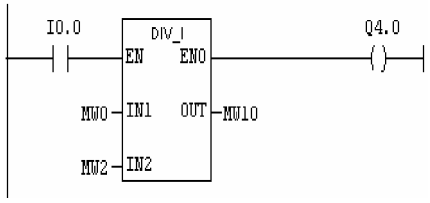
Dữ liệu vào và ra:

EN: BOOL IN1: INT
IN2: INT OUT: IN ENO: BOOL

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện nhân hai số nguyên 16 bits MW0 với MW2. Kết quả được cất vào MW10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

4/ Chia hai số nguyên 16 bits:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 0 L MW 2 /I T MW 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-15: Khối thực hiện chức năng chia hai số nguyên 16 bits

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện chia hai số nguyên 16 bits MW0 với MW2. Kết quả được cất vào MW10.

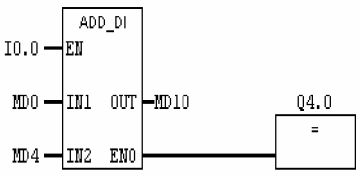
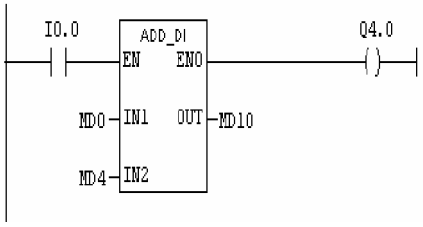
Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

4.9.2.Nhóm hàm làm việc với số nguyên 32 bits:

1/ Cộng hai số nguyên 32 bits:

Dữ liệu vào và ra:

EN: BOOL IN1: DINT
IN2: DINT OUT: DINT ENO: BOOL

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 +D T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-16: Khối thực hiện chức năng cộng hai số nguyên 32 bits

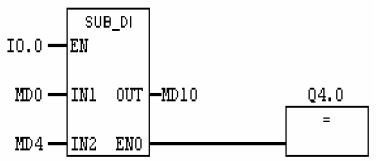
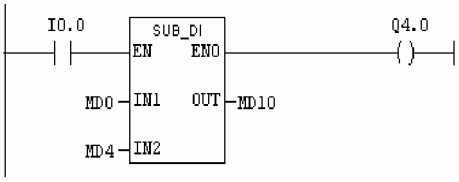
Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện công hai số nguyên 32 bits MD0 với MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

2/ Trừ hai số nguyên 32 bits:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện trừ hai số nguyên 32 bits MD0 với MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 -D T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-17: Khối thực hiện chức năng trừ hai số nguyên 32 bits

Dữ liệu vào và ra:

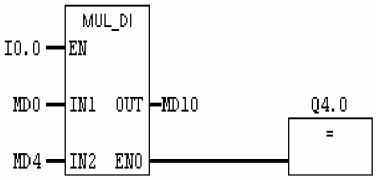
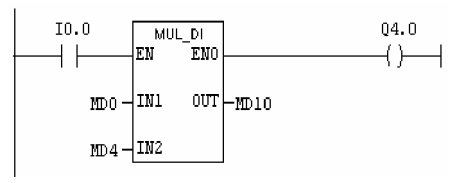
EN: BOOL IN1: DINT

IN2: DIN OUT: DINT ENO: BOOL

3/ Nhân hai số nguyên 32 bits:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện nhân hai số nguyên 32 bits MD0 với MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 *D T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-18: Khối thực hiện chức năng nhân hai số nguyên 32 bit

Dữ liệu vào và ra:

EN: BOOL IN1: DINT

IN2: DINT OUT: DINT ENO: BOOL

4/ Chia hai số nguyên 32 bits :

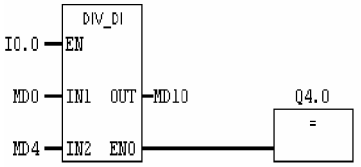
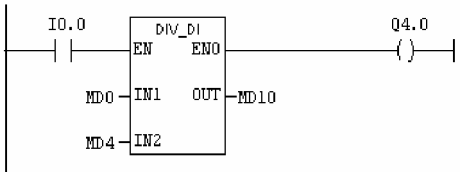
Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện chia hai số nguyên 32 bits MD0 với MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

Dữ liệu vào và ra:

EN: BOOL IN1: DINT

IN2: DINT OUT: DINT ENO: BOOL

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 /D T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

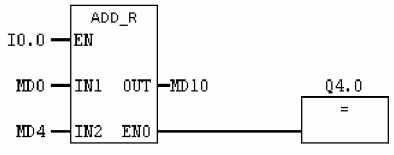
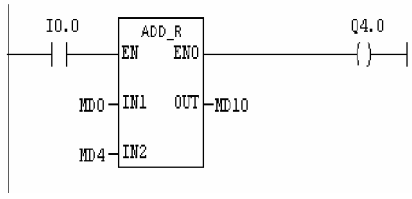
Hình 4-19: Khối thực hiện chức năng chia hai số nguyên 32 bits

4.9.3.Nhóm hàm làm việc với số thực:

1/ Cộng hai số thực:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện cộng hai số thực MD0 + MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 +R T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-20: Khối thực hiện chức năng cộng hai số thực

Dữ liệu vào và ra:

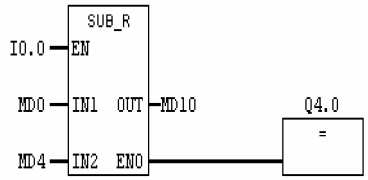
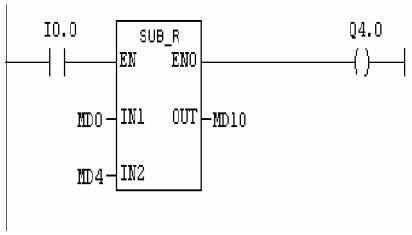
EN: BOOL IN1: REAL

IN2: REAL OUT: REAL ENO: BOOL

2/ Hàm trừ hai số thực:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện trừ hai số thực MD0 - MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 -R T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-21: Khối thực hiện chức năng trừ hai số thực.

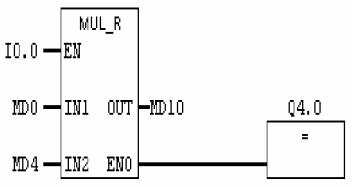
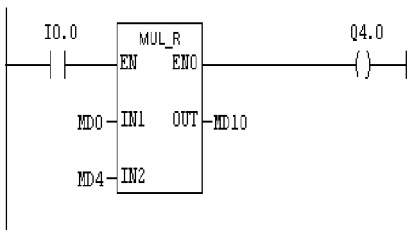
Dữ liệu vào và ra:

EN: BOOL IN1: REAL

IN2: REAL OUT: REAL ENO: BOOL

3/ Nhân hai số thực:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện nhân hai số thực MD0 . MD4. Kết quả được cất vào MD10.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 *R T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-22: Khối thực hiện chức năng nhân hai số thực.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

Dữ liệu vào và ra:

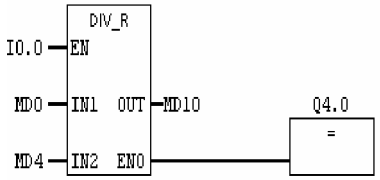
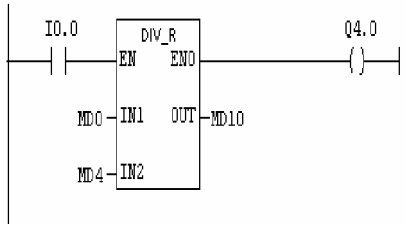
EN: BOOL IN1: REAL

IN2: REAL OUT: REAL ENO: BOOL

4/ Chia hai số thực:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiện chia hai số thực MD0 : MD4. Kết quả được cất vào MD10.

Trong trường hợp tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L MD 4 /R T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-23: Khối thực hiện chức năng nhân hai số thực

Dữ liệu vào và ra:

EN: BOOL IN1: REAL

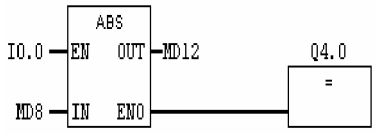
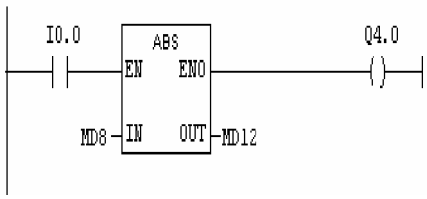
IN2: REAL OUT: REAL

ENO: BOOL

5/ Hàm lấy giá trị tuyệt đối : ABS

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiên chức năng lấy gía trị tuyệt đối của MD8 rồi cất vào MD12

Khi tín hiệu vao I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 8 ABS T MD 12 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-24: Khối thực hiện chức năng lấy giá trị thuyệt đối.

Dữ liệu vào và ra:

EN: BOOL IN: REAL

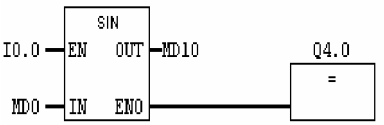
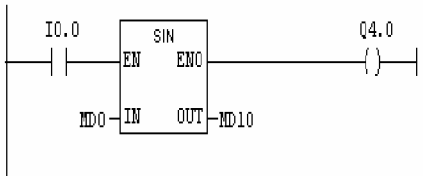
OUT: REAL ENO: BOOL

Ví dụ: MD8= -6,234 x 10-3 thi sau khi thực hiên chức năng ABS giá trị MD12 = 6,234 x 10-3.

6/ Hàm SIN, COS, TAN, ASIN, ACOS, ATAN:

Khi tín hiệu vào I0.0 = 1 đầu ra Q4.0 = 1 và hàm sẽ thực hiên chức năng tinh SIN, COS, TAN, ASIN, ACOS, ATAN của MD0 rồi cất vào MD10.

Khi tín hiệu vào I0.0 = 0 đầu ra Q4.0 = 0 và hàm sẽ không thực hiện chức năng.

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 SIN T MD 10 AN OV SAVE CLR _001: A BR = Q 4.0

Hình 4-25: Khối thực hiện chức năng tính hàm Sin.

Dữ liệu vào và ra:

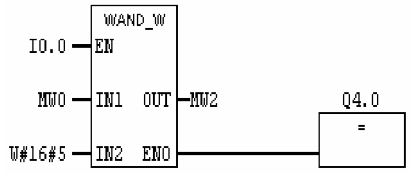
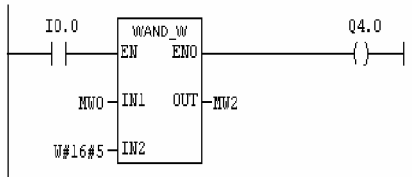
EN: BOOL IN: REAL

OUT: REAL ENO: BOOL

4.10.Các hàm Logic thực hiện trên thanh ghi :

1. *Hàm AND hai số có độ dài là 16 bits.*

-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 0 L W#16#5 AW T MW 2 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-63: sử dụng khối AND 16 bits

-Nguyênlý hoạt động:

Hàm sẽ thực hiện chức năng nhân hai số nhị phân tai đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT (MW2) khi có tín hiệu kích tại chân EN (I0.0 =1).

Tín hiệu ở đầu ra ENO (Q4.0 = 1) khi hàm thực hiện chức năng.

-Ví dụ:

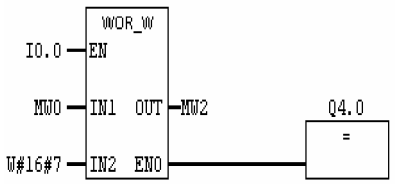
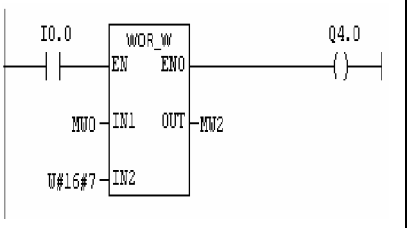
IN1 = 0101010101010101 Số thứ nhất

IN2 = 0100000000001111 Số thứ hai

OUT = 0100000000000101 Kết quả

2. *Hàm OR hai số có độ dài là 16 bits:*

-Sơ đồ khối :

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 0 L W#16#7 OW T MW 2 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-64: Sử dụng khối OR 16 bits.

-Nguyên lý hoạt động:

Hàm sẽ thực hiện chức năng OR hai số nhị phân tai đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT (MW2) khi có tín hiệu kích tại chân EN (I0.0 = 1).

Tín hiệu ở đầu ra ENO (Q4.0 = 1) khi hàm thực hiện chức năng.

-Ví dụ:

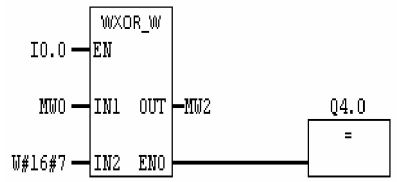
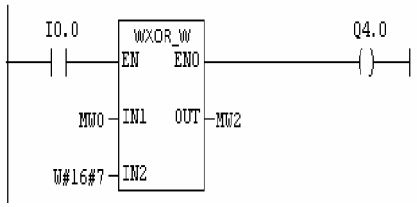
IN1 = 0101010101010101 Số thứ nhất

IN2 = 0000000000001111 Số thứ Hai

OUT = 0101010101011111 Kết quả

3.Hàm XOR hai số có độ dài 16 bits:

-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 JNB _001 L MW 0 L W#16#7 XOW T MW 2 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-65: sơ đồ khối XOR 16 bits.

-Nguyênlý hoạt động:

Hàm sẽ thực hiện chức năng XOR hai số nhị phân tai đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT khi có tín hiệu kích tại chân EN.

Tín hiệu ở đầu ra ENO khi hàm thực hiện chức năng.

-Ví dụ:

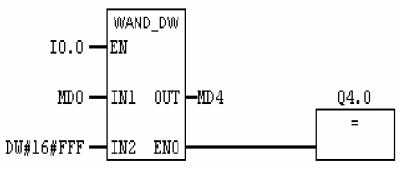
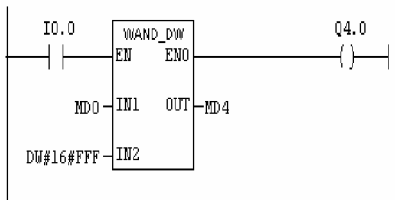
IN1 = 0101010101010101 Số thứ nhất

IN2 = 0000000000001111 Số thứ Hai

OUT = 0101010101011010 Kết quả

4.Hàm AND hai từ kép:

-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L DW#16#FFF AD T MD 4 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-66: Sử dụng khối AND hai từ kép .

EN(I0.0): BOOL - tín hiệu kích
IN1: DWORD - Vào 1
IN2: DWORD - vào2
OUT: DWORD - Ra
ENO: BOOL - Tín hiệu ra của khối.

-Nguyên lý hoạt động:

Hàm sẽ thực hiện chức năng AND hai số nhị phân tại đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT khi có tín hiệu kích tại chân EN.

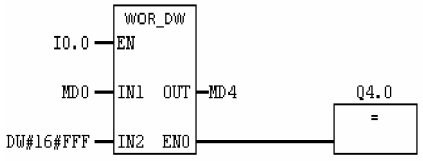
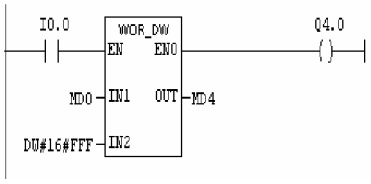
Tín hiệu ở đầu ra ENO khi hàm thực hiện chức năng.

-Ví dụ:

IN1 = 0101010101010101 0101010101010101
IN2 = 0000000000000000 0000111111111111
OUT = 0000000000000000 0000010101010101

5.Hàm OR hai từ kép:

-Sơ đồ khối:

FBD	LAD	STL
		A I 0.0 JNB _001 L MD 0 L DW#16#FFF OD T MD 4 SET SAVE CLR _001: A BR = Q 4.0

Hình 4-67: Sử dụng khối OR hai từ kép.

