

I. Nhóm lệnh cơ bản.

1. Lệnh INC

- Lệnh tăng nội dung thanh ghi ACCU1(lệnh +)

Cú Pháp: INC <toán hạng>

Ví dụ:

A I0.0 //Bits để kích RLO lên =1 để thực hiện.
L #Vùng nhớ 1 //Load “#Vùng nhớ 1” vào thanh ghi ACCU1 để thực hiện lệnh cộng INC.
INC 2 // “#Vùng nhớ 1” sẽ được cộng với 2 nếu I0.0=1.
T #Vùng nhớ 1 // Transfer(chuyển) dữ liệu vào lại ACCU1 để có thể thực hiện lại lần nữa nếu muốn

Network 1: Title:

Comment:

A	I	0.0	//toan hang de cho phep bat dau chay	
L	#So_hang_1		// Load vung nho vao ACCU1	#So_hang_1
INC	1		//thuc hien lenh cong 1	
T	#So_hang_1		//transfer lai vao vung nho ACCU1	#So_hang_1

2. Lệnh DEC

- Lệnh giảm nội dung thanh ghi ACCU1(lệnh -)

Cú Pháp: DEC <toán hạng>

Ví dụ:

A I0.0 //Bits để kích RLO=1 để thực hiện.
L #Vùng nhớ 1 //Load #Vùng nhớ 1 vào thanh ghi ACCU1
INC 1 // #Vùng nhớ 1 sẽ được trừ đi 1(INC là lệnh -)
T #Vùng nhớ 1 // #Vùng nhớ 1 sẽ được Transfer lại vào ACCU1 để có thể thực hiện lại lần nữa nếu muốn.

Network 1: Title:

Comment:

A	I	0.0	//toan hang de cho phep bat dau chay	
L	#So_hang_1		// Load vung nho vao ACCU1	#So_hang_1
DEC	1		//thuc hien lenh tru di 1	
T	#So_hang_1		//transfer lai vao vung nho ACCU1 de co the thuc hien lai	#So_hang_1

3. Lệnh kết thúc không điều kiện BEU

Cú pháp:

BEU <không có toán hạng>

Gặp lệnh này chương trình sẽ kết thúc vô điều kiện ngay.

Ví dụ:

A	I0.0	
S	Q0.0	
BEU		*1
A	Q0.0	
S	M0.0	

Giải thích

***1:** Nếu I0.0=1 thì sét Q0.0=1 nhưng gặp lệnh BEU chương trình sẽ dừng ở đây không chạy xuống dưới nữa có nghĩa là đoạn chương trình bên dưới sẽ không được thực hiện dù Q0.0 =1

4. Lệnh kết thúc có điều kiện

Thực hiện kết thúc chương trình nếu RLO = 1.

Cú pháp

BEC <không có toán hạng>

Ví dụ:

A	I0.0	
BEC		*1
A	I0.1	
S	Q0.1	

Giải thích

***1:** Nếu I0.0=1 (thì RLO=1) nhưng gặp lệnh BEC chương trình sẽ dừng ở đây không chạy xuống dưới nữa có nghĩa là đoạn chương trình bên dưới sẽ không được thực hiện dù I0.0 =1. Nếu I0.0=0 thì sẽ cho thực hiện đoạn chương trình dưới Q0.1=1 nếu I0.1=1.

II. Nhóm lệnh đại số.

1. Lệnh Cộng, trừ, nhân, chia.

Cú Pháp:

Phép toán	Lệnh	Kiểu dữ liệu
- Cộng số nguyên 16 bits	+I	- Interger(I)
- Cộng số nguyên 32 bits	+D	- Double Interger(DI)
- Cộng số thực	+R	- Số thực(Real)
- Trừ số nguyên 16 bits	-I	- Interger(I)
- Trừ số nguyên 32 bits	-D	- Double Interger(DI)
- Trừ số thực	-R	- Số thực(Real)
- Nhân số nguyên 16 bits	*I	- Interger(I)
- Nhân số nguyên 32 bits	*D	- Double Interger(DI)
- Nhân số thực	*R	- Số thực(Real)
- Chia số nguyên 16 bits	/I	- Interger(I)
- Chia số nguyên 32 bits	/D	- Double Interger(DI)
- Chia số thực	/R	- Số thực(Real)
- Lệnh lấy phần nguyên	RND	
- Lệnh lấy phần dư	MOD	

