Bài giảng Thiết bị Siemens
S7-300

Nội dung bài giảng:

Giới thiệu sơ lược về dòng sản phẩm của Siemens:

Logo: Dòng sản phẩm sơ cấp ,được sử dụng cho những ứng dụng nhỏ,có tác dụng thay thế cho những ứng dụng sử dụng nhiều rơle trung gian,timer....,nhằm giảm không gian lắp đặt tủ điện.

Do logo chỉ là những Logic Modul do vậy chỉ được sử dụng cho việc thay thế những mạch số đơn giản với số lượng I/O nhỏ 24In,16Out

Ưu điểm của Logo là dễ dàng sử dụng,dễ dàng lập trình và thay đổi dữ liệu .Có thể lập trình trực tiếp trên Logo bằng cách sử dụng Logo có màn hình.Giá thành tương đối hợp lí.

Ứng dụng : Chủ yếu trong hệ thống chiếu sáng công cộng,hệ thống chiếu sáng trong toà nhà,sử dụng trong các máy xà gồ,và một số máy đơn giản.....

S7_200: Dòng sản phẩm trung cấp,được sử dụng trong những ứng dụng trung bình với số lượng I/O vừa phải (khoảng 128),đối với dòng sản phẩm S7_200 này đã được tích hợp đầy đủ những hàm toán cho tất cả những ứng dụng cần thiết cho mọi hệ thống tự động,ngôn ngữ cũng như giao diện lập trình dễ hiểu ,thân thiện,giúp cho mọi người đều có thể dễ dàng tiếp cận.Tuy nhiên,thông thường S7-200 vẫn được sử dụng cho những ứng dụng riêng lẻ,còn trường hợp muốn mở rộng mạng thì vẫn nên sử dụng S7_300

Ứng dụng : Trong các nghành đá,Bê tông ,Gốm sứ,Ximăng,sắt thép.....Có thể sử dụng cho hệ thống SCADA nhỏ (kết nối S7_200 với máy tính thông qua PC Access,để có thể truy cập và quản lí dữ liệu : Trạm trộn Bê Tông....)

S7_300,400: Dòng sản phẩm cao cấp ,được dùng cho những ứng dụng lớn với những yêu cầu I/O nhiều và thời gian đáp ứng nhanh,yêu cầu kết nối mạng,và có khả năng mở rộng cho sau này.

Ngôn ngữ lập trình đa dạng cho phép người sử dụng có quyền chọn lựa.Đặc điểm nổi bật của S7_300 đó là ngôn ngữ lập trình cung cấp những hàm toán đa dạng cho những yêu cầu chuyên biệt như : Hàm SCALE.....

Hoặc ta có thể sử dụng ngôn ngữ chuyên biệt để xây dựng hàm riêng cho ứng dụng mà ta cần.

Ngoài ra S7-300 còn xây dựng phần cứng theo cấu trúc Modul,nghĩa là đối với S7-300 sẽ có những Modul tích hợp cho những ứng dụng đặc biệt như Modul PID,Modul Đọc xung tốc độ cao....

Màn Hình : Siemens còn cung cấp tất cả các loại màn hình ứng dụng trong công nghiệp như màn hình màu,màn hình đen trắng,màn hình máy tính công nghiệp.....

Các màn hình này có thể kết nối với các loại PLC để có thể dễ dàng thay đổi dữ liệu ,hoặc có thể kết nối thành mạng ProfiBus.

Dòng C7: Dòng C7 về nguyên tắc nó được xem như là sự kết hợp giữa PLC và màn hình,tức là ta có thể hiểu C7 là màn hình có thể kết hợp với I/O cho những ứng dụng trong công nghiệp,được kết nối theo mạng Profibus.

I/Đại số Boolean:

1/ Biến và hàm số 2 trị:

Khi mô tả đối tượng bằng mô hình toán học ta phải biểu diễn các đại lượng vào ra của đối tượng,các đại lượng này là các hàm phụ thuộc theo thời gian.

Biến hai trị hay còn gọi là biến Boolean là loại hàm số mà miền giá trị của nó chỉ có 2 phần tử,hai phần tử đó là 0 và 1

Vd: Công tắc là một biến ngõ vào 2 trị : đóng (kí hiệu là 1) và mở (kí hiệu là 0)

Đèn hiệu là một biến ngõ ra 2 trị : sáng (kí hiệu là 1) và tắt (kí hiệu là 0)

Hai biến được gọi là độc lập nhau nếu sự thay đổi của biến này không ảnh hưởng đến biến kia.

2/Các phép toán trên hàm 2 trị:

a/Phép Not: x = not(y)

y	X
0	1
1	0

b/Phép hợp (phép cộng): x= y+z

y	z	X
0	0	0
0	1	1
1	0	1
1	1	1

c/Phép giao : x=y^z

y	z	X
0	0	0
0	1	0
1	0	0
1	1	1

- Chuyển đổi từ mạch rơle nút nhấn ,đèn sang viết chương trình PLC

II/Biểu diễn số nguyên dương:

1/Trong hệ cơ số 10 (hệ thập phân):

Một số nguyên dương U_n bất kì,trong hệ cơ số 10 bao giờ cũng được biểu diễn đầy đủ bằng dãy con số nguyên từ 0 đến 9

Vd: U_n = 349 được biểu diễn trong hệ cơ số 10:

$349 = 3*10^2 + 4*10^1 + 9*10^0$.

2/Trong hệ cơ số 2 (hệ nhị phân):

Cách biểu diễn U_n trong hệ cơ số 10 chưa phù hợp với nguyên tắc mạch điện (hay nguyên tắc hàm 2 trị).Để sử dụng nguyên tắc hàm 2 trị (số 0 hoặc 1) ta đưa ra 1 khái niệm ***Bit***

1 Bit bao gồm 2 trị 0 hoặc 1,do vậy 1 số có thể biểu diễn trong hệ nhị phân dưới dạng ***Bit***

vd: U_n = 205 có thể được biểu diễn dưới dạng 8 Bit:

1 1 0 0 1 1 0 1

$205 = 1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

3 / Trong hệ cơ số 16 (hệ Hexadecimal):

Cũng giống như hệ cơ số 10,1 nguyên dương bất kì cũng có thể biểu diễn trong hệ cơ số 16 như sau:

Vd: 7723 = 1E2B

$= 1* 16^3 + 14* 16^2 + 2* 16^1 + 11*16^0$

E B

4/ Mã BCD của số nguyên dương:

Vd: số 259 dưới dạng mã BCD:

0010 0101 1001

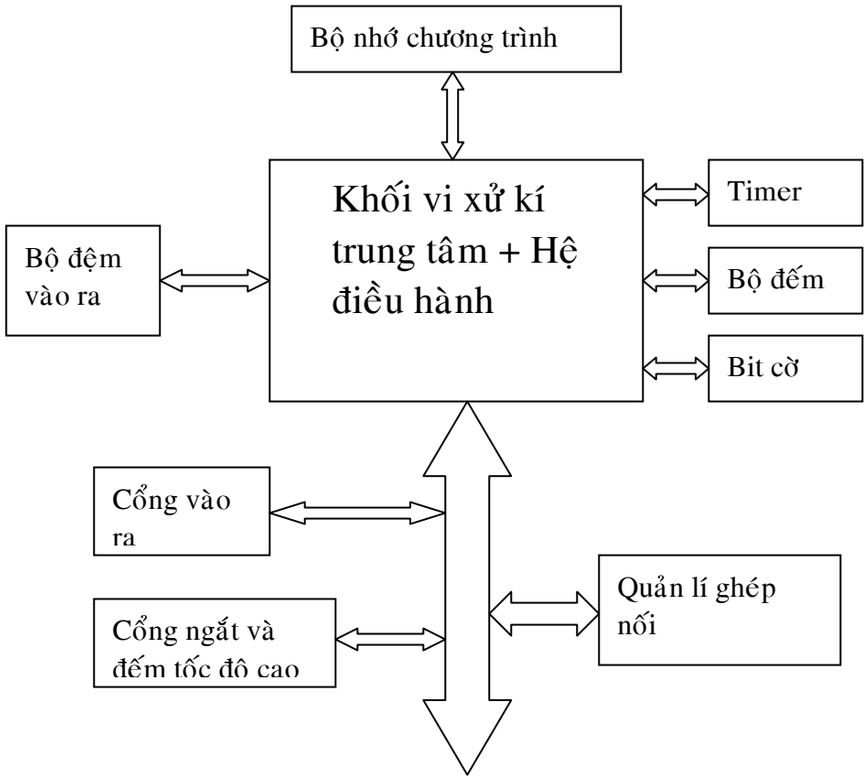
2 5 9

III/ Thiết bị điều khiển Logic khả trình:

1/ PLC (Progranable Logic Control) : Thiết bị điều khiển Logic khả trình PLC

Là loại thiết bị cho phép thực hiện linh hoạt các thuật toán điều khiển số thông qua một ngôn ngữ lập trình ,thay cho việc phải thể hiện thuật toán đó bằng mạch số .Như vậy với chương trình điều khiển trong mình ,PLC trở thành bộ điều khiển nhỏ gọn ,dễ thay đổi thuật toán và đặc biệt dễ trao đổi thông tin với môi trường xung quanh (với các PLC khác hoặc với máy tính).Toàn bộ chương trình được lưu trong bộ nhớ.dưới dạng các khối chương trình (OB,FC,FB..) và được thực hiện với chu kì quét.

Để có thể thực hiện một chương trình điều khiển.Tất nhiên PLC phải có tính năng như một máy tính .Nghĩa là phải có một bộ vi xử lí trung tâm (CPU),một hệ điều hành,một bộ nhớ chương trình để lưu chương trình cũng như dữ liệu và tất nhiên phải có các cổng vào rađể giao tiếp với các thiết bị bên ngoài..Bên cạnh đó ,nhằm phục vụ bài toán điều khiển số ,PLC phải có các khối hàm chức năng như Timer,Counter,và các hàm chức năng đặc biệt khác.



2/ Các Tín hiệu kết nối với PLC:

a/Tín hiệu số : Là các tín hiệu thuộc dạng hàm Boolean, dạng tín hiệu chỉ có 2 trị 0 hoặc 1.

Đối với PLC Siemens :

Mức 0 : tương ứng với 0V hoặc hở mạch

Mức 1 : Tương ứng với 24V

Vd: Các tín hiệu từ nút nhấn ,từ các công tắc hành trình..... đều là những tín hiệu số

b/ Tín hiệu tương tự : Là tín hiệu liên tục, từ 0-10V hay từ 4-20mA....

Vd: Tín hiệu đọc từ Loadcell,từ cảm biến lưu lượng...

c/ Tín hiệu khác : Bao gồm các tín hiệu giao tiếp với máy tính ,với các thiết bị ngoại vi khác bằng các giao thức khác nhau như giao thức RS232,RS485,Modbus....

3/ Các Modul của PLC S7_300:

Thông thường để tăng tính mềm dẻo trong ứng dụng thực tế mà ở đó phần lớn các đối tượng điều khiển có số tín hiệu đầu vào đầu ra cũng như chủng loại tín hiệu vào ra khác nhau mà các bộ điều khiển PLC được thiết kế không bị cứng hoá về cấu hình .Chúng được chia nhỏ thành các Modul.số các Modul được sử dụng nhiều hay ít tuỳ theo từng bài toán ,song tối thiểu bao giờ cũng có Modul chính là Modul CPU. Các Modul còn lại là những Modul nhận truyền tín hiệu với đối tượng điều khiển ,các Modul chức năng chuyên dụng như PID,điều khiển động cơ..... Chúng được gọi chung là Modul mở rộng.Tất cả các Modul được gắn trên những thanh ray (Rack).

a/ Modul CPU: Modul CPU là loại Modul chứa vi xử lí,hệ điều hành ,bộ nhớ ,các bộ thời gian ,bộ đếm,cổng truyền thông (RS485).... Và có thể còn có một vài cổng vào ra số.Các cổng vào ra số trên CPU được gọi là cổng vào ra Onboard.

Trong họ PLC S7_300 có nhiều loại CPU khác nhau : CPU 312,CPU 314 ,CPU 315....

Những Modul cùng sử dụng một loại bộ vi xử lí ,nhưng khác nhau về cổng vào ra onboard cũng như các khối hàm đặc biệttích hợp sẵn trong thư viện của hệ điều hành phục vụ việc sử dụng các cổng vào ra onboard này sẽ được phân biệt với nhau tong tên gọi bằng tên cụm chữ cái IFM(viết tắt của Intergrated Function Module).Ví dụ Module CPU 312IFM,Modul314 IFM....

Ngoìa ra còn có các loại module hai cổng truyền thông,trong đó cổng truyền thông thứ 2 có chức năng chính là phục vụ việc nối mạng phân tán .Các loại module CPU được phân biệt với những loại CPU khác bằng thêm cụm từ DP (Distrubited port)trong tên gọi. Ví dụ module CPU 315-DP

b/Các Modul mở rộng:

Các Modul mở rộng được chia thành 5 loại chính:

i/ PS(Power Supply): Modul nguồn nuôi

ii/ SM (Signal Module): Modul tín hiệu vào ra bao gồm:

DI (Digital Input)

DO (Digital Output)

DI/DO (Digital In/Output)

AI (Analog Input)

AO (Analog Output)

AI/AO (Analog In/Output)

iii / 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 từng một khối và được quản lí chung bởi một module 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 module mở rộng (không kể module CPU ,module nguồn nuôi). Một module CPU có thể làm việc trực tiếp với nhiều nhất 4 Rack,và các Rack này phải được nối với nhau bằng Module IM

IM360 :truyền IM361:nhận.

- FM (Function Module): Các Modul điều khiển riêng,như điều khiển Servo,điều khiển PID.....
- CP(Communication Module):Module truyền thông

2/ Tín Hiệu: Thông thường có 2 tín hiệu

Tín hiệu số: Tín hiệu mức 1 hoặc mức 0 (true hoặc False),

Vd: I0.0,Q0.0....

Tín hiệu tương tự : Là tín hiệu analog được đọc từ các Modul analog

Vd : PIW256....

3/Kiểu dữ liệu và phân chia bộ nhớ:

a/Kiểu Bool: True hoặc False (0 hoặc 1)

VD: M0.0

b/Kiểu Byte : gồm 8 Bit

c/Kiểu Word

d/Kiểu Int
e/Kiểu Dint
f/ Kiểu Real
h/Kiểu S5T:
k/ Kiểu Char
i/Kiểu Date
j/Kiểu Tod:

4/ Bộ nhớ PLC: gồm 3 vùng chính.

a/Vùng chứa chương trình ứng dụng : Vùng chứa chương trình được chia thành 3 miền :

i/ OB (Organisation block) : miền chứa chương trình tổ chức.

ii/ FC (Function) : Miền chứa chương trình con ,được tổ chức thành hàm và có biến hình thức để trao đổi dữ liệu

iii/ FB (Function block) : Miền chứa chương trình con ,được tổ chức thành hàmvà có khả năng trao đổi dữ liệu với bất cứ 1 khối chương trình nào khác .Các dữ liệu này phải được xây dựng thành một khối dữ liệu riêng (Data Block khối DB)

b/ Vùng chứa tham số của hệ điều hành: Chia thành 7 miền khác nhau

I (Process image input) : Miền dữ liệu các cổng vào số,trước khi bắt đầu thực hiện chương trình ,PLC sẽ đọc giá trị logic của tất cả các cổng 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.

Q (Process Image Output): Miền bộ đệm các dữ liệu cổng ra số .Kết thúc giai đoạn thực hiện chương trình,PLC sẽ chuyển giá trị logic của bộ đệm Q tới các cổng ra số.Thông thường chương trình không trực tiếp gán giá trị tới tận cổng ra mà chỉ chuyển chúng tới bộ đệm Q.

M (Miền các biến cờ): CHương trình ứng dụng sử dụng những biến này để lưu giữ các tham số cần thiết và có thể truy nhập nó theo Bit (M) ,byte (MB),từ (MW) hay từ kép (MD).

T (Timer): Miền nhớ phục vụ bộ thời gian (Timer) bao gồm việc lưu trữ giá trị thời gian đặt trước (PV-Preset Value),giá trị đếm thời gian tức thời (CV –Current Value) cũng như giá trị Logic đầu ra của bộ thời gian.

C (Counter): Miền nhớ phục vụ bộ đếm bao gồm việc lưu trữ giá trị đặt trước (PV- Preset Value),giá trị đếm tức thời (CV _ Current Value)và giá trị logic đầu ra của bộ đệm.

PI : Miền địa chỉ cổng vào của các Modul tương tự (I/O External input). Các giá trị tương tự tại cổng vào của modul tương tự sẽ được module đọc và chuyển tự động theo những địa chỉ.Chương trình ứng dụng có thể truy cập miền nhớ PI theo từng Byte (PIB),từng từ PIW hoặc từng từ kép PID .

PQ: Miền địa chỉ cổng ra cho các module tương tự (I/O External Output).Các giá trị theo những địa chỉ này sẽ được module tương tự chuyển tới các cổng ra tượng tự .Chương trình ứng dụng có thể truy nhập miền nhớ PQ theo từng Byte (PQB),từng từ (PQW) hoặc theo từng từ kép (PQD).

c/ Vùng chứa các khối dữ liệu: được chia làm 2 loại:

DB(Data Block):Miền chứa dữ liệu được tổ chức thành khối .Kích thước cũng như số lượng khối do người sử dụng quy định ,phù hợp với từng bài toán điều khiển.Chương trình có thể truy nhập miền này theo từng bit (DBX),byte (DBB),từ (DBW) hoặc từ kép (DBD).

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

5/ Vòng quét chương trình:

PLC thực hiện chương trình theo chu kì lặp .Mỗi vòng lặp được gọi là vòng quét (Scan) .Mỗi vòng quét được bắt đầu bằng giai đoạn 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 thực hiện từ lệnh đầu tiên đến lệnh kết thúc của khối OB (Block End).Sau giai đoạn thực hiện chương trình là giai đoạn chuyển các nội dung của bộ đệm ảo Qtới các cổng ra số .Vòng quét được kết thúc bằng giai đoạn truyền thông nội bộvà kiểm tra lỗi.

Chú ý rằng bộ đệm I và Q không liên quan tới các cổng vào ra tương tự nên các lệng truy nhập cổng tương tự được thực hiện trực tiếp với cổng vật lí chứ không thông qua bộ đệm.

Thời gian cần thiết để PLC thực hiện 1 vòng quét 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 và khối dữ liệu truyền thông trong vòng quét đó.

Như vậy giữa việc đọc dữ liệu từ đối tượng để xử lí ,tính toán và việc gởi 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 như 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 được thực hiện tại mọi điểm trong vòng quét chứ không bị gò ép là phải ở trong giai đoạn thực hiện chương trình.Chẳng hạn nếu 1 tín hiệu báo ngắt xuất hiện khi PLC đang ở giai đoạn truyền thông và kiểm tra nội bộ,PLC sẽ ngừng công việc truyền thông ,kiểm tra để thực hiện khối chương trình tương ứng với tín hiệu báo ngắt đó .Với hình thức xử lí tín hiệu 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 lí 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 đệm ảo của cổng trong vùng nhớ tham số.Việc truyền thông giữa bộ đệm ảo với ngoại vi trong các giai đoạn 1 và 3 do hệ điều hành CPU quản lí .Ở 1 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 lí ngắt,để thực hiện lệnh trực tiếp với cổng vào ra.

6 / Cấu trúc chương trình:

Chương trình trong S7_300 được lưu trong bộ nhớ của PLC ở vùng giành riêng cho chương trình và có thể được lập với 2 dạng cấu trúc khác nhau.

a/ Lập trình tuyến tính: toàn bộ chương trình nằm trong một khối trong bộ nhớ .Loại hình cấu trúc tuyến tính này phù hợp với những bài toán tự động nhỏ,không phức tạp .Khối được chọn phải là khối OB1 ,là khối mà PLC luôn quét và thực hiện các lệnh trong đó thường xuyên,từ lệnh đầu tiên đến lệnh cuối cùng và quay lại lệnh đầu tiên.

b/ Lập trình có cấu trúc: Chương trình được chia thành những phần nhỏ và mỗi phần thực thi những nhiệm vụ chuyên biệt riêng của no,từng phần này nằm trong những khối chương trình khác nhau .Loại hình cấu trúc này phù hợp với những bài toán điều khiển nhiều nhiệm vụ và phức tạp .PLC S7_300 có 4 loại khối cơ bản sau:

- Loại khối OB (Organization Block) : Khối tổ chức và quản lí chương trình điều khiển .Có nhiều loại khối OB với những chức năng khác nhau ,chúng được phân biệt với nhau bằng một số nguyên đi sau nhóm kí tự OB.

  Ví dụ: OB1,OB35,OB40,OB80,.....
- Loại khối FC (Program block): Khối chương trình với những chức năng riêng giống như 1 chương trình con hoặc một hàm (chương trình con có biến hình thức).Một chương trình ứng dụng có thể có nhiều khối FC và các khối FC này được phân biệt với nhau bằng một số nguyên sau nhóm kí tự FC.

  Ví dụ: FC1,FC2....
- Loại khối FB (Function Block) :Là loại khối FC đặc biệt có khả năng trao đổi 1 lượng dữ liệu lớn với các khối chương trình khác .Các dữ liệu này phhải được tổ chức thành khối dữ liệu riêng có tên gọi là Data block.Một chương trình ứng dụng có thể có nhiều khối FB và các khối Fb này được phân biệt với nhau bằng một số nguyên sau nhóm kí tự FB.Chẳng hạn như FB1,FB2...
- Loại khối DB (Data Block) : Khối chứa các dữ liệu cần thiết để thực hiện chương trình .Các tham số của khối do người dùng tự đặt .Một chương trình ứng dụng có thể có nhiều khối DB và các khối DB này được phân biệt với nhau bằng một số nguyên sau nhóm kí tự DB

  Ví dụ: DB1,DB2....

Chương trình trong các khối được liên kết với nhau bằng các lệng gọi khối ,chuyển khối.Xem những phần chương trình trong các khối như là các chương trình con thì S7_300 cho phép gọi chương trình con lồng nhau ,tức là chương trình con này gọi một chương trình con khác và từ một chương trình con được gọi lại gọi tới một chương trình con thứ 3 ... Số các lệnh gọi lồng nhau phụ thuộc vào từng chủng loại module CPU mà ta đang sử dụng. Ví dụ đối với module CPU 314 thì số lệnh gọi lồng nhau nhiều nhất có thể cho phép là 8.Nếu số lần gọi khối lồng nhau mà vượt quá con số giới hạn cho phép ,PLC sẽ tự chuyển qua chế độ Stop và đặt cờ báo lỗi.

7/ Các khối OB đặc biệt:

Trong khi khối OB được thực hiện đều đặn ở từng vòng quét trong giai đoạn thực hiện chương trình thì các khối OB khác chỉ được thực hiện khi xuất hiện tín hiệu báo ngắt tương ứng ,nói cách khác chương trình viết cho các khối OB này chính là chương trình xử lí tín hiệu ngắt (event).Chúng bao gồm:

OB10 (Time of Day Interrupt):Chương trình trong khối sẽ được thực hiện khi giá trị của đồng hồ thời gian thực nằm trong một khoảng thời gian đã được quy định.OB10 có thể gọi một lần ,nhiều lần cách đều nhau từng phút, từng giơ,từng ngày ....Việc quy định khoảng thời gian hay số lần gọi OB10 được thực hiện nhờ chương trình hệ thống SFC28 hoặc trong bảng tham số của module CPU nhờ phần mềm Step 7.

OB20 (Time Day Interrupt): Chương trình trong khối sẽ được thực hiện sau một khoảng thời gian trễ đặt trước kể từ khi gọi chương trình hệ thống SFC32 để đặt thời gian trễ.

OB35 (Cyclic Interrupt): Chương trình trong OB35 sẽ được thực hiện cách đều nhau 1 khoảng thời gian cố định.Mặc định khoảng thời gian này sẽ là 100ms,xong ta có thể thay đổi nó trong bảng tham số của module CPU ,nhờ phần mềm Step7.

OB40 (Hardware Interrupt) : Chương trình trong OB sẽ được thực hiện khi xuất hiện 1 tín hiệu báo ngắt từ ngoại vi đưa vào module CPU thông qua các cổng vào ra số onboard đặc biệt,hoặc thông qua các module SM,CP,FM

OB80 (Cycle Time Fault): Chương trình trong khối OB80 sẽ được thực hiện khi thời gian vòng quét(Scan time) vượt quá khoảng thời gian cực đại đã được quy 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 thời gian Scan time cực đại là 150ms ,nhưng có thể thay đổi nó thông qua bảng tham số của module CPU nhờ phần mềm Step 7.

OB81 (Power Supply fault): CPU sẽ gọi chương trình trong khối OB81 khi phát hiện thấy có lỗi về nguồn nuôi.

OB82(Diagnostic Interrupt):Chương trình trong OB82 được gọi khi CPU phát hiện sự cố từ các Modul vào ra

OB85(Not Load fault):Chương trình trong OB82 được gọi khi CPU phát hiện thấy chương trình ứng dụng có sử dụng chế độ ngắt nhưng chương trình sử lí tín hiệu ngắt lại không có trong khối OB tương ứng.

OB87 (Communication fault):Khối OB87 sẽ được gọi khi CPU phát hiện thấy lỗi trong truyền thông ví dụ như không có tín hiệu trả lời từ các đối tác.

OB100 (Start Up Information):Khối OB100 sẽ được thực hiện 1 lần khi CPU chuyển trạng thái Stop sang Run.

OB121 (Synchronous error):Khối OB121 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 CPU.

OB122 (Synchronous error):Khối OB122 sẽ được gọi khi CPU phát hiện thấy lỗi truy cập module trong chương trình,ví dụ chương trình có lệnh truy nhập module vào ra mở rộng nhưng lại không tìm thấy module này.

8/Các Loại CPU:

CPU 312: Bộ nhớ làm việc 16KB ,chu kì lệnh 0.1us

CPU 312C : Bộ nhớ làm việc 16KB,chu kì lệnh 0.1us,tích hợp sẵn 10DI/6DO,2 Xung tốc độ cao 2.5KHz,2 kênh đọc xung tốc độ cao 10Khz.

CPU 312IFM : Bộ nhớ làm việc 6KB,chu kì lệnh 0.6us,tích hợp sẵn 10DI/6DO

CPU 313 : Bộ nhớ làm việc 12KB,chu kì lệnh 0,6us

CPU 313C : Bộ nhớ làm việc 32KB,chu kì lệnh 0,1us, tích hợp sẵn 24DI,16DO,5AI ,2AO ,3 Kênh xuất xung tốc độ cao (2.5Khz) ,3 kênh đọc xung tốc độ cao (30Khz)

CPU 313C-2DP: Bộ nhớ làm việc 32KB,chu kì lệnh 0.1us ,tích hợp sẵn 24DI,16DO,5AI ,2AO ,3 Kênh xuất xung tốc độ cao (2.5Khz) ,3 kênh đọc xung tốc độ cao (30Khz),có 2 cổng giao tiếp .

CPU 313C-2PtP : Bộ nhớ làm việc 32KB,chu kì lệnh 0.1us ,tích hợp sẵn 24DI,16DO,5AI ,2AO ,3 Kênh xuất xung tốc độ cao (2.5Khz) ,3 kênh đọc xung tốc độ cao (30Khz),có 2 cổng giao tiếp .

MPI+ PtP connector (RS-422/485 (ASCII, ..)

CPU 314: Bộ nhớ làm việc 24KB ,chu kì lệnh 0.3us

CPU 314IFM : Bộ nhớ làm việc 24KB,chu kì lệnh 0.3us, tích hợp sẵn 20DI/16DO ,4AI / 1AO

CPU 314C-2DP:Bộ nhớ làm việc 48KB,chu kì lệnh 0.1us,tích hợp sẵn 24DI / 16DO ,5AI / 2AO , 4 kênh xuất xung tốc độ cao,4 kênh đọc xung tốc độ cao.2 cổng giao tiếp .

CPU 314C-2PtP:Bộ nhớ làm việc 48KB,chu kì lệnh 0.1us,tích hợp sẵn 24DI / 16DO ,5AI / 2AO , 4 kênh xuất xung tốc độ cao,4 kênh đọc xung tốc độ cao.2 cổng giao tiếp .

CPU 315 : Bộ nhớ làm việc 48KB ,chu kì lệnh 0.3us

CPU 315-2DP: Bộ nhớ làm việc 48KB ,chu kì lệnh 0.3us , MPI + DP

CPU 315F-2DP : Bộ nhớ làm việc 128KB,chu kì lệnh 0.3us , 2 cổng giao tiếp.