EN(I0.0): BOOL - tín hiệu kích
IN1: DWORD - Vào 1
IN2: DWORD - vào2
OUT: DWORD - Ra
ENO: BOOL - Tín hiệu ra của khối.

-Nguyên lý hoạt động:

Hàm sẽ thực hiện chức năng OR hai số có độ dài 2 từ tại đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT khi có tín hiệu kích tại chân EN.

Tín hiệu ở đầu ra ENO khi hàm thực hiện chức năng.

-Ví dụ:

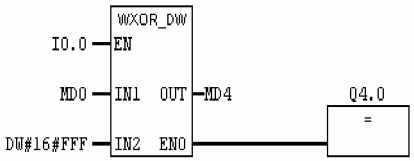
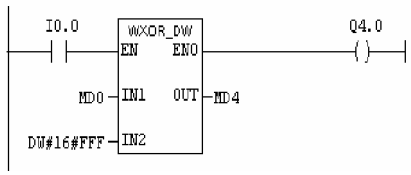
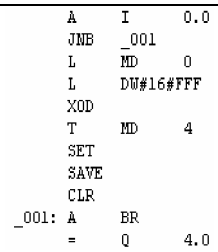
IN1 = 0101010101010101 0101010101010101

IN2 = 0000000000000000 0000111111111111

OUT = 0101010101010101 0101111111111111

6.Hàm XOR hai từ kép :

-sơ đồ nguyên lý:

FBD	LAD	STL
		

Hình 4-68: Sử dụng khối XOR hai từ kép.

EN(I0.0): BOOL - tín hiệu kích

IN1: DWORD - Vào 1

IN2: DWORD - vào2

OUT: DWORD - Ra

ENO: BOOL - Tín hiệu ra của

-Nguyên lý hoạt động:

Hàm sẽ thực hiện chức năng XOR hai số có độ dài 2 từ tại đầu vào IN1 và đầu vào IN2 kết quả được cất ở OUT khi có tín hiệu kích tại chân EN.

Tín hiệu ở đầu ra ENO khi hàm thực hiện chức năng.

-Ví dụ:

IN1 = 0101010101010101 0101010101010101

IN2 = 0000000000000000 0000111111111111

OUT = 0101010101010101 0101101010101010

Chương 5: Chương trình con và xử lý tín hiệu tương tự

KỸ THUẬT LẬP TRÌNH

Chương trình người dùng thường được chia nhỏ thành từng khối logic theo kiểu chương trình cấu trúc, giúp cho việc lập trình và sữa lỗi thuận tiện. Có nhiều loại khối logic:

- Khối tổ chức OB (Organization blocks)
- Khối hàm hệ thống SFB (System function blocks) và hàm hệ thống SFC (system functions) tích hợp trong PLC
- Khối hàm FB (Function blocks) trong thư viện hay người dùng tự viết
- Hàm FC (Functions) trong thư viện hay người dùng tự viết
- Khối dữ liệu Instance (Instance Data Blocks) liên kết với FB/SFB
- Khối dữ liệu chia xẻ (Shared Data Blocks)

Khối tổ chức OB là giao diện giữa chương trình người dùng và hệ điều hàmh của PLC. OB được gọi bởi hệ điều hành theo chu kỳ hay khi có ngắt, có sự cố hay khi khởi động PLC. Có nhiều khối OB và có ưu tiên khác nhau, khối OB có số ưu tiên cao hơn có thể ngắt khối OB số ưu tiên thấp hơn. Tuỳ theo loại CPU, số lượng khối OB sử dụng được sẽ khác nhau, bảng sau liệt kê các loại OB

Loại OB	**Ý nghĩa**	**Ưu tiên**
OB1	Được gọi khi kết thúc khởi động hay kết thúc OB1, thực hiện theo chu kỳ	1
OB10, OB11, OB12, OB13 OB14, OB15, OB16, OB17	Ngắt theo thời gian trong ngày, tháng, năm	2
OB20 OB21 OB22 OB23	Ngắt trì hoãn	3 4 5 6
OB30 OB31 OB32 OB33 OB34 OB35 OB36 OB37 OB38	Ngắt chu kỳ (mặc định 5s) Ngắt chu kỳ (mặc định 2s) Ngắt chu kỳ (mặc định 1s) Ngắt chu kỳ (mặc định 500ms) Ngắt chu kỳ (mặc định 200ms) Ngắt chu kỳ (mặc định 100ms) Ngắt chu kỳ (mặc định 50ms) Ngắt chu kỳ (mặc định 20ms) Ngắt chu kỳ (mặc định 10ms)	7 8 9 10 11 12 13 14 15

OB40 OB41 OB42 OB43 OB44 OB45 OB46 OB47	Ngắt cứng	16 17 18 19 20 21 22 23
OB60	Gọi bởi SFC35 "MP_ALM"	25
OB70 OB72 OB 73	Lỗi I/O redundancy (H CPU) Lỗi CPU redundancy (H CPU) Lỗi Communication redundancy (H CPU)	25 28 25
OB80 OB81 OB82 OB83 OB84 OB85 OB86 OB87	Sự cố chu kỳ quét Lỗi nguồn Ngắt chẩn đoán Ngắt do thêm bớt module Lỗi phần cứng CPU Lỗi chương trình Lỗi module mở rộng Lỗi truyền thông	26, 28
OB90	Warm or cold restart or delete a block being executed in OB90 or load an OB90 on the CPU or terminate OB90	29, 0
OB100 OB101 OB102	Khởi động ấm Khởi động nóng Khởi động lạnh	27
OB121 OB122	Sai lập trình Sai I/O	Ưu tiên của tác nhân gây ra sự cố

5.1/ Khối OB1

OB1 được gọi sau khi kết thúc quá trình khởi động và sau khi kết thúc chính nó, mọi OB trừ OB90 có thể ngắt OB1. Khi OB1 đã được thực hiện, hệ điều hành gởi đi dữ liệu toàn cục. Trước khi gọi lại OB1, hệ điều hành chuyển bộ nhớ đệm ra module xuất , cập nhật bộ đệm nhập và nhận dữ liệu toàn cục. Khi thực hiện OB1, chương trình trong khối được thực hiện, dữ liệu xuất ra module xuất được cấp tạm trong bộ nhớ. Chương trình trong OB1 có thể gọi các hàm hay khối hàm.

Thời gian thực hiện OB1 gọi là thời gian quét, hệ điều hành ấn định thời gian quét tối đa (150ms) và tối thiểu, có thể cài đặt bằng Step 7. Nếu chu kỳ quét kéo dài thì gọi OB80 hay chuyển sang STOP, nếu chu kỳ quét ngắn quá thì thêm trì hoãn hay gọi OB90.

OB1 gồm phần mã chương trình, do người dùng viết; bảng biến cục bộ (local block) còn gọi là bảng khai báo biến (variable declaration table) gồm 20 byte

Address	Declaration	Name	Type	Initial value	Comment
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Sc
2.0	temp	OB1_PRIORITY	BYTE		1 (Priority of 1 is lowest)
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milli
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (millisecond
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (millisecond
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

Cột thứ nhất là địa chỉ trong vùng biến cục bộ, cột thứ hai khai báo loại biến, temp nghĩa là tạm thời, giá trị của biến thay đổi sau mỗi vòng quét của OB, cột thứ ba là các tên của dữ liệu, có ý nghĩa như sau (giải thích trong cột chú thích 6):

OB1_EV_CLASS: giá trị B#16#11 có nghĩa OB1 tích cực

OB1_SCAN_1: B#16#01: hoàn tất warm restart
B#16#02: hoàn tất hot restart
B#16#03: hoàn tất chu kỳ
B#16#04: hoàn tất cold restart

OB1_PRIORITY: giá trị 1

OB1_OB_NUMBR: số OB là 1

OB1_RESERVED_1: dự trữ

OB1_RESERVED_2: dự trữ

OB1_PREV_CYCLE: thời gian vòng quét trước (ms)

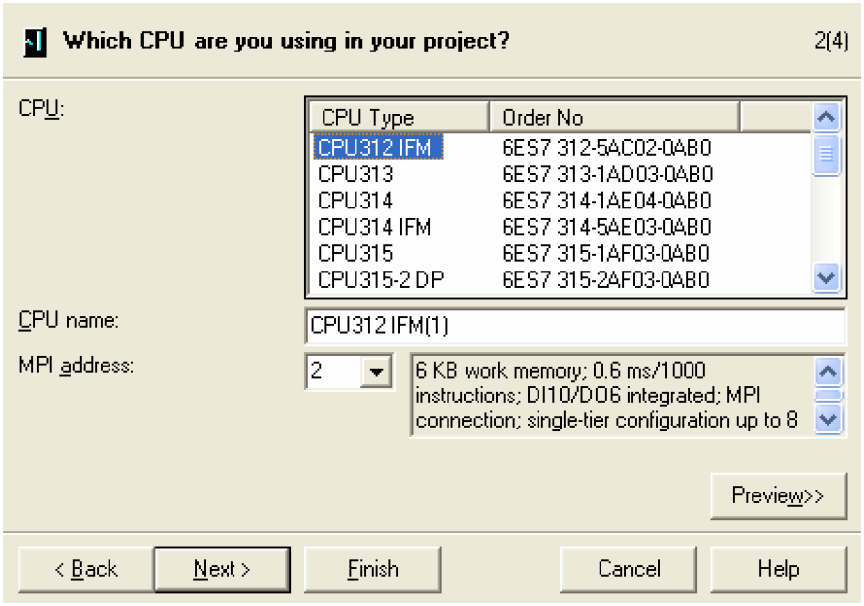
OB1_MIN_CYCLE: thời gian vòng quét ngắn nhất

OB1_MAX_CYCLE: thời gian vòng quét dài nhất

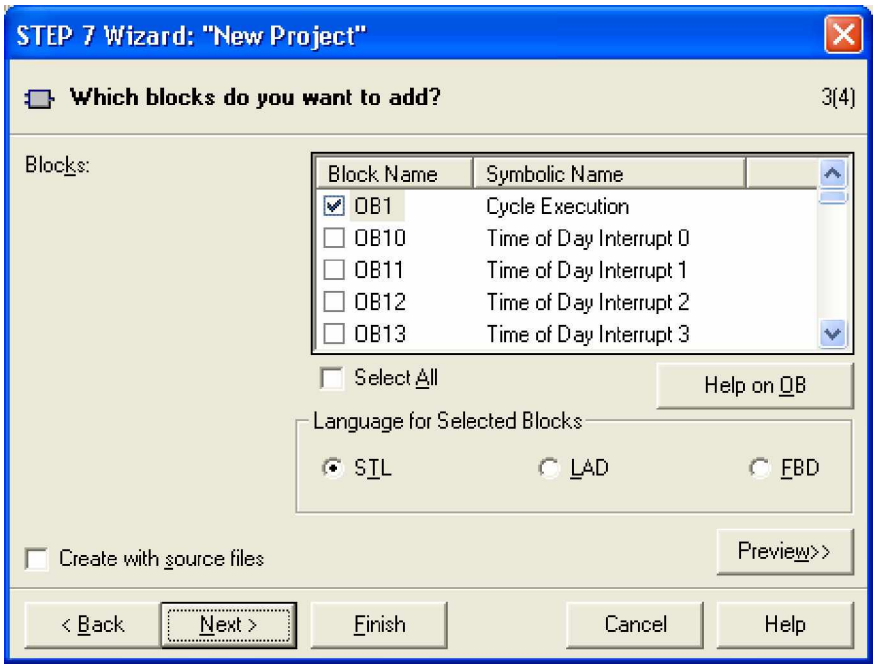
OB1_DATE_TIME: ngày giờ OB1 bắt đầu thực hiện (8 byte)

Các giá trị trên người dùng không thay đổi được, người dùng có thể thêm các biến vào từ địa chỉ 20.0 trở đi, các biến này là biến tạm, thay đổi sau mỗi vòng quét. Các biến thêm vào sử dụng cho việc gọi các chương trình con FC, SFC, FB, SFB.

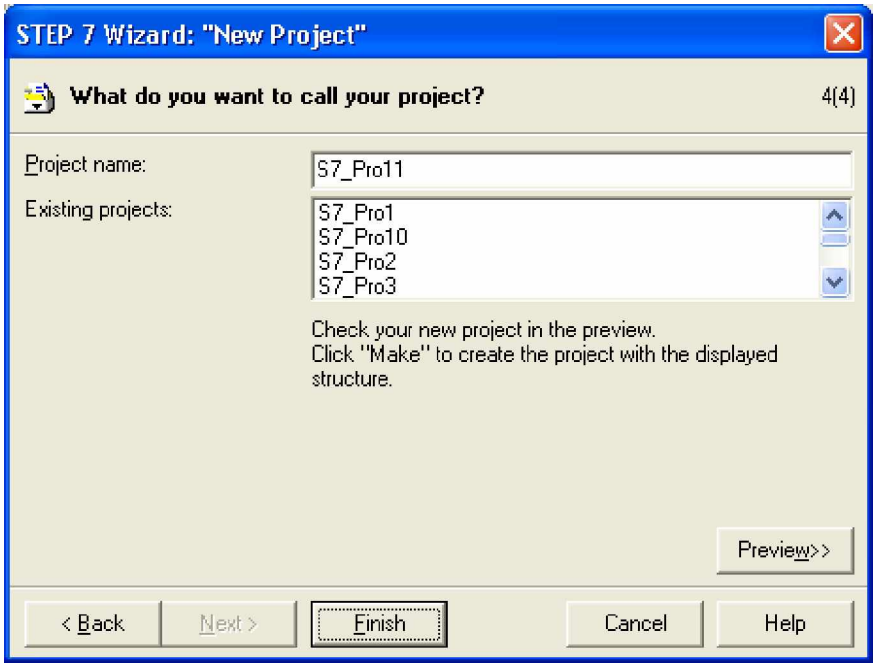
Chương trình STEP 7 dùng để lập trình cho PLC S7-300, S7-400. Chương trình này có version 5.0 dùng cho Win 98, Version 5.1 và 5.3 dùng cho Win XP. Khi kích chuột vào biểu tượng Simatic Manager sẽ xuất hiện cửa sổ Hình , bấm Next để chọn loại CPU

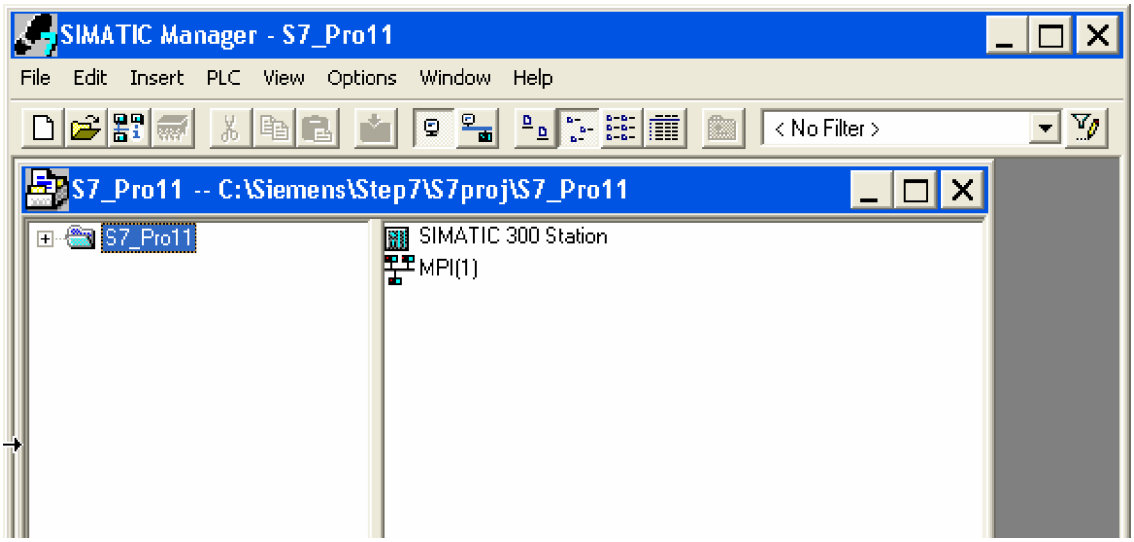


Bấm tiếp Next để chọn các khối OB, bắt buộc là OB1, các OB khác có thể thêm vào sau.