Ví dụ: Thực hiện phép cộng, trừ đại số 2 số hạng.

```
A      I0.0           //toán hạng để bắt đầu thực hiện phép toán dưới.
L      #so_hang_1     //Load dữ liệu vào ACCU1.
L      #so_hang_2     //Load dữ liệu vào ACCU1.***
+I                     //Thực hiện phép cộng.
T      #Tong          // Tranfer giá trị vào ACCU1.
```

Giải thích:

***:Lệnh load đầu tiên sẽ load giá trị “#so_hang_1” #so_hang_1” vào vùng nhớ của thanh ghi ACCU1. Sau đó lệnh chạy xuống gặp lệnh load giá trị “#so_hang_2”. Vì đây là lệnh sau nên sẽ được tiếp tục load giá trị vào ACCU1. Nhưng trên ACCU1 đang chứa giá trị của “#so_hang_1” nên giá trị này được load sang ACCU2 để nhường chỗ cho ACCU1 chứa giá trị của “#so_hang_1”. Các lệnh load liên sau sẽ được ưu tiên load giá trị vào ACCU1. Còn giá trị hiện thời trên ACCU1 sẽ được load sang ACCU2 để nhường vị trí. *Trường hợp khác cũng tương tự nhé.*

```

A      I0.0           //toán hạng để bắt đầu thực hiện phép toán dưới.
L      #so_hang_1       //Load dữ liệu vào ACCU1.
L      #so_hang_2       //Load dữ liệu vào ACCU1.***
*D           //Thực hiện phép nhân.
T      #Tong           // Tranfer giá trị vào ACCU1

```

- viết trên phần mềm:

FC3 : Title:

Comment:

Network 1: Title:

Comment:

```

A      I      0.0           //cho phép thực hiện phép toán.
L      #So_hang_1          //load số giá trị vào ACCU1      #So_hang_1
L      #So_hang_2          //Load giá trị vào ACCU1        #So_hang_2
*D           //phép nhân
T      #Tong              //Tranfer giá trị vào ACCU1      #Tong

```

2. Nhóm lệnh so sánh số nguyên 16

Phép toán	Lệnh	Kiểu dữ liệu
Lệnh so sánh bằng	= I	- I
Lệnh so sánh khác	< >I	- I
Lệnh so sánh lớn hơn	>I	- I
Lệnh so sánh nhỏ hơn	<I	- I
Lệnh so sánh lớn hơn hoặc bằng	>=I	- I
Lệnh so sánh nhỏ hơn hoặc bằng	<=I	- I

Ví dụ: Thực hiện so sánh 2 số nguyên.

```

A      I0.0           //toán hạng cho phép chương trình được thực hiện.
L      #so_hang_1       //Load số hạng 1 vào ACCU1**
L      #so_hang_2       //Load số hạng 2 vào ACCU1
>I           //So sánh lớn hơn giữa “#so_hang_1” và “#so_hang_2”

```

= Q0.0*** //Nếu “#so_hang_1” > “#so_hang_2” thì Q0.0=1

Giải thích:

- ***: Nếu “#so_hang_1” > “#so_hang_2” thì sét RLO=1. Có nghĩa là sét Q0.0=1. Các trường hợp khác thì RLO=0
- Tất cả các phép toán đại số khác như so sánh nhỏ hơn, so sánh bằng, so sánh khác cũng tương tự như vậy. Nếu thỏa mã điều kiện so sánh thì để sét RLO=1(hiểu là sét 1 bits sau lệnh so sánh lên =1)

Chương trình trên phần mềm:

FC3 : Title:

Comment:

Network 1: Title:

Comment:

```
A      I      0.0          //cho phép thực hiện phép toán.
L      #So_hang_1          //load số giá trị vào ACCU1
L      #So_hang_2          //Load giá trị vào ACCU1
==I                                //so sánh bằng
S      Q      0.0          //nếu 2 số bằng nhau thì set Q0.0=1
//neu 2 so bang nhau thi set Q0.0=1
```

#So_hang_1
#So_hang_2

Giải thích:

**:Lệnh load đầu tiên sẽ load giá trị “#so_hang_1” vào vùng nhớ của thanh ghi ACCU1. Sau đó lệnh chạy xuống gặp lệnh load giá trị “#so_hang_2”. Vì đây là lệnh sau nên sẽ được tiếp tục load giá trị vào ACCU1. Nhưng trên ACCU1 đang chứa giá trị của “#so_hang_1” nên giá trị này được load sang ACCU2 để nhường chỗ cho ACCU1 chứa giá trị của “#so_hang_1”. Các lệnh load liên sau sẽ được ưu tiên load giá trị vào ACCU1. Còn giá trị hiện thời trên ACCU1 sẽ được load sang ACCU2 để nhường vị trí. Trường hợp khác cũng tương tự nhé.