CPU 316 : Bộ nhớ làm việc 128KB ,chu kì lệnh 0.3us

CPU 316-2DP: Bộ nhớ làm việc 128KB,chu kì lệnh 0.3us , 2 cổng MPI + DP

CPU 317-2: Bộ nhớ làm việc 512KB ,chu kì lệnh 0.3us,2 cổng giao tiếp MPI + DP

CPU 317F-2: Bộ nhớ làm việc 512KB,chu kì lệnh 0.3us ,2 cổng giao tiếp MPI + DP (DP master hoặc Slave)

CPU 318-2: Bộ nhớ làm việc 256KB ,chu kì lệnh 0.3us ,2 cổng giao tiếp MPI + DP (DP Master hoặc Slave).

CPU 614: Bộ nhớ làm việc 192KB,chu kì lệnh 0.3us , tích hợp sẵn 512DI/DO

CPU M7: RS232,MPI 64KB SRAM

10/ Các loại Function Module : FM300:

Controller Module:

FM 355C PID Module: Module điều khiển PID ,4 kênh điều khiển PID ,4AI + 8DI + 4AO

FM 355S PID Module: Module điều khiển PID ,4 kênh điều khiển PID ,4AI + 8DI + 8DO

FM 355 Temperature Controller: Module điều khiển nhiệt độ 4 kênh 4AI + 8DI + 4AO

FM 355 Temperature Controller: Module điều khiển nhiệt độ 4 kênh 4AI + 8DI + 8DO

M7 Application Module: 4MB,RS232,64KB Sram.

CAM Controller : FM352 CAM Module : 1 kênh điều khiển.

CNC Controller : NCU 570 FM-NC : Điều khiển CNC

Counter Module : FM350-1 Counter Module : Module đếm ở tốc độ cao 1 kênh

FM350-2 : Module đếm ở tốc độ cao 8 kênh

Possition Module : Module điều khiển vị trí

FM351 FIXED SPEED POS : Module điều khiển vị trí

FM353 Fstepper Motor : Module điều khiển động cơ bước 4DI + 4DO

FM354 F Servo Motor : Module điều khiển động cơ Servo 4DI + 4DO

Modul Nguồn :Có 3 loại Modul nguồn 10A,5A,2A **PS 307**

Modul Rack **:** Chọn Rack (Rail) để định dạng cho cấu hình phần cứng.

Modul SM300 : Bao gồm các loại Modul :

Modul Analog Input (AI): Modul đọc 2 kênh,đọc 8 kênh với các loại tín hiệu khác nhau như dòng :4-20mA (theo cách đấu 2 dây và 4 dây) ,đọc tín hiệu áp 0-10VDC , đọc tín hiệu RTD,TC...

Modul AI/AO : Modul vừa đọc AI ,vừa xuất tín hiệu Analog OutPut

Modul AO : Modul xuất tín hiệu Analog Output.

Modul DI : Modul đọc tín hiệu số

Modul DO: Modul xuất tín hiệu số

Modul DI/DO : Modul vừa đọc và xuất tín hiệu số.

1/ Các vùng nhớ S7_300:

Trong S7_300 có các vùng nhớ sau:

I: Input, các ngõ vào số.

Q:Output, các ngõ ra số.

M: Internal Memory, vùng nhớ nội.

DB: Data Block, dữ liệu. Khi sử dụng vùng nhớ này thì phải khai báo trong phần mềm.

PIW: Analog Input, ngõ vào analog.

PQW: Analog Output, ngõ ra analog.

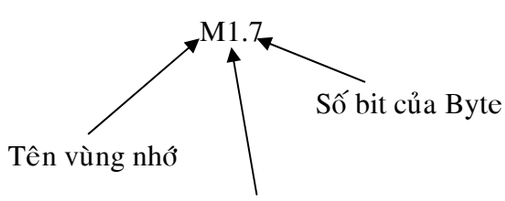
T: Timer.

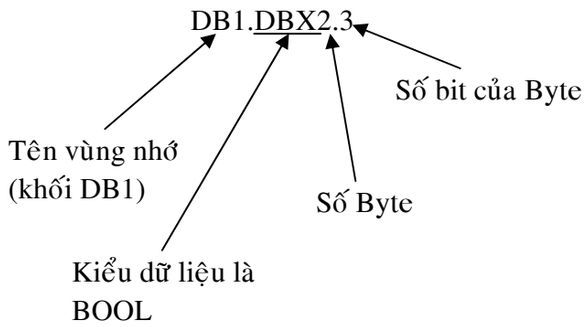
C: Counter.

Định dạng dữ liệu:

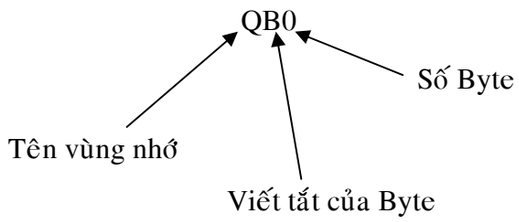
* Kiểu Bool:

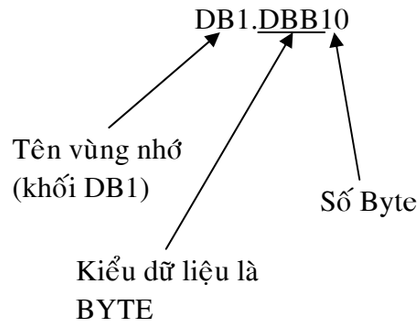
VD: Q0.0, I0.0, DB1.DBX2.3, M1.7....



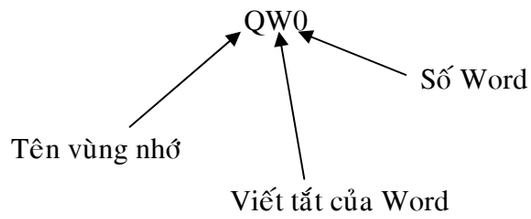


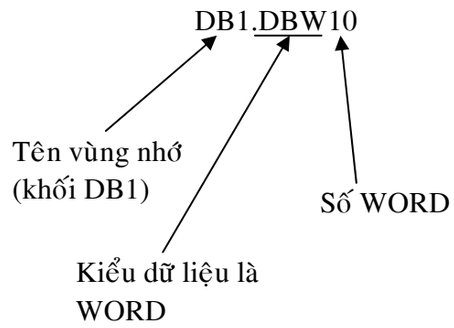
Một biến kiểu Bool chỉ có 2 giá trị là 0 hoặc 1 (TRUE hoặc FALSE).
Đối với ngõ IN
Trạng thái mức 0: OV
Trạng thái mức 1:24V
Đối với ngõ OUT:
Trạng thái mức 0: xuất 0V hoặc hở tiếp điểm
Trạng thái mức 1: xuất 24V hoặc đóng tiếp điểm
* Kiểu Byte:
1 Byte = 8 Bit. Suy ra, giá trị 1 Byte trong khỏang: 0 -(2^8-1) hay 0-255
VD: QB0, MB3, VB10, SMB2, DB1,DBB10...



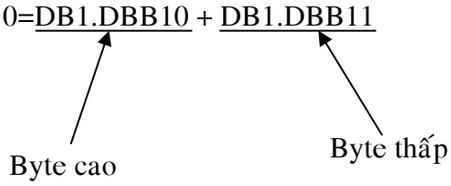


* Kiểu Word:
1 Word = 2 Byte = 16 Bit. Suy ra, giá trị 1 Word trong khỏang: 0 -(2^{16}-1)
VD: IW0, QW0, MW3, DB1.DBW10,...

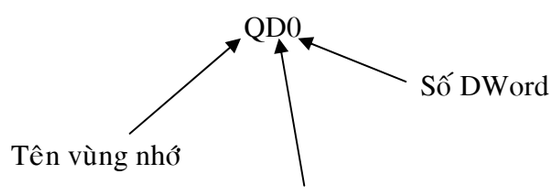




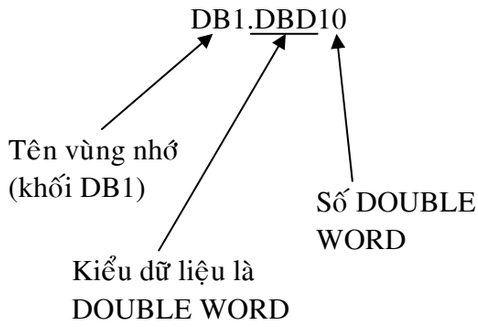
QW0=QB0+QB1, Trong đó, QB0 là byte cao, QB1 là Byte thấp.
DB1.DBW10=DB1.DBB10 + DB1.DBB11



* Kiểu DWord:
1 DWord = 2 Word = 4 Byte = 32 Bit. Suy ra, giá trị 1 Word trong khỏang: 0 -(2^{32}-1)
VD: ID0, QD0, MD3, DB1.DBD10, ...



Viết tắt của DWord



MD0=MW0+MW2=MB0+MB1+MB2+MB3, Trong đó, MB0 là byte cao nhất,MB3 là Byte thấp nhất.

* Kiểu Int: Số nguyên

Một biến kiểu Int tường đương một Word, nghĩa là dung lượng của 1 biến kiểu Int cũng gồm 16 bit. Tuy nhiên, biến kiểu Int và Word cũng có những điểm khác nhau như sau:

1/ Biến kiểu Word là biến ko dấu, biến kiểu Int có dấu(bit trọng số cao nhất là bit dấu).

2/ Giá trị 1 Word: 0 -(2^{16}-1), giá trị một Int (-2^{15}) – (2^{15}-1)

3/ Định dạng một biến kiểu Word phải có W#16# đứng đầu, còn Int thì không.

VD: W#16#1234, W#16#ABCD: một Word

1,5,100,250...: một Int

* Kiểu DInt: Số nguyên

Một biến kiểu DInt tường đương một DWord, nghĩa là dung lượng của 1 biến kiểu Int cũng gồm 32 bit. Tuy nhiên, biến kiểu DInt và DWord cũng có những điểm khác nhau như sau:

1/ Biến kiểu DWord là biến ko dấu, biến kiểu DInt có dấu(bit trọng số cao nhất là bit dấu).

2/ Giá trị 1 DWord: 0 -(2^{32}-1), giá trị một Int (-2^{31}) – (2^{31}-1)

3/ Định dạng một biến kiểu DWord phải có DW#16# đứng đầu.

Định dạng một biến kiểu DInt phải có L# đứng đầu.

VD: DW#16#12345678, DW#16#ABCDABCD: một DWord

L#1,L#5,L# -2,L#12345: một Dint

* Kiểu Real: Số thực.

Một biến kiểu Real 32 bit, nghĩa là vùng nhớ cũng là Dword.

Định dạng: phải có dấu "." Thập phân.

VD: 1.5, 2.3, 0.09, 1.0, 100.2 ...

I/Tập lệnh trong S7_300:

Kí hiệu: KQ là kết quả thu được sau phép tính

KT là kết quả trước phép tính

A.Thanh Ghi Trạng Thái:

Khi thực hiện lệnh ,CPU sẽ ghi nhận lại trạng thái của phép tính trung gian cũng như của kết quả vào một thanh ghi đặc biệt 16 Bits,được gọi là thanh ghi trạng thái (Status Word) >Mặc dù thanh ghi trạng thái này có độ dài 16 Bits nhưng chỉ sử dụng 9 Bits với cấu trúc như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC

-FC (First check) : Khi phải thực hiện một dãy các lệnh logic liên tiếp nhau gồm các phép tính giao ,hợp và nghịch đảo,bit FC có giá trị bằng 1,hay nói cách khác ,FC=0 khi dãy lệnh Logic tiếp điểm vừa được kết thúc.

-RLO (Result of logic operation) : Kết quả tức thời của phép tính logicvừa được thực hiện

-STA (Status bit) : Bit trạng thái này luôn có giá trị logic của tiếp điểm được chỉ định trong lệnh.

-OR :Ghi lại giá trị của phép tính logic giao cuối cùng được thực hiện để phụ giúp cho việc thực hiện phép toán hợp sau đó.Điều này là cần thiết vì trong một biểu thức hàm 2 trị ,phép tính giao bao giờ cũng phải được thực hiện trước các phép tính hợp.

-OS (Stored overflow bit) : Ghi lại giá trị Bit bị tràn ra ngoài mảng ô nhớ.

-OV(Overflow Bit): Bit báo cáo kết quả phép tính bị tràn ra ngoài mảng ô nhớ.

-CC0 và CC1 (Condition code) : Hai bit báo trạng thái của kết quả phép tính với số nguyên,số thực phép dịch chuyển hoặc phép tính logic trong ACCU

CC1	CC0	Ý nghĩa
0	0	Kết quả bằng 0 (=0)
0	1	Kết quả nhỏ hơn 0 (< 0)
1	0	Kết quả lớn hơn 0 (> 0)

Khi thực hiện lệnh toán học như cộng trừ nhân chia với số nguyên hoặc số thực

CC0	CC1	Ý Nghĩa
0	0	Kết quả quá nhỏ khi thực hiện lệnh cộng (+I,+D)
0	1	Kết quả quá nhỏ khi thực hiện lệnh nhân (*I,*D) hoặc quá lớn khi thực hiện lệnh cộng trừ (+I,+D,-I,-D)
1	0	Kết quả quá lớn khi thực hiện lệnh nhân chia (*I,*D,/I,/D) hoặc quá nhỏ khi thực hiện lệnh cộng trừ (+I,+D,-I,-D)
1	1	Kết quả bị tràn do thực hiện lệnh chia cho 0 (/I,/D)

Khi thực hiện lệnh toán học với số nguyên nhưng kết quả bị tràn ô nhớ

CC0	CC1	Ý Nghĩa
0	0	Kết quả có số mũ e quá lớn
0	1	Kết quả có mantissa quá nhỏ
1	0	Kết quả có mantissa quá lớn
1	1	Phép tính sai quy chuẩn

Khi thực hiện lệnh toán học với số thực nnhưng kết quả bị tràn ô nhớ

CC0	CC1	Ý Nghĩa
0	**0**	**Giá trị của bit bị đẩy ra bằng 0**
1	**0**	**Giá trị của Bit bị đẩy ra bằng 1**

Khi thực hiện lệnh dịch chuyển

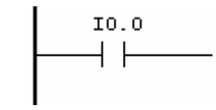
CC0	CC1	Ý Nghĩa
0	0	**Kết quả bằng 0**
1	0	**Kết quả khác 0**

Khi thực hiện lệnh logic trong ACCU

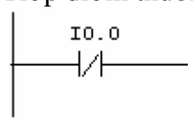
BR (Binary result bit) : Bit trạng thái cho phép liên kết hai loại ngôn ngữ lập trình STL và LAD .Chẳng hạn cho phép người sử dụng có thể viết một khối chương trình FB hoặc FC trên ngôn ngữ STL nhưng gọi và sử dụng chúng trong một chương trình khác viết trên LAD .Để tao ra được mối liên kết đó,ta cần phải kết thúc chương trình trong FB,FC bằng lệnh ghi
BR = 1 ,nếu chương trình chạy không có lỗi
BR = 0 ,nếu chương trình chạy có lỗi
Khi sử dụng các khối hàm đặc biệt của hệ thống (SFC hoặc SFB) ,trạng thái làm việc của chương trình cũng được thông báo ra ngoài qua bit trạng thái BR như sau:
BR=1 nếu SFC hay SFB thực hiện không có lỗi
BR=0 nếu có lỗi khi thực hiện SFC hay SFB

1/ Lệnh về bit:

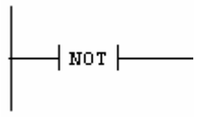
Tiếp điểm thường hở: KQ=KT nếu I0.0=1. KQ=0 nếu I0.0=0



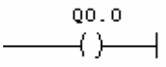
Tiếp điểm thường đóng : KQ=KT nếu I0.0=0. KQ=0 nếu I0.0=1



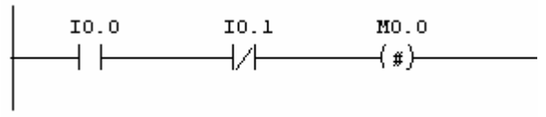
Lệnh Not: KQ thu được bằng đảo giá trị của KT
Nếu KT=1 thì KQ=0 ; Nếu KT=0 thì KQ=1



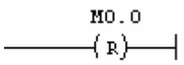
Ngõ ra (cuộn coil) : Gán KQ cho ngõ ra Q0.0



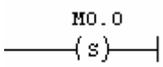
Xác định kết quả: Gán KQ tại vị trí mà lệnh được chèn
Vd: M0.0 lưu kết quả sau 2 phép tính qua I0.0 và I0.1



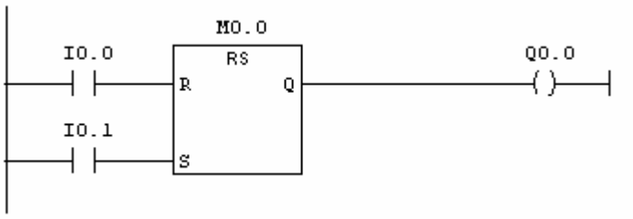
Lệnh Reset Bit: Gán giá trị 0 cho M0.0



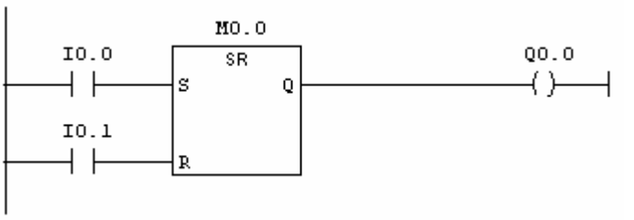
Lệnh Set Bit: Gán giá trị 1 cho M0.0



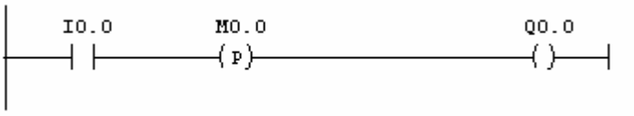
Lệnh RS:
Nếu I0.0=1 , I0.1=0 thì M0.0=1, Q0.0=0
Nếu I0.0=0 ,I0.1=1 thì M0.0=0 ,Q0.0=1
Nếu I0.0=I0.1=0 Thì không có gì thay đổi.
Nếu I0.0=I0.1=1 thì M0.0=Q0.0=1



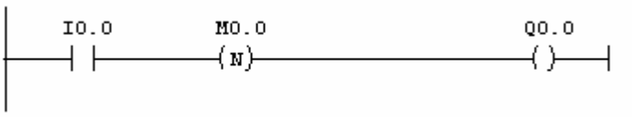
Lệnh SR:
Nếu I0.0=1 , I0.1=0 thì M0.0=1, Q0.0=1
Nếu I0.0=0 ,I0.1=1 thì M0.0=0 ,Q0.0=0
Nếu I0.0=I0.1=0 Thì không có gì thay đổi.
Nếu I0.0=I0.1=1 thì M0.0=Q0.0=0



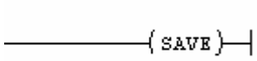
Vi phân cạnh lên :
M0.0 lưu giá trị KQ ở vòng quét trước
Khi I0.0 chuyển trạng thái từ 0 sang 1 và M0.0 =0 thì Q0.0 =1



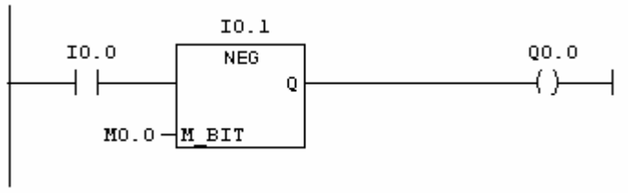
Vi phân cạnh xuống:
M0.0 lưu giá trị KQ ở vòng quét trước
Khi I0.0 chuyển trạng thái từ 1 xuống 0 và M0.0=1 thì Q0.0=1



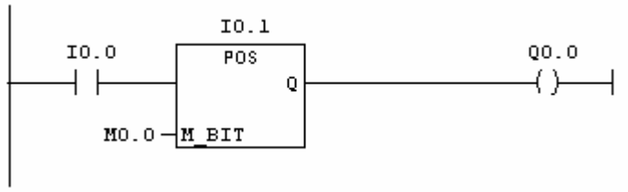
Như vậy trong cả 2 lệnh vi phân cạnh xuống và vi phân cạnh lên thì Q0.0 chỉ ON trong 1 chu kì tại thời điểm thoả điều kiện.
Lệnh Save : Lưu giá trịRLO (KQ) vào Bit cờ BR (Binary Result Bit)



Lệnh NEG: Khi I0.0=1 và I0.1 chuyển trạng thái từ 1 xuống 0 thì Q0.0 ON trong 1 chu kì
Hay nói cách Khác Q0.0 chỉ ON tại thời điểm thoả điều kiện bài toán.

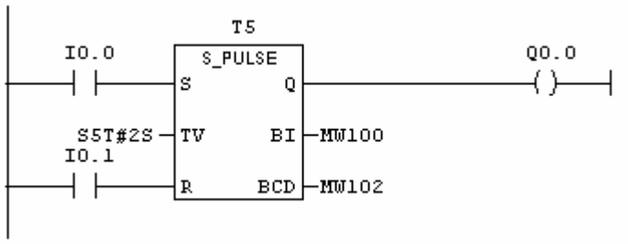


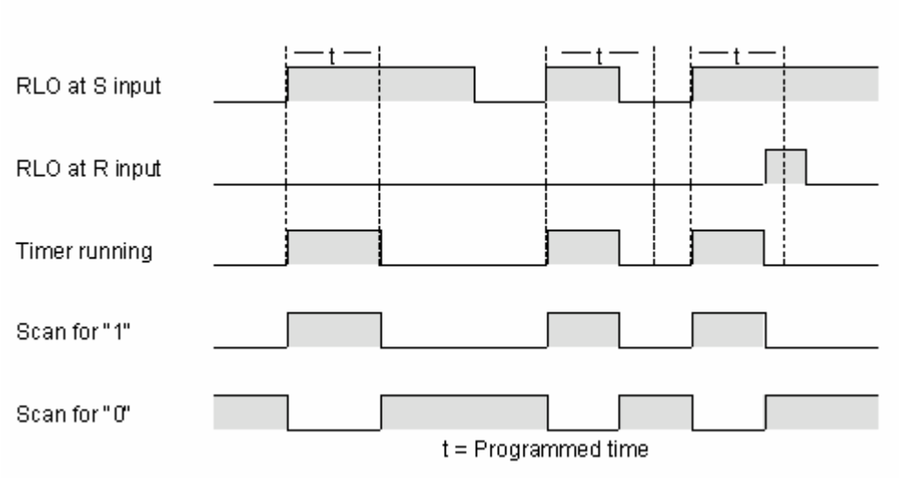
Lệnh POS: : Khi I0.0=1 và I0.1 chuyển trạng thái từ 0 lên 1 thì Q0.0 ON trong 1 chu kì
Hay nói cách Khác Q0.0 chỉ ON tại thời điểm thoả điều kiện bài toán.



2/ Lệnh về Timer :

Lệnh S_PULSE:
Nếu I0.0=1 Timer được kích chạy,khi I0.0=0 hoặc chạy đủ thời gian đặt 2s thì Timer dừng
Hoặc có tín hiệu I0.1 thì Timer cũng dừng
Timer chỉ chạy lại khi có tín hiệu mới từ I0.0 (tức là I0.0 chuyển trạng thái từ 0 lên 1)
Q0.0=1 khi Timer đang chạy.
MW100 lưu giá trị đếm của Timer theo dạng Integer
MW102 lưu giá trị của Timer theo dạng BCD
Chức năng của Timer này là tạo xung có thời gian được đặt sẵn





Lệnh S_PEXT:

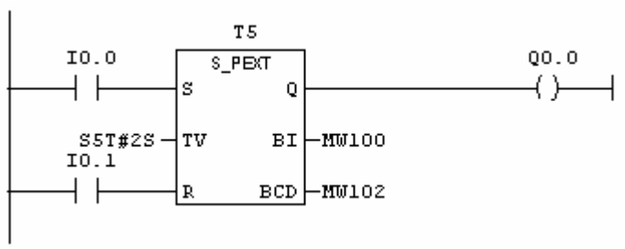
Timer kích có nhớ,Khi có tín hiệu cạnh lên ở I0.0 Timer T5 chạy,nếu đủ thời gian đặt Timer dừng.

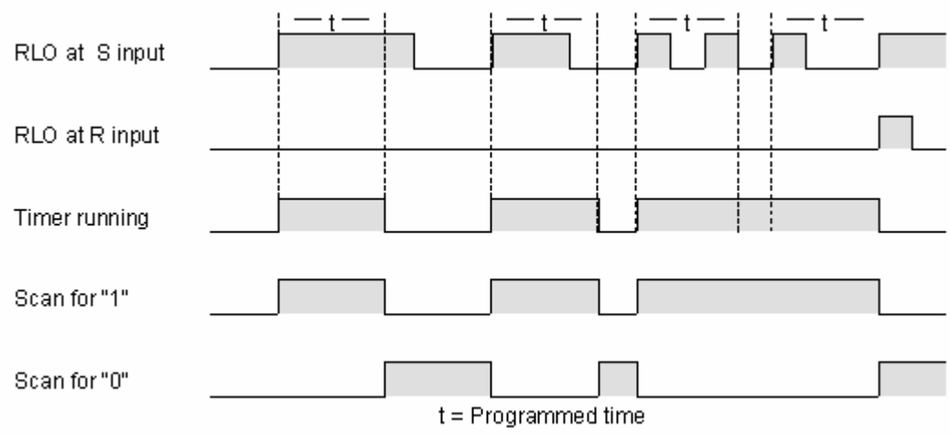
Trong quá trình chạy nếu có tín hiệu mới từ chân I0.0 thì thời gian Timer lại được tính lại từ đầu.

Trong quá trình chạy nếu có tín hiệu I0.1 thì Timer dừng

Q0.0 =1 khi Timer đang chạy.

Các ô nhớ MW100 và MW102 lưu giá trị hiện thời của Timer theo dạng Integer và dạng BCD

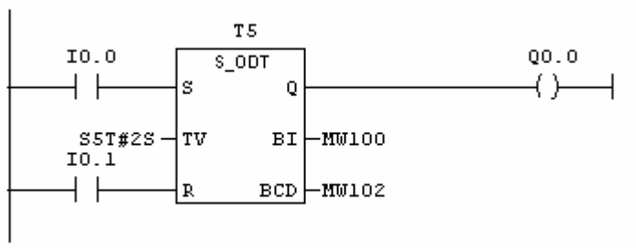


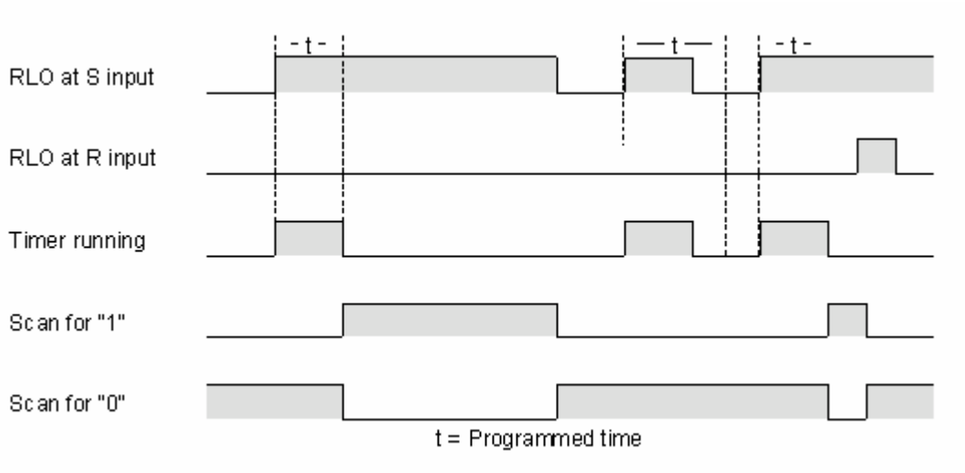


Lệnh S_ODT:

Nếu I0.0=1 Timer bắt đầu chạy khi đủ thời gian thì ngưng khi đó ngõ Q0.0 sẽ lên 1 nếu I0.0 vẫn còn giữ trạng thái 1,khi có tín hiệu I0.1 thì tất cả phải được Reset về 0

Các ô nhớ MW100 và MW102 lưu giá trị hiện thời của Timer theo dạng Integer và dạng BCD

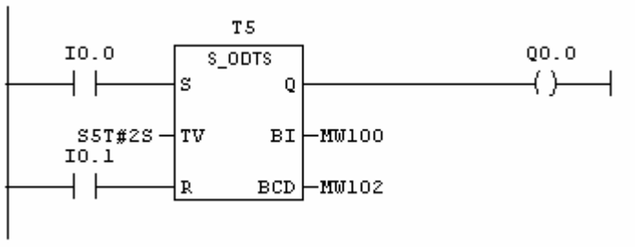


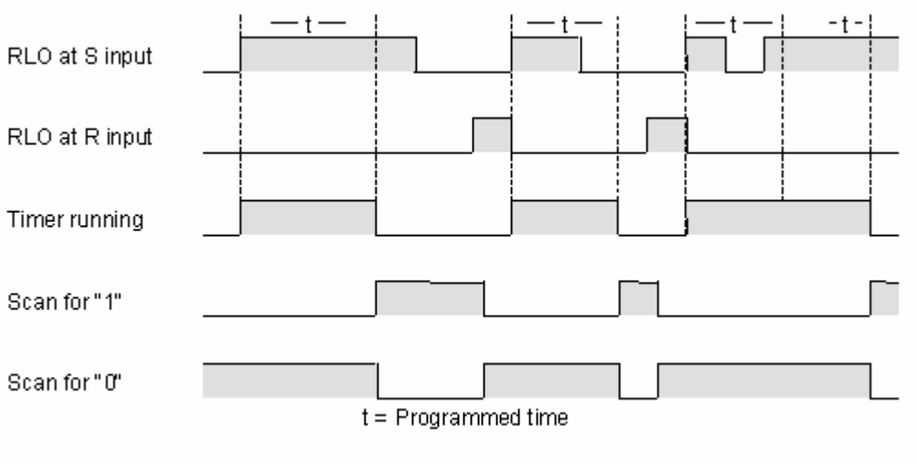


Lệnh S_ODTS:

Timer kích có nhớ,khi có xung cạnh lên ở I0.0 Timer bắt đầu chạy ,ngõ ra Q0.0=1 khi Timer ngưng và chỉ tắt khi có tín hiệu Reset (tín hiệu I0.1)

Trong quá trình Timer chạy nếu có sự chuyển đổi tín hiệu từ chân I0.0 them 1 lần nữa thì Timer sẽ nhớ và tiếp tục chạy khi hết thời gian lần trước.