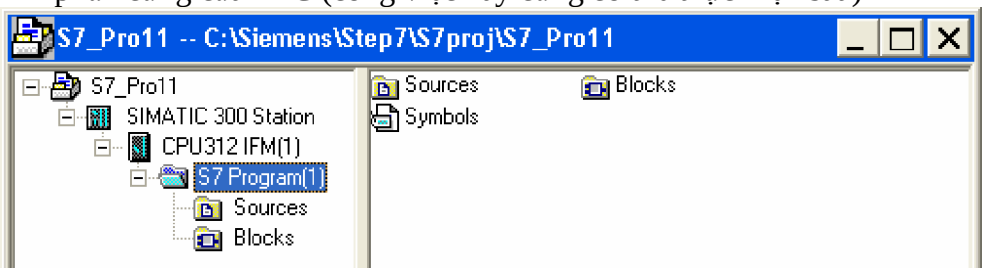


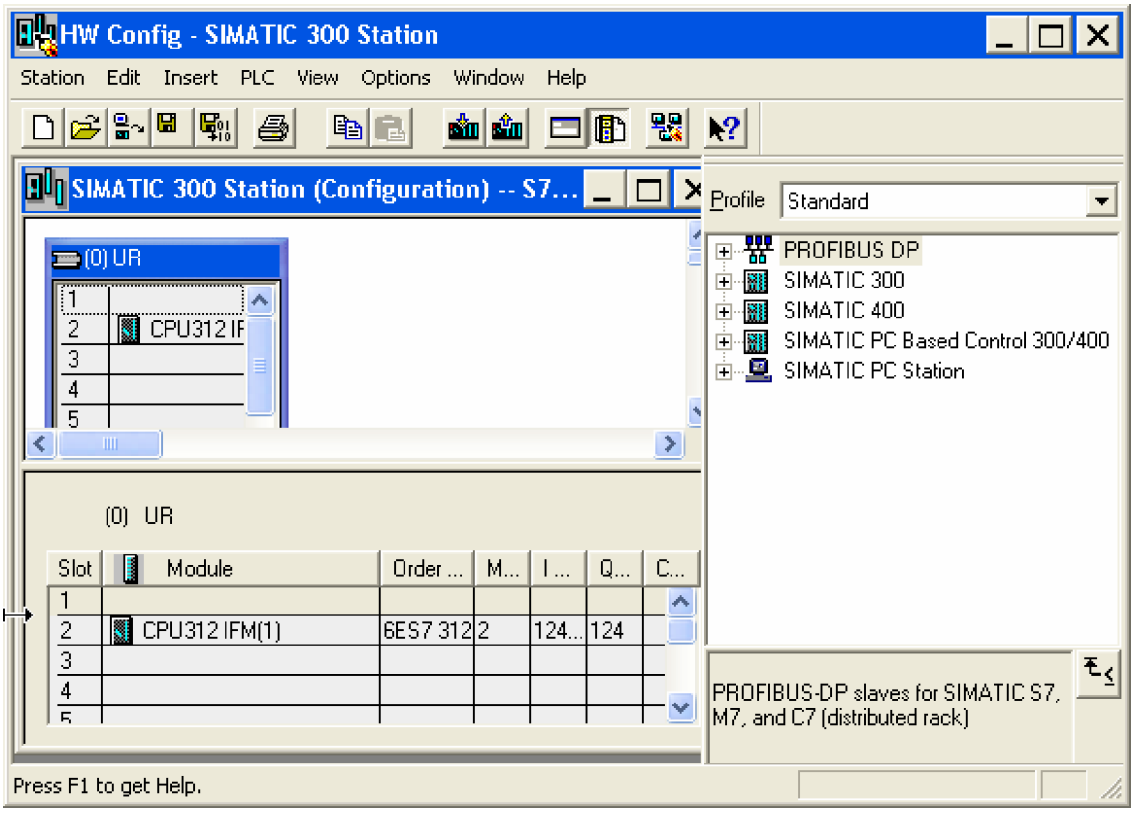
Chọn cách lập trình STL, LAD hay FBD, trong lúc lập trình có thể tuỳ ý thay đổi. Bấm tiếp Next đặt tên cho Project, sau đó bấm Finish, xuất hiện cửa sổ lập trình



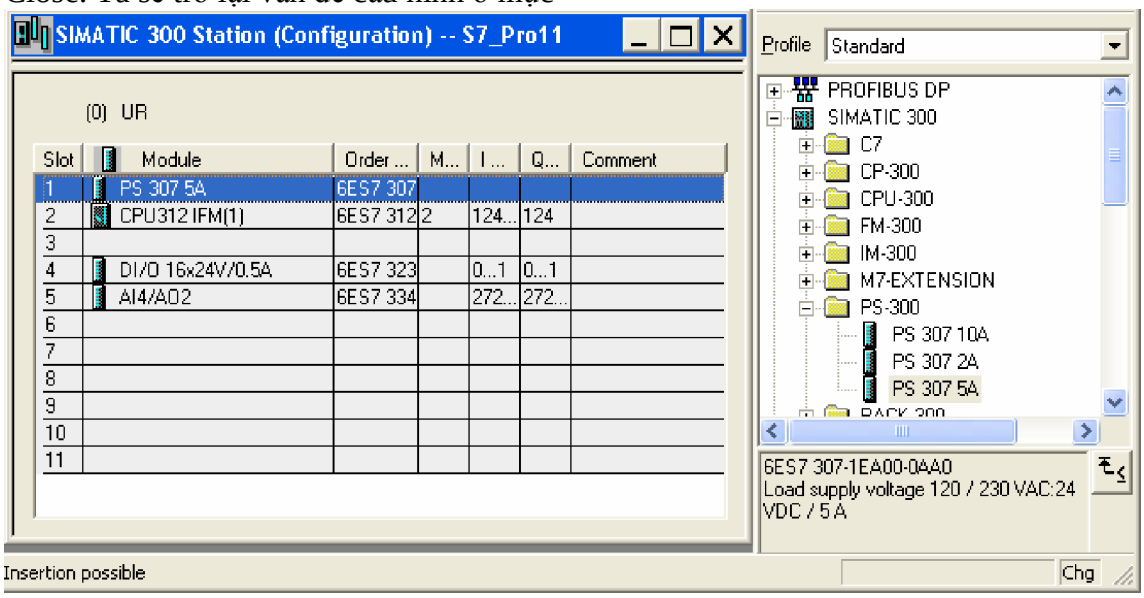


Nửa cửa sổ bên trái sắp xếp dạng thư mục, kích chuôt vào đó để mở ra các mục con. Bấm vào dòng SIMATIC 300 STATION bên trái rồi bấâm tiếp vào Hardware bên phải để đặt cấu hình phần cứng của PLC (công việc này cũng có thể thực hiện sau)

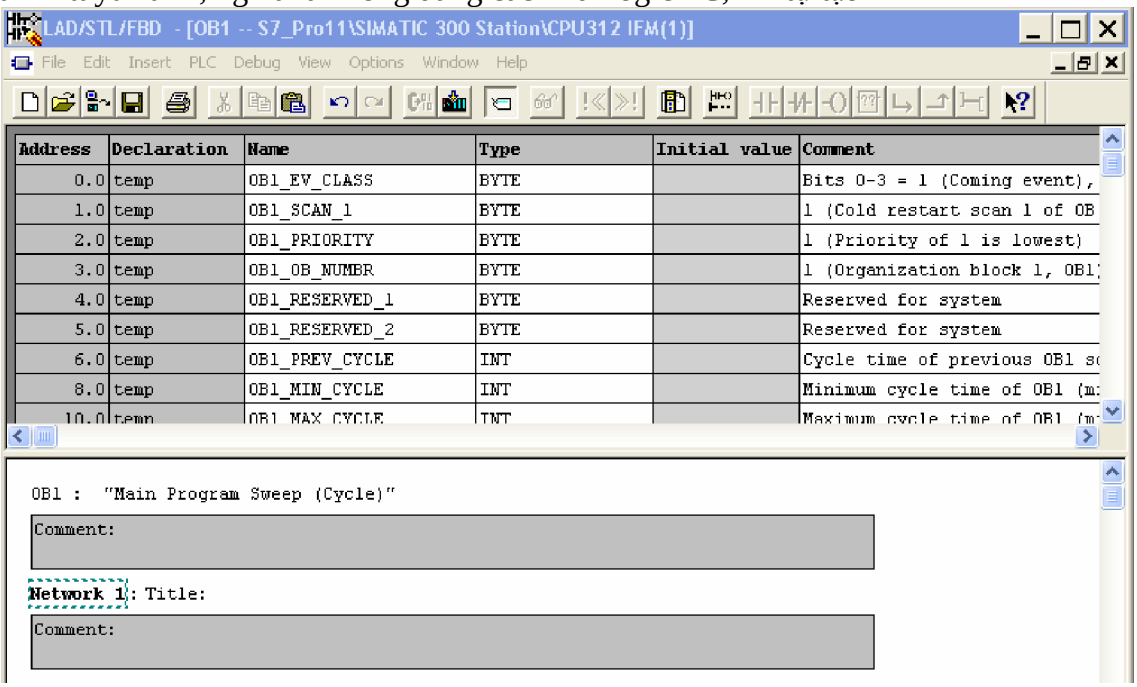




Gỉa sử cấu hình đơn giản gồm các moduke DI/DO, AI/AO, ta kích chuột vào dòng SIMATIC 300, SM- 300 , chọn các module phù hợp, dùng chuột kéo vào các slot của Station từ số 4 trở đi, (slot 3 dùng cho module IM), sau đó vào menu Station – Save rồi Close. Ta sẽ trở lại vấn đề cấu hình ở mục

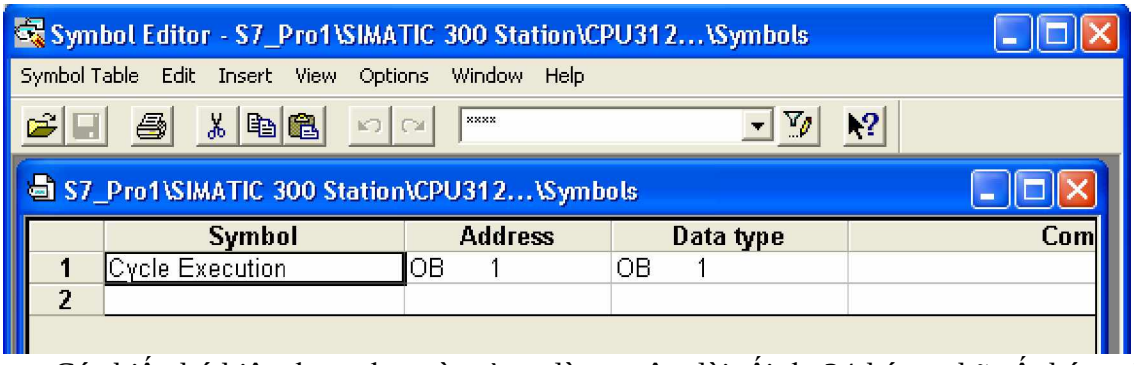


Trở lại Project, bấm vào mục Blocks, ta thấy xuất hiện OB1, bấm vào OB1 nếu lập trình tuyến tính, nghĩa là không dùng các khối logic FC, FB tự tạo



Bấm vào menu View, chọn STL, LAD, FBD chọn cách lập trình. Khi lập trình ta có thể dùng địa chỉ tuyệt đối (I0.0, MW2, T5…) hay địa chỉ ký hiệu (Start, Speed, Delay…). Địa chỉ ký hiệu giúp chương trình dễ hiểu hơn. Có hai loại là ký hiệu cục bộ và ký hiệu toàn cục (hay chia xẻ) , ký hiệu cục bộ khai báo trong bảng khai báo biến của khối và chỉ có ý nghĩa trong phạm vi khối đó, ký hiệu toàn cục khai báo trong bảng ký hiệu Symbols, có ý nghĩa trong toàn bộ các khối của project. Việc khai báo ký hiệu toàn cục thực hiện trước hay sau khi viết mã. Khối logíc có thể có tối đa 999 network, mỗi network có tối đa 2000 hàng , mỗi hàng gồm nhãn , lệnh, địa chỉ và chú thích (sau //)

Thủ tục lập ký hiệu toàn cục như sau: bấm chuột vào đối tượng Symbols (Xem hình).

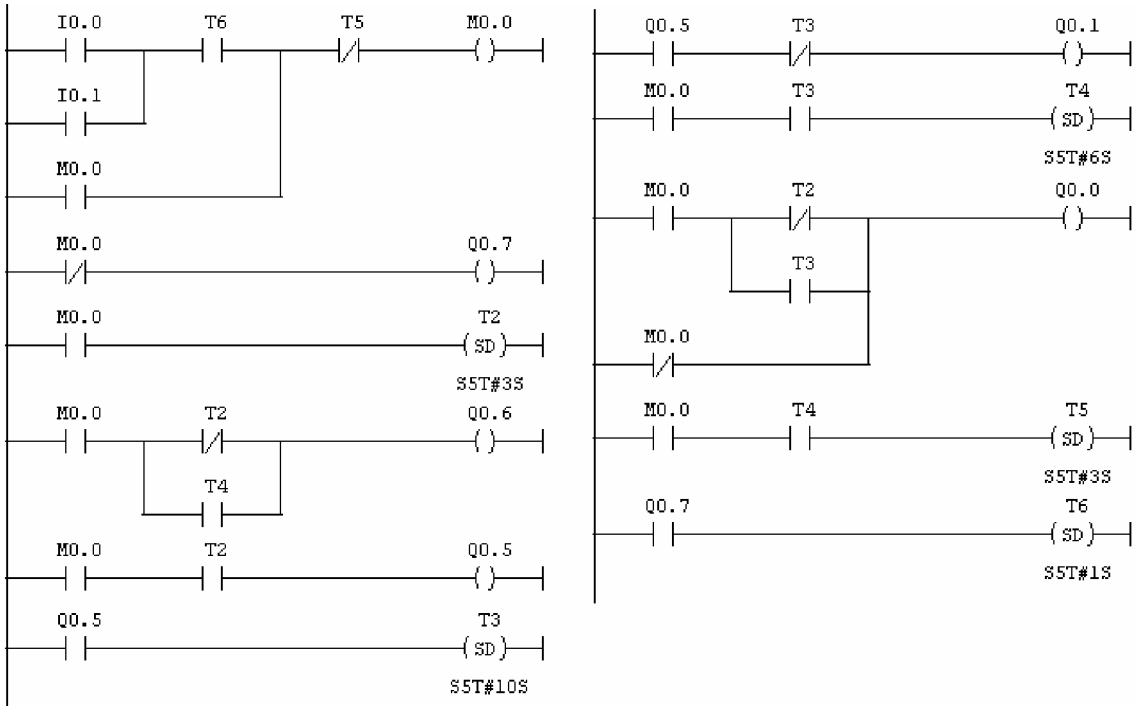


Các biến ký hiệu được đưa vào từng dòng một, dài tối đa 24 ký tụ chữ số, ký tự đặc biệt, trừ dấu nháy " , không phân biệt chữ hoa và chữ thường. Bảng ký hiệu chứa tối đa 16380 ký hiệu. Sau khi đã biên tập xong, vào menu Symbol Table- Save để lưu bảng. Vào cửa sổ biên tập của khối chọn View- Display with - Symbolic Representation để nhìn thấy địa chỉ ký hiệu trong chương trình, ký hiệu tuyệt đối được đóng khung bằng dấu ", còn ký hiệu cục bộ có dấu # đứng trước.