3. Nhóm lệnh so sánh số nguyên 32 Bits

Phép toán	Cú pháp	Kiểu dữ liệu
Lệnh so sánh bằng	= = D	- D
Lệnh so sánh khác	< >D	- D
Lệnh so sánh lớn hơn	>D	- D
Lệnh so sánh nhỏ hơn	<D	- D
Lệnh so sánh lớn hơn hoặc bằng	>=D	- D
Lệnh so sánh nhỏ hơn hoặc bằng	<=D	- D

- Các ví dụ cũng sẽ tương tự như so sánh số nguyên 16 bits.
- So sánh số thực câu lệnh, cú pháp cũng tương tự.

Ví dụ phần mềm:

Network 1: Title:

Comment:

```

A      I      0.0      //cho phép thực hiện phép toán.
L      #So_hang_1      //load số giá trị vào ACCU1
L      #So_hang_2      //Load giá trị vào ACCU1
<>D      //so sánh khác nhau số nguyên 32 Bits
S      Q      0.0
//neu 2 khác nhau thì set Q0.0=1

```

#So_hang_1
#So_hang_2

III. Nhóm lệnh điều khiển chương trình.

1. Lệnh JC nhảy có điều kiện khi RLO = 1

Cú pháp: **JC** <nhãn>

- Lệnh sẽ thực hiện nhảy tới nhãn nếu RLO bằng 1.

Ví dụ:

A **I0.0** **//1***

JC **m0** //tên nhãn, không được dài quá 4 ký tự. Có thể đặt tùy ý.

A **M0.0** **//2***

S **Q0.0**

m0: NOP **0** //Cú pháp.

A **M0.1**

R **Q0.0**

Giải thích:

- **1*:** Nếu I0.0=0 thì chương trình sẽ chạy xuống dưới như bình thường, nếu I0.0=1 thì sẽ nhảy đến nhãn m0 **bỏ qua đoạn chương trình giữa lệnh JC và nhãn m0**
- **2*:** Nếu I0.0=0 chương trình sẽ chạy xuống dưới bình thường. Nếu M0.0 =1 thì sẽ SET Q0.0=1. Nhưng nếu I0=1 thì lệnh JC thực hiện nhảy đến nhãn m0. Lúc này dù M0.0 có tác động thì cũng không SET được Q0.0 nữa.

2. Lệnh nhảy có điều kiện khi RLO = 0

Cú pháp: **JCN** <nhãn>

- Lệnh thực hiện nhảy tới nhãn khi RLO=0

Ví dụ:

A I0.0 //3*

JCN m0 //tên nhãn, không được dài quá 4 ký tự. Có thể đặt tùy ý.

A M0.0 //4*

S Q0.0

m0: NOP 0 //Cú pháp. K cần sử dụng có thể đặt luôn câu lệnh lên.

A M0.1

R Q0.0

Giải thích:

- **3*:** Nếu I0.0=1 thì chương trình sẽ chạy xuống dưới bình thường, nếu I0.0=0 thì sẽ nhảy đến nhãn m0 **bỏ qua đoạn chương trình giữa lệnh JCN và nhãn m0.** Chú ý lệnh JCN ngược hoàn toàn với lệnh JC nhé.
- **4*:** Nếu I0.0=1 chương trình sẽ chạy xuống dưới bình thường. Nếu M0.0 =1 thì sẽ SET Q0.0=1. Nhưng nếu I0=0 thì lệnh JCN thực hiện nhảy đến nhãn m0. Lúc này dù M0.0 có tác động thì cũng không SET được Q0.0 nữa.

3. Lệnh nhảy vô điều kiện JU

Cú pháp: **JU** <nhãn>

- Lệnh thực hiện nhảy đến nhãn vô điều kiện

Ví dụ:

A I0.0 //5*

JU m0 //tên nhãn, không được dài quá 4 ký tự. Có thể đặt tùy ý.