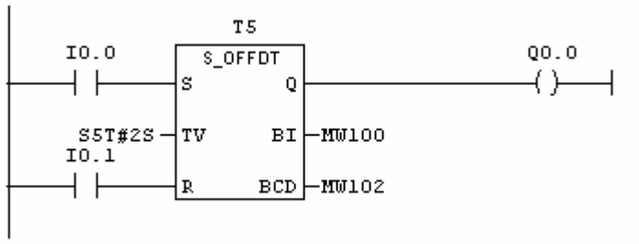


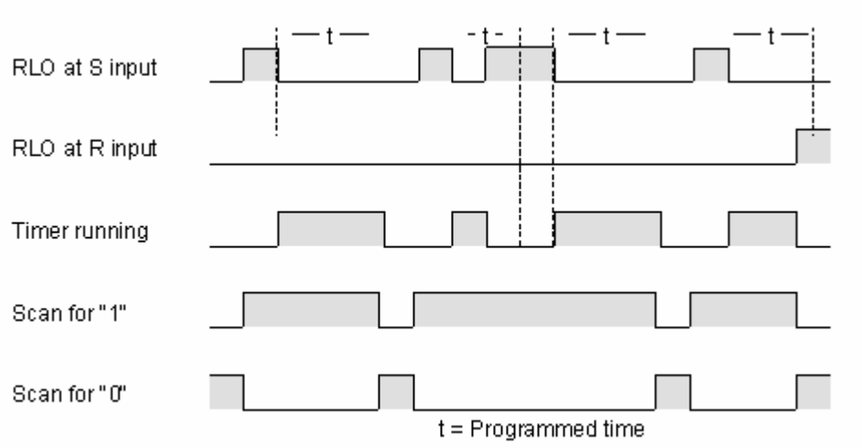


Lệnh S_OFFDT:

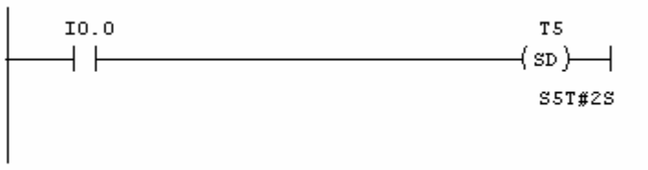
Khi I0.0 ON , Q0.0 =1 ,khi I0.0 OFF Timer bắt đầu chạy và Q0.0 chỉ tắt khi đủ thời gian và I0.0 vẫn OFF

Khi có tín hiệu Reset I0.1 thì tất cả tín hiệu đều OFF





Lệnh TON:



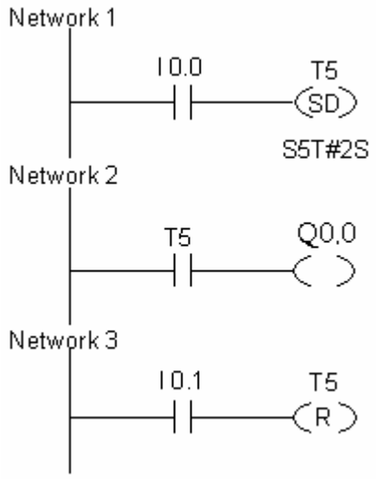
Parameter	Data Type	Memory Area	
<T no.>	TIMER	T	Số hiệu timer
<time value>	S5TIME	I, Q, M, L, D	Giá trị đặt cho timer

Số Timer trong S7_300 phụ thuộc vào loại CPU.
CPU 312: có 128 Timer
CPU 313 trở lên: có 256 Timer.
Có 2 cách cài đặt giá trị cho Timer:
1/ Cài thông số thời gian trực tiếp:
Để cài giá trị trực tiếp cho Timer ta phải thêm kí tự S5T# trước giá trị đặt. Các kí tự kế tiếp là thông số thời gian muốn cài đặt cho Timer.
Tổng quát như sau: S5T#aH_bM_cS_dMS. Trong đó:

H: giờ
M: phút
S: giây
MS: mili giây
a,b,c,d: các thông số cài đặt.

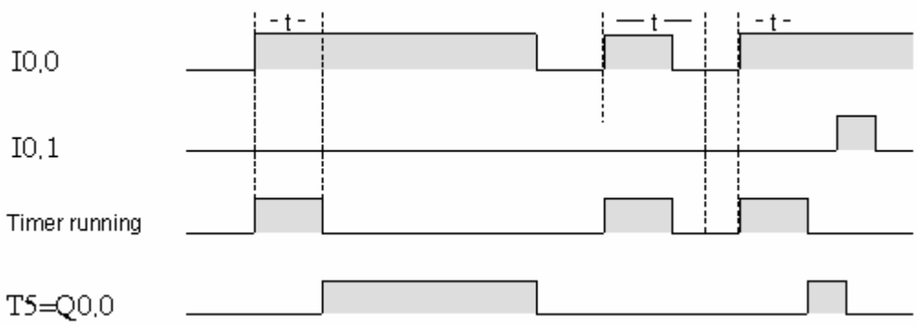
VD: S5T#3S: thời gian cài đặt là 3s
S5T#7S500MS: thời gian cài đặt là 7,5s
S5T#1M8S200MS: thời gian cài đặt là 1 phút 8 giây 200 ms
S5T#1H1M10S: thời gian cài đặt là 1 giờ 1 phú 10 giây.

Trong VD trên thì T0 được cài thời gian là 1 giờ 30 phút.



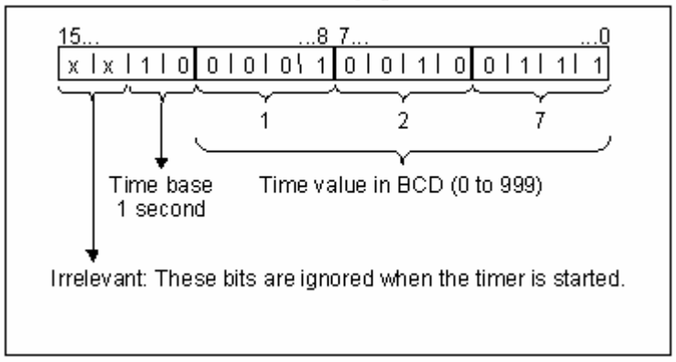
Trong VD trên, khi I0.0 ON, Timer T5 sẽ được kích hoạt. Đủ thời gian cài đặt là 2 s thì bit T5 tác động làm cho Q0.0 ON.

Khi ngõ vào I0.1 tác động thì Timer được reset. Giá trị hiện tại của Timer cũng như Bit T5 được Reset về 0.



2/ Cài đặt thông số thời gian thông qua biến nhớ:

Giá trị cài đặt cho timer thông qua một biến kiểu WORD 16 bit:



Hai bit cao nhất trong WORD không sử dụng

Hai bit kế tiếp (Time base) cài thông số đơn vị thời gian cho Timer, cụ thể:

Time Base	Binary Code for the Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

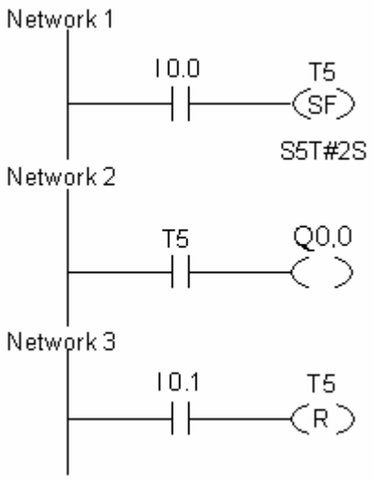
12 bit kế tiếp là giá trị cài đặt thời gian cho Timer dưới dạng số BCD (giá trị từ 0-999). Như trong VD trên thì giá trị cài đặt cho Timer sẽ là 127s.

Như vậy để có thể cài đặt giá trị cho Timer thay đổi theo ô nhớ:
Ta phải thực hiện các bước:
Giá trị Timer phải bé hơn hoặc bằng 999
Chuyển giá trị đó sang dạng BCD dùng lệnh I_BCD
Sau đó chọn Time Base theo mong muốn như bảng trên bằng cách chọn 4 Bit đầu.

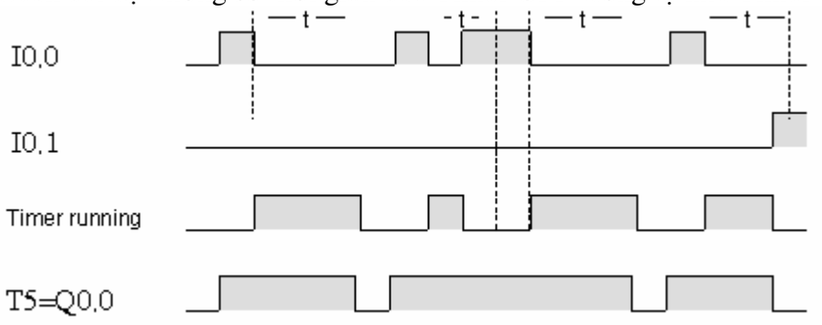
TOFF:

<T no.>
---(SF)
<time value>

Parameter	Data Type	Memory Area	
<T no.>	TIMER	T	Số hiệu timer
<time value>	S5TIME	I, Q, M, L, D	Giá trị đặt cho timer



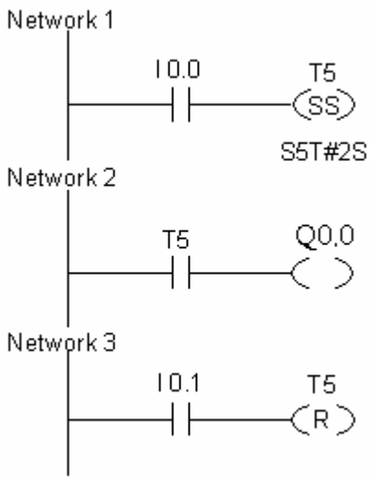
Trong VD trên, khi I0.0 ON, Bit T5 sẽ ON ngay khi I0.0 ON. Khi I0.0 chuyển từ ON sang OFF, Timer T5 sẽ được kích hoạt. Đủ thời gian cài đặt là 2 s thì Timer T5 tác động, bit T5 OFF làm cho Q0.0 OFF.
Khi ngõ vào I0.1 tác động thì Timer được reset. Giá trị hiện tại của Timer cũng như Bit T5 được Reset về 0.
Cách cài đặt thông số thời gian của Timer OFF tương tự như Timer ON.



TON có nhớ:

<T no.>
---(SS)
<time value>

Parameter	Data Type	Memory Area	
<T no.>	TIMER	T	Số hiệu timer
<time value>	S5TIME	I, Q, M, L, D	Giá trị đặt cho timer



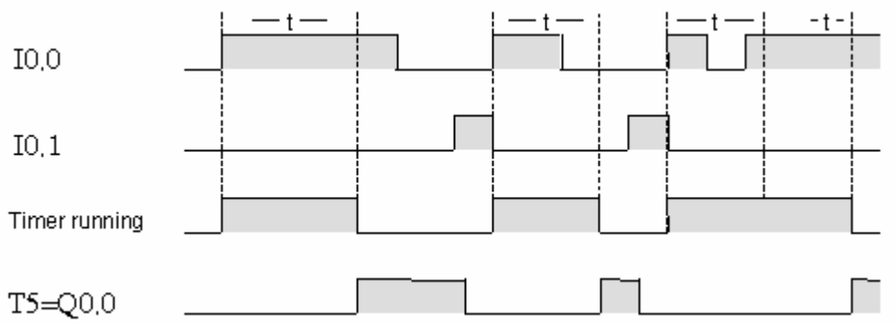
Trong VD trên, khi I0.0 ON, Timer T5 sẽ được kích hoạt. Đủ thời gian cài đặt là 2 s thì bit T5 tác động làm cho Q0.0 ON. Trong trường hợp thời gian chưa đủ 2S mà I0.0 chuyển OFF sang ON một lần nữa, giá trị đếm của Timer sẽ được khởi động lại.

Giữa Timer ON và Timer ON có nhớ còn khác nhau một điểm nữa như sau:

Timer ON: sau khi Timer tác động, Bit của Timer được bật ON, nếu tín hiệu kích Timer mất đi thì Timer sẽ được Reset, Bit timer sẽ OFF.

Timer ON có nhớ: sau khi Timer tác động, Bit của Timer được bật ON, nếu tín hiệu kích Timer mất đi thì Timer vẫn không Reset, Bit timer sẽ vẫn ON.

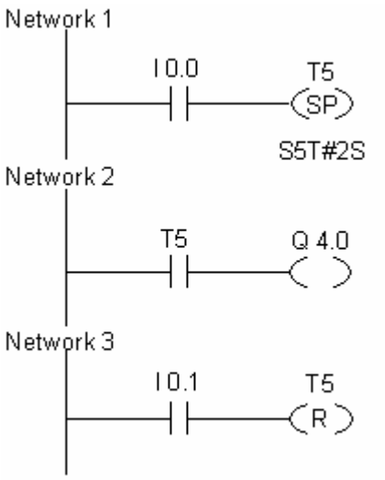
Khi ngõ vào I0.1 tác động thì Timer được reset. Giá trị hiện tại của Timer cũng như Bit T5 được Reset về 0.



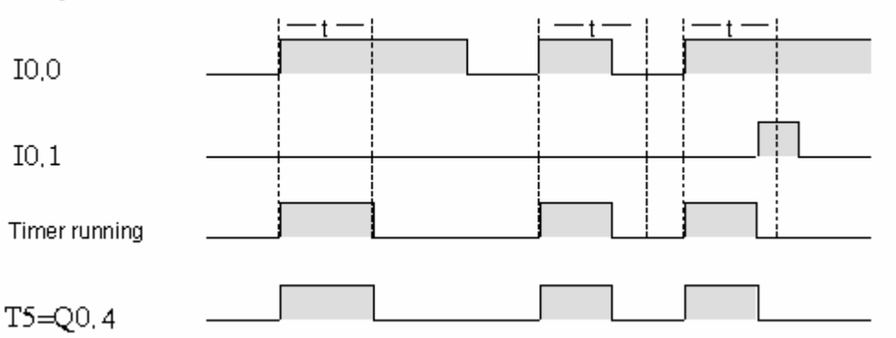
Timer xung:

<T no..>
---(SP)
<time value>

Parameter	Data Type	Memory Area	
<T no.>	TIMER	T	Số hiệu timer
<time value>	S5TIME	I, Q, M, L, D	Giá trị đặt cho timer



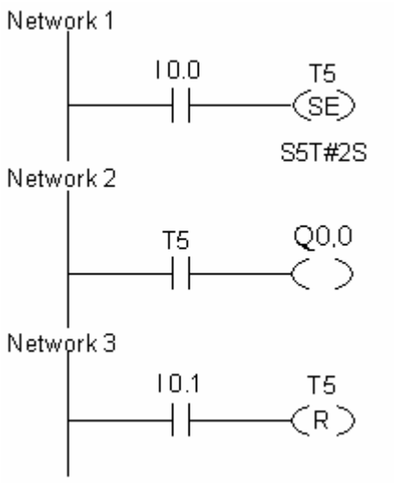
Mô tả:
Khi I0.0 chuyển từ 0 lên 1, Timer T5 sẽ được khởi động, ngõ ra bit T5 sẽ ON ngay lập tức. Khi hết thời gian cài đặt là 2s thì bit T5 OFF (nếu ngõ vào I0.0 vẫn còn ON). Trong trường hợp chưa đủ 2s mà ngõ vào I0.0 đã OFF, Timer sẽ được reset và ngõ ra bit T5 OFF.
Trong khi Timer chạy mà chưa đủ 2s, nếu I0.1 chuyển từ 0 lên 1. Ngõ ra bit T5 sẽ OFF và thời gian được reset.



Timer xung mở rộng:

<T no.>
---(SE)
<time value>

Parameter	Data Type	Memory Area	
<T no.>	TIMER	T	Số hiệu timer
<time value>	S5TIME	I, Q, M, L, D	Giá trị đặt cho timer



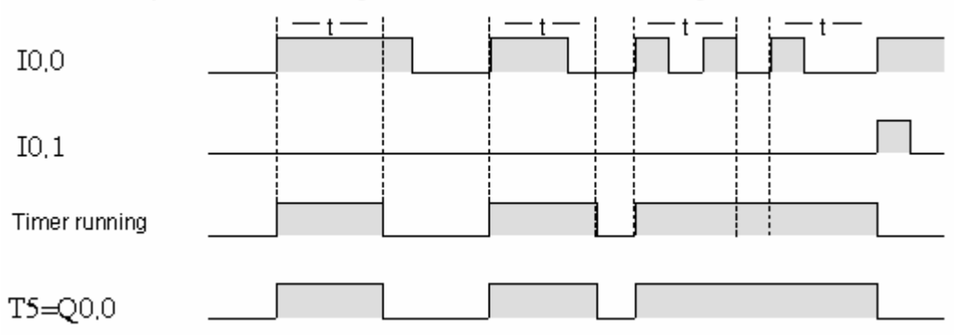
Mô tả:

Khi I0.0 chuyển từ 0 lên 1, Timer T5 sẽ được khởi động, ngõ ra bit T5 sẽ ON ngay lập tức.

Khi hết thời gian cài đặt là 2s thì bit T5 OFF (bất kể ngõ vào I0.0 vẫn còn ON hay đã OFF).

Trong trường hợp chưa đủ 2s mà ngõ vào I0.0 chuyển từ OFF lên ON một lần nữa, Timer sẽ được khởi động lại.

Khi I0.1 chuyển từ 0 lên 1. Ngõ ra bit T5 sẽ OFF và thời gian được reset.



3/ Counter:

Lệnh đếm lên xuống S_CUD:

Ngõ vào I0.2=1 : đưa giá trị đếm vào PV

Khi I0.0 chuyển trạng thái từ 0 lên 1 ,C0 đếm tăng lên 1

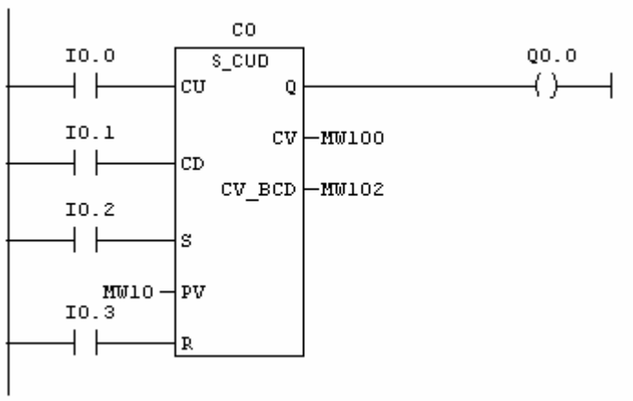
Khi I0.1 chuyển trạng thái từ 0 lên 1 ,C0 đếm giảm xuống 1

Khi cả I0.0 và I0.1 đều chuyển trạng thái thì C0 không thay đổi

Khi I0.3=1 thì C0 bị Reset về 0

Giá trị bộ đếm hiện thời nằm trong 2 ô nhớ MW100 và MW102 dưới dạng Integer và dạng BCD ,giá trị này có tầm từ 0 – 999.

Ngõ ra Q0.0=1 khi giá trị đếm lớn hơn 0



Lệnh đếm lên S_CU:

Ngõ vào I0.1=1 : đưa giá trị đếm vào PV

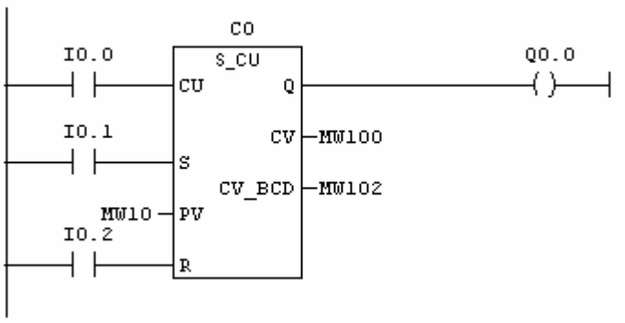
Khi I0.0 chuyển trạng thái từ 0 sang 1 , C0 đếm tăng lên 1

Khi I0.2 = 1 Counter bị Reset

Ngõ ra Q0.0=1 khi giá trị đếm lớn hơn 0

Giá trị bộ đếm hiện thời nằm trong 2 ô nhớ MW100 và MW102 dưới dạng Integer và dạng BCD ,giá trị này có tầm từ 0 – 999.

Ngõ ra Q0.0=1 khi giá trị đếm lớn hơn 0



Lệnh đếm xuống S_CD:

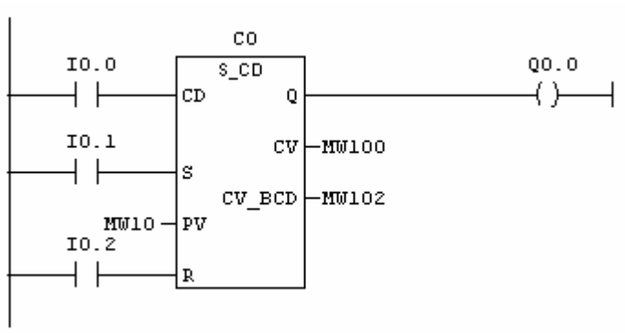
Ngõ vào I0.1=1 : đưa giá trị đếm vào PV

Khi I0.0 chuyển trạng thái từ 1 sang 0 , C0 giảm đi 1

Khi I0.2 = 1 Counter bị Reset

Ngõ ra Q0.0=1 khi giá trị đếm lớn hơn 0

Giá trị bộ đếm hiện thời nằm trong 2 ô nhớ MW100 và MW102 dưới dạng Integer và dạng BCD ,giá trị này có tầm từ 0 – 999.
Ngõ ra Q0.0=1 khi giá trị đếm lớn hơn 0

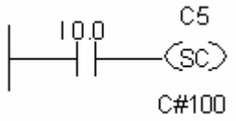


Lệnh Set Counter: (SC)

<C no.>
---(SC)
<preset value>

Parameter English	Parameter German	Data Type	Memory Area	
<C no.>	<Z no.>	COUNTER	C	Số hiệu Counter
<preset value>	<preset value>	WORD	I, Q, M, L, D or constant	Giá trị đặt cho Counter

S7_300 có 1000 counter (từ C0 đến C999)

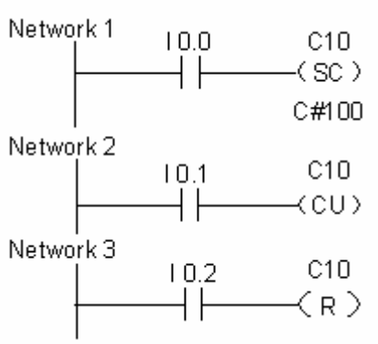


Mô tả:
Khi I0.0 ON, giá trị 100 sẽ được nạp cho Counter C5.
Khi I0.0 OFF, giá trị Counter sẽ phụ thuộc vào các tín hiệu kích đếm.

Lệnh đếm lên: (CU)

<C no.>
---(CU)

Parameter English	Parameter German	Data Type	Memory Area	
<C no.>	<Z no.>	COUNTER	C	Số hiệu Counter

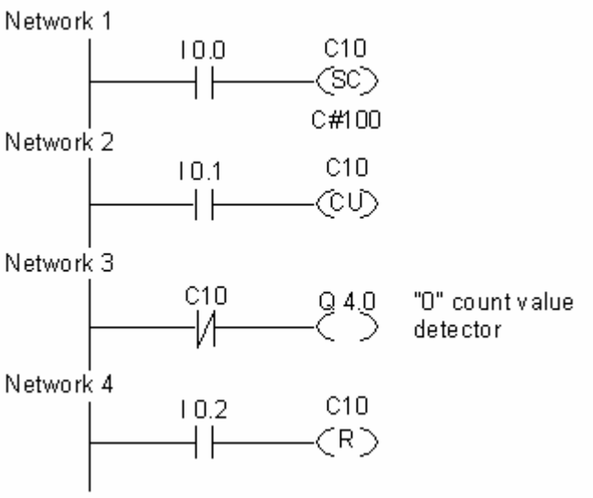


Mô tả:
Khi I0.0 chuyển từ 0 lên 1, giá trị 100 sẽ được nạp vào cho Counter C10.
Cứ mỗi xung cạnh lên ở ngõ vào I0.1, bộ đếm C10 sẽ tăng 1 đơn vị. Khi giá trị tăng đến 999 thì tín hiệu kích tăng không còn tác dụng.
Khi I0.2 chuyển từ 0 lên 1, giá trị bộ đếm và bit C10 sẽ được reset.

Lệnh đếm xuống: (CD)

<C no.>
---(CD)

Parameter English	Parameter German	Data Type	Memory Area	
<C no.>	<Z no.>	COUNTER	C	Số hiệu Counter

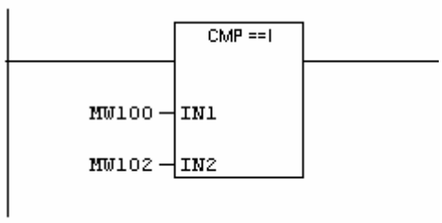


Mô tả:
Khi I0.0 chuyển từ 0 lên 1, giá trị 100 sẽ được nạp vào cho Counter C10.
Cứ mỗi xung cạnh lên ở ngõ vào I0.1, bộ đếm C10 sẽ giảm 1 đơn vị. Khi giá trị giảm đến 0 thì tín hiệu kích giảm không còn tác dụng, đồng thời lúc đó C10 sẽ OFF. Nếu bộ đếm khác 0, C10 sẽ ON.
Khi I0.2 chuyển từ 0 lên 1, giá trị bộ đếm và bit C10 sẽ được reset.

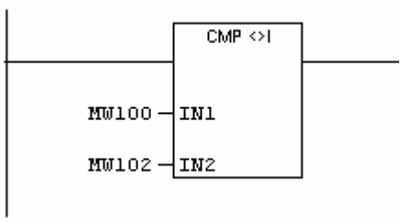
4/Lệnh So Sánh:

a/Lệnh so sánh số nguyên:

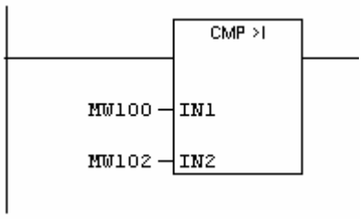
Lệnh EQ_I (Equal Integer): So sánh MW100 và MW102, nếu 2 số nguyên này bằng nhau thì KQ=KT



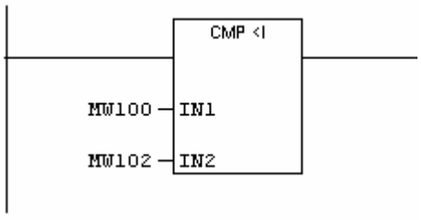
Lệnh NE_I (Not Equal Integer) : So sánh MW100 và MW102,nếu 2 số này khác nhau thì KQ=KT.