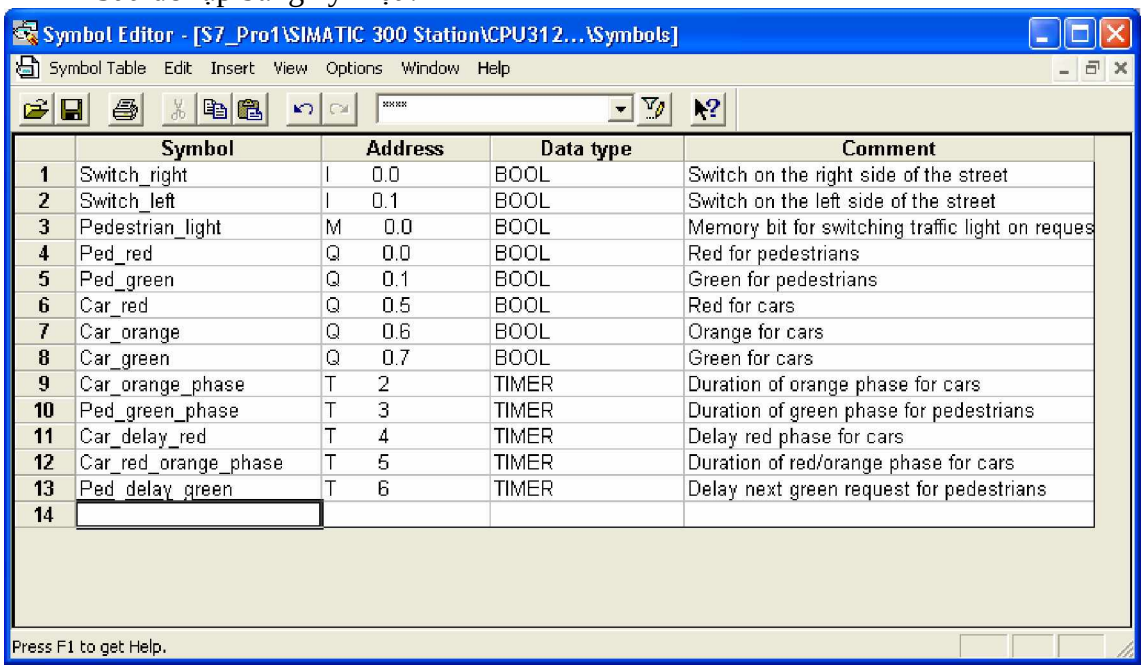
Ví dụ lập trình cho đèn bộ hành, bình thường khi không có yêu cầu qua đường (I0.0, I0.1), đèn xanh xe (Q0.7) và đèn đỏ bộ hành (Q0.0) sáng. Khi có yêu cầu đèn vàng xe (Q0.6) sáng trong 3s , sau đó đèn đỏ xe (Q0.5) sáng và đèn xanh bộ hành (Q0.1) sáng trong 10s, hết thời gian này đèn đỏ bộ hành và đỏ xe cùng sáng, sau 6s đèn vàng xe và đỏ xe cùng sáng và sau 3 s đèn xanh xe sáng , xóa yêu cầu qua đường

```
A(
A(
 O     I     0.0      // Có yêu cầu qua đường của khách bộ hành
 O     I     0.1
 )
 A     T     6
 O     M     0.0
 )
 AN    T     5        //xóa yêu cầu
 =     M     0.0      // ghi nhận yêu cầu

AN     M     0.0      // nếu không có yêu cầu thì
 =     Q     0.7      // đèn xanh xe sáng

A      M     0.0
 L     S5T#3S
SD     T     2
A      M     0.0
A(
ON     T     2
O      T     4
 )
=      Q     0.6      //Đèn vàng xe 3s
A      M     0.0
A      Q     0.5
L      S5T#10S
SD     T     3
A      Q     0.5
AN     T     3
=      Q     0.1 //Bật đèn xanh bộ hành, thời gian 10s

A      M     0.0
A      T     3
L      S5T#6S
SD     T     4  //Thời gian 6 s đỏ xe và đỏ bộ hành cùng sáng
A      M     0.0
A(
ON     T     2
O      T     3
)
ON     M     0.0
=      Q     0.0      // Bật đèn đỏ bộ hành

A      M     0.0      //Bật đèn đỏ và vàng xe
A      T     4
L      S5T#3S
SD     T     5 //Chuyển sang xanh xe sau
```

A T 2 = Q 0.5 //Đèn đỏ xe sau 3s	3s A Q 0.7 L S5T#1S SD T 6 //Thời gian trì hoãn 1s để nhận yêu cầu khi xanh xe vừa sáng



Sau đó lập bảng ký hiệu:



	Symbol	Address	Data type	Comment
1	Switch_right	I 0.0	BOOL	Switch on the right side of the street
2	Switch_left	I 0.1	BOOL	Switch on the left side of the street
3	Pedestrian_light	M 0.0	BOOL	Memory bit for switching traffic light on reques
4	Ped_red	Q 0.0	BOOL	Red for pedestrians
5	Ped_green	Q 0.1	BOOL	Green for pedestrians
6	Car_red	Q 0.5	BOOL	Red for cars
7	Car_orange	Q 0.6	BOOL	Orange for cars
8	Car_green	Q 0.7	BOOL	Green for cars
9	Car_orange_phase	T 2	TIMER	Duration of orange phase for cars
10	Ped_green_phase	T 3	TIMER	Duration of green phase for pedestrians
11	Car_delay_red	T 4	TIMER	Delay red phase for cars
12	Car_red_orange_phase	T 5	TIMER	Duration of red/orange phase for cars
13	Ped_delay_green	T 6	TIMER	Delay next green request for pedestrians
14				

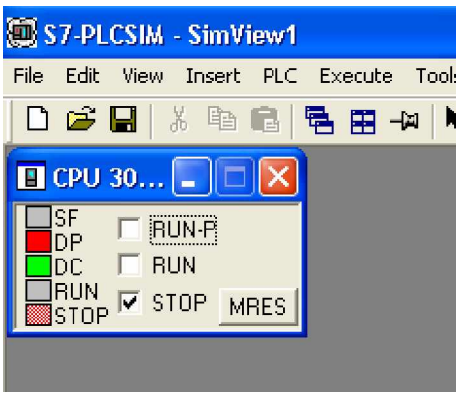
A(	A "Pedestrian_light"

```
A(
O    "Switch_right"
O    "Switch_left"
)
A    "Ped_delay_green"
O    "Pedestrian_light"
)
AN   "Car_red_orange_phase"
=    "Pedestrian_light"
AN   "Pedestrian_light"
=    "Car_green"
A    "Pedestrian_light"
L    S5T#3S
SD   "Car_orange_phase"
A    "Pedestrian_light"
A(
ON   "Car_orange_phase"
O    "Car_delay_red"
)
=    "Car_orange"
A    "Pedestrian_light"
A(
ON   "Car_orange_phase"
O    "Car_delay_red"
)
=    "Car_orange"
```

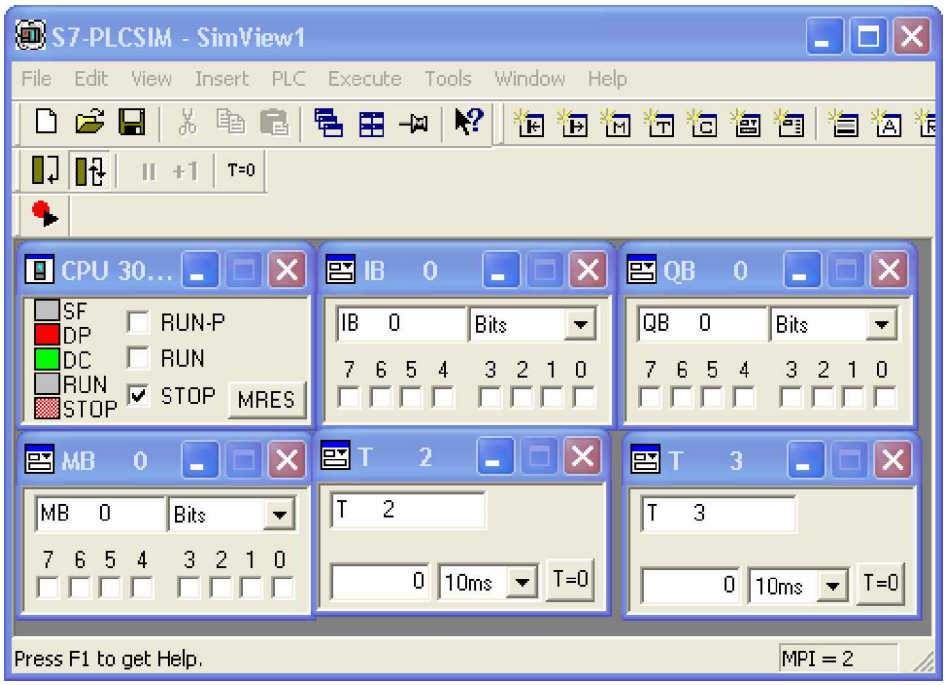
```
A    "Car_orange_phase"
=    "Car_red"
A    "Car_red"
L    S5T#10S
SD   "Ped_green_phase"
A    "Car_red"
AN   "Ped_green_phase"
=    "Ped_green"
A    "Pedestrian_light"
A    "Ped_green_phase"
L    S5T#6S
SD   "Car_delay_red"
A    "Pedestrian_light"
A(
ON   "Car_orange_phase"
O    "Ped_green_phase"
)
ON   "Pedestrian_light"
=    "Ped_red"
A    "Pedestrian_light"
A    "Car_delay_red"
L    S5T#3S
SD   "Car_red_orange_phase"
A    "Car_green"
L    S5T#1S
SD   "Ped_delay_green"
```

Sau khi biên soạn chương trình ta có thể chạy mô phỏng không cần PLC nhờ phần mềm S7 PLC Sim theo các bước sau:

- Vào menu Simatic Manager- Options- chọn Simulate Modules. Cửa sổ sau xuất hiện

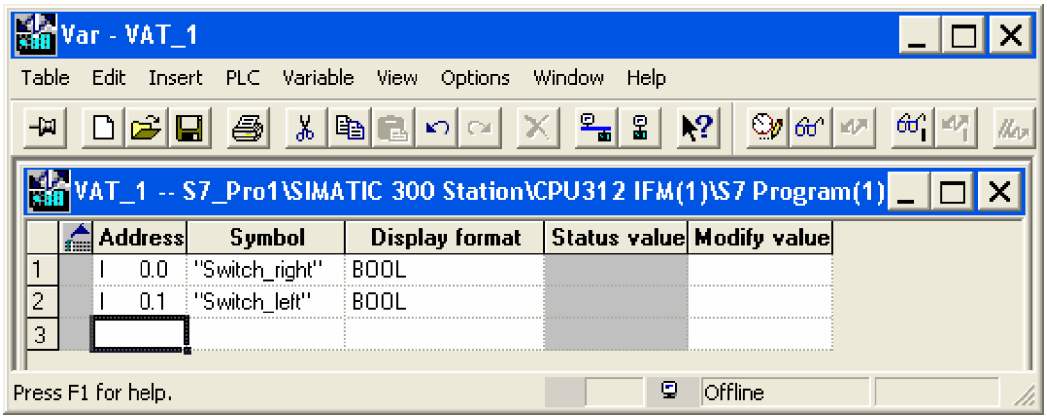


- Vào menu PLC- Download để nạp khối chương trình xuống PLC mô phỏng
- Vào cửa sổ S7-PLCSIM menu Insert chọn các vùng nhớ muốn quan sát



- Vào menu PLC- chọn Power On, vào menu Execute chọn Scan Mode – Continuous Scan.
- Chọn RUN hay RUN –P
- Tác độâng vào các bit I 0.0, I0.1 để xem hoạt động của chương trình.
- Trở lại Simatic Manager, chọn View- Online, mở khối logic muốn quan sát (OB1), bấm Debug- Monitor

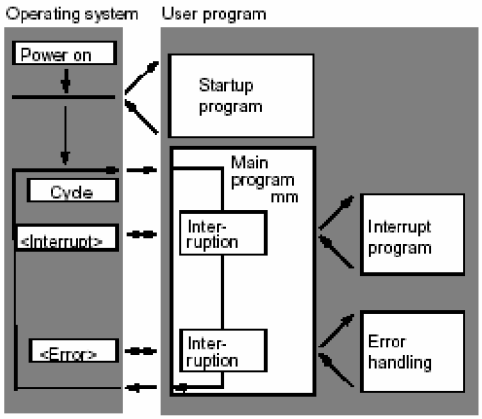
Trong trường hợp muốn tập trung các biến vào một chỗ để dễ quan sát, ta dùng bảng khai báo biến VAT (Variable Table). Trong cửa sổ Manager vào menu Insert- S7 Block- Variable Table (hay bấm chuột phải – Insert New Object- Variable Table) ta được khối VAT1, mở khối này ra và thêm vào các địa chỉ vùng nhớ muốn quan sát.

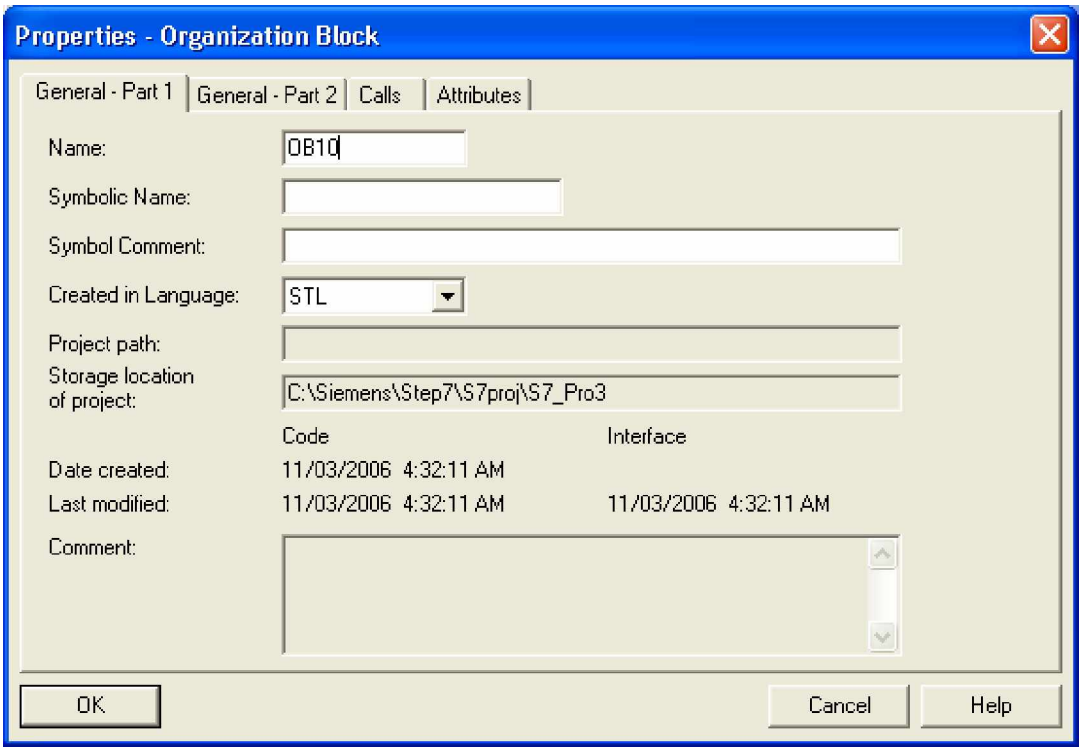


Trường hợp có sẵn PLC, đầu tiên ta phải kết nối máy tính với PLC thông qua cáp nối thích hợp, vào menu PLC- Display Accessible Nodes, sau đó PLC- Operating mode chọn chế độ PLC là Stop, PLC- Download nạp chương trình xuống PLC.