A M0.0 //6*

S Q0.0

m0: NOP 0 //Cú pháp. K cần sử dụng có thể đặt luôn câu lệnh lên.

Giải thích:

- **5*:** Cho dù $I0.0=0$ hay $I0.0=1$ thì khi chương trình chạy từ trên xuống dưới gặp lệnh JU sẽ nhảy đến nhãn ngay nên đa phần những lệnh JU sẽ đặt xuống cuối cùng của NetWork để nhảy đến 1 vị chỉ chung nào đó. Lưu ý nhiều lệnh có thể nhảy đến cùng 1 nhãn hoặc 1 nhãn do nhiều lệnh thực hiện cũng được.
- **6*:** Đoạn chương trình trong này sẽ không bao giờ được chạy đến nếu phía trên nó là 1 lệnh JU.
- Lệnh này dùng để viết cho chu trình máy (đây là ứng dụng quan trọng nhất của lệnh này)

4. Lệnh nhảy có điều kiện JP, JM

Cú pháp: **JP** <nhãn>
 JM <nhãn>

- Lệnh JP được thực hiện khi kết quả của phép trính trước nó kết quả dương. Có nghĩa là phép toán liền trước lệnh JP>0 thì sẽ nhảy đến nhãn.
- Lệnh JM được thực hiện khi kết quả của phép trính trước nó có kết quả âm. Có nghĩa là phép toán liền trước lệnh JP<0 thì sẽ nhảy đến nhãn.

Ví dụ:

```
L      MW0    //Load giá trị vào ACCU1
L      MW2    //Load giá trị vào ACCU1.Giá trị đang có load vào ACCU2
-I                      //1**
JP      m0     //2**tên nhãn, không được dài quá 4 ký tự. Có thể đặt tùy ý.
JM      m1     //3***
A      M0.0    //
S      Q0.0
m0: NOP    0    //Cú pháp. K cần sử dụng có thể đặt luôn câu lệnh lên.
A      M0.1
R      Q0.0
m1: NOP 0
S      #bits_bao_loi
```


Giải thích:

- **1****: Phép trừ đại số 2 số nguyên Interger được thực hiện và lưu giá trị lại ACCU1.
- **2****: Nếu phép tính thực hiện bên trên nó dương thì nó sẽ nhảy đến nhãn m0.
- **3*****: Nếu phép tính trên thực hiện ra kết quả âm nó sẽ chạy qua lệnh JP(do không thỏa mãn điều kiện kết quả dương được nhảy tới nhãn)gặp lệnh JM nó sẽ nhảy tới nhãn m1. Đây là 1 trường hợp của 2 lệnh lồng nhau. Cũng dùng rất là nhiều.

5. Lệnh xoay vòng LOOP

Cú pháp: LOOP <nhãn>

- Nếu gặp lệnh LOOP thanh ghi sẽ tự động trừ đi 1 đơn vị và kiểm tra.
 - + Nếu khác không thì sẽ nhảy đến nhãn
 - + Nếu bằng 0 sẽ thực hiện lệnh kế tiếp.

Ví dụ:

```
L          10          //Load giá trị vào ACCU1
M0:        NOP 0       //1*
.....
.....          //đoạn chương trình
.....
LOOP       M0          //2*
A          M0.0
AN         M0.2
S          Q0.0
```

Giải thích:

- **1****: Nhãn M0. Chương trình sẽ chạy ở nhãn M0 đến khi gặp lệnh LOOP.
- **2****:Chương trình sẽ so sánh ACCU1 có bằng 0 hay khác không.
 - + Nếu ACCU1 khác 0 thì sẽ nhảy lại về nhãn M0 để thực hiện tiếp
 - + Nếu ACCU1 =0 thì chương trình sẽ chạy qua lệnh LOOP để thực hiện các lệnh tiếp theo.
- Sử dụng cho các bài toán cần thực hiện theo số vòng quét. Ví dụ như chỉ kiểm tra lỗi trong chu kỳ quét đầu tiên(có thể thay thế khối OB100).