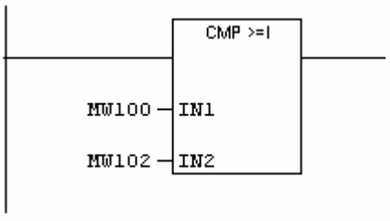
Lệnh GT_I (Greater than Integer) : So sánh 2 số MW100 và MW102 ,nếu MW100 lớn hơn MW102 thì KQ=KT



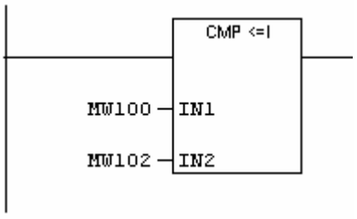
Lệnh LT_I (Less than Integer) : So sánh 2 số MW100 và MW102,Nếu MW100 bé hơn MW102 thì KQ=KT



Lệnh GE_I (Greater than or equal Integer) : So sánh 2 số MW100 và MW102, Nếu MW100 lớn hơn hoặc bằng MW102 thì KQ=KT

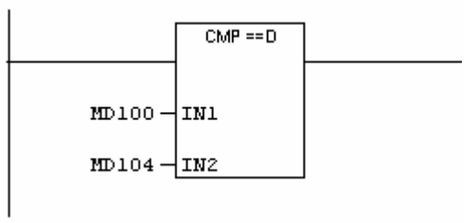


Lệnh LE_I (Less than or equal Integer) : So sánh 2 số MW100 và MW102, Nếu MW100 bé hơn hoặc bằng MW102 thì KQ=KT

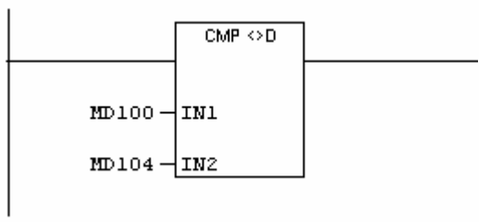


b/ Lệnh so sánh số Double Integer:

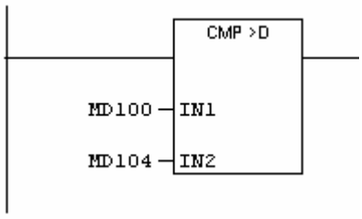
Lệnh EQ_D (Equal Double Integer): So sánh MD100 và MD104, nếu 2 số nguyên này bằng nhau thì KQ=KT



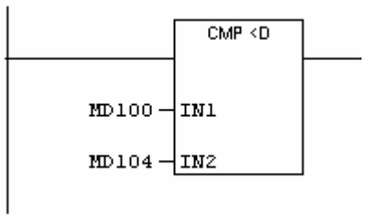
Lệnh NE_D (Not Equal Double Integer) : So sánh MD100 và MD104,nếu 2 số này khác nhau thì KQ=KT.



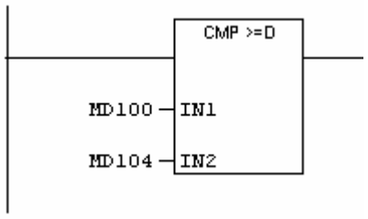
Lệnh GT_D (Greater than DoubleInteger) : So sánh 2 số MD100 và MD104 ,nếu MD100 lớn hơn MD104 thì KQ=KT



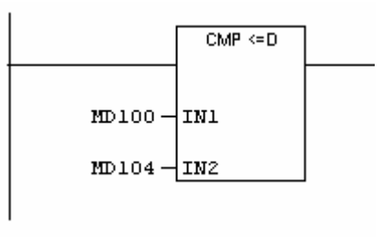
Lệnh LT_D (Less than DoubleInteger) : So sánh 2 số MD100 và MD104,Nếu MD100 bé hơn MD104 thì KQ=KT



Lệnh GE_D (Greater than or equal DoubleInteger) : So sánh 2 số MD100 và MD104, Nếu MD100 lớn hơn hoặc bằng MD104 thì KQ=KT

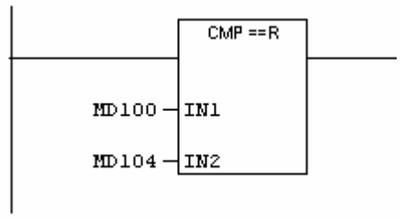


Lệnh LE_D (Less than or equal DoubleInteger) : So sánh 2 số MD100 và MD104, Nếu MD100 bé hơn hoặc bằng MD104 thì KQ=KT

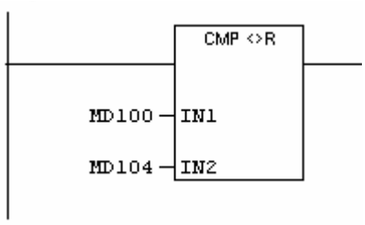


b/ Lệnh so sánh số thực (Real):

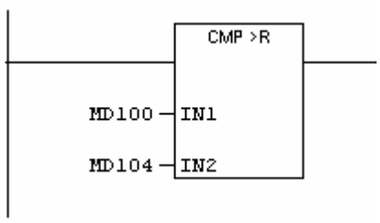
Lệnh EQ_R (Equal Real): So sánh MD100 và MD104, nếu 2 số nguyên này bằng nhau thì KQ=KT



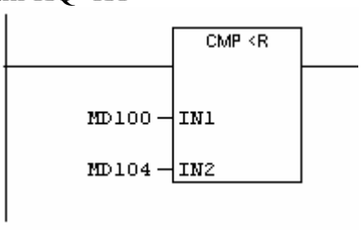
Lệnh NE_R (Not Equal Real) : So sánh MD100 và MD104,nếu 2 số này khác nhau thì KQ=KT.



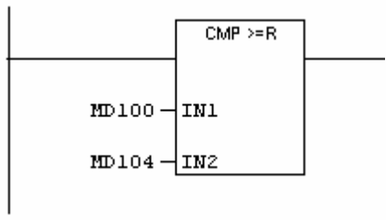
Lệnh GT_R (Greater than Real) : So sánh 2 số MD100 và MD104 ,nếu MD100 lớn hơn MD104 thì KQ=KT



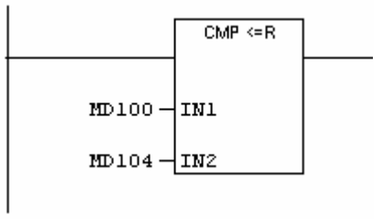
Lệnh LT_R (Less than Real) : So sánh 2 số MD100 và MD104,Nếu MD100 bé hơn MD104 thì KQ=KT



Lệnh GE_R (Greater than or equal Real) : So sánh 2 số MD100 và MD104, Nếu MD100 lớn hơn hoặc bằng MD104 thì KQ=KT



Lệnh LE_R (Less than or equal Real) : So sánh 2 số MD100 và MD104, Nếu MD100 bé hơn hoặc bằng MD104 thì KQ=KT



5 /Lệnh chuyển đổi:

Lệnh BCD_I : Chuyển đổi từ số định dạng dưới dạng BCD (chứa 3 Digit)sang số nguyên 16 Bit

Số BCD có tầm (+/- 999) chứa trong 12Bit.

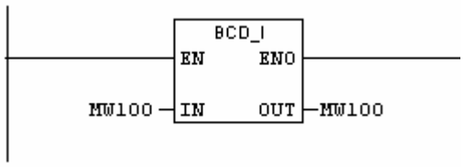
Vd: MW100 =22 được định dạng dưới dạng BCD như sau:

0000 0010 0010

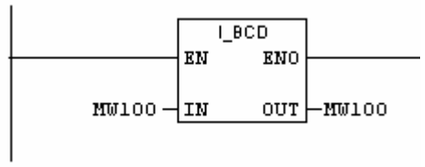
2 2

Sau khi thực hiện lệnh chuyển đổi thành số Integer 16 Bit được định dạng:

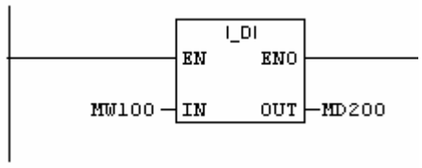
0000 0000 0001 0110 := $1*2^4 + 1*2^2 + 1*2^1 = 22$



Lệnh I_BCD: Chuyển đổi từ số nguyên sang số được định dạng dưới dạng BCD (chứa 3 Digit), do số BCD tối đa 999 nên số nguyên phải tối đa 999



Lệnh I_DI : Chuyển đổi số nguyên từ 16Bit sang số nguyên 32 Bit để thực hiện cho các phép toán trên số 32 Bit.



Lệnh BCD_DI : : Chuyển đổi từ số định dạng dưới dạng BCD (chứa 7 Digit)sang số nguyên 32 Bit

Số BCD có tầm (+/- 9999999) chứa trong 28Bit.

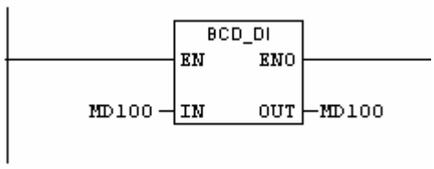
Vd: MD100 =22 được định dạng dưới dạng BCD như sau:

0000 0000 0000 0000 0000 0010 0010

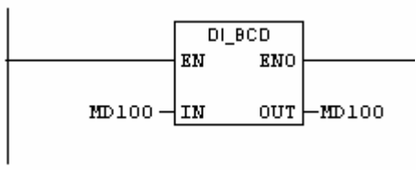
2 2

Sau khi thực hiện lệnh chuyển đổi thành số DoubleInteger 32 Bit được định dạng:

0000 0000 0000 0000 0000 0000 0001 0110 := $1*2^4 + 1*2^2 + 1*2^1$ =22

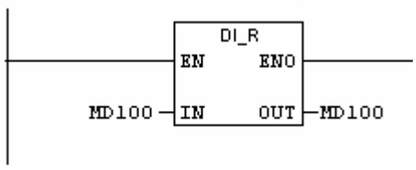


Lệnh DI_BCD: : Chuyển đổi từ số nguyên 32 Bit sang số được định dạng dưới dạng BCD (chứa 7 Digit), do số BCD tối đa 9999999 nên số nguyên phải tối đa 9999999



Lệnh DI_R : Chuyển đổi từ số nguyên 32 Bit sang số thực để phục vụ cho các phép toán trên số thực.

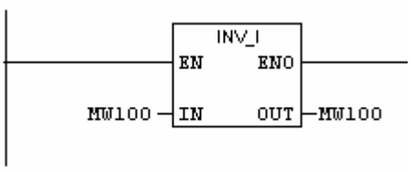
Vd: MD100= 457 ; Sau phép toán giá trị mới là MD100=457.0



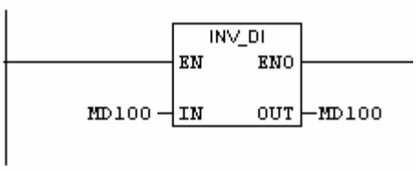
Lệnh INV_I : Đảo tất cả các Bit của số nguyên 16 Bit

Vd : MW100 ban đầu là 0101 0111 1000 0101 ; sau lệnh chuyển đổi

MW100 lúc sau 1010 1000 0111 1010

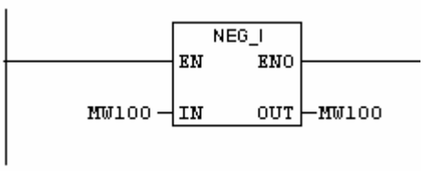


Lệnh INV_DI: : Đảo tất cả các Bit của số nguyên 32 Bit

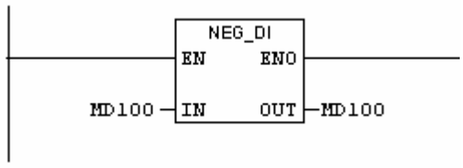


Lệnh NEG_I : Đổi dấu số nguyên 16 Bit

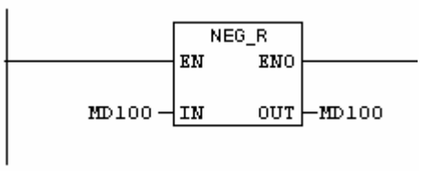
Vd: MW100 = 8 sau lệnh MW100 = -8



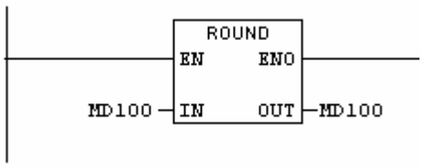
Lệnh NEG_DI : Đổi dấu số nguyên 32 Bit



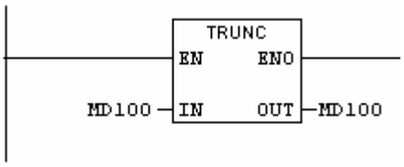
Lệnh NEG_R : Đổi dấu số thực



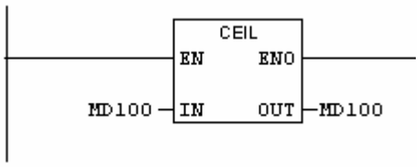
Lệnh Round : Lệnh chuyển đổi số thực thành số nguyên 32 Bit bằng cách làm tròn
Vd: MD100 = 20.35 làm tròn thành 20



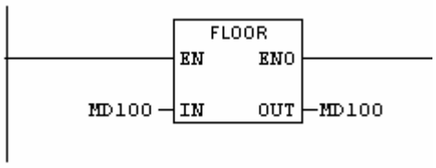
Lệnh Trunc: Lệnh chuyển đổi số thực thành số nguyên 32 Bit bằng cách cắt phần nguyên
Vd: MD100 = 20.56 chuyển thành 20



Lệnh Ceil: Lệnh chuyển đổi số thực thành số nguyên 32 Bit bằng cách làm tròn lên
Vd: MD100 = 20.04 làm tròn lên thành 21



Lệnh Floor: Lệnh chuyển đổi số thực thành số nguyên 32 Bit bằng cách làm tròn xuống
Vd: MD100 = 23.45 làm tròn xuống còn 23

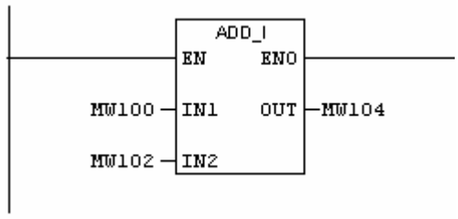


6 / Các lệnh về số học:

a/ Phép Toán trên số nguyên 16 Bit:

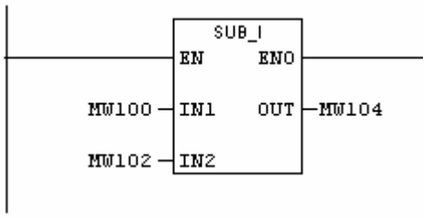
Lệnh ADD_I : Lệnh thực hiện việc cộng 2 số nguyên 16 Bit ,kết quả cất vào số nguyên 16 Bit,nếu kết quả vượt quá 16 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MW104 = MW100 + MW102



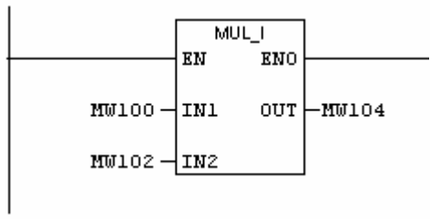
Lệnh SUB_I : Lệnh thực hiện việc trừ 2 số nguyên 16 Bit ,kết quả cất vào số nguyên 16 Bit , nếu kết quả vượt quá 16 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MW104 = MW100 - MW102



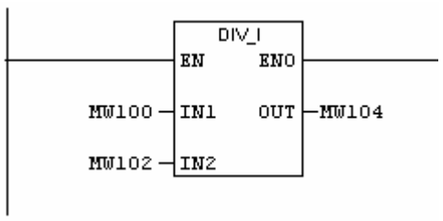
Lệnh MUL_I : : Lệnh thực hiện việc nhân 2 số nguyên 16 Bit ,kết quả cất vào số nguyên 16 Bit , nếu kết quả vượt quá 16 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MW104 = MW100 * MW102



Lệnh DIV_I : : Lệnh thực hiện việc chia 2 số nguyên 16 Bit ,kết quả cất vào số nguyên 16 Bit , nếu kết quả vượt quá 16 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

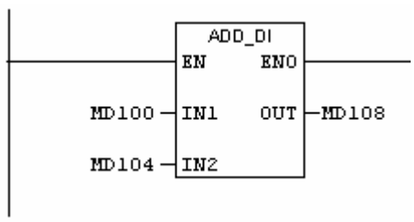
MW104 = MW100 : MW102



b/ Phép Toán trên số nguyên 32 Bit:

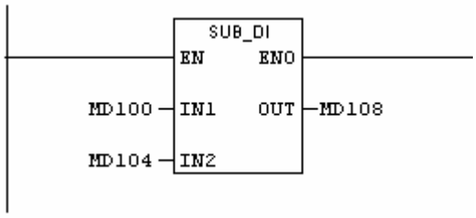
Lệnh ADD_DI : Lệnh thực hiện việc cộng 2 số nguyên 32 Bit ,kết quả cất vào số nguyên 32 Bit,nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 + MD104



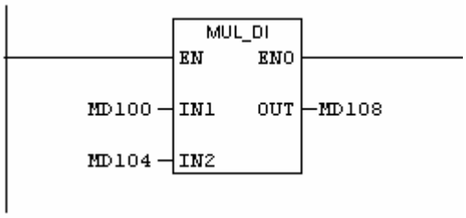
Lệnh SUB_DI : Lệnh thực hiện việc trừ 2 số nguyên 32 Bit ,kết quả cất vào số nguyên 32 Bit , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 - MD104



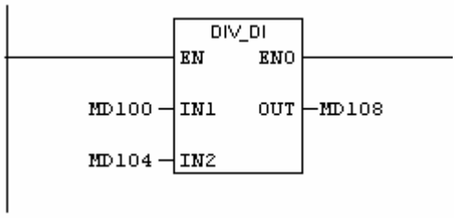
Lệnh MUL_DI : : Lệnh thực hiện việc nhân 2 số nguyên 32 Bit ,kết quả cất vào số nguyên 32 Bit , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 * MD104



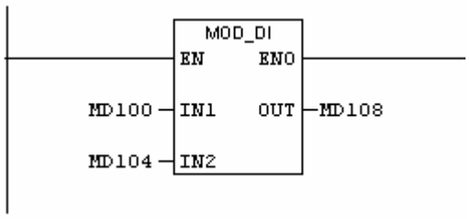
Lệnh DIV_DI : : Lệnh thực hiện việc chia 2 số nguyên 32 Bit ,kết quả cất vào số nguyên 32 Bit , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 : MD104



Lệnh MOD_DI : : Lệnh xác định phần dư của phép chia 2 số nguyên 32 Bit ,kết quả cất vào số nguyên 32 Bit , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

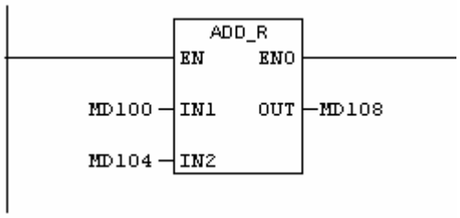
MD108 = MD100 mod MD104



c/ Phép Toán trên số nguyên 32 Bit (Floating Point Function):

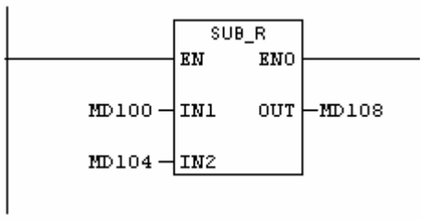
Lệnh ADD_R : Lệnh thực hiện việc cộng 2 số thực ,kết quả cất vào số thực,nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 + MD104



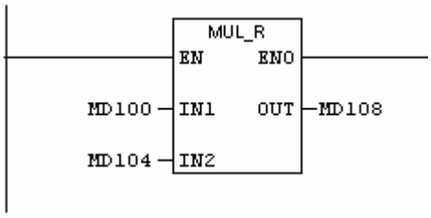
Lệnh SUB_R : Lệnh thực hiện việc trừ 2 số thực ,kết quả cất vào số thực , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 - MD104



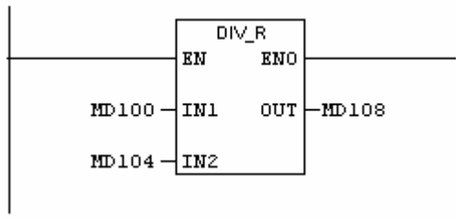
Lệnh MUL_R : : Lệnh thực hiện việc nhân 2 số thực ,kết quả cất vào số thực , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

MD108 = MD100 * MD104

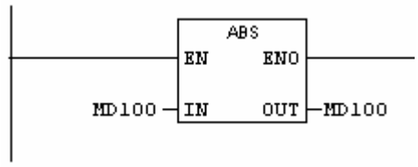


Lệnh DIV_R : : Lệnh thực hiện việc chia 2 số thực ,kết quả cất vào số thực , nếu kết quả vượt quá 32 Bit thì cờ OV sẽ bật lên 1 ,cờ OS sẽ lưu Bit bị tràn đó.

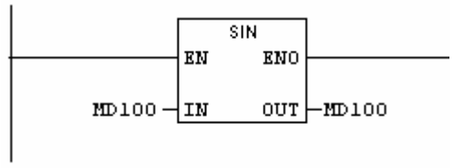
MD108 = MD100 : MD104



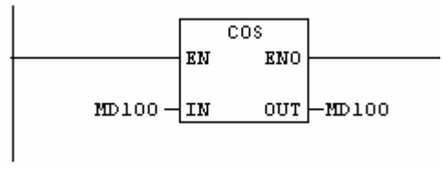
Lệnh ABS: Lệnh xác định giá trị tuyệt đối của số thực,kết quả cất vào số thực



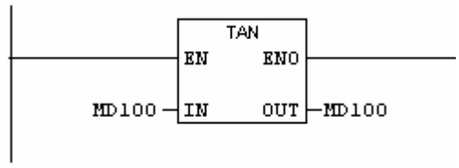
Lệnh SIN: Lệnh tính SIN của số thực,kết quả cất vào số thực.Nếu kết quả nằm ngoài khoảng [-1,1] thì cờ OV bật lên 1



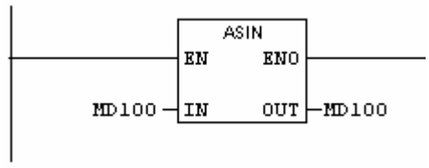
Lệnh COS: Lệnh tính COS của số thực,kết quả cất vào số thực.Nếu kết quả nằm ngoài khoảng [-1,1] thì cờ OV bật lên 1



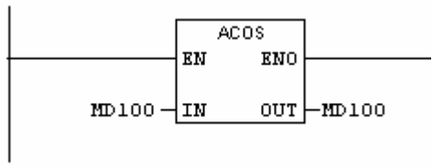
Lệnh TAN: Lệnh tính TAN của số thực,kết quả cất vào số thực.Nếu kết quả nằm ngoài khoảng 16Bit thì cờ OV bật lên 1



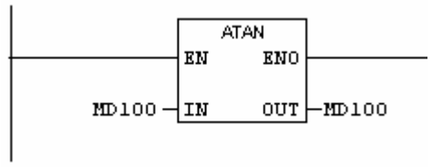
Lệnh ASIN: Lệnh tính Arcsin của số thực,số thực phải nằm trong khoảng [-1,1] kết quả là 1 số thực trong khoảng [-pi/2,pi/2]và được cất vào số thực.



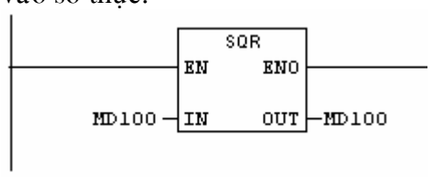
Lệnh ACOS: Lệnh tính Arccos của số thực,số thực phải nằm trong khoảng [-1,1] kết quả là 1 số thực trong khoảng [-pi,0]và được cất vào số thực.



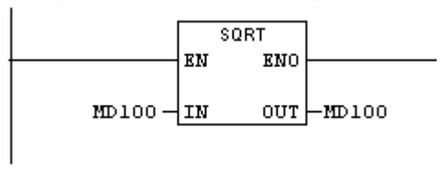
Lệnh ATAN: Lệnh tính Arctang của số thực,kết quả là 1 số thực trong khoảng [-pi/2,pi/2]và được cất vào số thực.



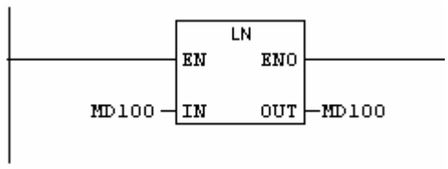
Lệnh SQR: Lệnh tính bình phương của số thực, kết quả là 1 số thực không âm được cất vào số thực.



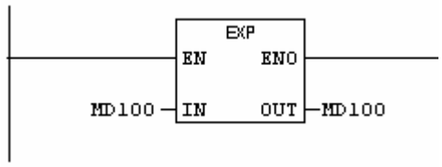
Lệnh SQRT: Lệnh tính căn bậc hai của số thực,số thực này phải là 1 số thực không âm, kết quả là 1 số thực không âm được cất vào số thực.



Lệnh Ln: Lệnh tính ln(x) của số thực,số thực này phải là 1 số thực không âm, kết quả là 1 số thực được cất vào số thực.

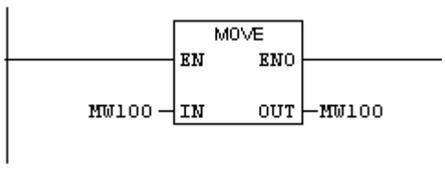


Lệnh EXP: Lệnh tính e^x của số thực, kết quả là 1 số thực không âm được cất vào số thực.



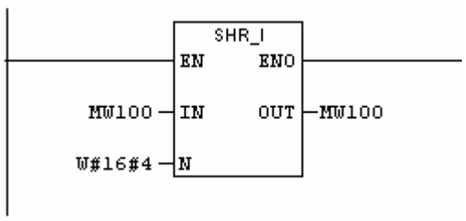
7/ Lệnh Di chuyển :

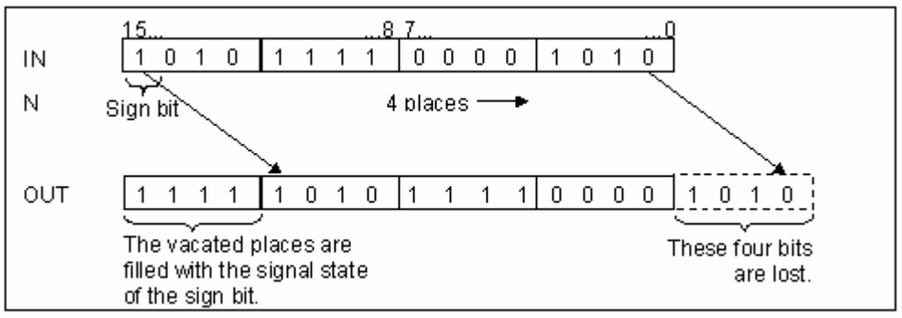
Lệnh MOV : Lệnh đưa giá trị một ô nhớ sang 1 ô nhớ khác,lệnh này có thể áp dụng cho mọi kiểu số khác nhau.(Int,Dint,Real,Byte....)



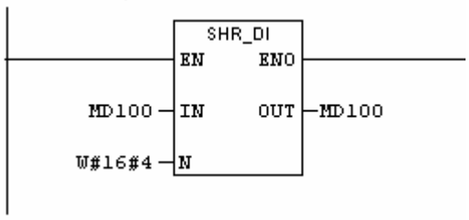
8/Lệnh Dịch Bit :

lệnh SHR_I: Lệnh thực hiện việc dịch phải ô nhớ 16Bit,kết quả cất vào ô nhớ 16 Bit,N là số Bit dịch.



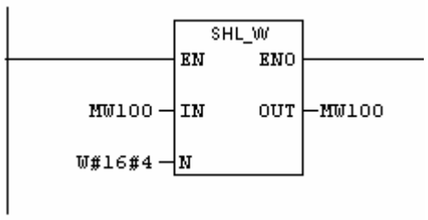


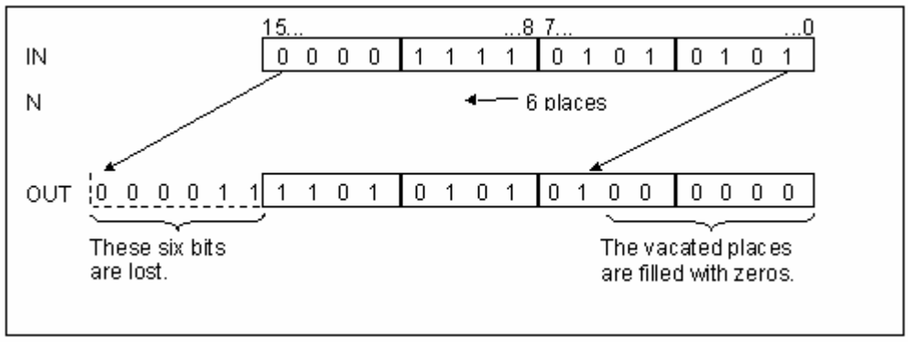
lệnh SHR_DI: Lệnh thực hiện việc dịch phải ô nhớ 32Bit,kết quả cất vào ô nhớ 32 Bit,N là số Bit dịch.