5.2/ Các khối ngắt

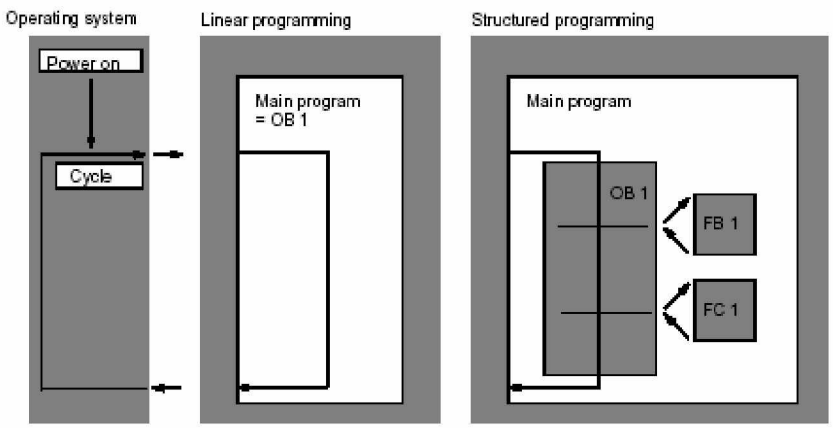
Khối OB1 được thực hiện theo chu kỳ, và có thể bị ngắt bởi các sự kiện khi ta cài đặt thêm các khối OB khác vào Project hoặc khi xảy ra các sự cố. Các khối OB phù hợp được gọi để xử lý ngắt nhờ các chương trình con được cài đặt. Khối OB ưu tiên cao có thể ngắt khối có ưu tiên thấp hơn. Ta có thể thay đổi ưu tiên của OB trong S7-400 và S7-300-CPU318. Thêm OB b ằng cách bấm chuột phải trong cửa sổ Project- Insert New Object- Organization block, chọn số OB, sau đó mở khối OB và lập trình





3 Tạo các khối logic

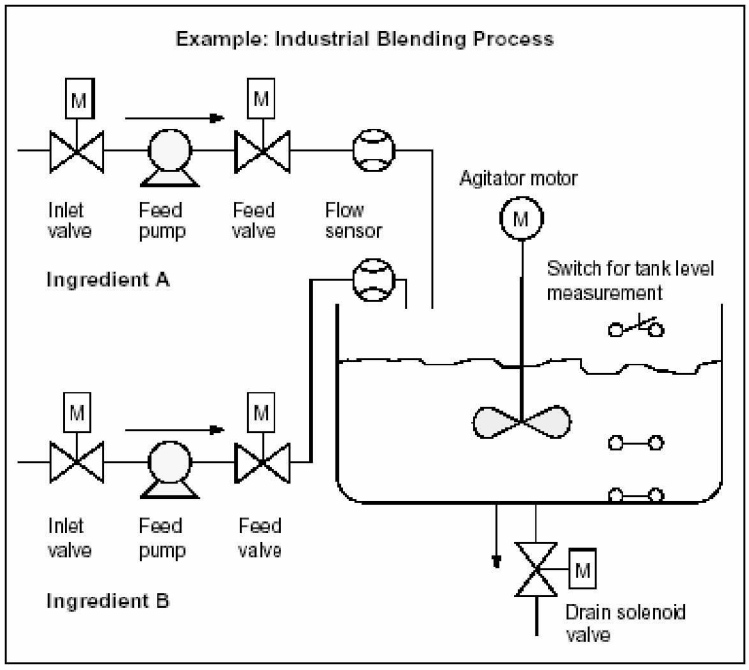
Các chương trình lớn thường được viết dạng cấu trúc, gồm khối OB1, các khối chương trình FC, FB, các khối chương trình hệ thống SFC, SFB. Sử dụng lập trình cấu trúc giúp chương trình dễ quản lý và sửa lỗi, thuận tiện cho việc lập trình theo nhóm. Khối OB1 và các khối FC, FB có thể gọi FC, FB, SFC, SFB

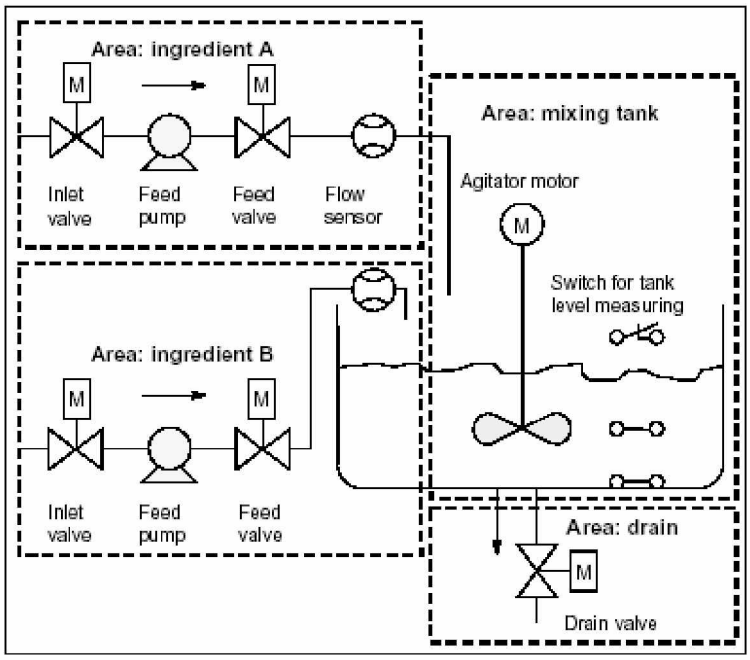


Lấy ví dụ lập trình cho hệ thống trộn hai chất lỏng A và B (H), ta chia quá trình thành nhiều khối nhỏ (H) : bơm chất A, bơm chất B, bồn trộn và van xả. Ta nhận thấy hai khối bơm lập trình giống nhau, chỉ khác ở các ngõ vào/ra. Trước khi lập trình ta phải có mô tả kỹ thuật cho hoạt động của các khối.

Khối A/B gồm có bơm và van vào, van ra

- **Bơm có công suất 100KW, vòng quay 1200 rpm, lưu lượng 400l/phút. Bơm được điều khiển bởi nút Start/Stop trên bảng điều khiển, số lần start được hiển thị để tiện bảo trì. Bơm được phép hoạt động khi:**





- bồn không đầy,
- van xả đóng,
- nút emergency không tác động.

Bơm tắt khi cảm biến lưu lượng báo không có dòng chảy sau 7 s kể từ khi khởi đông bơm hay khi cảm biến lưu lượng báo đã ngừng chảy.

- **Van được điều khiển bởi solenoid, mở khi có điện vào van. Van phải mở ít nhất 1s sau khi bơm chạy.**

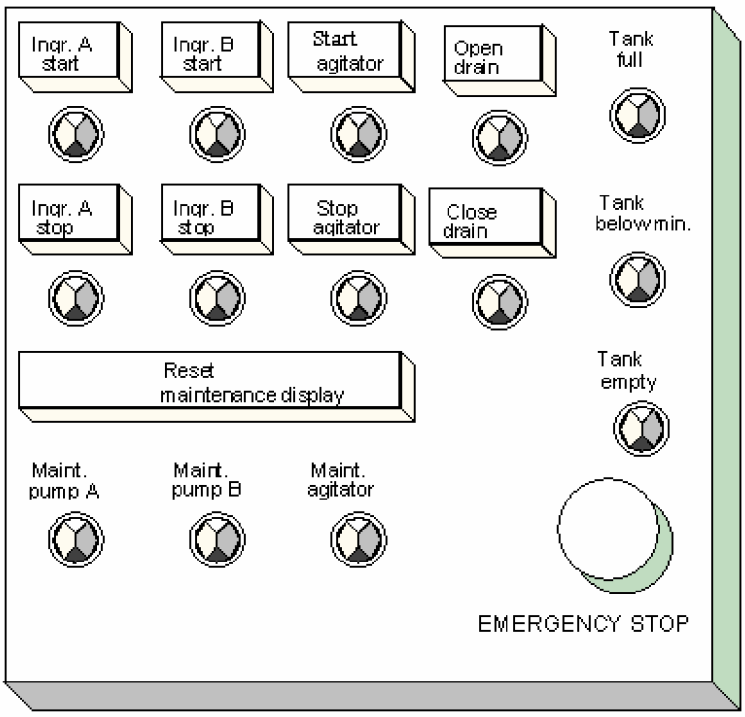
Khối bồn trộn có động cơ trộn, các cảm biến mức. có công suất 100KW, vòng quay 1200 rpm, lưu lượng 400l/phút. Động cơ được điều khiển bởi nút Start/Stop trên bảng điều khiển, số lần start được hiển thị để tiện bảo trì. Động cơ được phép chạy khi:

- Mức chất lỏng trên mức tối thiểu ,
- Van xả đóng
- Nút Emergency không tác động

Động cơ được tắt khi vận tốc không đạt định mức sau khi khởi động 10s. Có ba cảm biến mức dạng contact . Cảm biến đầy thường đóng, khi bồn đầy thì hở ra. Cảm biến mức tối thiểu thường hở , khi mực chất lỏng thấp thì đóng lại. Cảm biến cạn, hở nếu bồn cạn

Van xả được điều khiển từ bảng điều khiển. Van xả được hoạt động nếu động cơ trộn ngừng, cảm biến mức báo bồn chưa cạn, nút emergency không tác động. Van xả đóng nếu cảm biến mức báo bồn cạn.

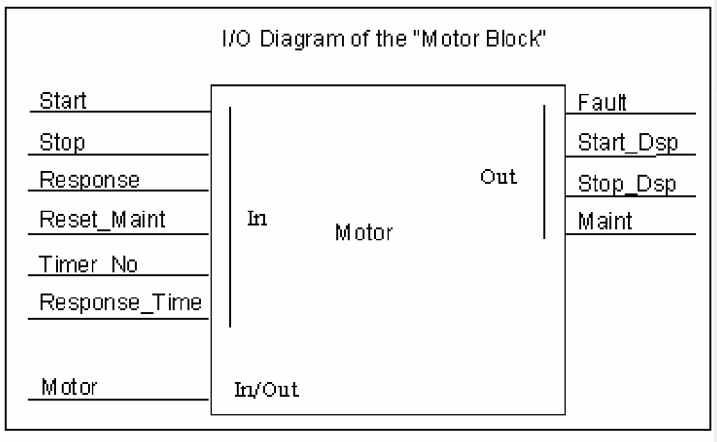
Bảng điều khiển dùng để điều khiển và báo trạng thái các động cơ, van xả, báo mức bồn, báo bảo trì và dừng khẩn cấp.

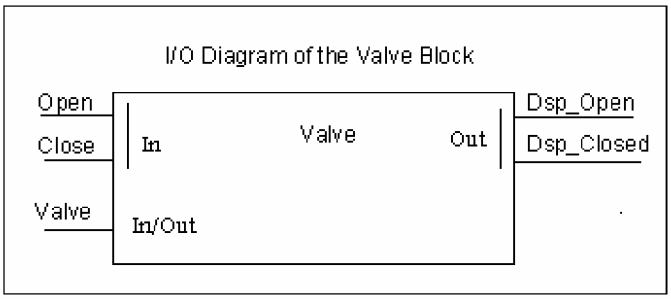


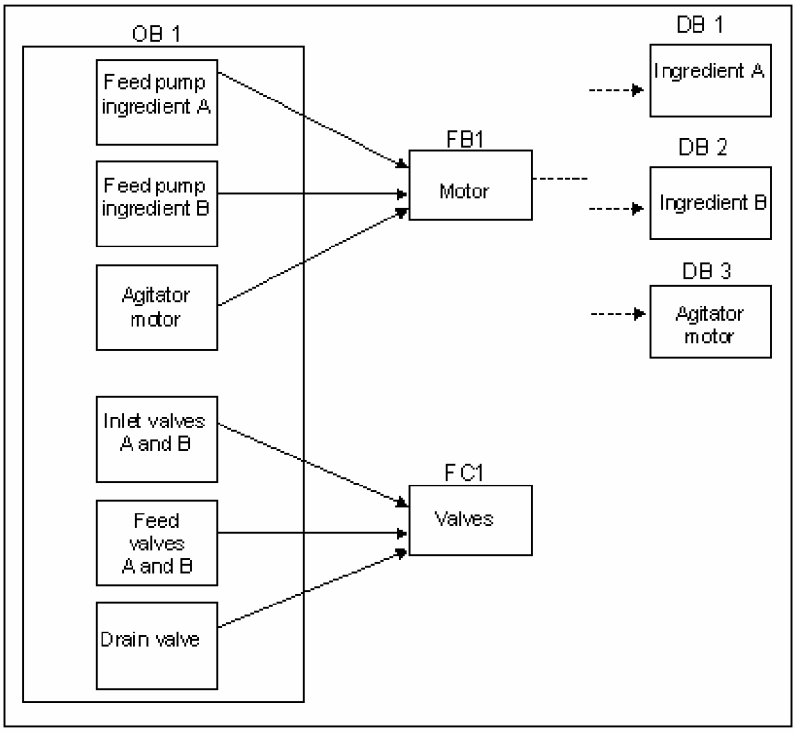
Có ba động cơ có thể lập trình bằng khối logic chung (Hình) . Sáu ngõ vào là hai nút nhấn Sart/Stop, nút nhấn Reset_Maint xóa đèn bảo trì, tín hiệu báo động cơ chạy, ngừng (Response) số hiệu Timer (Timer No) và thời gian timer (Response_Time). Bốn ngõ ra là báo lỗi (Fault), đèn báo động cơ chạy, ngừng (Start_Dsp, Stop_Dsp), báo bảo trì (Maint), Tín hiệu vào/ra là điều khiển Motor. Khối logic này lập trình dưới dạng khối hàm FB vì cần lưu trữ giá trị biến.

Các van cũng được điều khiển bằng khối logic FC (Hình) . Hai tín hiệu vào là nút nhấn mở /đóng van (Open/ Close) Tín hiệu ra là đèn báo trạng thái van (Dsp_Open, Dsp_Closed). Tín hiệu vào/ ra điều khiển van (Valve). Khối này không có lưu biến và thực hiện bằng FC.