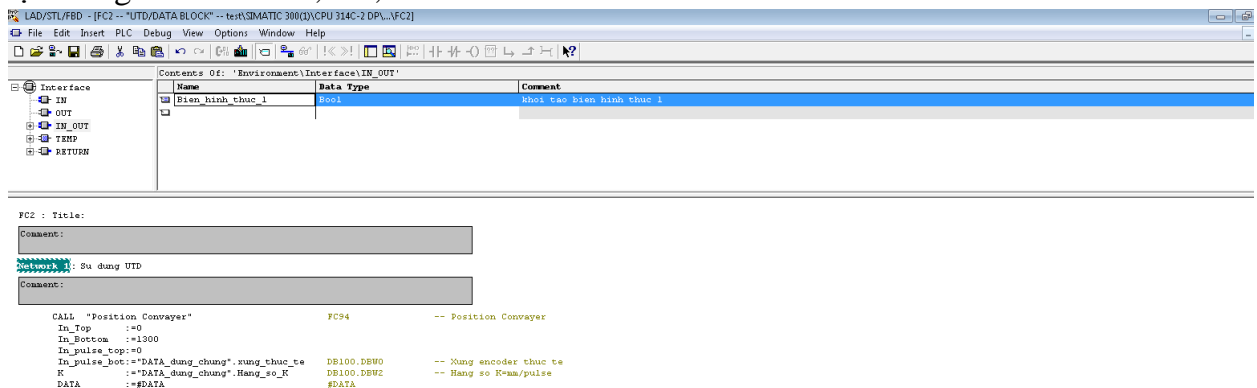
I. Các kiểu dữ liệu của PLC S7 300

1. Các kiểu dữ liệu.

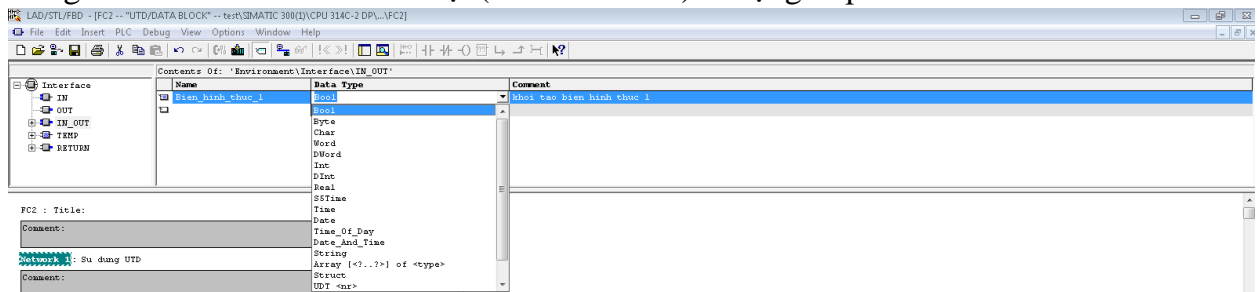
- Kiểu **BOOL**: Kiểu Bits trạng thái, có giá trị 0 và 1 (kiểu tiếp điểm thường đóng, thường hở, cuộn hút)
- Kiểu **BYTE**: Gồm 8 bits. Dùng để biểu diễn các số nguyên từ 0-255. Đây là kiểu dữ liệu nhập cho lệnh Jump List(JL).
- Kiểu **WORD**: gồm 2 byte biểu diễn số nguyên trong khoảng 0 – 65535.
- Kiểu **INT**: Kiểu số nguyên Integer. Thường dùng để nhập dữ liệu cho các biến hình thức trong FB, FC. Biểu diễn số nguyên trong khoảng -32767 đến 32767.
- Kiểu **DINT**: 4 Byte hoặc Double Integer. Dùng để biểu diễn những số nguyên dài trong khoảng từ -2147483648 đến 2147483648.
- Kiểu **REAL**: Dùng để biểu diễn số thực.
- Kiểu **S5T**: Kiểu thời gian nhập cho Timer.

2. Biến hình thức

- Biến hình thức là một dạng biến không cố định về kiểu dữ liệu được quy định trong các khối OB, FB, FC. Có vô hạn số lượng biến hình thức có thể được khởi tạo trong các khối OB, FB, FC.



Ví dụ trên là tạo một biến hình thức có tên “Bien_hinh_thuc_1” được khởi tạo trong khối hàm FC2. Kiểu dữ liệu(DATA TYPE) là dạng tiếp điểm BOOL.



“Click” vào DATA TYPE ta có thể chọn được các kiểu dữ liệu của biến hình thức

⇒ Lưu ý:

- Tên biến hình thức không sử dụng được dấu cách, các ký tự đặc biệt...