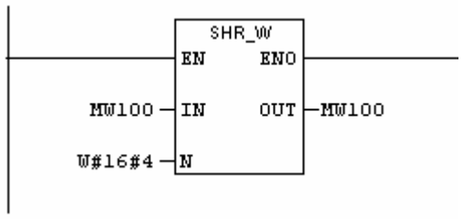
lệnh SHL_W: Lệnh thực hiện việc dịch trái ô nhớ16Bit,kết quả cất vào ô nhớ 16 Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.

Nếu N lớn hơn 16 thì MW100 =0 và cờ CC0,OV trong thanh ghi trạng thái đều bằng 0

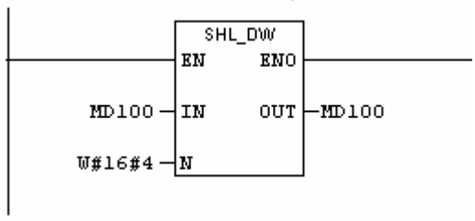




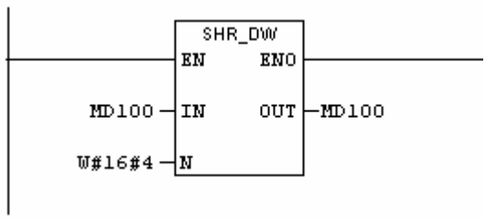
lệnh SHR_W: Lệnh thực hiện việc dịch phải ô nhớ16Bit,kết quả cất vào ô nhớ 16 Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.



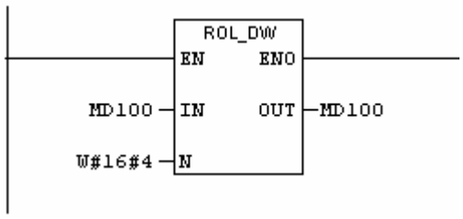
lệnh SHL_DW: Lệnh thực hiện việc dịch trái ô nhớ 32Bit,kết quả cất vào ô nhớ 32 Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.

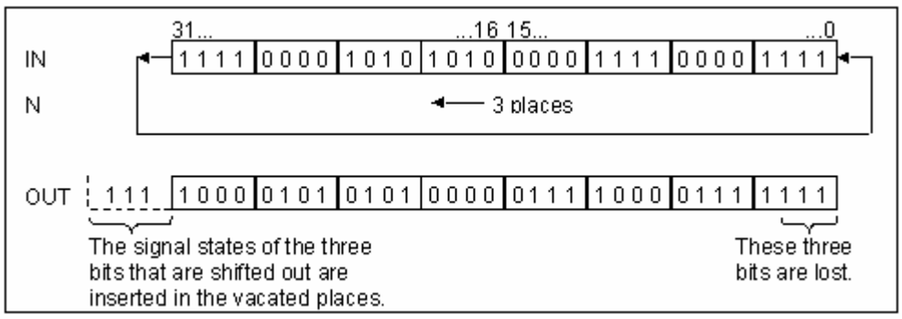


lệnh SHR_DW: Lệnh thực hiện việc dịch phải ô nhớ 32Bit,kết quả cất vào ô nhớ 32 Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.

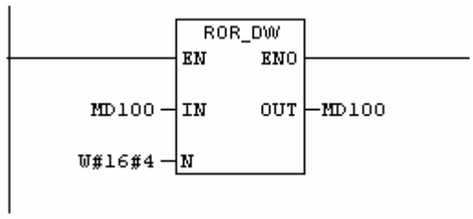


lệnh ROL_DW: Lệnh thực hiện việc dịch trái xoay tròn ô nhớ 32Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.



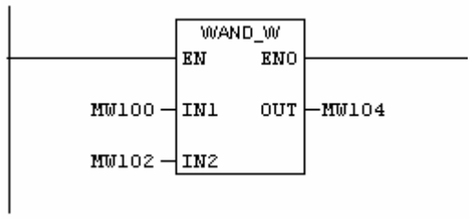


lệnh ROR_DW: Lệnh thực hiện việc dịch phải xoay tròn ô nhớ 32Bit,N là số Bit dịch. Ô nhớ này được định dạng theo kiểu Word.

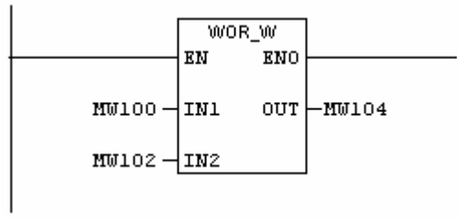


9/ Các phép tính trên Word:

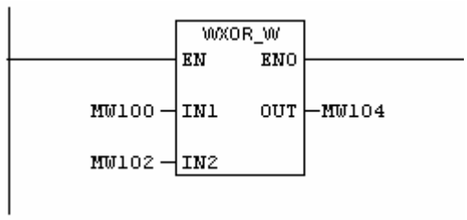
Lệnh WAND_W : Lệnh thực hiện việc giao 2 Word,kết quả được cất vào ô Word.



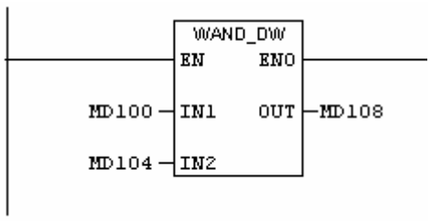
Lệnh WOR_W : Lệnh thực hiện việc hợp 2 Word,kết quả được cất vào ô Word.



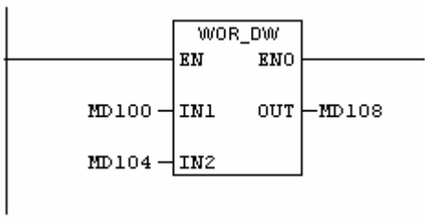
Lệnh WXOR_W : Lệnh thực hiện việc Xor 2 Word,kết quả được cất vào ô Word.



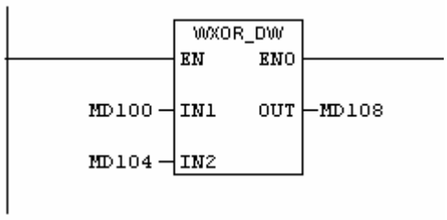
Lệnh WAND_DW : Lệnh thực hiện việc giao 2 DoubleWord,kết quả được cất vào ô DoubleWord.



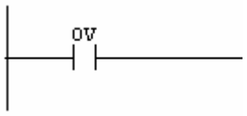
Lệnh WOR_DW : Lệnh thực hiện việc hợp 2 DoubleWord,kết quả được cất vào ô DoubleWord.

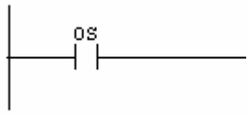


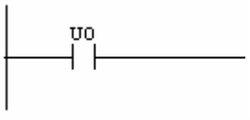
Lệnh WXOR_DW : Lệnh thực hiện việc Xor 2 DoubleWord,kết quả được cất vào ô DoubleWord.

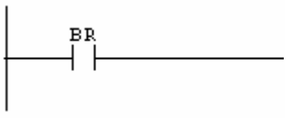


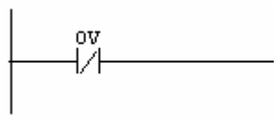
10/Các phép tính trên thanh ghi trạng thái :

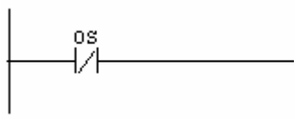


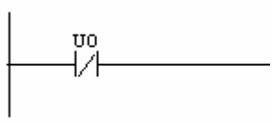


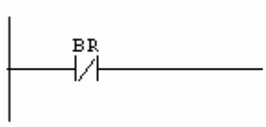




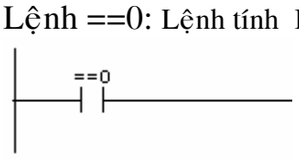




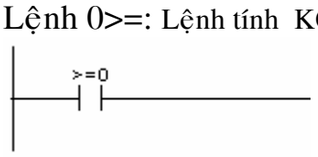




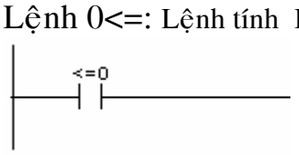
Lệnh ==0: Lệnh tính KQ:= KT + ($\overline{CC0}$^$\overline{CC1}$)



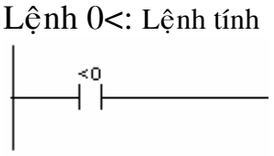
Lệnh 0>=: Lệnh tính KQ:= KT + $\overline{CC0}$



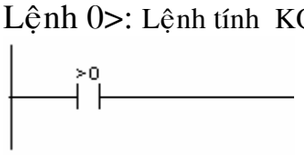
Lệnh 0<=: Lệnh tính KQ:= KT + $\overline{CC1}$



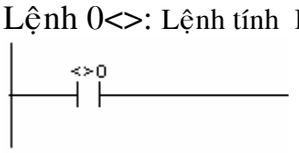
Lệnh 0<: Lệnh tính KQ:= KT + (CC0^$\overline{CC1}$)



Lệnh 0>: Lệnh tính KQ:= KT + ($\overline{CC0}$^CC1)

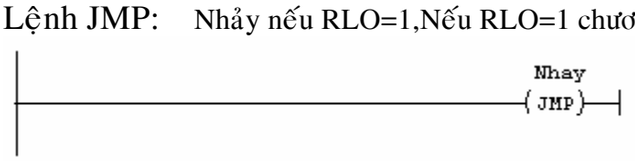


Lệnh 0<>: Lệnh tính KQ:= KT + [($\overline{CC0}$^CC1)+(CC0^$\overline{CC1}$)]

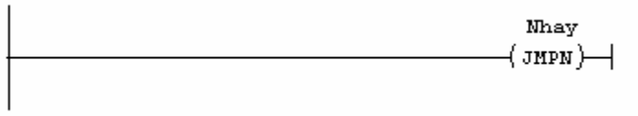


11 / Lệnh nhảy:

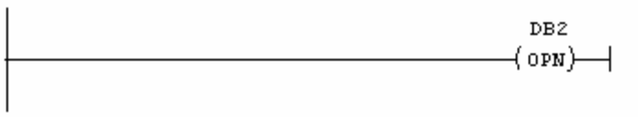
Lệnh JMP: Nhảy nếu RLO=1,Nếu RLO=1 chương trình sẽ nhảy đến nhãn "nhảy"



Lệnh JMPN: Nhảy nếu RLO=0,Nếu RLO=0 chương trình sẽ nhảy đến nhãn "nhảy"



Lệnh OPN : Lệnh mở khối DB để có thể truy cập trực tiếp tới khối này



B. Ngôn ngữ lập trình STL:

Một Chương trình viết trên LAD hoặc FBD có thể chuyển được sang STL,nhưng ngược lại thì không.Trong STL có nhiều lệnhkhông có trong LAD hay FBD.

Đối với người mới nhập môn thì ngôn ngữ LAD là ngôn ngữ dễ tiếp cận nhất,nhưng ngôn ngữ STL là hết sức cần thiết cho tương lai,do vậy chúng tôi giới thiệu thêm tập lệnh trong ngôn ngữ STL.

1/ Nhóm lệnh logic tiếp điểm:

Lệnh gán:

Cú pháp **=** < toán hạng >

Toán hạng là địa chỉ bit **I,Q,M,L,D**

Lệnh gán giá trị logic của **RLO** tới ô nhớ có địa chỉ được chỉ thị trong toán hạng

```
Ví dụ :     A     I0.0    // Đọc nội dung của I0.0 vào RLO
            =     Q0.0    // Đưa kết quả ra cổng Q0.0
```

Lệnh thực hiện phép tính giao:

Cú pháp **A** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic của toán hạng vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính giao giữa RLO với toán hạng và ghi lại kết quả vào RLO.

```
Ví dụ:      A     I0.0    // Đọc nội dung I0.0 đưa vào RLO
            A     I0.1    // Giao RLO với I0.1 kết quả đưa vào RLO
            =     Q0.0    // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép tính giao với giá trị nghịch đảo:

Cú pháp **AN** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic nghịch đảo của toán hạng vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính giao giữa RLO với giá trị nghịch đảo của toán hạng và ghi lại kết quả vào RLO.

```
Ví dụ :     A     I0.0    // Đọc nội dung của I0.0 đưa vào RLO
            AN    I0.1    // Giao RLO với giá trị nghịch đảo của I0.1 kết quả đưa vào
RLO
```

```
=       Q0.0    // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép tính hợp:

Cú pháp **O** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic của toán hạng vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính hợp giữa RLO với giá trị nghịch đảo của toán hạng và ghi lại kết quả vào RLO.

Ví dụ :
```
A       I0.0    // Đọc nội dung của I0.0 đưa vào RLO
O       I0.1    // hợp RLO với giá trị I0.1 kết quả đưa vào RLO
=       Q0.0    // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép tính hợp với giá trị nghịch đảo:

Cú pháp **ON** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic nghịch đảo của toán hạng vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính hợp giữa RLO với giá trị nghịch đảo của toán hạng và ghi lại kết quả vào RLO.

Ví dụ :
```
A       I0.0    // Đọc nội dung của I0.0 đưa vào RLO
ON      I0.1    // hợp RLO với giá trị nghịch đảo của I0.1 kết quả đưa vào RLO
=       Q0.0    // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép tính giao với giá trị một biểu thức:

Cú pháp **A** (

Lệnh không có toán hạng

Nếu FC=0 lệnh sẽ gán giá trị logic của biểu thức trong dấu ngoặc sau nó vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính giao giữa RLO với giá trị logic của biểu thức trong dấu ngoặc sau nó ghi lại kết quả vào RLO.

Ví dụ :
```
A(
O       I0.0
O       I0.1
)               // Giá trị biểu thức I0.0+I0.1 được chuyển vào RLO
A(
ON      I0.2
O       I0.3
)
=       Q0.0
```

Lệnh thực hiện phép tính hợp với giá trị một biểu thức:

Cú pháp **O** (

Lệnh không có toán hạng

Nếu FC=0 lệnh sẽ gán giá trị logic của biểu thức trong dấu ngoặc sau nó vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính hợp giữa RLO với giá trị logic của biểu thức trong dấu ngoặc sau nó ghi lại kết quả vào RLO.

Ví dụ :

```
A     M0.0
O(
O     I0.0
O     I0.1
)           // Giá trị biểu thức I0.0+I0.1 được chuyển vào RLO
=     Q0.0
```

Lệnh thực hiện phép tính giao với giá trị nghịch đảo của một biểu thức:

Cú pháp **AN (**

Lệnh không có toán hạng

Nếu FC=0 lệnh sẽ gán giá trị logic của biểu thức trong dấu ngoặc sau nó vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính giao giữa RLO với giá trị nghịch đảo logic của biểu thức trong dấu ngoặc sau đó ghi lại kết quả vào RLO.

Ví dụ :

```
AN(
O     I0.0
O     I0.1
)           // Giá trị biểu thức I0.0+I0.1 được chuyển vào RLO
=     Q0.0  // Giá trị Q0.0 bằng giá trị nghịch đảo của RLO
```

Lệnh thực hiện phép tính hợp với giá trị nghịch đảo một biểu thức:

Cú pháp **ON (**

Lệnh không có toán hạng

Nếu FC=0 lệnh sẽ gán giá trị logic nghịch đảo của biểu thức trong dấu ngoặc sau nó vào RLO.Ngược lại khi FC=1 nó sẽ thực hiện phép tính hợp giữa RLO với giá trị nghịch đảo logic nghịch đảo của biểu thức trong dấu ngoặc sau đó ghi lại kết quả vào RLO.

Ví dụ :

```
A     M0.0
ON(
O     I0.0
O     I0.1
)           // Giá trị biểu thức I0.0+I0.1 được chuyển vào RLO
=     Q0.0  //
```

Lệnh thực hiện phép exclusive or:

Cú pháp **x** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic của toán hạng vào RLO.Ngược lại khi FC=1 lệnh sẽ kiểm tra xem nội dung của RLO và giá trị logic của toán hạng có khác nhau không .Trong trường hợp khác nhau thì ghí vào RLO,ngược lại thì ghi 0.Nói cách khác ,lệnh sẽ đảo nội dung của RLO nếu toán hạng có giá trị là1.

Ví dụ :

```
A     I0.0  // Đọc nội dung của I0.0 đưa vào RLO
X     I0.1  // nghịch đảo giá trị RLO nếu I0.1 =1
=     Q0.0  // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép exclusive or not:

Cú pháp **XN** < toán hạng >

Toán hạng làdữ liệu kiểu Bool hoặc địa chỉ bit **I,Q,M,L,D,T,C**

Nếu FC=0 lệnh sẽ gán giá trị logic nghịch đảo của toán hạng vào RLO.Ngược lại khi FC=1 lệnh sẽ kiểm tra xem nội dung của RLO và giá trị logic của toán hạng có khác nhau không .Trong trường hợp khác nhau thì ghí 1 vào RLO,ngược lại thì ghi 0.Nói cách khác ,lệnh sẽ đảo nội dung của RLO nếu toán hạng có giá trị là 0.

```
Ví dụ :     A     I0.0   // Đọc nội dung của I0.0 đưa vào RLO
            XN    I0.1   // nghịch đảo giá trị RLO nếu I0.1 =0
            =     Q0.0   // Gán giá trị RLO cho Q0.0
```

Lệnh thực hiện phép exclusive or với giá trị của biểu thức:

Cú pháp **X(**

Lệnh không có toán hạng.

Nếu FC=0 lệnh sẽ gán giá trị logic của biểu thức trong dấu ngoặc vào RLO.Ngược lại khi FC=1 lệnh sẽ đảo nội dung của RLO khi biểu thức trong dấu ngoặc sau nó có giá trị 1.

Lệnh thực hiện phép exclusive or not với giá trị của biểu thức:

Cú pháp **XN(**

Lệnh không có toán hạng.

Nếu FC=0 lệnh sẽ gán giá trị logic nghịch đảo của biểu thức trong dấu ngoặc vào RLO.Ngược lại khi FC=1 lệnh sẽ đảo nội dung của RLO khi biểu thức trong dấu ngoặc sau nó có giá trị 0.

Lệnh ghi giá trị logic 1 vào RLO:

Cú pháp SET

Lệnh không có toán hạng và có tác dụng ghi 1 vào RLO

Lệnh ghi giá trị logic 0 vào RLO:

Cú pháp CLR

Lệnh không có toán hạng và có tác dụng ghi 0 vào RLO

Lệnh đảo giá trị RLO:

Cú pháp NOT

Lệnh không có toán hạng và có tác dụng đảo nội dung của RLO

Lệnh gán có điều kiện giá trị logic 1 vào ô nhớ:

Cú pháp S < toán hạng >

Toán hạng là địa chỉ Bit I,Q,M,L,D

Nếu RLO =1,lệnh sẽ ghi giá trị 1 vào ô nhớ có địa chỉ trong toán hạng

Lệnh gán có điều kiện giá trị logic 0 vào ô nhớ:

Cú pháp R < toán hạng >

Toán hạng là địa chỉ Bit I,Q,M,L,D

Nếu RLO =1,lệnh sẽ ghi giá trị 0 vào ô nhớ có địa chỉ trong toán hạng

Lệnh phát hiện sườn lên

Cú pháp FP < toán hạng >

Toán hạng là địa chỉ Bit I,Q,M,L,D và được sử dụng như một biến cờ để ghi nhận lại giá trị của RLO tại vị trí này trong chương trình ,nhưng của vòng quét trước

Tại mỗi vòng quét lệnh sẽ kiểm tra:nếu biến cờ (toán hạng)có giá trị 0 và RLO có giá trị 1 thì sẽ ghi 1 vào RLO,các trường hợp khác thì ghi 0,đồng thời chuyển nội dung của RLO vào lại biến cờ.Như vậy RLOsẽ có giá trị 1 trong vòng quét khi có sườn lên trong RLO

Nếu RLO =1,lệnh sẽ ghi giá trị 0 vào ô nhớ có địa chỉ trong toán hạng

Ví dụ: Đoạn lệnh sau:

```
A     I0.0
FP    M10.0
=Q0.0
```

Sẽ tương đương với đoạn lệnh:

```
A     I0.0
AN    M10.0
=     Q0.0
A     I0.0
=     M10.0
```

Lệnh phát hiện sườn xuống

Cú pháp FN < toán hạng >

Toán hạng là địa chỉ Bit I,Q,M,L,D và được sử dụng như một biến cờ để ghi nhận lại giá trị của RLO tại vị trí này trong chương trình ,nhưng của vòng quét trước

Tại mỗi vòng quét lệnh sẽ kiểm tra:nếu biến cờ (toán hạng)có giá trị 1 và RLO có giá trị 0 thì sẽ ghi 1 vào RLO,các trường hợp khác thì ghi 0,đồng thời chuyển nội dung của RLO vào lại biến cờ.Như vậy RLO sẽ có giá trị 1 trong vòng quét khi có sườn xuống trong RLO

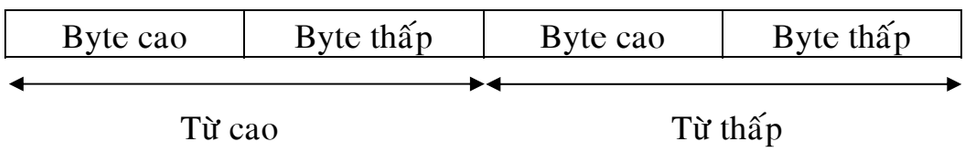
Lệnh chuyển giá trị của RLO vào BR

Cú pháp SAVE

Lệnh chuyển nội dung của RLO vào bit trạng BR.Lệnh không làm thay đổi nội dung các bit còn lại của thanh ghi trạng thái.

2/ Lệnh đọc,ghi và đảo vị trí bytes trong thanh ghi ACCU :

Các CPU của S7_300 thường có hai thanh ghi Accumulator (ACCU) kí hiệu là ACCU1 và ACCU2.Hai thanh ghi ACCU có cùng kích thước 32 bits (1 từ kép).Mọi phép tính toán trên số thực ,số nguyên,các phép tính logic với mảng nhiều bit ...đều được thực hiện trên hai thanh ghi này



Lệnh đọc vào ACCU:

Cú pháp **L** **< Toán hạng >**

Toán hạng là dữ liệu (số nguyên , thực , nhị phân) hoặc địa chỉ . Nếu là địa chỉ thì

- Byte IB,QB,PIB,MB,LB,DBB,DIB trong khoảng 0 - 255
- Từ IW,QW,PIW,MW,LW,DBW,DIW trong khoảng 0 - 2^{16} - 1
- Từ kép ID,QD,PID,MD,LD,DBD,DID trong khoảng 0 - 2^{32} – 1

Nếu là kiểu dữ liệu:

```
L     +5    : Ghi 5 vào từ thấp của ACCU1 ( số nguyên 16 Bit)
L     B#(1,8) : Ghi 1 vào Byte cao của từ thấp và ghi 8 vào Byte thấp của từ thấp
L     L#5 : Ghi 5 vào ACCU1 ( số nguyê 32 Bit)
L     B#16#2E :Dữ liệu dạng cơ số 16
L     2#10001110 : Dữ liệu dạng cơ số 2
```

L ‘AB’ :Dữ liệu dạng kí tự
L C#1000 : Dữ liệu dạng đặt trước cho bộ đếm (PV)
L S5TIME#2S : Dữ liệu dạng đặt trước cho Timer (PV)
L P#M10.2 : Dữ liệu là địa chỉ ô nhớ (dùng con trỏ)
L D#2006-1-1: Dữ liệu là giá trị về ngày/tháng /năm (16 bit)
L T#0H_1M_10S : Dữ liệu về thời gian giờ / phút /giây (32 Bit)

Lệnh L có tác dụng chuyển nội dung của ô nhớ có địa chỉ là toán hạng vào thanh ghi ACCU1 . Nội dung cũ của ACCU1 được chuyển vào ACCU2.Trong trường hợp giá trị chuyển vào nhỏ hơn từ kép thì chúng sẽ được ghivào theo thứ tự Byte thấp của từ thấp,Byte thấp của từ cao,Byte cao của từ cao.Nhũng Bit còn trống trong ACCU1 được ghi 0.

Ví dụ : Lệnh L IB0

Sẽ chuyển nội dung IB0 vào Byte thấp của từ thấp thanh ghi ACCU1

Lệnh chuyển nội dung của ACCU tới ô nhớ:

Cú pháp **T** **< Toán hạng >**

Toán hạng là địa chỉ:

- Byte IB,QB,PIB,MB,LB,DBB,DIB trong khoảng 0 - 255
- Từ IW,QW,PIW,MW,LW,DBW,DIW trong khoảng 0 - 2^{16} - 1
- Từ kép ID,QD,PID,MD,LD,DBD,DID trong khoảng 0 - 2^{32} – 1

Lệnh chuyển nội dung của ACCU1 vào ô nhớ có địa chỉ là toán hạng

Ví dụ: T VB100 // Chuyển nội dung Byte thấp của từ thấp thanh ghi ACCU1 vào ô nhớ VB100.

Lệnh đọc nội dung thanh ghi trạng thái vào ACCU1:

Cú pháp **L** **STW**

Lệnh chuyển nội dung thanh ghi trạng thái vào từ thấp của ACCU1

Lệnh ghi nội dung của ACCU1 vào thanh ghi trạng thái :

Cú pháp **T** **STW**

Lệnh chuyển 9 bits của từ thấp của ACCU1 vào thanh ghi trạng thái .

Lệnh chuyển nội dung của ACCU2 vào ACCU1 :

Cú pháp **POP**

Lệnh chuyển nội dung của ACCU2 vào ACCU1,nội dung của thanh ghi ACCU2 không đổi .

Lệnh chuyển nội dung của ACCU1 vào ACCU2 :

Cú pháp **PUSH**

Lệnh chuyển nội dung của ACCU1 vào ACCU2,nội dung của thanh ghi ACCU1 không đổi .

Lệnh đảo nội dung của hai thanh ghi ACCU1 và ACCU2 :

Cú pháp **TAK**

Lệnh chuyển nội dung của ACCU1 vào ACCU2 và ngược lại .

Lệnh đảo nội dung hai Byte của từ thấp trong thanh ghi ACCU1 :

Cú pháp **CAW**

Lệnh có tác dụng đảo nội dung hai byte của từ thấp trong thanh ghi ACCU1.

Lệnh đảo nội dung các Byte trong thanh ghi ACCU1 :

Cú pháp **CAD**

Lệnh có tác dụng đảo nội dung tất cả 4 Byte trong thanh ghi ACCU1.

Lệnh đảo giá trị các Bits trong từ thấp của thanh ghi ACCU1 :

Cú pháp **INVI**

Lệnh có tác dụng đảo nội dung tất cả các Bits trong từ thấp của thanh ghi ACCU1.

Lệnh đảo giá trị các Bits trong thanh ghi ACCU1 :

Cú pháp **INVD**

Lệnh có tác dụng đảo nội dung tất cả các Bits trong thanh ghi ACCU1.

3/ Các lệnh Logic thực hiện trên thanh ghi ACCU:

Lệnh thực hiện phép giao giữa các bits trong từ thấp của ACCU1,ACCU2:

Cú pháp: **AW [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính giao giữa các bits thuộc từ thấp của hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào từ thấp của thanh ghi ACCU1.
- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 16 bits.Khi đó lệnh thực hiện phép tính giao giữa dữ liệu với từ thấp của ACCU1.Kết quả được ghi lại vào từ thấp của ACCU1

Lệnh thực hiện phép giao giữa các bits của hai thanh ghi ACCU1,ACCU2:

Cú pháp: **AD [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính giao giữa hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào thanh ghi ACCU1.
- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 32 bits.Khi đó lệnh thực hiện phép tính giao giữa dữ liệu với thanh ghi ACCU1.Kết quả được ghi lại vào thanh ghi ACCU1

Lệnh thực hiện phép hợp giữa các bits trong từ thấp của ACCU1,ACCU2:

Cú pháp: **OW [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính hợp giữa các bits thuộc từ thấp của hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào từ thấp của thanh ghi ACCU1.
- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 16 bits.Khi đó lệnh thực hiện phép tính hợp giữa dữ liệu với từ thấp của ACCU1.Kết quả được ghi lại vào từ thấp của ACCU1

Lệnh thực hiện phép giao giữa các bits của hai thanh ghi ACCU1,ACCU2:

Cú pháp: **OD [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính hợp giữa hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào thanh ghi ACCU1.
- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 32 bits.Khi đó lệnh thực hiện phép tính hợp giữa dữ liệu với thanh ghi ACCU1.Kết quả được ghi lại vào thanh ghi ACCU1

Lệnh thực hiện phép tính exclusive or 16 bits:

Cú pháp: X**OW [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính exclusive or giữa các bits của hai từ thấp của hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào từ thấp của thanh ghi ACCU1.

- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 16 bits.Khi đó lệnh thực hiện phép tính exclusive giữa dữ liệu với từ thấp của ACCU1.Kết quả được ghi lại vào từ thấp của ACCU1

Lệnh thực hiện phép tính exclusive or 16 bits:

Cú pháp: X**OD [< Dữ liệu hằng >]**

Lệnh có thể hoặc không có toán hạng

- Nếu không có toán hạng,lệnh thực hiện phép tính exclusive or giữa các bits của hai hai thanh ghi ACCU1 và ACCU2. Kết quả ghi vào thanh ghi ACCU1.
- Nếu có toán hạng thì toán hạng phải là dữ liệu hằng 32 bits.Khi đó lệnh thực hiện phép tính exclusive giữa dữ liệu với thanh ghi ACCU1.Kết quả được ghi lại vào thanh ghi ACCU1

4/ Nhóm lệnh tăng giảm nội dung thanh ghi ACCU:

Lệnh tăng nội dung thanh ghi ACCU1:

Cú pháp **INC < Toán hạng >**

Toán hạng là số nguyên 8 bits

Lệnh thực hiện phép cộng giữa byte thấp trong ACCU1 với toán hạng .Kết quả được ghi vào byte thấp của từ thấp của ACCU1.Nội dung của các Byte khác không thay đổi.

Lệnh giảm nội dung thanh ghi ACCU1:

Cú pháp **DEC < Toán hạng >**

Toán hạng là số nguyên 8 bits

Lệnh thực hiện phép trừ giữa byte thấp trong ACCU1 với toán hạng .Kết quả được ghi vào byte thấp của từ thấp của ACCU1.Nội dung của các Byte khác không thay đổi.

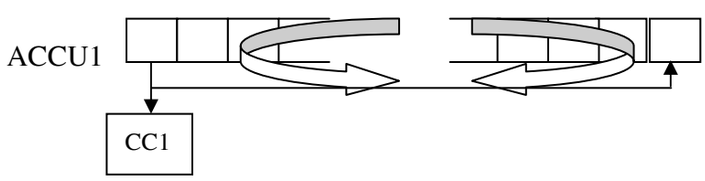
5/ Nhóm lệnh dịch chuyển nội dung thanh ghi ACCU:

Lệnh xoay tròn các bits của ACCU1 theo chiều trái.

Cú pháp **RLD [< toán hạng >]**

-Lệnh có thể có hoặc không có toán hạng là số nguyên không dấu trong khoảng 0 – 32.Khi đó lệnh thực hiện phép tính xoay tròn các Bits của ACCU1 theo chiều trái.Số Bits được xoay được chỉ định trong toán hạng .Tại mỗi lần xoay ,bit thứ 31 (bit cuối) bị đẩy ra khỏi ACCU1sẽ được ghi đồng thời vào CC1 và vào bit 0 (bit đầu tiên).Nếu toán hạng là một số 0,lệnh sẽ không làm gì cả.Nếu toán hạng bằng 32,nội dung của ACCU1không bị thay đổi và bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1. Hai bits CC0 và 0V trong thanh ghi trạng thái sẽ bằng 0 khi toán hạng là một số lớn hơn 0.

- Nếu không có toán hạng ,lệnh thực hiện phép tính xoay tròn các bits của ACCU1 theo chiều trái .Số bits được xoay tròn được chỉ thị trong byte thấp của từ thấp trong ACCU2.Tại mỗi lần xoay bit thứ 31 (bit cuối) bị đẩy ra khỏi ACCU1 sẽ được ghi đồng thời vào CC1 và vào bit thứ 0 (bit đầu tiên) .Nếu byte thấp của từ thấp trong thanh ghi ACCU2 bằng 0 thì lệnh không làm gì cả ,và nếu bằng 32 thì nội dung thanh ghi ACCU1 không bị thay đổi gì cảvà bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1.Hai bits CC0 và 0V trong thanh ghi trạng thái sẽ bằng 0 khi nội dung của byte thấp của từ thấp trong ACCU2 là một số lớn hơn 0.

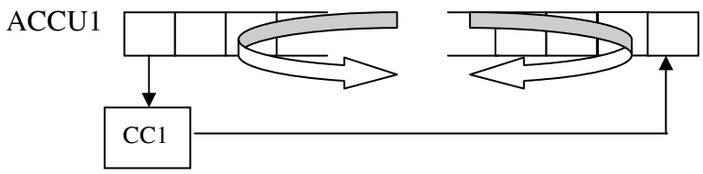


Lệnh xoay tròn ACCU1 theo chiều trái 1 bit.

Cú pháp **RLDA**

Lệnh không có toán hạng

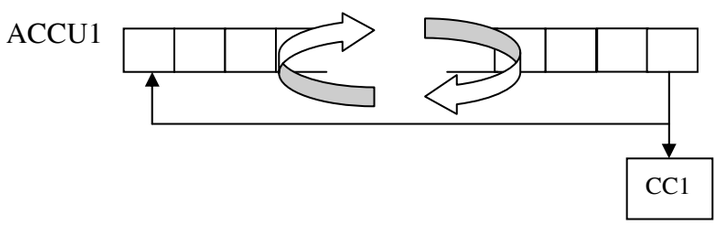
Lệnh thực hiện xoay tròn theo chiều trái 1 bit. Bit thứ 31 (bit cuối) bị đẩy ra khỏi ACCU1 được ghi vào CC1,nội dung bit CC1 được chuyển vào bit 0(bit đầu tiên).



Lệnh xoay tròn các bits của ACCU1 theo chiều phải.

Cú pháp **RDD [< toán hạng >]**

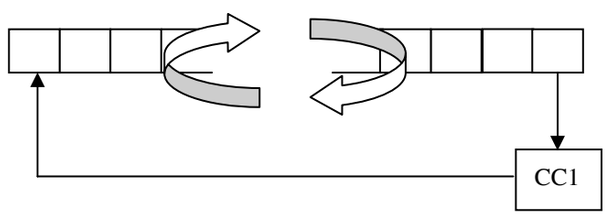
Lệnh có thể có hoặc không có toán hạng



Lệnh xoay tròn ACCU1 theo chiều phải 1 bit.

Cú pháp **RRDA**

Lệnh không có toán hạng



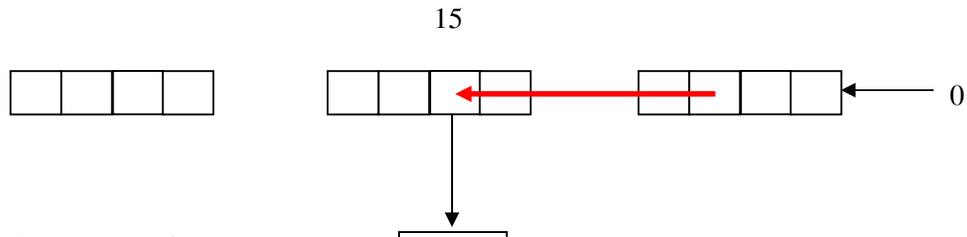
Lệnh dịch trái các bits của từ thấp của ACCU1.

Cú pháp **SLW [< toán hạng >]**

Lệnh có thể có hoặc không có toán hạng

-Nếu có toán hạng thì toán hạng là số nguyên không dấu trong khoảng 0 – 16.Khi đó lệnh thực hiện dịch trái các bits trong từ thấp của ACCU1.Số bits được dịch được chỉ thị trong toán hạng .Nội dung của từ cao trong ACCU1 không bị thay đổi .Tại 1 lần dịch ,bit thứ 15 bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1 còn bit đầu (bit thứ 0) được ghi 0.Nếu toán hạng là một số 0,lệnh sẽ không làm gì cả Nếu toán hạng bằng 16 ,nội dung của thanh ghi ACCU1 không thay đổi và bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1.Hai bit CC0 và OV sẽ bằng 0 khi toán hạng là 1 số lớn hơn 0.

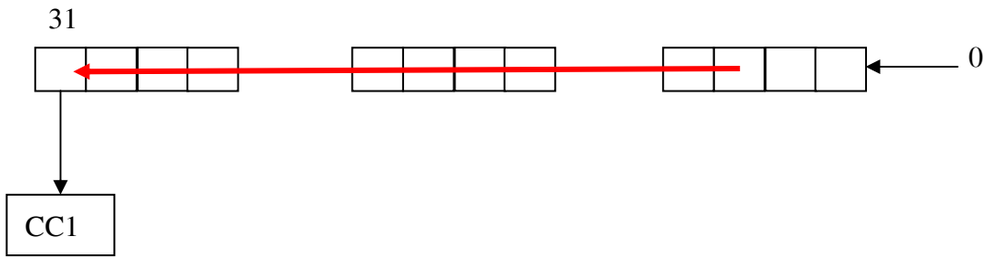
-Nếu không có toán hạng ,lệnh thực hiện phép tính dịch trái của các bit trong từ thấp của ACCU1 với số bit được dịch là nội dung của byte thấp trong từ thấp của ACCU2.Nội dung của từ cao trong ACCU1 không bị thay đổi .Tại mỗi lần dịch ,bit thứ 15 bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1,còn bit đầu (bit thứ 0) được ghi 0.Nếu byte thấp của từ thấp trong ACCU2 là một số lớn hơn 0,hai bits CC0 và OV sẽ bằng 0.



Lệnh dịch trái các bits của ACCU1.

Cú pháp **SLD [< toán hạng >]**

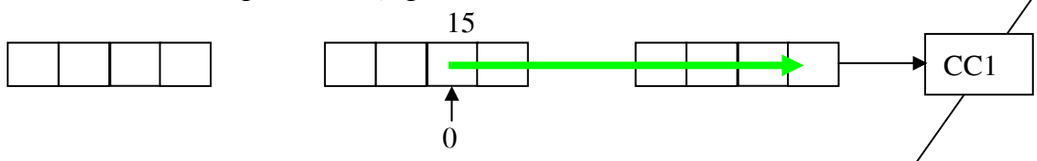
Lệnh có thể có hoặc không có toán hạng



Lệnh dịch phải các bits của từ thấp của ACCU1.

Cú pháp **SRW [< toán hạng >]**

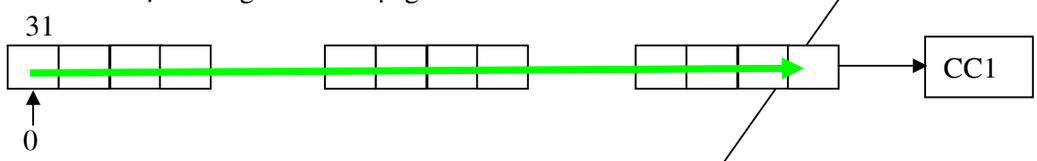
Lệnh có thể có hoặc không có toán hạng.



Lệnh dịch phải các bits của ACCU1.

Cú pháp **SRD [< toán hạng >]**

Lệnh có thể có hoặc không có toán hạng.



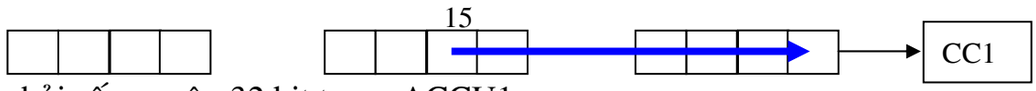
Lệnh dịch phải số nguyên 16 bit trong ACCU1.

Cú pháp **SSI [< toán hạng >]**

Lệnh có thể có hoặc không có toán hạng.

- Nếu có toán hạng thì toán hạng là số nguyên không dấu trong khoảng 0 – 16 . Khi đó lệnh thực hiện phép tính dịch phải các bits trong từ thấp của ACCU1.Số bits được dịch là toán hạng .Nội dung của từ cao trong ACCU1 không bị thay đổi .Tại mỗi lần dịch ,bit 0 (bit đầu) bị đẩy từ ACCU1 sang CC1còn bit thứ 15 được ghi lại đúng bằng giá trị cũ của nó .Nếu toán hạng là số lớn hơn 0 ,hai bits CC0 và OV sẽ bằng 0
- Nếu không có toán hạng ,lệnh thực hiện phép tính dịch phải các bits trong từ thấp của ACCU1 .Số bit được dịch là nội dung của byte thấp trong từ thấp của ACCU2.Nội dung của từ cao trong

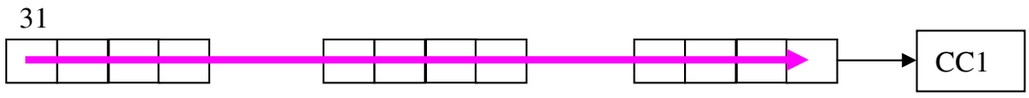
ACCU1 không bị thay đổi .Tại mỗi lần dịch ,bit thứ 0 bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1 ,bit thứ 15 được ghi lại đúng bằng giá trị cũ của nó .Nếu byte thấp của từ thấp trong ACCU2 là một số lớn hơn 0 ,hai bits CC0 và OV sẽ bằng 0



Lệnh dịch phải số nguyên 32 bit trong ACCU1.

Cú pháp **SSD [< toán hạng >]**

Lệnh có thể có hoặc không có toán hạng.



6 / Nhóm lệnh so sánh số nguyên 16 bits:

Trong tất cả những lệnh so sánh hai số nguyên 16 bits nằm trong 2 từ thấp của 2 thanh ghi ACCU1 và ACCU2 được trình bày sau đây đều tác động vào thanh ghi trạng thái như sau:

CC1	*CC0*	*Ý Nghĩa*
0	***0***	***Từ thấp ACCU2= từ thấp ACCU1***
0	***1***	***Từ thấp ACCU1< từ thấp ACCU1***
1	***0***	***Từ thấp ACCU> từ thấp ACCU1***

Lệnh so sánh 2 số nguyên 16 bits:

Cú pháp : **= = I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU1 có nội dung giống như số nguyên trong từ thấp của ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh không bằng nhau 2 số nguyên 16 bits:

Cú pháp : **<> I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU1 có nội dung khác số nguyên trong từ thấp của ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn 2 số nguyên 16 bits:

Cú pháp : **>I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU2 lớn hơn số nguyên trong từ thấp của ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh nhỏ hơn 2 số nguyên 16 bits:

Cú pháp : **< I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU2 nhỏ hơn số nguyên trong từ thấp của ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn hoặc bằng 2 số nguyên 16 bits:

Cú pháp : **>= I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU2 lớn hơn hoặc bằng số nguyên trong từ thấp của ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh bé hơn hoặc bằng 2 số nguyên 16 bits:

Cú pháp : **<= I**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 16 bits nằm trong hai từ thấp trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong từ thấp của ACCU2 bé hơn hoặc bằng số nguyên trong từ thấp của ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

7 / Nhóm lệnh so sánh số nguyên 32 bits:

Trong tất cả những lệnh so sánh hai số nguyên 32 bits nằm trong 2 thanh ghi ACCU1 và ACCU2 được trình bày sau đây đều tác động vào thanh ghi trạng thái như sau:

CC1	*CC0*	*Ý Nghĩa*
0	*0*	*ACCU2= ACCU1*
0	*1*	*ACCU2< ACCU1*
1	*0*	*ACCU2> ACCU1*

Lệnh so sánh 2 số nguyên 32 bits:

Cú pháp : **= = D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU1 có nội dung giống như số nguyên trong thanh ghi ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh không bằng nhau 2 số nguyên 32 bits:

Cú pháp : **<> D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU1 có nội dung khác số nguyên trong thanh ghi ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn 2 số nguyên 32 bits:

Cú pháp : **>D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU2 lớn hơn số nguyên trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh nhỏ hơn 2 số nguyên 32 bits:

Cú pháp : **< D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU2 nhỏ hơn số nguyên trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn hoặc bằng 2 số nguyên 32 bits:

Cú pháp : **>= D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU2 lớn hơn hoặc bằng số nguyên trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh bé hơn hoặc bằng 2 số nguyên 32 bits:

Cú pháp : **<= D**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số nguyên 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số nguyên trong thanh ghi ACCU2 bé hơn hoặc bằng số nguyên trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

8 / Nhóm lệnh so sánh số thực 32 bits:

Trong tất cả những lệnh so sánh hai số thực 32 bits nằm trong 2 thanh ghi ACCU1 và ACCU2 được trình bày sau đây đều tác động vào thanh ghi trạng thái như sau:

CC1	*CC0*	*Ý Nghĩa*
0	*0*	*ACCU2= ACCU1*
0	*1*	*ACCU2< ACCU1*
1	*0*	*ACCU2> ACCU1*

Lệnh so sánh 2 số thực 32 bits:

Cú pháp : **= = R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU1 có nội dung giống như số thực trong thanh ghi ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh không bằng nhau 2 số thực 32 bits:

Cú pháp : **<> R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU1 có nội dung khác số thực trong thanh ghi ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn 2 số thực 32 bits:

Cú pháp : **>R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU2 lớn hơn số thực trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh nhỏ hơn 2 số thực 32 bits:

Cú pháp : **< R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU2 nhỏ hơn số thực trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh lớn hơn hoặc bằng 2 số thực 32 bits:

Cú pháp : **>= R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU2 lớn hơn hoặc bằng số thực trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

Lệnh so sánh bé hơn hoặc bằng 2 số thực 32 bits:

Cú pháp : **<= R**

Lệnh không có toán hạng .

Lệnh thực hiện phép so sánh hai số thực 32 bits nằm trong hai thanh ghi ACCU1 và ACCU2.Nếu số thực trong thanh ghi ACCU2 bé hơn hoặc bằng số thực trong thanh ghi ACCU1 thì bit trạng thái RLO sẽ nhận giá trị 1,ngược lại sẽ có giá trị 0.

9 / Các lệnh toán học:

Tất cả những lệnh toán học thực hiện với nội dung hai thanh ghi ACCU1 và ACCU2 được trình bày sau đây đều tác động vào thanh ghi trạng thái như sau:

CC1	*CC0*	*Ý Nghĩa*
0	*0*	***Kết quả bằng 0 (= 0)***
0	*1*	***Kết quả nhỏ hơn 0 (< 0)***
1	*0*	***Kết quả lớn hơn 0 (> 0)***

a/ Nhóm lệnh làm việc với số nguyên 16 bits:

Lệnh cộng:

Cú pháp **+ I**

Lệnh thực hiện phép cộng hai số nguyên nằm trong từ thấp của ACCU1 và ACCU2.Kết quả được ghi lại vào từ thấp của ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm – 32768 – 32767 thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh trừ:

Cú pháp **- I**

Lệnh thực hiện phép trừ hai số nguyên nằm trong từ thấp của ACCU1 và ACCU2.Kết quả được ghi lại vào từ thấp của ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm – 32768 – 32767 thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh nhân:

Cú pháp *** I**

Lệnh thực hiện phép nhân hai số nguyên nằm trong từ thấp của ACCU1 và ACCU2.Kết quả là số nguyên 32 Bits được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm – 32768 – 32767 thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh chia:

Cú pháp **/ I**

Lệnh thực hiện phép chia hai số nguyên nằm trong từ thấp của ACCU2 cho từ thấp của ACCU1.Kết quả được ghi lại vào từ thấp của ACCU1,phần dư được ghi vào từ cao thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm – 32768 – 32767 thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

b/ Nhóm lệnh làm việc với số nguyên 32 bits:

Lệnh cộng:

Cú pháp **+ D**

Lệnh không có toán hạng

Lệnh thực hiện phép cộng hai số nguyên 32 bit nằm trong hai thanh ghi ACCU1 và ACCU2.Kết quả được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh trừ:

Cú pháp **- D**

Lệnh không có toán hạng

Lệnh thực hiện phép trừ hai số nguyên 32 bit nằm trong hai thanh ghi ACCU1 và ACCU2.Kết quả được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh nhân:

Cú pháp *** D**

Lệnh thực hiện phép nhân hai số nguyên 32 bit trong hai thanh ghi ACCU1 và ACCU2.Kết quả là số nguyên 32 Bits được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh chia:

Cú pháp **/ D**

Lệnh thực hiện phép chia hai số nguyên 32 bit trong ACCU2 cho số nguyên 32 bit trong thanh ghi ACCU1.Kết quả là số nguyên 32 bit sẽ được ghi lại vào thanh ghi ACCU1.Nếu giá trị vượt tầm(– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh lấy phần dư:

Cú pháp **MOD**

Lệnh không có toán hạng và xác định phần dư của phép chia số nguyên 32 bit trong ACCU2 cho số nguyên 32 bit trong ACCU1.Kết quả là số nguyên 32 bit được ghi lại vào ACCU1. Nếu giá trị vượt tầm(– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

c/ Nhóm lệnh làm việc với số thực:

Lệnh cộng:

Cú pháp **+ R**

Lệnh không có toán hạng

Lệnh thực hiện phép cộng hai số thực nằm trong hai thanh ghi ACCU1 và ACCU2.Kết quả được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh trừ:

Cú pháp **- R**

Lệnh không có toán hạng

Lệnh thực hiện phép trừ hai số thực nằm trong hai thanh ghi ACCU1 và ACCU2.Kết quả được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh nhân:

Cú pháp *** R**

Lệnh thực hiện phép nhân hai số thực trong hai thanh ghi ACCU1 và ACCU2.Kết quả là số thực được ghi lại vào thanh ghi ACCU1.Nội dung của thanh ghi ACCU2 không bị thay đổi .Nếu giá trị vượt tầm (– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh chia:

Cú pháp **/ R**

Lệnh thực hiện phép chia hai thực trong ACCU2 cho số thực trong thanh ghi ACCU1.Kết quả là số thực sẽ được ghi lại vào thanh ghi ACCU1.Nếu giá trị vượt tầm(– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh lấy phần dư:

Cú pháp **MOD**

Lệnh không có toán hạng và xác định phần dư của phép chia số nguyên 32 bit trong ACCU2 cho số nguyên 32 bit trong ACCU1.Kết quả là số nguyên 32 bit được ghi lại vào ACCU1. Nếu giá trị vượt tầm(– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh lấy giá trị tuyệt đối:

Cú pháp **ABS**

Lệnh không có toán hạng và xác định giá trị tuyệt đối của số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1.Đặc biệt lệnh này không làm thay đổi nội dung của các bit trạng thái.

Lệnh tính Sin:

Cú pháp **SIN**

Lệnh không có toán hạng và xác định sin của số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1.

Lệnh tính Cos:

Cú pháp **COS**

Lệnh không có toán hạng và xác định cos của số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1.

Lệnh tính Tan:

Cú pháp **TAN**

Lệnh không có toán hạng và xác định tang của số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1. Nếu giá trị vượt tầm(– 2147483648 , 2147483648) thì hai bit OV và OS sẽ cùng nhận giá trị là 1.

Lệnh tính Arsin:

Cú pháp **ASIN**

Lệnh không có toán hạng và xác định arcsin của số thực trong ACCU1,số thực này phải nằm trong khoảng (-1,1).Kết quả là một số thực trong khoảng (-pi/2,pi/2)sẽ được ghi lại vào ACCU1.

Lệnh tính Arcos:

Cú pháp **ACOS**

Lệnh không có toán hạng và xác định arccos của số thực trong ACCU1,số thực này phải nằm trong khoảng (-1,1).Kết quả là một số thực trong khoảng (-pi,0)sẽ được ghi lại vào ACCU1.

Lệnh tính Artg:

Cú pháp **ATAN**

Lệnh không có toán hạng và xác định arctg của số thực trong ACCU1.Kết quả là một số thực trong khoảng (-pi/2,pi/2)sẽ được ghi lại vào ACCU1.

Lệnh tính bình phương:

Cú pháp **SQR**

Lệnh không có toán hạng và xác định giá trị bình phương của số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1.

Lệnh tính căn bậc hai:

Cú pháp **SQRT**

Lệnh không có toán hạng và xác định căn bậc hai của số thực trong ACCU1,số thực này phải là số thực không âm.Kết quả là một số thực không âm sẽ được ghi lại vào ACCU1.

Lệnh đảo dấu:

Cú pháp **NERG**

Lệnh không có toán hạng và có tác dụng đổi dấu số thực trong ACCU1.Kết quả sẽ được ghi lại vào ACCU1.Đặc biệt lệnh này không làm thay đổi nội dung của các bit trạng thái.

10 / Lệnh đổi kiểu dữ liệu:

Trong ngôn ngữ lập trình STL của S7_300 có nhiều dạng dữ liệu khác nhau như:

- Số nguyên 16 Bits.
- Số nguyên 32 Bits
- Số nguyên dạng BCD
- Số thực dấu phẩy động
- Và một số dạng dữ liệu khác

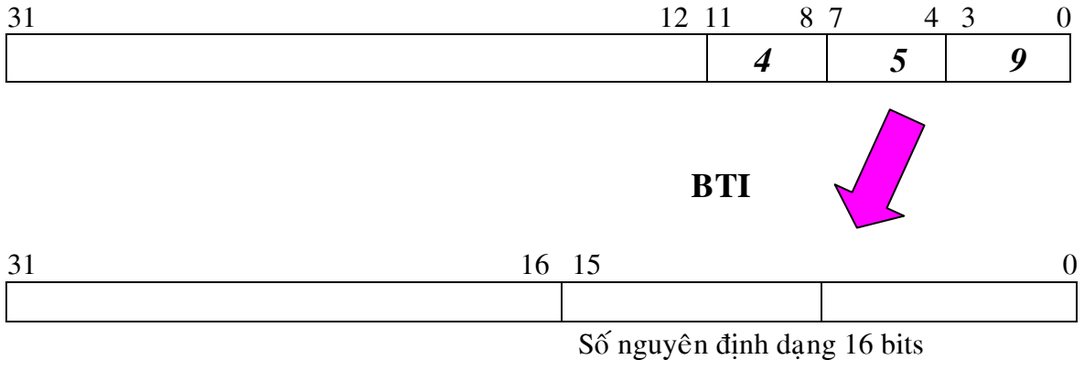
Việc làm với nhiều dạng dữ liệu khác nhau đặt ra cho ta vấn đề cần phải chuyển đổi chúng .Ví dụ khi đọc tín hiệu tương tự từ cổng tương tự ta nhận được số liệu dạng nguyên 16 bits mang giá trị tín hiệu tương tự chứ không phải bản thân giá trị đó,bởi vậy để xử lý tiếp thì cần thiết phải chuyển số nguyên đó thành đúng giá trị thực,dấu phẩy động của tín hiệu tượng tự ở cổng.

a/ Chuyển đổi số BCD thành số nguyên và ngược lại:

Lệnh chuyển đổi BCD thành số nguyên 16 bits:

Cú pháp **BTI**

Lệnh không có toán hạng và chuyển đổi một số BCD có 3 chữ số nằm trong 12 Bits đầu của ACCU1 thành số nguyên 16 bits.Kết quả được cất lại vào 16 bits cuối (từ thấp) của ACCU1.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái.

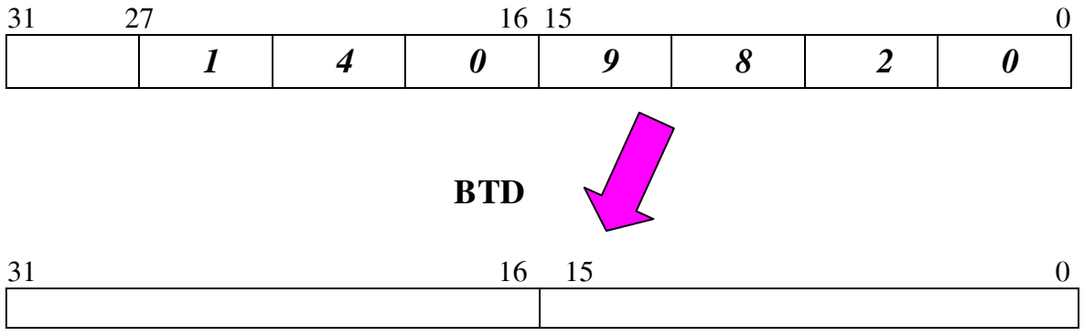


Nếu số BCD cần chuyển đổi có cấu trúc sai ,ví dụ như có 1 chữ số 4 bits nhị phân không nằm trong khoảng từ 0 đến 9,CPU sẽ gọi chương trình ngắt xử lí lỗi OB121 hoặc chuyển qua chế độ Stop (nếu OB121 không có chương trình).

Lệnh chuyển đổi BCD thành số nguyên 32 bits:

Cú pháp **BTD**

Lệnh không có toán hạng và chuyển đổi một số BCD có 7 chữ số nằm trong 28 Bits đầu của ACCU1 thành số nguyên 32 bits.Kết quả được cất lại vào thanh ghi ACCU1.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái.



Số nguyên định dạng 32 bits

Nếu số BCD cần chuyển đổi có cấu trúc sai ,ví dụ như có 1 chữ số 4 bits nhị phân không nằm trong khoảng từ 0 đến 9,CPU sẽ gọi chương trình ngắt xử lí lỗi OB121 hoặc chuyển qua chế độ Stop (nếu OB121 không có chương trình).

Lệnh chuyển đổi số nguyên 16 bits thành số BCD:

Cú pháp **ITB**

Lệnh không có toán hạng và chuyển đổi một số nguyên 16 bits thành số BCD có 3 chữ số.Kết quả được cất lại vào từ thấp của thanh ghi ACCU1. Nếu số nguyên 16 bits cần chuyển đổi có giá trị tuyệt đối lớn hơn 999 ,CPU sẽ thông báo trong thanh ghi trạng thái dưới dạng kết quả tràn

Lệnh chuyển đổi số nguyên 32 bits thành số BCD:

Cú pháp **DTB**

Lệnh không có toán hạng và chuyển đổi một số nguyên 32 bits thành số BCD có 7 chữ số.Kết quả được cất lại vào thanh ghi ACCU1. Nếu số nguyên 32 bits cần chuyển đổi có giá trị tuyệt đối lớn hơn 9999999 ,CPU sẽ thông báo trong thanh ghi trạng thái dưới dạng kết quả tràn

b/ Chuyển đổi số nguyên 16 bits thành số nguyên 32 bits:

Cú pháp : **ITD**

Lệnh không có toán hạng và thực hiện việc chuyển đổi một số nguyên 16 bits trong từ thấp của ACCU1 thành số nguyên 32 bits .Kết quả được cất vào lại ACCU1.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái.

c/ Chuyển đổi số nguyên 32 bits thành số thực:

Cú pháp : **DTR**

Lệnh không có toán hạng và thực hiện việc chuyển đổi một số nguyên 32 bits trong thanh ghi ACCU1 thành số thực .Kết quả được cất vào lại ACCU1.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái.

d/ Chuyển đổi số thực thành số nguyên 32 bits:

Lệnh làm tròn:

Cú pháp : **RND**

Lệnh không có toán hạng và thực hiện việc làm tròn số thực nằm trong thanh ghi ACCU1.Kết quả là số nguyên 32 bits được cất vào lại ACCU1.

Lệnh làm tròn lên:

Cú pháp : **RND+**

Lệnh không có toán hạng và thực hiện việc làm tròn lên số thực nằm trong thanh ghi ACCU1.Kết quả là số nguyên 32 bits được cất vào lại ACCU1.

Lệnh làm tròn xuống:

Cú pháp : **RND -**

Lệnh không có toán hạng và thực hiện việc làm tròn xuống số thực nằm trong thanh ghi ACCU1.Kết quả là số nguyên 32 bits được cất vào lại ACCU1.

Lệnh lấy phần nguyên:

Cú pháp : **TRUNC**

Lệnh không có toán hạng và thực hiện việc lấy phần nguyên số thực nằm trong thanh ghi ACCU1.Kết quả là số nguyên 32 bits được cất vào lại ACCU1.

11/ Các lệnh điều khiển chương trình:

a/ Nhóm lệnh kết thúc chương trình:

Lệnh kết thúc vô điều kiện:

Cú pháp : **BEU**

Lệnh không có toán hạng và thực hiện kết thúc chương trình trong khối một cách vô điều kiện.

Lệnh kết thúc có điều kiện:

Cú pháp : **BEC**

Lệnh không có toán hạng và thực hiện kết thúc chương trình trong khối nếu như RLO có giá trị bằng 1.

b/ Nhóm lệnh rẽ nhánh theo bit trạng thái:

Lệnh rẽ nhánh theo bit trạng thái là loại lệnh thực hiện bước nhảy nhằm bỏ qua một đoạn chương trình để tới đoạn chương trình khác được đánh dấu bằng nhãn nếu điều kiện kiểm tra trong thanh ghi trạng thái được thoả mãn .Nơi lệnh nhảy tới phải thuộc cùng một khối chương trình với lệnh.Không thể nhảy từ một khối chương trình này sang một khối chương trình khác,ví dụ không thể nhảy từ khối FC1 sang FC10.

Nhãn là một dãy với nhiều nhất 4 kí tự hoặc số và phải được bắt đầu bằng một kí tự.Khoảng cách bước nhảy tính theo ô nhớ chương trình ,phải ít hơn 32767 từ .Nơi nhảy đến có thể nằm trước hay nằm sau lệnh nhảy.

Rẽ nhánh khi BR=1

Cú pháp: **JBI < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit **BR** trong thanh ghi trạng thái bằng 1

Rẽ nhánh khi BR=0

Cú pháp: **JNBI < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit **BR** trong thanh ghi trạng thái bằng 0

Rẽ nhánh khi RLO=1

Cú pháp: **JC < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit **RLO** trong thanh ghi trạng thái bằng 1

Rẽ nhánh khi RLO=0

Cú pháp: **JCN < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit **RLO** trong thanh ghi trạng thái bằng 0

Rẽ nhánh khi CC1=0 và CC0=1

Cú pháp: **JM < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 = 0 và CC0 = 1).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả âm.

Rẽ nhánh khi CC1=1 và CC0=0

Cú pháp: **JP < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 = 1 và CC0 = 0).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả dương.

Rẽ nhánh khi CC1=0 và CC0=0

Cú pháp: **JZ < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 = 0 và CC0 = 0).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả bằng 0.

Rẽ nhánh khi CC1 khác CC0

Cú pháp: **JN < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 khác CC0).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả khác 0.

Rẽ nhánh khi CC1=CC0=0 hoặc (CC1=0 và CC0=1)

Cú pháp: **JMZ < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 =CC0= 0 hoặc (CC1=0 và CC0 = 1)).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả là một số không dương.

Rẽ nhánh khi CC1=CC0=0 hoặc (CC1=1 và CC0=0)

Cú pháp: **JPZ < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn nếu như Bit (**CC1 =CC0= 0 hoặc (CC1=1 và CC0 = 0)).**Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả là một số không âm.

Rẽ nhánh vô điều kiện

Cú pháp: **JU < nhãn >**

Lệnh sẽ thực hiện việc nhảy tới Nhãn vô điều kiện không phụ thuộc vào bit trạng thái nào.

12/ Lệnh xoay vòng:

Cú pháp **LOOP < nhãn >**

Khi gặp lệnh LOOP ,CPU của S7_300 sẽ tự giảm nội dung của từ thấp trong thanh ghi ACCU1 đi một đơn vị và kiểm tra xem kết quả có bằng 0 hay không .Nếu kết quả khác 0 ,CPU sẽ thực hiện bước nhảy đến đoạn chương trình được đánh dấu bởi "nhãn" .Ngược lại CPU sẽ thực hiện lệnh kế tiếp.

Lệnh xoay vòng này có thể được sử dụng để mô phỏng nguyên tắc làm việc giống như lệnh For...của C bằng cách thực hiện bước nhảy ngược.Đoạn chương trình nằm giữa nhãn và lệnh LOOP sẽ được thực hiện cho tới khi nội dung thanh ghi ACCU1 bằng 0

Lệnh không làm thay đổi nội dung của thanh ghi trạng thái

13/ Bộ thời gian (Timer):

a/Khai báo sử dụng:

Việc khai báo sử dụng Timer bao gồm các bước :

-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

-Khai báo loại Timer được sử dụng (SD,SS,SP,SE,SF).

-Khai báo tín hiệu xoá Timer nếu muốn sử dụng chế độ reset chủ động.

Trong tất cả 5 bước trên,các bước 2,3,4 là bắt buộc

i/Khai báo tín hiệu enable (chủ động kích)

Cú pháp **A** **< Địa chỉ bit >**

FR **< tên Timer >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu sẽ được sử dụng làm tín hiệu chủ động kích cho Timer có tên cho trong toán hạng thứ hai.

ii/Khai báo tín đầu vào

Cú pháp **A** **< Địa chỉ bit >**

" địa chỉ bit" trong toán hạng xác định đầu vào u(t) cho Timer.

iii/Khai báo thời gian trễ mong muốn:

Cú pháp **L** **< hằng số >**

"Hằng số" trong toán hạng xác định giá trị thời gian trễ T đặt trước cho Timer .Hằng số này có 2 dạng

-S5T#giờH_phútM_giâyS_miligiâyMS.Đây là dạng dữ liệu thời gian trực tiếp.

-Dạng một số nguyên 16 bits có cấu trúc như sau:

		1	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***1***	***0***	***0***	***1***	***1***	***1***
		1s			1			2			7				

iv/Khai báo loại Timer:

S7_300 có 5 loại timer được khai báo bằng các lệnh :

- SD: trễ có sườnlên không có nhớ
- SS: Trễ theo sườn lên có nhớ
- SP: Tạo xung không có nhớ
- SE: Tạo xung có nhớ
- SF : Trễ theo sườn xuống.

a/Trễ theo sườn lên không nhớ (On delay timer):

Cú pháp **SD** **< tên timer >**

Thời gian giữ trễ được bắt đầu khi có sườn lên của tín hiệu đầu vào (hoặc có sườn lên của tínhiệu enable đồng thời tín hiệu vào bằng 1),tức thời ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T-WORD (CV) .Trong khoảng thời gian trễ T-bit có giá trị bằng 1 .Như vậy T-bit có giá trị bắng khi T-Word = 0

Khoảng thời gian trễ chính là khoảng thời gian giữa thời điểm xuất hiện sườn lên của tín hiệu vào và sườn lên của T-bit

Khi tín hiệu vào bằng 0 ,T-bit và T-Word cùng nhận giá trị 0

b/Trễ theo sườn lên có nhớ (On delay timer):

Cú pháp **SS** **< tên timer >**

Thời gian giữ trễ được bắt đầu khi có sườn lên của tín hiệu đầu vào (hoặc có sườn lên của tínhiệu enable đồng thời tín hiệu vào bằng 1),tức thời ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T-WORD (CV) .Khi hết thời gian trễ ,tức là T-Word bằng 0 ,T-bit có giá trị 1

Khoảng thời gian trễ chính là khoảng thời gian giữa thời điểm xuất hiện sườn lên của tín hiệu vào và sườn lên của T-bit

Với bộ timer có nhớ ,thời gian trễ vẫn được tính cho dù lúc đó tín hiệu đầu vào đã về 0

c/Timer tạo xung không nhớ (Pulse timer):

Cú pháp **SP** **< tên timer >**

Thời gian giữ trễ được bắt đầu khi có sườn lên của tín hiệu đầu vào (hoặc có sườn lên của tínhiệu enable đồng thời tín hiệu vào bằng 1),tức thời ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T-WORD (CV) .Trong khoảng thời gian trễ ,tức là khi T-Word #0,T-bit có giá trị bằng 1.Ngoài khoảng thời gian trễ T-bit có giá trị bằng 0.

Nếu chưa hết thời gian trễ mà tín hiệu đầu vào về 0 thì T-bit và T-Word cũng về giá trị 0

d/Timer tạo xung có nhớ (Extended Pulse timer):

Cú pháp **SE** **< tên timer >**

Thời gian giữ trễ được bắt đầu khi có sườn lên của tín hiệu đầu vào (hoặc có sườn lên của tínhiệu enable đồng thời tín hiệu vào bằng 1),tức thời ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T-WORD (CV) .Trong khoảng thời gian trễ ,tức là khi T-Word #0,T-bit có giá trị bằng 1.Ngoài khoảng thời gian trễ T-bit có giá trị bằng 0.

Nếu chưa hết thời gian trễ mà tín hiệu đầu vào về 0 thì thời gian trễ vẫn được tính tiếp tục ,tức là T-bit và T-Word không về 0theo tín hiệu đầu vào.

e/Timer trễ theo sườn xuống (Off delay timer):

Cú pháp **SF** **< tên timer >**

Thời gian giữ trễ được bắt đầu khi có sườn xuống của tín hiệu đầu vào ,tức là ở thời điểm xuất hiện sườn xuống của tín hiệu đầu vào, giá trị PV được chuyển vào thanh ghi T-WORD (CV) .Trong khoảng thời gian giữa sườn lên của tín hiệu vào hoặc T-Word #0,T-bit có giá trị bằng 1.Ngoài khoảng thời gian trễ T-bit có giá trị bằng 0.

f/Khai báo tín hiệu xoá (Reset)

Cú pháp **A** **< địa chỉ bit >**

R **< Tên timer >**

Toán hạng thứ nhất “địa chỉ bit” xác định tín hiệu sẽ được sử dụng làm tín hiệu chủ động xoá cho timer có tên trong toán hạng thứ 2

Khi tín hiệu xoá bằng 1 ,T-Word (Thanh ghi CV) và T-bit cùng đồng thời được đưa về 0.Nếu tín hiệu xoá về 0,Timer sẽ chờ được kích lại.

g/Đọc nội dung thanh ghi T-Word (CV)

Nội dung thanh ghi T-Word là CV có thể được đọc vào ACCU1 theo hai cách :

1/ Đọc số đếm tức thời (không có độ phân giải)

Cú pháp **L** **< tên timer >**

Toán hạng là tên timer mà thanh ghi T-Word của nó sẽ được đọc vào ACCU1.

Giá trị đọc được là một số nguyên dương xác định số đếm tức thời (không có thứ nguyên),tức là chỉ là chỉ là tỉ số giữa khoảng thời gian kể từ khi Timer được kích ,và độ phân giải.

2/ Đọc thời gian trễ tức thời:

Cú pháp **LC** **< Tên timer >**

Toán hạng là tên timer mà thanh ghi T-Word của nó sẽ được đọc vào ACCU1.Giá trị đọc được gồm 2 phần: Một số BCD xác định số đếm tức thời (không có thứ nguyên) và độ phân giải

13/ Bộ đếm (Counter):

a/Khai báo sử dụng:

Việc khai báo sử dụng Counter bao gồm các bước :

-Khai báo tín hiệu enable nếu muốn sử dụng tín hiệu chủ động kích.

-Khai báo tín hiệu đầu vào CU được đếm tiến

-Khai báo tín hiệu đầu vào CD được đếm lùi

-Khai báo tín hiệu đặt (set) và giá trị đặt trước (PV)

-Khai báo tín hiệu xoá (reset).

Trong tất cả 5 bước trên,các bước 2,3 là bắt buộc

i/Khai báo tín hiệu enable (kích đếm)

Cú pháp **A** **< Địa chỉ bit >**

FR **< tên Counter >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu sẽ được sử dụng làm tín hiệu chủ động kích cho bộ đếm có tên cho trong toán hạng thứ hai.Tên của bộ đếm có dạng **Cx** với x trong khoảng [0,255]

ii/Khai báo tín hiệu được đếm tiến theo sườn lên

Cú pháp **A** **< Địa chỉ bit >**

CU **< Tên Counter >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu mà sườn lên của nó được bộ đếm với tên cho trong toán hạng thứ hai đếm tiến .Tên của bộ đếm có dạng Cx .Mỗi khi xuất hiện một sườn lên của tín hiệu ,bộ đếm sẽ tăng nội dung của thanh ghi C-Word (CV) lên 1 đơn vị.

iii/Khai báo tín hiệu được đếm lùi theo sườn lên

Cú pháp **A** **< Địa chỉ bit >**

CD **< Tên Counter >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu mà sườn lên của nó được bộ đếm với tên cho trong toán hạng thứ hai đếm lùi .Tên của bộ đếm có dạng Cx .Mỗi khi xuất hiện một sườn lên của tín hiệu ,bộ đếm sẽ giảm nội dung của thanh ghi C-Word (CV) đi 1 đơn vị.

iv/Khai báo tín hiệu đặt (set) giá trị đặt trước (PV)

Cú pháp **A** **< Địa chỉ bit >**

L **C# < hằng số >**

S **< Tên Counter >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu mà mỗi khi xuất hiện sườn lên của nó ,hằng số PV cho trong lệnh thứ hai dưới dạng BCD sẽ được chuyển vào thanh ghi C-Wordcủa bộ đếm có tên trong toán hạng của lệnh thứ ba.

iv/Khai báo tín hiệu xoá (reset)

Cú pháp **A** **< Địa chỉ bit >**

R **< Tên Counter >**

Toán hạng thứ nhất " địa chỉ bit" xác định tín hiệu mà mỗi khi xuất hiện sườn lên của nó ,thanh ghi C-Word của bộ đếm có tên trong toán hạng của lệnh thứ hai sẽ được xoá về

g/Đọc nội dung thanh ghi C-Word

Nội dung thanh ghi C-Word là CV có thể được đọc vào ACCU1 theo hai cách :

1/ Đọc số đếm tức thời (không có độ phân giải)

Cú pháp **L < tên counter >**

Toán hạng là tên bộ đếm mà thanh ghi C-Word của nó sẽ được đọc vào ACCU1.

Giá trị đọc được là một số nguyên dương xác định số đếm tức thời

2/ Đọc số đếm tức thời dạng BCD:

Cú pháp **LC < Tên counter >**

Toán hạng là tên bộ đếm mà thanh ghi C-Word của nó sẽ được đọc vào ACCU1.Giá trị đọc được là số BCD

14/ Kỹ thuật sử dụng con trỏ (Pointer)

Con trỏ (Pointer) là một công cụ mạnh ,rất được ưa dùng trong các chương trình điều khiển .Việc sử dụng con trỏ được hiểu là sự truy nhập gián tiếp tới một ô nhớ trong bộ nhớ .Nhưng thế nào là sự truy nhập gián tiếp .Ta hãy xét lệnh đọc nội dung của ô nhớ MW0 vào ACCU1 làm ví dụ

L MW0 // Đọc giá trị của ô nhớ MW0 vào thanh ghi ACCU1

Lệnh này là truy nhập trực tiếp ô nhớ MW0 vì địa chỉ của ô nhớ đó là MW0 đã được cho trực tiếp trong lệnh dưới dạng toán hạng .Như vậy có thể hình dung ra là lệnh đọc nội dung ô nhớ MW0 mà địa chỉ ô nhớ đó không cho trực tiếp trong lệnh sẽ là lệnh truy nhập gián tiếp.

Trong truy nhập gián tiếp ,địa chỉ ô nhớ được truy nhập sẽ là nội dung của một ô nhớ khác mà ta gọi là con trỏ .Ví dụ việc truy nhập trực tiếp ô nhớ MW0 ở trên tương đương với việc truy nhập gián tiếp nhờ con trỏ MD10 như sau:

```
L   0
T   MD10
L   MW[MD10]
```

a/ Sử dụng từ MW hoặc từ kép MD làm con trỏ :

Ta có thể sử dụng một ô nhớ thuộc vùng nhớ M có kích thước là từ (MW) hoặc từ kép (MD) để làm con trỏ .Trong những trường hợp như vậy ,con trỏ MW hoặc MD chỉ có thể là con trỏ địa phương (chỉ chứa phần số của địa chỉ).

Do phần số của địa chỉ có hai dạng thể hiện :

- Địa chỉ byte :20 ,22 ,100,.....
- Địa chỉ bit : 20.0 ,22.2 ,100.5.....

Nên con trỏ địa phương cũng có hai hình thái

- Con trỏ địa phương chỉ vị trí byte trong vùng
- Và con trỏ địa phương chỉ vị trí bit trong từng vùng.

i/ Con trỏ địa phương chỉ vị trí Byte: Với hình thái con trỏ này ta dùng được cả hai loại kích thước từ (MW) hoặc từ kép (MD) .Con trỏ chỉ chứa phần số xác định địa chỉ byte .Nếu ô nhớ cần được truy nhập gián tiếp có kích thước lớn hơn 1 byte (từ,từ kép hay một dãy các byte) thì nội dung của con trỏ là địa chỉ byte đầu tiên trong dãy các byte đó .

ví dụ:

```
L   20
T   MD10
L   DIB[MD10]
T   MW[MD10]
```

ii/ Con trỏ địa phương chỉ vị trí Bit:

Với hình thái này ta phải dùng loại con trỏ có kích thước từ kép (MD,DBD,LD).Con trỏ này chứa cả phần số xác định địa chỉ byte và phần số xác định số thứ tự của bit trong byteđó theo cấu trúc.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	b b b b b b b b b b b b b b b b	x x x
Không sử dụng	Địa chỉ byte (0 – 65535)	Địa chỉ Bit (0-7)

Cấu trúc dữ liệu này của con trỏ chỉ địa chỉ bit được khai báotrong S7-300 bằng toán hạng dạng:

P# < địa chỉ byte > . < Số thứ tự >

15/ Sử dụng thanh ghi con trỏ AR1 và AR2:

S7-300 có hai thanh ghi 32bits được dùng làm con trỏ thay vì phải sử dụng một từ (MW,DBW,LW) hay từ kép (MD,DBD,LD).Hai thanh ghi này có tên là AR1 và AR2.Đặc biệt tuy hai thanh ghi con trỏ này chỉ chứa địa chỉ bit(có thể có hoặc không có phần chữ của địa chỉ),song lại có thể sử dụng để truy nhập ô nhớ có kích thước nhiều hơn một bit như byte,từ hoặc từ kép.

Ta phân biệt hai trường hợp :

- AR là con trỏ địa phương chỉ vị trí bit trong từng vùng ,không chứa phần chữ của địa chỉ (area internal register)
- AR là con trỏ toàn cục chỉvị trí bit trong bộ nhớ ,chứa cả phần chữ và phần số của địa chỉ (area crssing register)

a/Khai báo giá trị thanh ghi AR: Hai thanh ghi AR được gán giá trị bằng lệnh

Cú pháp: **LAR1 [P# <địa chỉ bit >]**

LAR2 [P# <địa chỉ bit >]

Toán hạng của lệnh gán gia 1trị có cấu trúc:

P#[< tên vùng bộ nhớ >] < địa chỉ byte > . < số thứ tự bit >

Lệnh có thể có hoặc không có toán hạng .Nếu không có toán hạng ,lệnh sẽ chuyển nội dung của ACCU1 vào thanh ghi AR1 hoặc AR2.Trường hợp có toán hạng ,lệnh chuyển giá trị toán hạng vào thanh ghi AR1 hoặc AR2.Lệnh này không làm thay đổi nội dung thanh ghi trạng thái .

Giá trị chuyển vào thanh ghi AR phải có cấu trúc đúng của một con trỏ chỉ bit với dạng như sau

Một điểm khác biệt nữa của việc truy nhập gián tiếp thông qua con trỏ AR so với con trỏ kiểu MD là ô nhớ được truy nhập có một khoảng cách nhất định theo chiều tăng (offset) so với ô nhớ mà AR chỉ vào (hình 2.24).Offset có đơn vị nhỏ nhất tính theo bit với cấu trúc trong lệnh truy nhập như sau:

<tên lệnh> <vùng và kích thước> [Arx,P# <số byte >.<Số bit>]

Offset

Trừ trường hợp truy nhập bit (A,O,=,....),trong lệnh phải ghi rõ kích thước mảng bit của ô nhớ được truy nhập (B,W hay D).Nếu con trỏ được sử dụng là con trỏ địa phương ,thì còn phải cho biết vùng bộ nhớ được truy nhập trong bộ nhớ (M,P,I,Q,DB hay DI)

Đặc biệt thanh ghi AR không chỉ tới được vùng đệm PQ của các cổng ra tương tự .Giá trị P#P.... của toán hạng chỉ địa chỉ được tự động hiểu là địa chỉ của cổng vào tương tự.

Ví dụ 1:

```
LAR1 P#1.0          //Thanh ghi AR1 được dùng làm con trỏ địa phương
LAR2 P#M10.0        //Thanh ghi AR2 được dùng làm con trỏ toàn cục
A    [ AR2,P#1.3]   //Truy nhập ô nhớ M11.3
=    Q[AR1,P#0.2]   //Đưa giá trị ra cổng Q1.2
L    IB[AR1,P#0.0]  //Đọc 8 cổng vào IB1 ( I1.0 – I1.7)
T    B[AR2,P#0.0]   //Chuyển giá trị đọc được vào byte MB10
```

```
L     W[AR2,P#5.0]      // Đọc MW15
T     MW[AR1,P#2.0]     //Chuyển vào MW3
L     DBD[AR1,P#9.0]    //Đọc DBD10
T     D[AR2,P#20.0]
```

Ví dụ 2: Quay lại ví dụ về chương trình nhập dữ liệu từ cổng tương tự PIW304 và cất vào bộ đệm đã được trình bày trong mục trước nhưng sữa lại bằng cách dùng thanh ghi con trỏ toàn cục AR.Bộ đệm là vùng nhớ gồm 10 từ MW0 – MW18.Dữ liệu vừa đọc được sẽ được cất vào từ nhớ cuối cùng của bộ đệm .Các dữ liệu đã có trong bộ đệm sẽ được chuyển dần lên.Dữ liệu đầu tiên trong bộ đệm sẽ bị đẩy ra khỏi bộ đệm.Chương trình sử dụng MB24 chứa số đếm:

```
      LAR1 P#M0.0      //Địa chỉ ô nhớ đầu tiên
      L    9
Next: T    MB24       //Chỉ số đếm
      L    W[AR1,P#2.0]
      T    W[AR1,P#0.0]
      +AR1 P#2.0
      L    MB24
      LOOP Next
      L    PIW304
      T    MW18
```

b/ Tăng giảm nội dung thanh ghi AR:

Trong ví dụ trên ta đã sử dụng lệnh tăng nội dung thanh ghi AR.Lệnh này có cấu trúc chung như sau :

Cú Pháp: **+AR1 [P# < Bytes > . < Bits >]**
+AR1 [P# < Bytes > . < Bits >]

Lệnh có thể có hoặc không có toán hạng .Trong trường hợp không có toán hạng,lệnh sẽ cộng nội dung của thanh ghi AR với nội dung của từ thấp trong ACCU1 và cất lại kết quả vào thanh ghi AR.

Trong trường hợp có toán hạng ,thì toán hạng phải là một số có cấu trúc giống như Offset,khi đó lệnh sẽ cộng nội dung của toán hạng với nội dung của thanh ghi AR và cất lại nội dung vào thanh ghi AR .Lệnh không làm thay đổi nội dung thanh ghi trạng thái.

Ví dụ:

```
LAR1    P#M0.0    // Địa chỉ ô nhớ M0.0 được ghi vào thanh ghi AR1
+AR1    P#2.0     // AR1 chứa địa chỉ ô nhớ M2.0
```

c/ Cất giữ nội dung thanh ghi AR:

Ngoài các lệnh khai báo ,tăng giảm ,ta còn có các lệnh cất giữ nội dung thanh ghi AR với cấu trúc:

Cú Pháp: **TAR1 [< Địa chỉ từ kép >]**
TAR2 [< Địa chỉ từ kép >]

Lệnh có thể hoặc không có toán hạng ,trong trường hợp không có toán hạng ,lệnh sẽ chuyển nội dung thanh ghi AR vào ACCU1 .Nếu có toán hạng ,lệnh sẽ chuyển nội dung thanh ghi AR vào từ kép có địa chỉ được chỉ thị trong toán hạng

Ví dụ:

```
TAR1    MD0   //Chuyển nội dung thanh ghi AR1 vào từ kép MD0
```

d/ Đảo nội dung hai thanh ghi AR:

Cú Pháp: CAR

Lệnh không có toán hạng va thực hiện việc đảo nội dung của hai thanh ghi AR1,AR2.Nội dung của AR1 được chuyển sang AR2 và ngược lại nội dung của AR2 được ghi vào AR1

16/ Khai báo và sử dụng khối DB:

S7-300 có vùng M được sử dụng làm các ô nhớ lưu trữ giá trị trung gian ,các biến cờ .Bên cạnh vùng nhớ M,S7-300 còn cung cấp thêm một vùng đặc biệt khác để tổ chức lưu giữ dữ liệu dưới dạng khối và có tên chung là các khối dữ liệu Data Block (DB).Kích thước vùng nhớ này phụ thuộc vào từng loại CPU,riêng đối với CPU 314 thì nó có kích thước là 8Kbytes.Ta có thể khai báo nhiều khối DB cùng một lúc (tối đa 65535),được phân biệt với nhau nhờ chỉ số khối như DB1,DB2.....DB65535.Kích thước của các khối có thể khác nhau ,nhưng tổng kích thước của tất cả các khối DB không được vượt quá kích thước vùng nhớ đã cho (không được vượt quá 8Kbytes với CPU 314).Mọi khối DB đều có thể truy nhập từng bit.

a/Khai báo một khối dữ liệu:

Khối dữ liệu (DB) được khai báo nhờ phần mềm soạn thảo Step7.

Để khai báo một khối DB ta thực hiện các bước sau:

- Đặt tên biến
- Khai báo kiểu biến .Bên cạnh những kiểu biến thông dụng như BOOL (1 bit),Byte (8 bits), Int (16 bits),Real (32 bits)..... ta còn có thể sử dụng các kiểu biến phức hợp như String (chuỗi kí tự) ,Array (mảng dữ liệu)....
  - Đặt giá trị ban đầu cho biến (có thể bỏ qua)
  - Chú thích (có thể bỏ qua)

i/ Kiểu String: Đây là 1 biến có dạng một dãy kí tự .Dãy kí tự có độ dài tính theo byte là số cho trong dấu ngoặc vuông .Kích thước của biến bằng độ dài của dãy kí tự cộng thêm 2 byte chứa mã kết thúc chuỗi kí tự đó

ii/Kiểu biến ARRAY: Đây là biến dạng mảng gồm nhiều phần tử cùng cấu trúc (**CHAR,BYTE,WORD,INT,DWORD** hay **REAL**).Mảng này có thể 1 chiều ,song cũng có thể nhiều chiều.Mảng 2 chiều được khai báo bởi ARRAY [1..x,1..y],trong đó x là độ dài chiều thứ nhất và y là độ dài chiều thứ hai.

b/Truy nhập và quản lí khối dữ liệu:

i/ Truy nhập xa:

Tất cả các lệnh truy nhập ô nhớ đã biết đều sử dụng được với khối dữ liệu thông qua toán hạng:

< Tên khối dữ liệu > . < Kích thước ô nhớ và vị trí >

ví dụ: DB1,DB5	**DBX**	**Truy nhập bit**	**DBX1.3**
	DBB	**Truy nhập Byte**	**DBB2**
	DBW	**Truy nhập từ**	**DBW4**
	DBD	**Truy nhập từ kép**	**DBD6**

Cách truy nhập như trên còn gọi là truy nhập xa .Kiểu truy nhập này có ưu điểm là có thể tác động tới tất cả các khối dữ liệu nhưng có hạn chế cơ bản là chậm và không thể sử dụng kỹ thuật con trỏ để truy nhập xa.

Ví dụ:

```
A   DB1.DBX1.5 //Đọc nội dung bit thứ 5 thuộc byte thứ 0 của khối DB1
A   DB5.DBX2.3 //Thực hiện phép ^ với giá trị của bit thứ 3 ,byte 2 của khối
DB5
```

= DB10.DBX2.4 //Chuyển vào bit thứ 4 byte 2 của khối DB10.

ii/Truy nhập gần:

Bên cạnh truy nhập xa,S7-300 còn cung cấp thêm những lệnh truy nhập gần.Đó là kiểu truy nhập các khối dữ liệu có tên đã được ghi vào 1 trong hai thanh ghi chỉ khối dữ liệu DB (DB register).Việc ghi tên khối dữ liệu vào hai thanh ghi đó được thực hiện bằng lệnh mở khối có cấu trúc như sau:

Cú Pháp: OPN DB < Chỉ số của khối dữ liệu >
OPN DB < Chỉ số của khối dữ liệu >

Lệnh thứ nhất sẽ ghi tên khối dữ liệu có chỉ số cho trong toán hạng vào DB-register thứ nhất.Thanh ghi này sẽ được ta gọi là thanh ghi DB.Lệnh thứ hai ghi tên khối dữ liệu với chỉ số cho trong toán hạng vào DB-register thứ hai có tên gọi là thanh ghi DI

Ví dụ:

```
OPN   DB1     //Mở khối dữ liệu DB1( Ghi tên khối DB1 vào thanh ghi DB)
L     DBW35   //Đọc nội dung từ DBW35 của DB1 vào ACCU1
T     MW22    //Chuyển vào ô nhớ MW22
OPN   DI20    //Mở khối dữ liệu DB20( Ghi tên khối DB20 vào thanh ghi DB)
L     DIB12   //Đọc nội dung byte 12 của khối DB20 và chuyển vào ACCU1
T     DBB37   //Chuyển ACCU1 vào byte 37 của khối dữ liệu DB1
```

Các ô nhớ của khối dữ liệu đã được mở bằng lệnh OPN sẽ được truy nhập thông qua toán hạng:

< Kích thước ô nhớ và vị trí >

Thông qua thanh ghi DB		Thông qua thanh ghi DI	
DBX	Truy nhập bit	DIX	Truy nhập bit
DBB	Truy nhập byte	DIB	Truy nhập byte
DBW	Truy nhập từ	DIW	Truy nhập từ
DBD	Truy nhập từ kép	DID	Truy nhập từ kép

Khác với việc truy nhập xa ,ở chế độ truy nhập gần ta có thể sử dụng kỹ thuật con trỏ .

Ví dụ các lệnh sau thực hiện việc chuyển nội dung DB10.DBW0 tới DB10.DBW2

```
OPN     DB10
LAR1    P#DBX0.0
L       W[AR1,P#0.0]
T       W[AR1,P#2.0]
```

1/Đọc chỉ số khối dữ liệu có tên trong thanh ghi DB hoặc DI

Cú Pháp L DBNO
L DINO

Lệnh đọc chỉ số của khối dữ liệu có tên trong thanh ghi DB (DBNO) hoặc trong thanh ghi DI (DINO) và chuyển kết quả đọc được vào ACUU1 dưới dạng số nguyên.Lệnh không làm thay đổi nội dung thanh ghi trạng thái.Nội dung cũ của ACCU1 được chuyển vào ACCU2

2/Đọc độ dài khối dữ liệu có tên trong thanh ghi DB hoặc DI

Cú Pháp L DBLG
L DILG

Lệnh đọc độ dài tính theo byte của khối dữ liệu có tên trong thanh ghi DB (DBLG) hoặc trong thanh ghi DI (DILG) và chuyển kết quả đọc được dưới dạng số nguyên 32 bits vào ACCU1.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái .Nội dung cũ của ACCU1 được chuyển vào ACCU2.

3/Đảo nội dung hai thanh ghi DB và DI

Cú pháp CDB

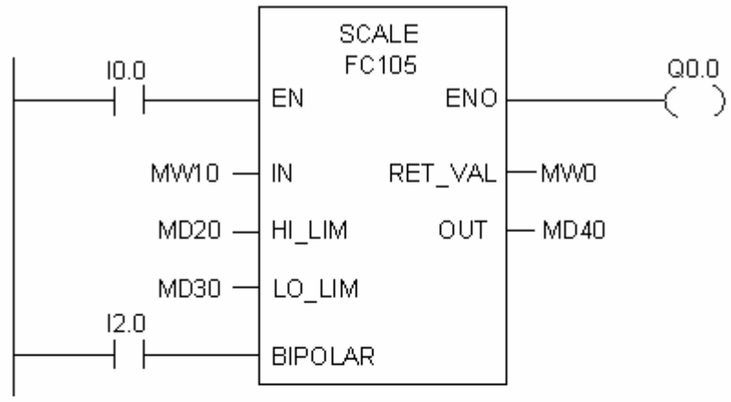
Lệnh chuyển nội dung của thanh ghi DB sang thanh ghi DI và ngược lại nội dung DI sang DB

.Lệnh không làm thay đổi nội dung của thanh ghi trạng thái

17/ Tín hiệu Analog :

a/ Đọc tín hiệu Analog:

- Để đọc tốt tín hiệu Analog trước hết ta phải xác định tín hiệu đọc Analog là tín hiệu loại gì (0-10V,4-20mA,cách đấu 2 dây,cách đấu 4 dây.....)
- Bước kế tiếp là phải chọn đúng loại tín hiệu trên phần cứng (Chọn loại tín hiệu trên Modul đọc kênh Analog) và chọn đúng trên cấu hình phần cứng cho phù hợp,nếu chọn 2 bước này không tương thích thì đèn System Fault của Modul Analog sẽ sáng và kênh Analog sẽ đọc sai.
- Xác định đúng tín hiệu sử dụng,đơn cực hay lưỡng cực
- Xác định địa chỉ cho từng kênh Analog (vd: PIW256...)
- Sử dụng hàm SCALE như sau :



Before execution:

IN ⟶ MW10 = 22

HI_LIM ⟶ MD20 = 100.0

LO_LIM ⟶ MD30 = 0.0

OUT ⟶ MD40 = 0.0

BIPOLAR ⟶ I2.0 = TRUE

After execution:

OUT ⟶ MD40 = 50.03978588

Hàm SCALE sẽ thực hiện việc kênh chỉnh từ tín hiệu IN kết quả sẽ được lưu ở OUT

OUT = (Float (IN)-K1)/(K2-K1) * (HI_LIM – LO_LIM) + LO_LIM

Tín hiệu **BIPOLAR** : K1 = -27648.0 , K2 = 27648.0

Tín hiệu **UNBIPOLAR** : K1 = 0.0 , K2 = 27648.0

RET_VAL : Trả về lỗi nếu việc thực hiện hàm SCALE có vấn đề

- Việc xuất tín hiệu Analog cũng sử dụng hàm SCALE,Tín hiệu Analog Out sẽ được đưa ra Modul xuất tín hiệu Analog tương ứng

Bài tập :

1/ Đọc khối lượng từ đầu cân Redlion:

Đầu cân Redlion có tích hợp sẵn các Card Analog (0-10V,4-20mA) và 1 số Card khác như RS232,RS485,ModBus,Profibus,Device Net...)

Tín hiệu Analog sẽ tương ứng với khối lượng hiện thị trên đầu cân tuỳ thuộc vào việc Set giá trị Analog tương ứng trên đầu cân.

Ví dụ : Sử dụng kênh Analog là 0-10VDC,chọn giá trị Min là 0Kg,giá trị Max là 100Kg thì Nếu khối lượng trên đầu cân là 50Kg,thì điện áp đọc về tương ứng là 5VDC.

2/ Xuất tín hiệu Analog Out điều khiển biến tần :

Có 1 cách thông dụng để thay đổi tốc độ của động cơ là điều khiển biến tần bằng cách thay đổi cấp điện áp tương ứng,hoặc dòng tương ứng . Tuỳ thuộc vào từng bài toán cụ thể,tốc độ động cơ sẽ được thay đổi tương ứng cho phù hợp.

18 / Đọc xung tốc độ cao :

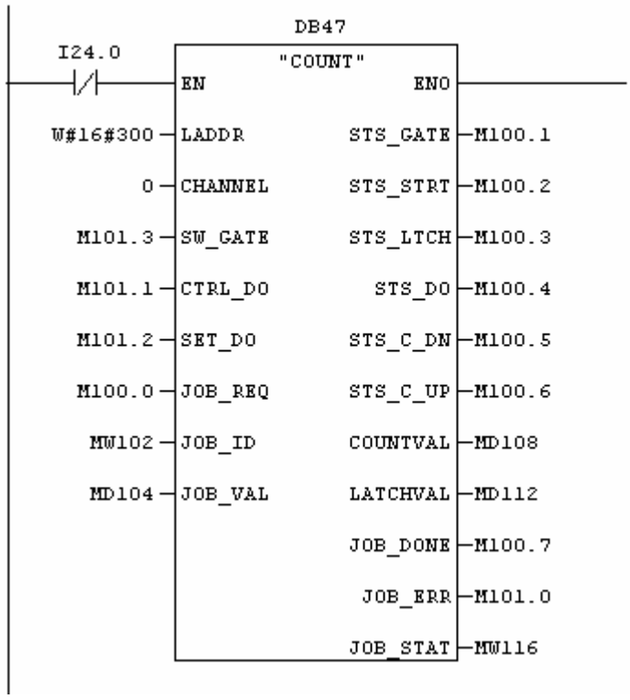
Xung tốc độ cao được đọc thông qua những Modul đọc xung tốc độ cao hoặc có thể được đọc thông qua CPU có tích hợp sẵn những I/O có khả năng đọc xung tốc độ cao như CPU 312C,313C.....

Việc đọc xung tốc độ cao là hết sức cần thiết cho những ứng dụng đọc xung Encoder, hay đọc xung của những Input tốc độ cao.

Tuỳ thuộc từng loại CPU cũng như Modul đọc xung tốc độ cao mà có những cách thức đấu nối dây khác nhau,do vậy việc đấu nối dây cần phải xem tài liệu trước khi thực hiện.

Cần phải xác định chế độ đọc xung trước khi đấu nối (vd : chế độ đọc 2 xung,chế độ đọc 1 xung .....).

Hàm đọc xung tốc độ cao **: SFB 47**



Dữ liệu được lưu vào khối DB47 theo nguyên tắc sau:

Parameter	Declaration	Data type	Address (Instance DB)	
LADDR	INPUT	WORD	0	Mặc định W#16#0300 Địa chỉ của kênh đọc xung tốc độ cao,địa chỉ này thường được cài đặt trong cấu hình phần cứng khi thực hiện việc định cấu hình phần cứng.
CHANNEL	INPUT	INT	2	Số kênh,0-1 : CPU 312C, 0-2 : CPU 313C 0 – 3 : CPU 314C
SW_GATE	INPUT	BOOL	4.0	Bit cho phép đếm (bắt đầu đếm và kết thúc đếm) Được điều khiển bằng phần mềm
CTRL_DO	INPUT	BOOL	4.1	Cho phép ngõ ra
SET_DO	INPUT	BOOL	4.2	Điều khiển ngõ ra
JOB_REQ	INPUT	BOOL	4.3	
JOB_ID	INPUT	WORD	6	Mặc định W#16#0000 Giá trị cho việc ghi dữ liệu cho bộ đếm. Trạng thái cổng bên trong Trạng thái cổng bắt đầu đếm từ bên ngoài Trạng thái ngõ vào chốt Trạng thái ngõ ra Trạng thái hướng ngược Trạng thái hướng thuận.

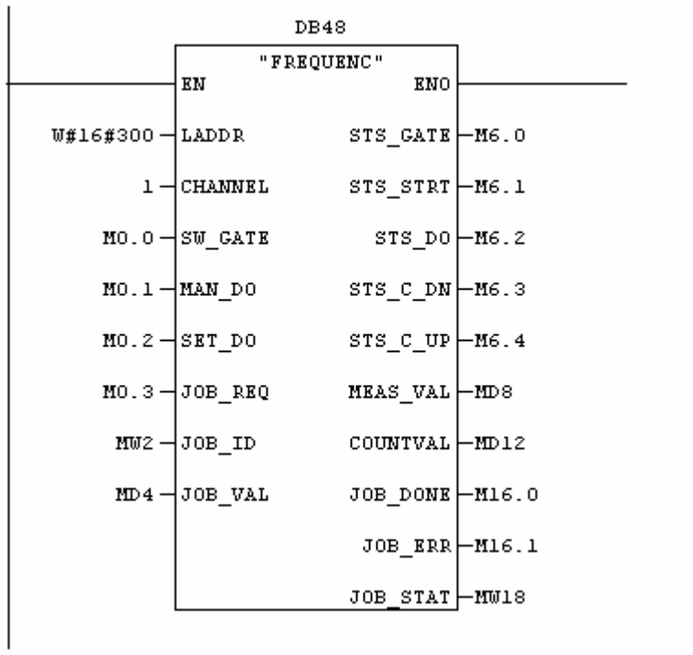
JOB_VAL	INPUT	DINT	8			
STS_GATE	OUTPUT	BOOL	12.0			
STS_STRT	OUTPUT	BOOL	12.1			
STS_LTCH	OUTPUT	BOOL	12.2			
STS_DO	OUTPUT	BOOL	12.3			
STS_C_DN	OUTPUT	BOOL	12.4			
STS_C_UP	OUTPUT	BOOL	12.5			
COUNTVAL	OUTPUT	DINT	14	-2^{31} up to $+2^{31}-1$	0	Giá trị đếm hiện tại
LATCHVAL	OUTPUT	DINT	18	-2^{31} up to $+2^{31}-1$	0	Giá trị chốt tại thời điểm cuối cùng
JOB_DONE	OUTPUT	BOOL	22.0	TRUE/FALSE	TRUE	Có sự kiện mới bắt đầu
JOB_ERR	OUTPUT	BOOL	22.1	TRUE/FALSE	FALSE	Trạng thái lỗi
JOB_STAT	OUTPUT	WORD	24	0 to W#16#FFFF	0	Giá trị lỗi.
STS_CMP	STATIC	BOOL	26.3			Trạng thái so sánh
STS_OFLW	STATIC	BOOL	26.5			Trạng thái tràn
STS_UFLW	STATIC	BOOL	26.6			Trạng thái dưới
STS_ZP	STATIC	BOOL	26.7			Trạng thái Zero
						Giá trị đếm ngõ ra
						Bit Reset lỗi.

JOB_OVAL	STATIC	DINT	28
RES_STS	STATIC	BOOL	32.2

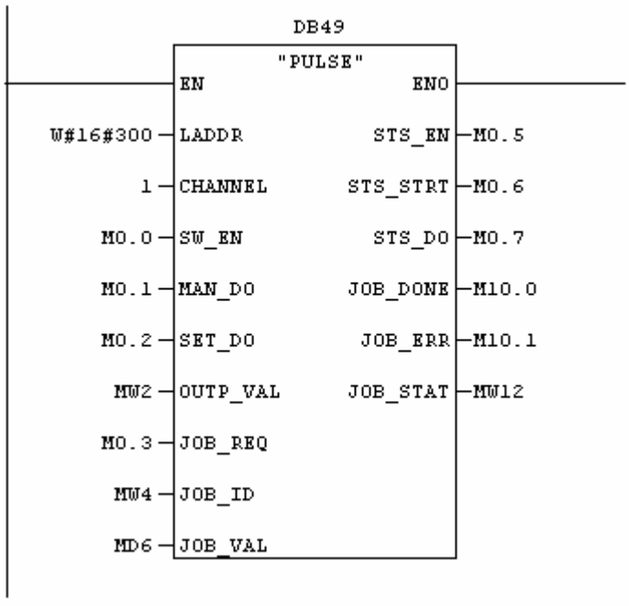
Chi tiết cụ thể có thể xem trong File Help của SFB47 bằng cách chọn SFB47 rồi bấm F1.

- Ngoài việc đọc xung tốc độ cao bằng hàm SFB47,ta còn có thể đọc tần số bằng hàm SFB48

- Cách thức định dạng hàm SFB48 cũng hoàn toàn tương tự hàm SFB47,chỉ khác ngõ ra là tần số.Chi tiết cụ thể có thể chọn hàm SFB48 rồi bấm F1.



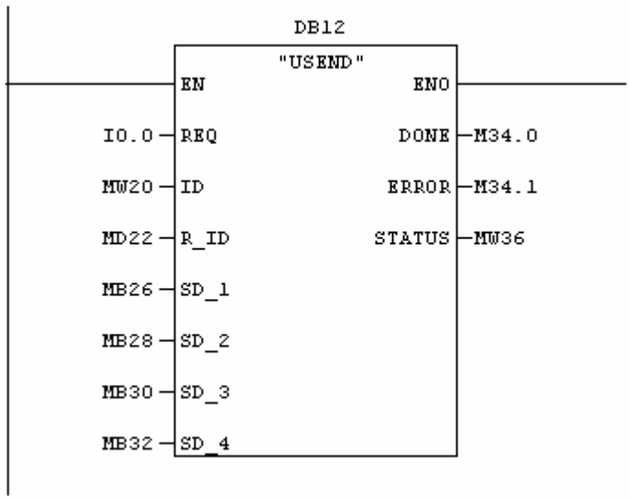
Xác định độ rộng xung bằng hàm SFB49



Cách thức định dạng hàm cũng như các bit ngõ vào ngõ ra hoàn toàn tương tự,chỉ khác ngõ ra Output là dạng độ rộng xung từ 0 -1.

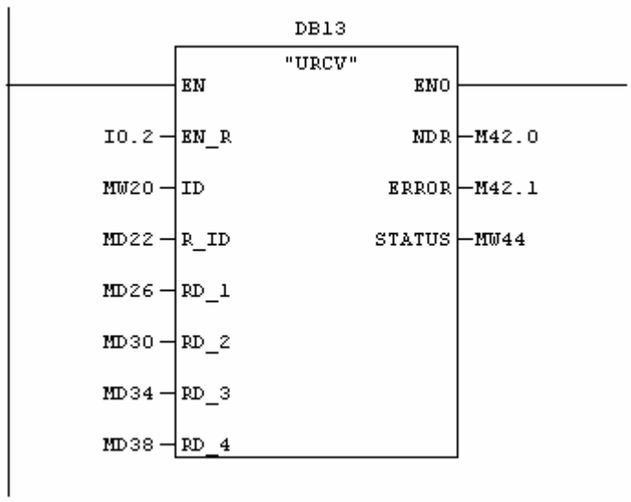
- **Truyền dữ liệu qua CP bằng lệnh SFB8:**

Cứ có 1 xung cạnh lên I0.0 thì ID,R_ID,SD_1 sẽ được truyền trong trường hợp CPU300,
SD_2,SD_3,SD_4 sẽ được truyền trong trường hợp CPU S7_400
SD có thể được dùng là Byte,Word hoặc Dword
Ngoài ra còn có các Bit lỗi , Bit thực hiện và thanh ghi trạng thái.
Dữ liệu được lưu vào DB12.



- **Nhận dữ liệu qua CP bằng lệnh SFB9:**

Cứ có 1 xung cạnh lên I0.2 thì ID,R_ID,SD_1 sẽ được nhận trong trường hợp CPU300,
SD_2,SD_3,SD_4 sẽ được nhận trong trường hợp CPU S7_400
SD có thể được dùng là Byte,Word hoặc Dword
Ngoài ra còn có các Bit lỗi , Bit thực hiện và thanh ghi trạng thái.
Dữ liệu được lưu vào DB13.



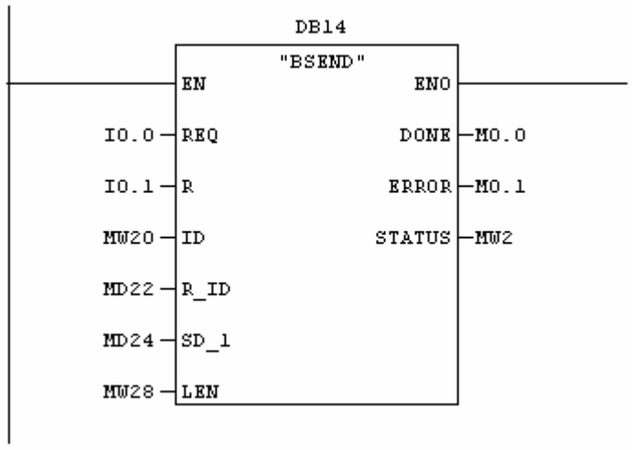
- **Truyền dữ liệu qua CP bằng lệnh SFB12:**

Cứ có 1 xung cạnh lên I0.0 thì ID,R_ID,SD_1 sẽ được truyền trong trường hợp CPU300,số Byte được truyền được quyết định bởi chiều dài Len MW28,vị trí Byte truyền được quyết định bởi SD_1.

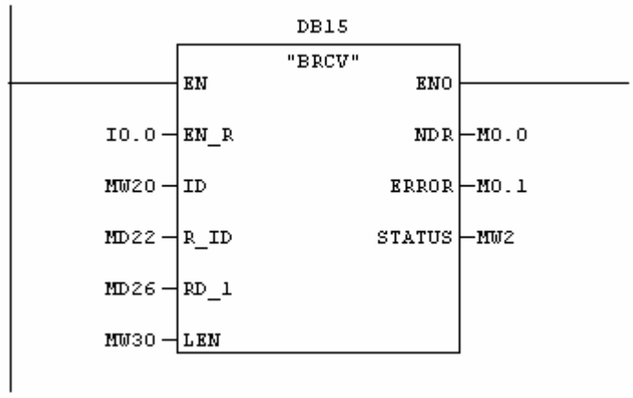
SD có thể được dùng là Byte,Word hoặc Dword

Ngoài ra còn có các Bit lỗi , Bit thực hiện và thanh ghi trạng thái.

Dữ liệu được lưu vào DB14.

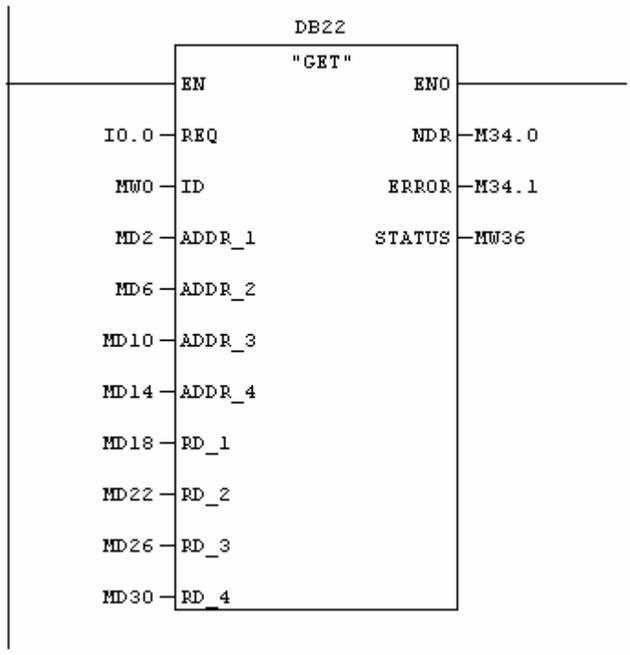


Nhận dữ liệu qua cổng COM thông qua hàm SFB13

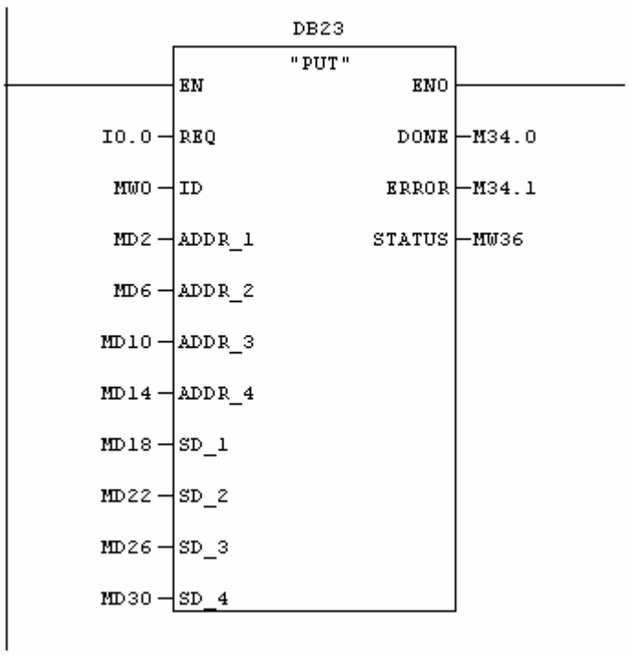


Cách sử dụng hàm nhận dữ liệu thông qua cổng COM hoàn toàn tương tự cách dùng của hàm truyền dữ liệu thông qua cổng COM.

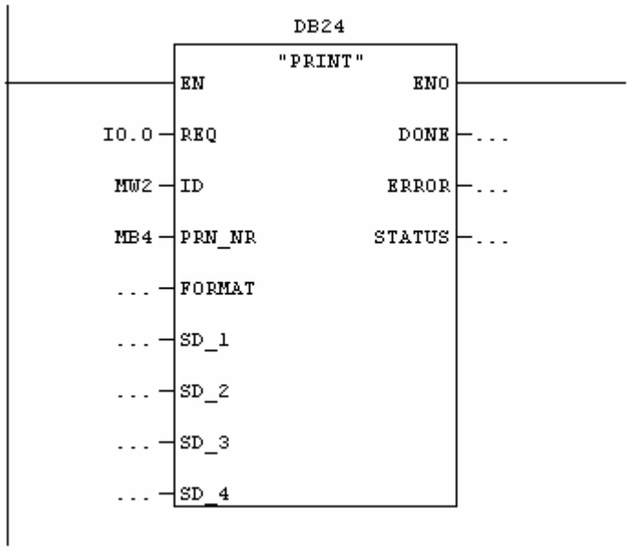
Đọc dữ liệu từ CPU thông qua hàm SFB14



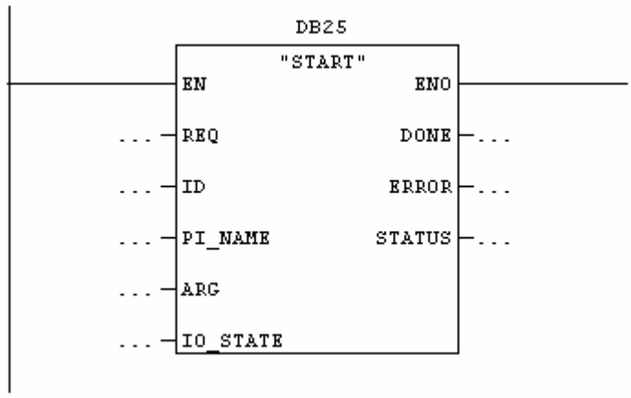
Ghi dữ liệu vào CPU thông qua hàm SFB15



Truyền dữ liệu ra cổng máy in thông qua hàm SFB16



Khởi động Thiết bị thông qua hàm SFB19



Dừng thiết bị thông qua hàm SFB20