Cấu trúc chương trình như Hình . Chương trình chính OB1 gọi hàm FB1 điều khiển động cơ, có ba động cơ ứng với ba khối dữ liệu DB1, DB2, DB3. Hàm FC1 được OB1 gọi khi điều khiển van. Các khối FB và FC phải được lập trình trước khối OB. Vào cửa sổ Project –Symbols lập bảng ký hiệu cho các biến (Bảng)







Symbolic Name	Address	Data Type	Description
Feed_pump_A_start	I0.0	BOOL	Starts the feed pump for ingredient A
Feed_pump_A_stop	I0.1	BOOL	Stops the feed pump for ingredient A
Flow_A	I0.2	BOOL	Ingredient A flowing
Inlet_valve_A	Q4.0	BOOL	Activates the inlet valve for ingredient A
Feed_valve_A	Q4.1	BOOL	Activates the feed valve for ingredient A
Feed_pump_A_on	Q4.2	BOOL	Lamp for ”feed pump ingredient A running"
Feed_pump_A_off	Q4.3	BOOL	Lamp for ”feed pump ingredient A not running"
Feed_pump_A	Q4.4	BOOL	Activates the feed pump for ingredient A
Feed_pump_A_fault	Q4.5	BOOL	Lamp for ”feed pump A fault"
Feed_pump_A_maint	Q4.6	BOOL	Lamp for ”feed pump A maintenance"
Feed_pump_B_start	I0.3	BOOL	Starts the feed pump for ingredient B
Feed_pump_B_stop	I0.4	BOOL	Stops the feed pump for ingredient B
Flow_B	I0.5	BOOL	Ingredient B flowing

Inlet_valve_B	Q5.0	BOOL	Activates the inlet valve for ingredient A
Feed_valve_B	Q5.1	BOOL	Activates the feed valve for ingredient B
Feed_pump_B_on	Q5.2	BOOL	Lamp for ”feed pump ingredient B running"
Feed_pump_B_off	Q5.3	BOOL	Lamp for ”feed pump ingredient B not running"
Feed_pump_B	Q5.4	BOOL	Activates the feed pump for ingredient B
Feed_pump_B_fault	Q5.5	BOOL	Lamp for ”feed pump B fault"
Feed_pump_B_maint	Q5.6	BOOL	Lamp for ”feed pump B maintenance"
Agitator_running	I1.0	BOOL	Response signal of the agitator motor
Agitator_start	I1.1	BOOL	Agitator start button
Agitator_stop	I1.2	BOOL	Agitator stop button
Agitator	Q8.0	BOOL	Activates the agitator
Agitator_on	Q8.1	BOOL	Lamp for "agitator running"
Agitator_off	Q8.2	BOOL	Lamp for "agitator not running"
Agitator_fault	Q8.3	BOOL	Lamp for ”agitator motor fault"
Agitator_maint	Q8.4	BOOL	Lamp for ”agitator motor maintenance"
Tank_below_max	I1.3	BOOL	Sensor ”mixing tank not full"
Tank_above_min	I1.4	BOOL	Sensor ”mixing tank above minimum level"
Tank_not_empty	I1.5	BOOL	Sensor ”mixing tank not empty"
Tank_max_disp	Q9.0	BOOL	Lamp for "mixing tank full"
Tank_min_disp	Q9.1	BOOL	Lamp for "mixing tank below minimum level"
Tank_empty_disp	Q9.2	BOOL	Lamp for "mixing tank empty"
Drain_open	I0.6	BOOL	Button for opening the drain valve
Drain_closed	I0.7	BOOL	Button for closing the drain valve
Drain	Q9.5	BOOL	Activates the drain valve
Drain_open_disp	Q9.6	BOOL	Lamp for "drain valve open"
Drain_closed_disp	Q9.7	BOOL	Lamp for "drain valve closed"
EMER_STOP_off	I1.6	BOOL	EMERGENCY STOP switch
Reset_maint	I1.7	BOOL	Reset switch for the maintenance lamps on all motors
Motor_block	FB1	FB1	FB for controlling pumps and motor
Valve_block	FC1	FC1	FC for controlling the valves
DB_feed_pump_A	DB1	FB1	Instance DB for controlling feed pump A
DB_feed_pump_B	DB2	FB1	Instance DB for controlling feed pump B
DB_agitator	DB3	FB1	Instance DB for controlling the agitator motor

5.3 Lập trình khối FB

FB là khối logíc với các biến in, out, in_out, static và temp, được tạo ra trong bảng biến địa phương đi kèm. Các biến in, out, in- out là các tham số hìmh thức có địa chỉ cụ thể do chương trình gọi truyền đến, biến static là biến trong chương trình FB được lưu lại khi ra khỏi khối FB, biến temp mất giá trị khi ra khỏi khối FB. Kèm với FB là khối dữ liệu data block chứa các biến in, out, in- out và static. Có thể có nhiều data block cho một FB khi một FB dùng cho các nhiệm vụ khác nhau, gọi là instance data block. Khi chương trình gọi FB cần phải kèm theo instance data block tương ứng. Ta vào cửa sổ Project bấm chuột phải - Insert New Object – Function block thêm vào khối FB1. Bấm chuột vào khối FB1 để soạn chương trình cho khối. Ta vào bảng khai báo biến để khai báo các biến hình thức cho khối theo thứ tự in, out, in_out, static và temp. Với ví dụ ở trên, bảng biến của FB1 "Motor_Block" như sau:

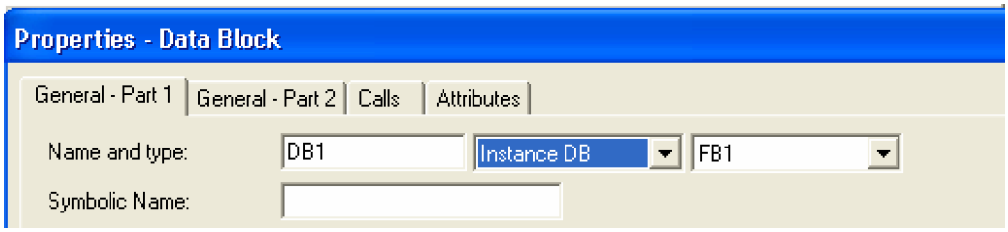
Address	Declaration	Name	Type	Initial Value
0.0	IN	Start	BOOL	FALSE
0.1	IN	Stop	BOOL	FALSE
0.2	IN	Response	BOOL	FALSE
0.3	IN	Reset_Maint	BOOL	FALSE
2.0	IN	Timer_No	TIMER	
4.0	IN	Response_Time	S5TIME	S5T#0MS
6.0	OUT	Fault	BOOL	FALSE
6.1	OUT	Start_Dsp	BOOL	FALSE
6.2	OUT	Stop_Dsp	BOOL	FALSE
6.3	OUT	Maint	BOOL	FALSE
8.0	IN_OUT	Motor	BOOL	FALSE
10.0	STAT	Time_bin	WORD	W#16#0
12.0	STAT	Time_BCD	WORD	W#16#0
14.0	STAT	Starts	INT	0
16.0	STAT	Start_Edge	BOOL	FALSE

Các biến STAT Time_ bin và Time_BCD lư u thời gian timer, Starts lưu số lần khởi động motor, Start_ Edge phục vụ cho lệnh lấy cạnh lên

```
Network 1 Start/stop and latching
A(
O    #Start
O    #Motor
)
AN  #Stop
=      #Motor
Network 2 Startup monitoring
A   #Motor
L   #Response_Time
SD  #Timer_No
AN  #Motor
R #Timer_No
L #Timer_No
T #Timer_bin
Network 4 Stop lamp
AN #Response
= #Stop_Dsp
Network 5 Counting the starts
A #Motor
FP #Start_Edge
JCN lab1
L #Starts
+ 1
T #Starts
lab1: NOP 0
Network 6 Maintenance lamp
L #Starts
L 50
>=I
```

LC #Timer_No T #Timer_BCD A #Timer_No AN #Response S #Fault R #Motor **Network 3 Start lamp and fault reset** A #Response = #Start_Dsp R #Fault	= #Maint **Network 7 Reset counter for number of starts** A #Reset_Maint A #Maint JCN END L 0 T #Starts END: NOP 0

Thêm khối DB project với các tên DB1, DB2, DB3 loại Instance DB và thuộc FB1



Các biến trong DB1 sẽ tự tạo ra theo bảng khai báo biến của khối FB1, tương tự cho các DB2 và DB3.

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Start	BOOL	FALSE	
0.1	in	Stop	BOOL	FALSE	
0.2	in	Response	BOOL	FALSE	
0.3	in	Reset_Maint	BOOL	FALSE	
2.0	in	Timer_No	TIMER		
4.0	in	Response_Time	S5TIME	S5T#0MS	
6.0	out	Fault	BOOL	FALSE	
6.1	out	Start_Dsp	BOOL	FALSE	
6.2	out	Stop_Dsp	BOOL	FALSE	
6.3	out	Maint	BOOL	FALSE	
8.0	in_out	Motor	BOOL	FALSE	
10.0	stat	Time_bin	WORD	W#16#0	
12.0	stat	Time_BCD	WORD	W#16#0	
14.0	stat	Starts	INT	0	
16.0	stat	Start_Edge	BOOL	FALSE	

5.4 Lập trình khối FC

Khối FC có các biến hình thức in, out và in_ out do chương trình gọi cung cấp các địa chỉ cụ thể, ngoài ra còn có biến temp sử dụng nội bộ. Khối FC không có bộ nhớ nên dữ liệu mất đi khi ra khỏi khối. Ta thêm vào project khối FC1 và khai báo các biến trong bảng khai báo biến kèm theo. Sau đó lập trình cho FC1

Address	Declaration	Name	Type	Initial Value
0.0	IN	Open	BOOL	FALSE
0.1	IN	Close	BOOL	FALSE
2.0	OUT	Dsp_Open	BOOL	FALSE
2.1	OUT	Dsp_Closed	BOOL	FALSE
4.0	IN_OUT	Valve	BOOL	FALSE

Network 1 Open/close and latching

```
A(
O #Open
O #Valve
)
AN #Close
= #Valve
```

Network 2 Display "valve open"

```
A #Valve
= #Dsp_Open
```

Network 3 Display "valve closed"

```
AN#Valve
= #Dsp_Closed
```

Bước tiếp theo là lập trình cho OB1, ta khai báo các biến cho OB1

Address	Declaration	Name	Type
0.0	**TEMP**	**OB1_EV_CLASS**	**BYTE**
1.0	**TEMP**	**OB1_SCAN1**	**BYTE**
2.0	**TEMP**	**OB1_PRIORITY**	**BYTE**
3.0	**TEMP**	**OB1_OB_NUMBR**	**BYTE**
4.0	**TEMP**	**OB1_RESERVED_1**	**BYTE**
5.0	**TEMP**	**OB1_RESERVED_2**	**BYTE**
6.0	**TEMP**	**OB1_PREV_CYCLE**	**INT**
8.0	**TEMP**	**OB1_MIN_CYCLE**	**INT**
10.0	**TEMP**	**OB1_MAX_CYCLE**	**INT**
12.0	**TEMP**	**OB1_DATE_TIME**	**DATE_AND_TIME**
20.0	TEMP	Enable_motor	BOOL
20.1	TEMP	Enable_valve	BOOL
20.2	TEMP	Start_fulfilled	BOOL
20.3	TEMP	Stop_fulfilled	BOOL
20.4	TEMP	Inlet_valve_A_open	BOOL

20.5 TEMP Inlet_valve_A_closed BOOL

Chương trình OB1

Network 1 Interlocks for feed pump A
A "EMER_STOP_off"
A "Tank_below_max"
AN "Drain"
= #Enable_Motor

Network 2 Calling FB Motor for ingredient A
A "Feed_pump_A_start"
A #Enable_Motor
= #Start_Fulfilled
A(
O "Feed_pump_A_stop"
ON #Enable_Motor
)
= #Stop_Fulfilled
CALL "Motor_block",
"DB_feed_pump_A"
Start :=#Start_Fulfilled
Stop :=#Stop_Fullfilled
Response :="Flow_A"
Reset_Maint :="Reset_maint"
Timer_No :=T12
Reponse_Time:=S5T#7S
Fault :="Feed_pump_A_fault"
Start_Dsp :="Feed_pump_A_on"
Stop_Dsp :="Feed_pump_A_off"
Maint :="Feed_pump_A_maint"
Motor :="Feed_pump_A"

Network 3 Delaying the valve enable ingredient A
A "Feed_pump_A"
L S5T#1S
SD T 13
AN "Feed_pump_A"
R T 13
A T 13
= #Enable_Valve

Network 4 Inlet valve control for ingredient A
AN "Flow_A"
AN "Feed_pump_A"
= #Close_Valve_Fulfilled
CALL "Valve_block"
Open :=#Enable_Valve
Close :=#Close_Valve_Fulfilled
Dsp_Open :=#Inlet_Valve_A_Open
Dsp_Closed:=#Inlet_Valve_A_Closed

Network 8 Delaying the valve enable ingredient B
A "Feed_pump_B"
L S5T#1S
SD T 15
AN "Feed_pump_B"
R T 15
A T 15
= #Enable_Valve

Network 9 Inlet valve control for ingredient B
AN "Flow_B"
AN "Feed_pump_B"
= #Close_Valve_Fulfilled
CALL "Valve_block"
Open :=#Enable_Valve
Close :=#Close_Valve_Fulfilled
Dsp_Open :=#Inlet_Valve_B_Open
Dsp_Closed:=#Inlet_Valve_B_Closed
Valve :="Inlet_Valve_B"

Network 10 Feed valve control for ingredient B
AN "Flow_B"
AN "Feed_pump_B"
= #Close_Valve_Fulfilled
CALL "Valve_block"
Open :=#Enable_Valve
Close :=#Close_Valve_Fulfilled
Dsp_Open :=#Feed_Valve_B_Open
Dsp_Closed:=#Feed_Valve_B_Closed
Valve :="Feed_Valve_B"

Network 11 Interlocks for agitator
A "EMER_STOP_off"
A "Tank_above_min"
AN "Drain"
= #Enable_Motor

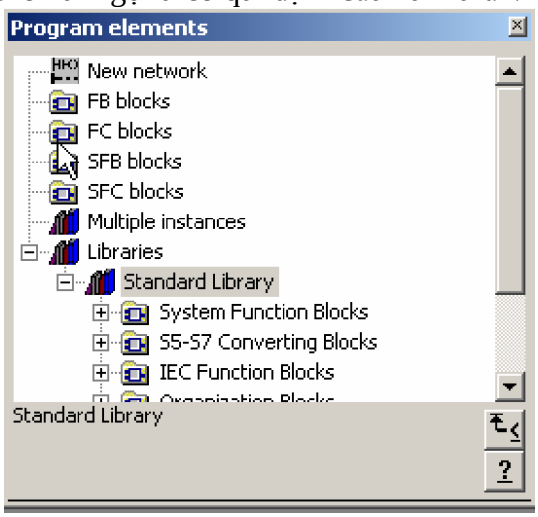
Network 12 Calling FB Motor for agitator
A "Agitator_start"
A #Enable_Motor
= #Start_Fulfilled
A(
O "Agitator_stop"
ON #Enable_Motor
)
= #Stop_Fulfilled

Valve :="Inlet_Valve_A"
Network 5 Feed valve control for ingredient A
AN"Flow_A"
AN"Feed_pump_A"
=#Close_Valve_Fulfilled
CALL"Valve_block"
Open:=#Enable_Valve
Close:=#Close_Valve_Fulfilled
Dsp_Open:=#Feed_Valve_A_Open
Dsp_Closed:=#Feed_Valve_A_Closed
Valve :="Feed_Valve_A"
Network 6 Interlocks for feed pump B
A "EMER_STOP_off"
A "Tank_below_max"
AN "Drain"
= "Enable_Motor
Network 7 Calling FB Motor for ingredient B
A "Feed_pump_B_start"
A #Enable_Motor
= #Start_Fulfilled
A(
O "Feed_pump_B_stop"
ON #Enable_Motor
)
= #Stop_Fulfilled
CALL "Motor_block",
"DB_feed_pump_B"
Start :=#Start_Fulfilled
Stop :=#Stop_Fullfilled
Response :="Flow_B"
Reset_Maint :="Reset_maint"
Timer_No :=T14
Reponse_Time:=S5T#7S
Fault :="Feed_pump_B_fault"
Start_Dsp :="Feed_pump_B_on"
Stop_Dsp :="Feed_pump_B_off"
Maint :="Feed_pump_B_maint"
Motor :="Feed_pump_B"

CALL "Motor_block", "DB_Agitator"
Start :=#Start_Fulfilled
Stop :=#Stop_Fullfilled
Response :="Agitator_running"
Reset_Maint :="Reset_maint"
Timer_No :=T16
Reponse_Time:=S5T#10S
Fault :="Agitator_fault"
Start_Dsp :="Agitator_on"
Stop_Dsp :="Agitator_off"
Maint :="Agitator_maint"
Motor :="Agitator"
Network 13 Interlocks for drain valve
A"EMER_STOP_off"
A"Tank_not_empty"
AN"Agitator"
= "Enable_Valve
Network 14 Drain valve control
A "Drain_open"
A #Enable_Valve
= #Open_Drain
A(
O "Drain_closed"
ON #Enable_Valve
)
= #Close_Drain
CALL "Valve_block"
Open :=#Open_Drain
Close :=#Close_Drain
Dsp_Open :="Drain_open_disp"
Dsp_Closed :="Drain_closed_disp"
Valve :="Drain"
Network 15 Tank level display
AN"Tank_below_max"
="Tank_max_disp"
AN"Tank_above_min"
="Tank_min_disp"
AN"Tank_not_empty"
= "Tank_empty_disp"

4. Sử dụng hàm thư viện

Các hàm thư viện do Siemens viết sẵn thuộc các loại FC, FB, SFC,SFB giúp người dùng thuận tiện trong lập trình. Muốn dùng các hàm thư viện trong khối logic nào thì ta mở khối logic đó ra, kích chuột vào chỗ lệnh CALL gọi hàm thư viện, vào menu Insert-Program Elements- Libraries chọn các hàm thư viện phù hợp rồi bấm chuột kép vào đó, hoặc gõ CALL tên hàm thư viện. Các hàm thư viện cần chuyển giá trị cho các biến hình thức và ta phải cung cấp cho hàm gọi theo qui định của hàm thư viện.