- Các biến hình thức này ngoài việc sử dụng trong chương trình còn để tạo các biến TAG để sử dụng cho màn hình cảm ứng Monitor

3. Biến cố định.

- Chính là các biến I, Q, M, MW, MD. Là các biến cố định không thể tùy ý thay đổi về kiểu dữ liệu biến.
- Một số trường hợp như biến M vẫn có thể đặt làm tác vụ cho màn hình Monitor

II. Tổng quan về các khối OB, FB, FC, DATA, UDT.

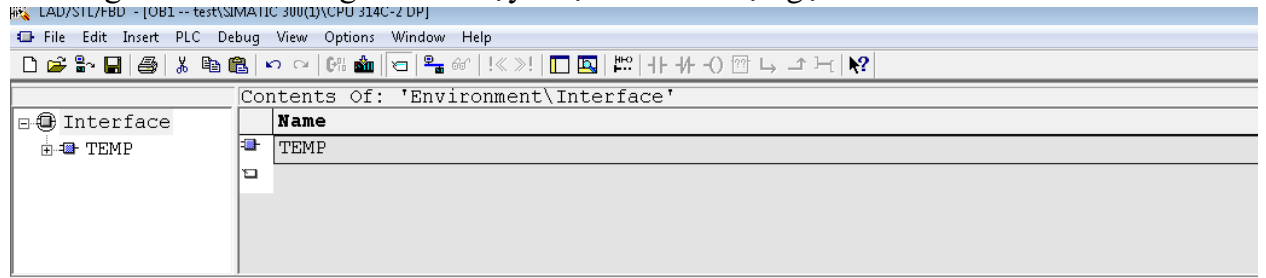
1. Các khối OB.

- Miền chứa chương trình tổ chức.

1.1. Khối chương trình chính OB1

Object name	Symbolic name	Created in language	Size in the work me...	Type	Version (Header)	Name (Header)	Unlinked	Author	Non-Retain	Standard block	Last interface char
System data				SDB							
OB1		STL	38	Organization Block	0.1						02/15/1996 04:51
FB1	test	STL	70	Function Block	0.1						08/12/2020 02:20
FC1		STL	312	Function	0.1						07/06/2020 03:07
FC2	UTD/DATA BLOCK	STL	400	Function	0.1						07/07/2020 03:23
FC3	TSETTTTTTTTTTTTTTTT...	STL	52	Function	0.1						08/05/2020 02:19
FC30	SUM	STL	368	Function	0.1						08/12/2020 10:36
FC34	Position Conveyor	STL	1034	Function	0.1						08/04/2020 02:33
DB1		DB	36	Instance data block ...	0.0						08/12/2020 10:24
DB47		DB	70	Instance data block ...	0.0			SIMATIC			02/26/2001 08:00
DB93	DAT1_SUM	DB	42	Data Block	0.1						07/06/2020 03:15
DB94	DAT1_position conveyor	DB	78	Data Block	0.1						08/04/2020 02:54
DB100	DATA_dung_chung	DB	42	Data Block	0.1						08/10/2020 04:23
UDT93	DATA_SUM	STL	---	Data Type	0.1						07/06/2020 02:59
UDT94	DATA_position conveyor	STL	---	Data Type	0.1						08/04/2020 02:32
SFB47	COUNT	STL	---	System function block	1.0	COUNT		SIMATIC			02/26/2001 08:00

- Đây là khối chương trình chính. Chương trình sẽ chỉ chạy trong khối OB1. Chương trình con bên ngoài chỉ chạy được khi nó được gọi “CALL” từ khối OB1.



OB1 : "Main Program Sweep (Cycle)"

Comment:

Network 1: Title:

Comment:

//cách gọi các khối hàm trong OB1

CALL FC 1

CALL "UTD/DATA BLOCK"

CALL "TSETTTTTTTTTTTTTTTTTT"