Chương 7: CÁC KHỐI TỔ CHỨC NGẮT

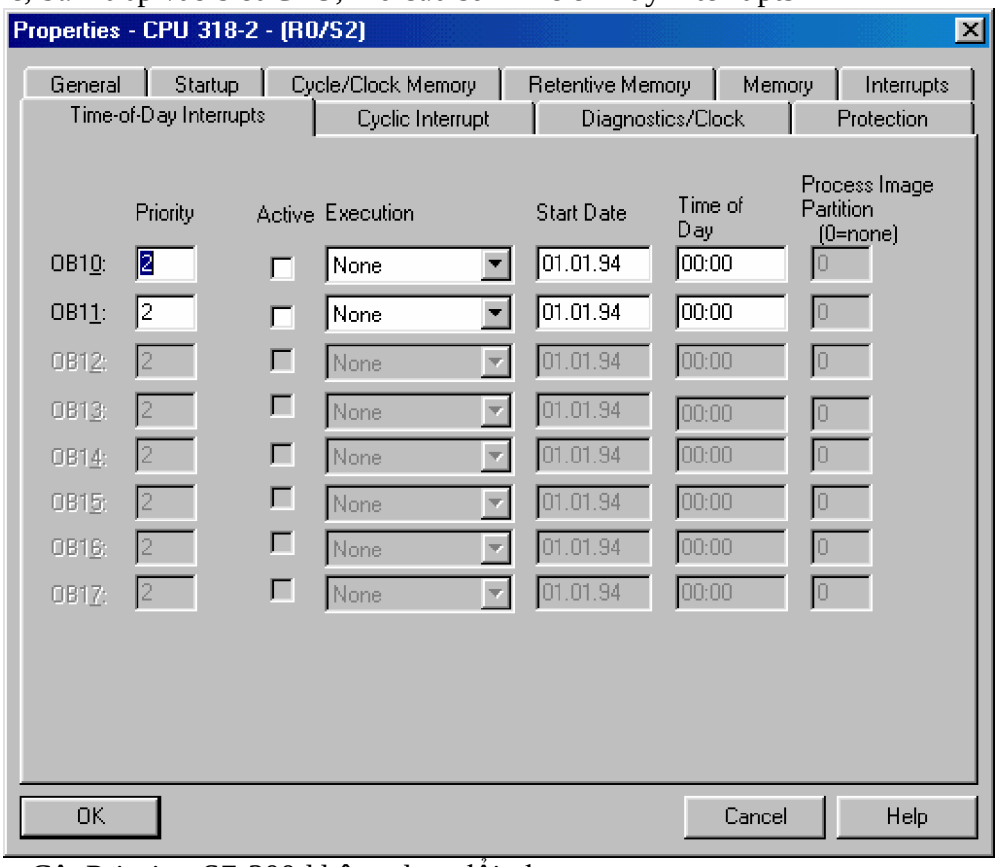
Các khối tổ chức ngắt

1/ Ngắt thời gian

Có tám OB từ OB10 đến OB17 gây ra ngắt ở một thời điểm xác định. Có thể cài đặt để các ngắt này xảy ra một lần, hay theo chu kỳ hàng giờ, hàng ngày, hàng tuần , hàng tháng dùng phần mềm Step7 cấu hình PLC hay dùng các hàm hệ thống. Số các ngắt sử dụng được tùy thuộc koại CPU

Loại	**Các ngắt thời gian**
CPU 312	Không có
CPU 313, 314, 315, 316	OB10
CPU 318, 412, 413	OB10, OB11
CPU 414	OB 10..OB13
CPU 416, 417	OB 10..OB17

Gỉa sử ta đã cài cấu hình PLC CPU 318 cho Project. Trong cửa sổ Project bấm vào Hardware, bấm tiếp vào slot CPU, mở cửa sổ Time of Day Interrupts



- Cột Priority: S7-300 không thay đổi được
- Cột Active: bấm chọn để tích cực OB tương ứng
- Cột Execution: có các tùy chọn None (không tác động), Once (một lần), Every minute, hour, day, week, month, year (theo chu kỳ phút, giờ, ngày, tháng, năm) end of month (cuối tháng)

- Cột Start Date và Time of Day: ghi ngày (mm.dd.yy) và giờ (hh:mm) bắt đầu gây ra ngắt. Nếu cài đặt xảy ra một lần thì ngày giờ này phải là tương lai so với giờ của PLC.

Nếu đã cài đặt các thông số xong, ta trở lại Project thêm vào khối OB ngắt tương ứng (ví dụ OB10), mở khối ra và lập trình cho khối. Các thông số và chương trình sẽ được truyền xuống PLC khi thực hiện download. Nếu trong chương trình không có khối OB tương ứng khi được gọi đến thì sẽ gây sự cố gọi OB85, nếu chưa cài OB85 thì PLC Stop.

Giờ của PLC cần phải chỉnh cho chính xác, với máy tính lập trình kết nối PLC, vào menu PLC- Diplay Accessible Nodes- MPI, sau đó chọn PLC- Set Time of Day .

Sau đây giới thiệu về cài đặt ngắt bằng chương trình. Trước hết trong cột Active ta phải tích cực OB, sau đó phải có khối OB đó trong Project; các hàm SFC 28 đến SFC 31 được sử dụng trong chương trình với các chức năng sau:

- Cài đặt thông số ngắt thời gian (SFC28 "SET_TINT")
- Hùy bỏ ngắt thời gian (SFC29 "CAN_TINT")
- Tích cực ngắt thời gian (SFC30 "ACT_TINT")
- Truy vấn (query) ngắt thời gian (SFC31 "QRY_TINT")

Trước khi khối OB ngắt thời gian được thực hiện, phải thỏa các điều kiện sau:

- Đặt ngày giờ dùng STEP 7 hay SFC28
- Tích cực dùng STEP 7 hay SFC30
- Cài đặt OB trong Project

Ngắt thời gian liên kết với khối dữ liệu địa phương

Variable	**Type**	**Description**
OB10_EV_CLASS	BYTE	Event class and identifiers: B#16#11 = interrupt is active
OB10_STRT_INFO	BYTE	B#16#11: start request for OB10 (B#16#12: start request for OB11) : : (B#16#18: start request for OB17)
OB10_PRIORITY	BYTE	Assigned priority class; default 2
OB10_OB_NUMBR	BYTE	OB number (10 to 17)
OB10_RESERVED_1	BYTE	Reserved
OB10_RESERVED_2	BYTE	Reserved
OB10_PERIOD_EXE	WORD	The OB is executed at the specified intervals: W#16#0000: once W#16#0201: once every minute W#16#0401: once hourly W#16#1001: once daily W#16#1201: once weekly W#16#1401: once monthly W#16#1801: once yearly
OB10_RESERVED_3	INT	Reserved
OB10_RESERVED_4	INT	Reserved
OB10_DATE_TIME	DATE_AND_TIME	Date and time of day when the OB was called

1.2 SFC28 "SET_TINT"

Bảng các tham số khi gọi SFC 28

Parameter	Declaration	Data Type	Memory Area	Description
OB_NR	INPUT	INT	I, Q, M, D, L, constant	Number of the OB started at the time SDT + multiple of PERIOD (OB10 to OB17).
SDT	INPUT	DT	D, L, constant	Start date and time: The seconds and milliseconds of the specified start time are ignored and set to 0.
PERIOD	INPUT	WORD	I, Q, M, D, L, constant	Periods from start point SDT onwards: W#16#0000 = once W#16#0201 = every minute W#16#0401 = hourly W#16#1001 = daily W#16#1202 = weekly W#16#1401 = monthly W#16#1801 = yearly W#16#2001 = at month's end
RET_VAL	OUTPUT	INT	I, Q, M, D, L,	If an error occurs while the function is active, the actual parameter of RET_VAL contains an error code.

Bảng giá trị trả về

Error Code (W#16#...)	Explanation
0000	No error occurred
8090	Incorrect parameter OB_NR
8091	Incorrect parameter SDT
8092	Incorrect parameter PERIOD
80A1	The set start time is in the past.

1.3 SFC29 CAN_TINT"

Parameter	Declaration	Data Type	Memory Area	Description
OB_NR	INPUT	INT	I, Q, M, D, L, constant	Number of the OB, in which the start date time will be canceled (OB10 to OB17).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is active, the actual parameter of RET_VAL contains an error code.

Bảng giá trị trả về

Error Code (W#16#...)	Explanation
0000	No error occurred.
8090	Incorrect parameter OB_NR
80A0	No start date/time specified for the time-of-day interrupt OB

1.4 SFC30 “ACT_TINT"

Parameter	Declaration	Data Type	Memory Area	Description
OB_NR	INPUT	INT	I, Q, M, D, L, constant	Number of the OB to be activated (OB10 to OB17).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is active, the actual parameter of RET_VAL contains an error code.

Bảng giá trị trả về

Error Code (W#16#...)	Explanation
0000	No error occurred.
8090	Incorrect parameter OB_NR.
80A0	Start date/time not set for the time-of-day interrupt OB.
80A1	The activated time is in the past. This error only occurs if execution=once is selected.

1.5 SFC31 "QRY_TINT"

Parameter	Declaration	Data Type	Memory Area	Description
OB_NR	INPUT	INT	I, Q, M, D, L, constant	Number of the OB, whose status will be queried (OB10 to OB17).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is active, the actual parameter of RET_VAL contains an error code.
STATUS	OUTPUT	WORD	I, Q, M, D, L	Status of the time-of-day interrupt; see following table.

STATUS

Bit	Value	Meaning
0	0	Time-of-day interrupt is enabled by operating system.
1	0	New time-of-day interrupts are accepted.
2	0	Time-of-day interrupt is not activated or has elapsed.
3	-	-
4	0	Time-of-day interrupt OB is not loaded.
5	0	The execution of the time-of-day interrupt OB is disabled by an active test function.

Error Code (W#16#...)	Explanation
0000	No error occurred.
8090	Incorrect parameter OB_NR

Ví dụ 1: Lập trình ngắt thời gian cho bài toán; từ 5:00 sáng thứ hai đến 8:00 tối thứ sáu, Q0.0 ON, các thời gian còn lại Q0.0 OFF. Báo sự cố ở Q4.1. I0.0 tích cực ngắt và I0.1 hủy ngắt.

Chương trình gồm:

OB1 gọi FC12
FC12 đặt thời điểm ngắt, tích cực ngắt, hủy ngắt
OB10 đặt/xóa Q4.0, ấn định lần ngắt kế
OB80 báo sự cố ở Q4.1
FC3 đổi ngày giờ ra dạng thích hợp
SFC20 truyền khối

Lập trình FC12

Khối biến địa phương:

Variable Name	Data Type	Declaration	Comment
IN_TIME	TIME_OF_DAY	TEMP	Start time
IN_DATE	DATE	TEMP	Start date
OUT_TIME_DATE	DATE_AND_TIME	TEMP	Start date/time converted
OK_MEMORY	BOOL	TEMP	Enable for setting time-of-day interrupt

STL (FC12)	Explanation
Network 1	
CALL SFC 31	SFC QRY_TINT
OB_NO := 10	Query STATUS of time-of-day interrupts
RET_VAL:= MW 208	
STATUS := MW 16	
Network 2:	
AN Q 4.0	Specify start time dependent on Q 4.0 (in variable
JC mond	#IN_DATE and #IN_TIME)
L D#1995–1–27	Start date is a Friday
T #IN_DATE	
L TOD#20:0:0.0	
T #IN_TIME	

JU cnvrt mond: L D#1995–1–23 T #IN_DATE L TOD#5:0:0.0 T #IN_TIME cnvrt: NOP 0	Start date is a Monday
Network 3: CALL FC 3 IN1 := #IN_DATE IN2 := #IN_TIME RET_VAL := #OUT_TIME_DATE **Network 4:** A I 0.0 AN M 17.2 A M 17.4 = #OK_MEMORY **Network 5:** A #OK_MEMORY JNB m001 CALL SFC 28 OB_NO := 10 SDT := #OUT_TIME_DATE PERIOD := W#16#1201 RET_VAL := MW 200 m001 A BR = M 202.3 **Network 6:** A #OK_MEMORY JNB m002 CALL SFC 30 OB_NO := 10 RET_VAL := MW 204 m002 A BR = M 202.4 **Network 7:** A I 0.1 JNB m003 CALL SFC 29 OB_NO := 10 RET_VAL := MW 210 m003 A BR = M 202.5	Convert format from DATE and TIME_OF_DAY to DATE_AND_TIME (for setting time-of-day interrupt) All requirements for setting time-of-day interrupt fulfilled? (Input for enable set and time-of-day interrupt not active and time-of-day interrupt OB is loaded) If so, set time-of-day interrupt... ...and activate time-of-day interrupt. If input for canceling time-of-day interrupts is set, cancel time-of-day interrupt.

Lập trình OB10

Bảng biến địa phương

Variable Name	Data Type	Declaration	Comment
STARTINFO	STRUCT	TEMP	Entire start event information of OB10 declared as structure

E_ID	WORD	TEMP	Event ID:
PR_CLASS	BYTE	TEMP	Priority class
OB_NO	BYTE	TEMP	OB number
RESERVED_1	BYTE	TEMP	Reserved
RESERVED_2	BYTE	TEMP	Reserved
PERIOD	WORD	TEMP	Periodicity of time-of-day interrupt
RESERVED_3	DWORD	TEMP	Reserved
T_STMP	STRUCT	TEMP	Structure for time-of-day details
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	
WDAY	INT	TEMP	Day of the week
IN_DATE	DATE	TEMP	Input variable for FC3 (conversion of time format)
IN_TIME	TIME_OF_DAY	TEMP	Input variable for FC3 (conversion of time format)
OUT_TIME_DATE	DATE_AND_TIME	TEMP	Output variable for FC3 and input variable for SFC28

STL (OB10)	**Explanation**
Network 1	
L #STARTINFO.T_STMP.MSEC_WDAY	Select day of week
L W#16#F	
AW	
T #WDAY	and store.
Network 2:	
L #WDAY	If day of week is not Monday,
L 2	then specify Monday, 5.00 am
<>I	as next starting time and reset
JC mond	output Q 4.0.
Network 3:	
L D#1995–1–27	
T #IN_DATE	Otherwise, if day of week is
L TOD#20:0:0.0	Monday, specify Friday, 8.00
T #IN_TIME	pm (20.00) as next starting
SET	time and set output Q 4.0.
= Q 4.0	
JU cnvrt	
mond: L D#1995–1–23	
T #IN_DATE	

L TOD#5:0:0.0 T #IN_TIME CLR = Q 4.0 cnvrt: NOP 0 **Network 4:** CALL FC 3 IN1 := #IN_DATE IN2 := #IN_TIME RET_VAL := #OUT_TIME_DATE **Network 5:** CALL SFC 28 OB_NO := 10 SDT := #OUT_TIME_DATE PERIOD := W#16#1201 RET_VAL := MW 200 A BR = M 202.1	Starting time specified. Convert specified starting time to format DATE_AND_TIME (for SFC28). Set time-of-day interrupt.
Network 6: CALL SFC 30 OB_NO := 10 RET_VAL := MW 204 A BR = M 202.2 **Network 7:** CALL SFC 20 SRCBLK := #STARTINFO.T_STMP RET_VAL := MW 206 DSTBLK := P#M 100.0 BYTE 8	Activate time-of-day interrupt. Block transfer: save time of day from start event information of OB10 to the memory area MB100 to MB107.

Lập trình OB1

CALL FC 12 Calls the function FC12

Lập trình OB80

Variable Name	Data Type	Declaration	Comment
STARTINFO	STRUCT	TEMP	Entire start event information of OB80 declared as structure
E_ID	WORD	TEMP	Event ID:
PR_CLASS	BYTE	TEMP	Priority class
OB_NO	BYTE	TEMP	OB number
RESERVED_1	BYTE	TEMP	Reserved
RESERVED_2	BYTE	TEMP	Reserved
A1_INFO	WORD	TEMP	Additional information about the event that caused the error
A2_INFO	DWORD	TEMP	Additional information about the event ID, priority class, and OB no. of the error

T_STMP	STRUCT	TEMP	Structure for time-of-day details
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	

STL (OB80)	**Explanation**
Network 1 AN Q 4.1 S Q 4.1 CALL SFC 20 SRCBLK := #STARTINFO RET_VAL := MW 210 DSTBLK := P#M 110.0 Byte 20	Set output Q 4.1 if time error occurred. Block transfer: save entire start event information to memory area MB110 to MB129.

2/ Ngắt trễ

Có bốn ngắt thời trễ OB20..OB23 được kích hoạt bởi hàm SFC32 (SRT_DINT). Sau khi gọi SFC32 một thời gian, OB tương ứng sẽ hoạt động. Khi chưa đến thời điểm kích hoạt có thể hủy OB bằng SFC33 (CAN_DINT).

CPU 312	Không
CPU 313..316	OB20
CPU 318, 412, 413	OB20, OB21
CPU 414, 416, 417	OB20..23

2,1/ SFC 32: khởi động ngắt

Parameter	**Declaration**	**Data Type**	**Memory Area**	**Description**
OB_NR	INPUT	INT	I, Q, M, D, L, constant	Number of the OB, to be started after a time delay (OB20 to OB23).
DTIME	INPUT	TIME	I, Q, M, D, L, constant	Length of the delay (1 to 60000 ms)
SIGN	INPUT	WORD	I, Q, M, D, L, constant	Identifier that is entered in the start event information of the OB when the time-delay OB is called.
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is active, the
				actual parameter of RET_VAL contains an error code.

Error Code (W#16#...)	Explanation
0000	No error occurred.
8090	Incorrect parameter OB_NR
8091	Incorrect parameter DTIME

2.2/ Truy vấn ngắt trì hoãn SFC34 "QRY_DINT"

Parameter	Declaration	Data	Memory Area	Description
		Type		
OB_NR	INPUT	INT	I, Q, M, D, L,	Number of the OB, whose STATUS will be
			constant	queried (OB20 to OB23).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is being
				processed, the actual parameter of RET_VAL
				contains an error code.
STATUS	OUTPUT	WORD	I, Q, M, D, L	Status of the time-delay interrupt, see following
				table.

OUTPUT STATUS

Bit	Value	Meaning
0	0	Time-delay interrupt is enabled by the operating system.
1	0	New time-delay interrupts are not rejected.
2	0	Time-delay interrupt is not activated or has elapsed.
3	-	-
4	0	Time-delay interrupt-OB is not loaded.
5	0	The execution of the time-delay interrupt OB is disabled by an active test
		function.

ERROR RET_VAL

Error Code (W#16#...)	Explanation
0000	No error occurred
8090	Incorrect parameter OB_NR

2.3/ Triệt tiêu ngắt trì hoãn SFC33 "CAN_DINT"

Parameter	Declaration	Data Type	Memory Area	Description
OB_NR	INPUT	INT	I, Q, M, D, L,	Number of the OB to be canceled (OB20 to
			constant	OB23).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs while the function is active,
				the actual parameter of RET_VAL

				contains an
				error code.

Error Code (W#16#...)	Explanation
0000	No error has occurred.
8090	Incorrect parameter OB_NR
80A0	Time-delay interrupt has not started.

Bảng biến địa phương của OB20

Variable	Type	Description
OB20_EV_CLASS	BYTE	Event class and identifiers: B#16#11: interrupt is active
OB20_STRT_INF	BYTE	B#16#21: start request for OB20 (B#16#22: start request for OB21) (B#16#23: start request for OB22) (B#16#24: start request for OB23)
OB20_PRIORITY	BYTE	Assigned priority class: default values 3 (OB20) to 6 (OB23)
OB20_OB_NUMBR	BYTE	OB number (20 to 23)
OB20_RESERVED_1	BYTE	Reserved
OB20_RESERVED_2	BYTE	Reserved
OB20_SIGN	WORD	User ID: input parameter SIGN from the call for SFC32 (SRT_DINT)
OB20_DTIME	TIME	Elapsed delay time in ms
OB20_DATE_TIME	DATE_AND_ TIME	Date and time of day when the OB was called

Ví dụ 2: Mỗi khi I0.0 ON thì 10 s sau Q0.0 ON. I0.2 ON thì reset Q0.0. Nếu trong khoảng delay mà I0.1 ON thì Q0.0 vẫn OFF

Bảng ký hiệu:

Address	Meaning
I0.0	Input to enable "start time-delay interrupt"
I0.1	Input to cancel a time-delay interrupt
I0.2	Input to reset output Q 4.0
Q4.0	Output set by the time-delay interrupt OB (OB20)
MB1	Used for edge flag and binary result (status bit BR) buffer for SFCs
MW4	STATUS of time-delay interrupt (SFC34 "QRY_TINT")
MD10	Seconds and milliseconds BCD-coded from the start event information of OB1
MW 100	RET_VAL of SFC32 "SRT_DINT"
MW102	RET_VAL of SFC34 "QRY_DINT"
MW104	RET_VAL of SFC33 "CAN_DINT"

MW106	RET_VAL of SFC20 "BLKMOV"
MB120 to MB139	Memory for start event information of OB20
MD140	Seconds and milliseconds BCD-coded from the start event information of OB20
MW144	Seconds and milliseconds BCD-coded from the start event information of OB1; acquired from start event information of OB20 (user-specific ID SIGN)

Bảng biến địa phương của OB20

Variable Name	**Data Type**	**Declaration**	**Comment**
STARTINFO	STRUCT	TEMP	Start information for OB20
E_ID	WORD	TEMP	Event ID:
PC_NO	BYTE	TEMP	Priority class
OB_NO	BYTE	TEMP	OB number
D_ID 1	BYTE	TEMP	Data ID 1
D_ID 2	BYTE	TEMP	Data ID 2
SIGN	WORD	TEMP	User-specific ID
DTIME	TIME	TEMP	Time with which the time-delay interrupt is started
T_STMP	STRUCT	TEMP	Structure for time-of-day details (time stamp)
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	

STL (OB20)	**Explanation**
Network 1 SET = Q 4.0	Set output Q 4.0 unconditionally
Network 2: L QW 4 T PQW 4	Activate output word immediately
Network 3: L #STARTINFO.T_STMP.SECONDS T MW 140 L #STARTINFO.T_STMP.MSEC_WDAY	Read seconds from start event information Read milliseconds and day of week from start event information

T MW 142 L MD 140 SRD 4 T MD 140 Network 4: L #STARTINFO.SIGN T MW 144 Network 5: CALL SFC 20 SRCBLK := STARTINFO RET_VAL := MW 106 DSTBLK := P#M 120.0 Byte 20	Eliminate day of week and write milliseconds back (now BCD-coded in MW 142) Read starting time of time-delay interrupt (= call SFC32) from start event information Copy start event information to memory area (MB120 to MB139)

Bảng biến địa phương của OB1

Variable Name	**Data Type**	**Declaration**	***Comment***
STARTINFO	STRUCT	TEMP	*Start information for OB1*
E_ID	WORD	TEMP	*Event ID:*
PC_NO	BYTE	TEMP	*Priority class*
OB_NO	BYTE	TEMP	*OB number*
D_ID 1	BYTE	TEMP	*Data ID 1*
D_ID 2	BYTE	TEMP	*Data ID 2*
CUR_CYC	INT	TEMP	*Current cycle time*
MIN_CYC	INT	TEMP	*Minimum cycle time*
MAX_CYC	INT	TEMP	*Maximum cycle time*
T_STMP	STRUCT	TEMP	*Structure for time-of-day details (time stamp)*
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	*TEMP*	

STL (OB1)	**Explanation**
Network 1 L #STARTINFO.T_STMP.SECONDS T MW 10 L #STARTINFO.T_STMP.MSEC_WDAY T MW 12 L MD 10	 Read seconds from start event information Read milliseconds and day of week from start event information Eliminate day of week and write milliseconds back (now BCD-coded

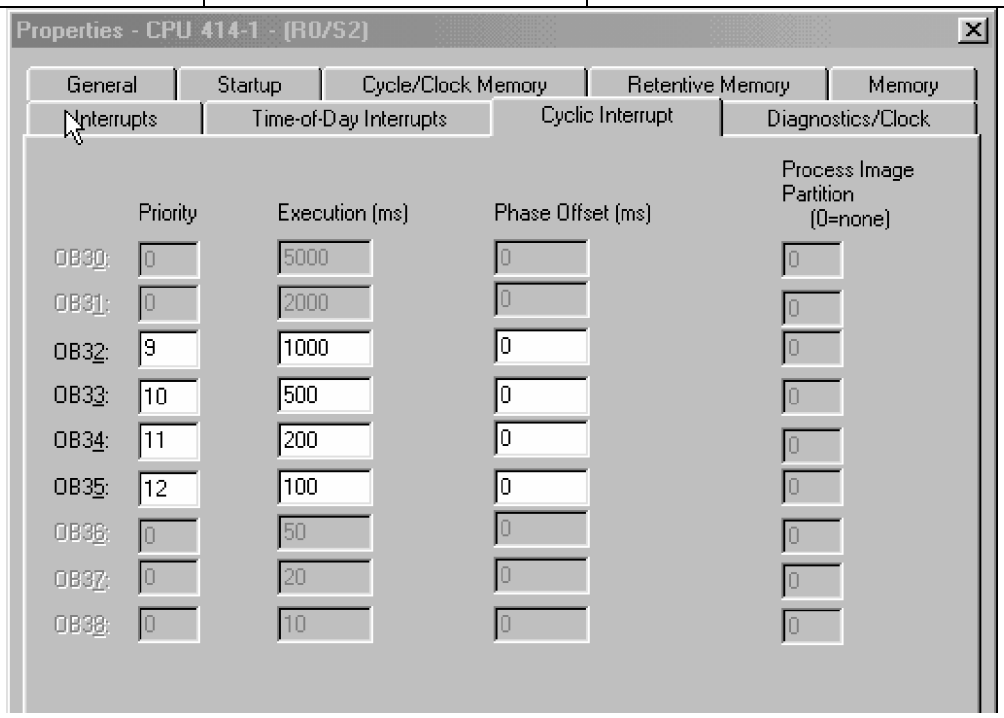
SRD 4	in MW 12)
T MD 10	Positive edge at input I 0.0?
Network 2:	
A I 0.0	
FP M 1.0	
= M 1.1	If so, start time-delay interrupt (starting time of time-delay interrupt assigned to the parameter SIGN)
Network 3:	
A M 1.1	
JNB m001	
CALL SFC 32	
OB_NO := 20	
DTME := T#10S	
SIGN := MW 12	
RET_VAL:= MW 100	
m001: NOP 0	Query status of time-delay interrupt (SFC QRY_DINT)
Network 4:	
CALL SFC 34	
OB_NO := 20	
RET_VAL:= MW 102	Positive edge at input I 0.1?
STATUS := MW 4	
Network 5:	
A I 0.1	
FP M 1.3	
= M 1.4	...and time-delay interrupt is activated (bit 2 of time-delay interrupt STATUS)?
Network 6:	Then cancel time-delay interrupt
A M 1.4	
A M 5.2	
JNB m002	
CALL SFC 33	Reset output Q 4.0 with input I 0.2
OB_NO := 20	
RET_VAL:= MW 104	
m002: NOP 0	
A I 0.2	
R Q 4.0	

3 . NGẮT CHU KỲ

Ngắt chu kỳ OB30..OB38 được gọi đến theo chu kỳ tuần hoàn. Thời gian thực hiện mỗi ngắt chu kỳ OB phải nhỏ hơn nhiều chu kỳ ngắt, nếu không OB80 sẽ được gọi.

Chu kỳ ngắt được xác định bởi khoảng (interval) là số nguyên chỉ chu kỳ(đơn vị ms) và lệch pha (phase offset) là thời gian trễ m (đơn vị ms), 0<= m < n, gọi OB ngắt chu kỳ khi đến thời điểm ấn định. Dùng phase offset để tránh các OB ngắt cùng được gọi đồng thời . Hai giá trị này được cài đặt bằng STEP 7 và có giá trị mặc định như bảng sau:

OB Number	**Default Interval**	**Default Priority Class**
OB30	5 s	7
OB31	2 s	8
OB32	1 s	9
OB33	500 ms	10
OB34	200 ms	11
OB35	100 ms	12
OB36	50 ms	13
OB37	20 ms	14
OB38	10 ms	15



Số các OB ngắt chu kỳ phụ thuộc CPU

CPU 312	Không có
CPU 313… 316	OB35
CPU 318, 412, 413	OB32, OB35
CPU 414	OB32..OB35
CPU 416, 417	OB30..OB38

Bảng biến địa phương

Variable	*Type*	**Description**
OB35_EV_CLASS	BYTE	Event class and identifiers B#16#11: interrupt is active
OB35_STRT_INF	BYTE	(B#16#31: start request for OB30) ,,, B#16#36: start request for OB35 (B#16#39: start request for OB38)
OB35_PRIORITY	BYTE	Assigned priority class: defaults 7 (OB30) to 15 (OB38)
OB35_OB_NUMBR	BYTE	OB number (30 to 38)
OB35_RESERVED_1	BYTE	Reserved
OB35_RESERVED_2	BYTE	Reserved
OB35_PHASE_OFFSET	WORD	Phase offset [ms]
OB35_RESERVED_3	INT	Reserved
OB35_EXC_FREQ	INT	Interval in milliseconds
OB35_DATE_TIME	DATE_AND_TIME	Date and time of day when the OB was called

Tài liệu tham khảo

1.Tự động hoá với Simatic S7-200. Nhà xuất bản nông nghiệp,1997- Doãn Minh Phước, Phan Xuân Minh.

2.S5-95U và phần mềm Step5. Giáo trình giảng dạy của trung tâm đào tạo Simens tự động hoá trường ĐHBK Hà nội, 1997- Doãn Minh Phước, Phan Xuân Minh.

3.SPS-Grundkurs, Volgel Buchverlag- Juergen Kaftan.

4.Speicherprogrammierte Steuerungen Aufgaben mit Loesungen, Europa-Fachbuchreihe.

5.Tự động hoá với Simatic S7-300. Nhà xuất bản khoa học và kỹ thuật, 2000-Doãn Minh Phước, Phan Xuân Minh, Vũ Văn Hà .