CALL "ham dk dong co 1" , DB1

Tất cả các khối FB, FC chỉ được thực thi khi gọi trên khối OB1

FC2

FC3

FB1

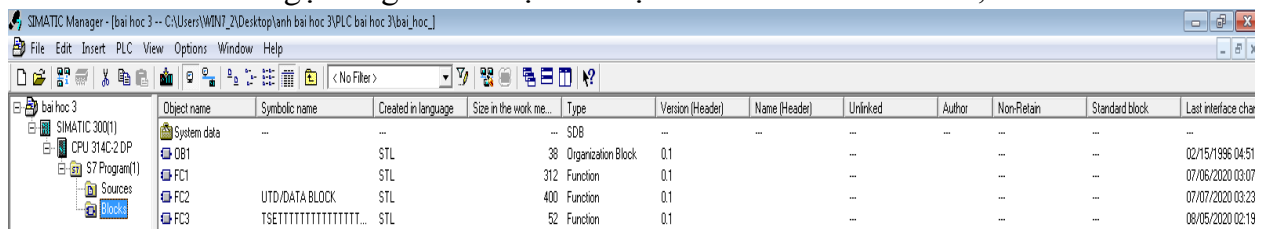
H1.1: Cách gọi các chương trình con từ khối OB1

1.2 Các khối OB đặc biệt khác

- OB100 được thực hiện 1 lần khi PLC Run
- OB121 được thực hiện khi CPU phát hiện thấy có lỗi trong chương trình....
- trong tài liệu tham khảo.

2. Các khối FC (Program Block)

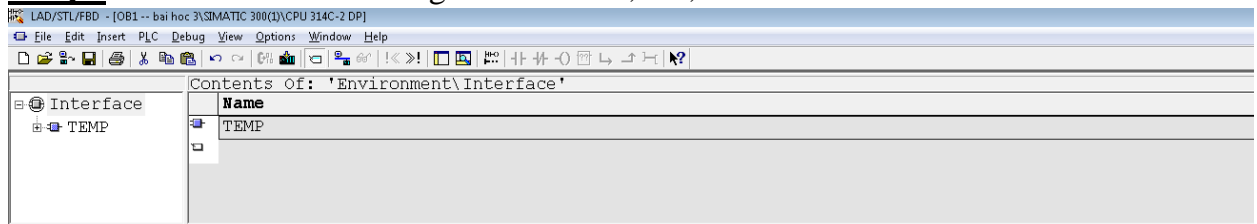
- Khối chương trình với chức năng riêng giống như một chương trình con hoặc 1 hàm. Một chương trình ứng dụng có thể có nhiều khối FC
- Khối FC thường được khởi tạo như là 1 hàm viết sẵn và sẽ được CALL ra trong các khối FC, FB, OB.
- Các khối FC có thể gọi lồng nhau được. Ví dụ FC1 có thể CALL FC2, FC3...



The screenshot shows the SIMATIC Manager interface. On the left, the project tree displays the hierarchy: SIMATIC 300(1) > CPU 314C-2 DP > S7 Program(1) > Sources > Blocks. On the right, a table lists the objects in the project.

Object name	Symbolic name	Created in language	Size in the work me...	Type	Version (Header)	Name (Header)	Unlinked	Author	Non-Retain	Standard block	Last interface char
System data	--	--	--	SDB	--	--	--	--	--	--	--
OB1		STL	38	Organization Block	0.1		--	--	--	--	02/15/1996 04:51
FC1		STL	312	Function	0.1		--	--	--	--	07/06/2020 03:07
FC2	UTD/DATA BLOCK	STL	400	Function	0.1		--	--	--	--	07/07/2020 03:23
FC3	TSETTTTTTTTTTTTTTTT...	STL	52	Function	0.1		--	--	--	--	08/05/2020 02:19

Ví dụ: Cách CALL FC từ trong các khối OB, FB, FC



OB1 : "Main Program Sweep (Cycle)"

Comment:

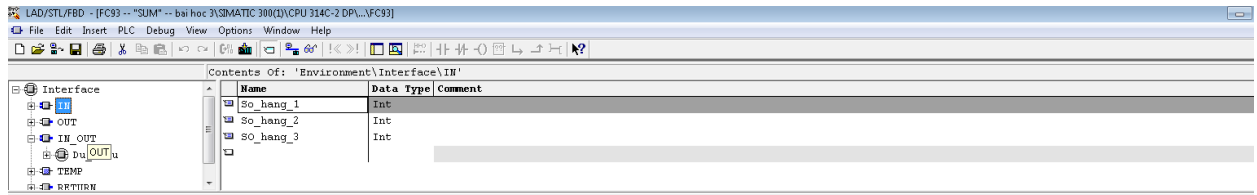
Network 1: Title:

Comment:

```
CALL "SUM"                                FC93
So_hang_1:=
So_hang_2:=
So_hang_3:=
Tung_binh:=
MAPP   :=
```

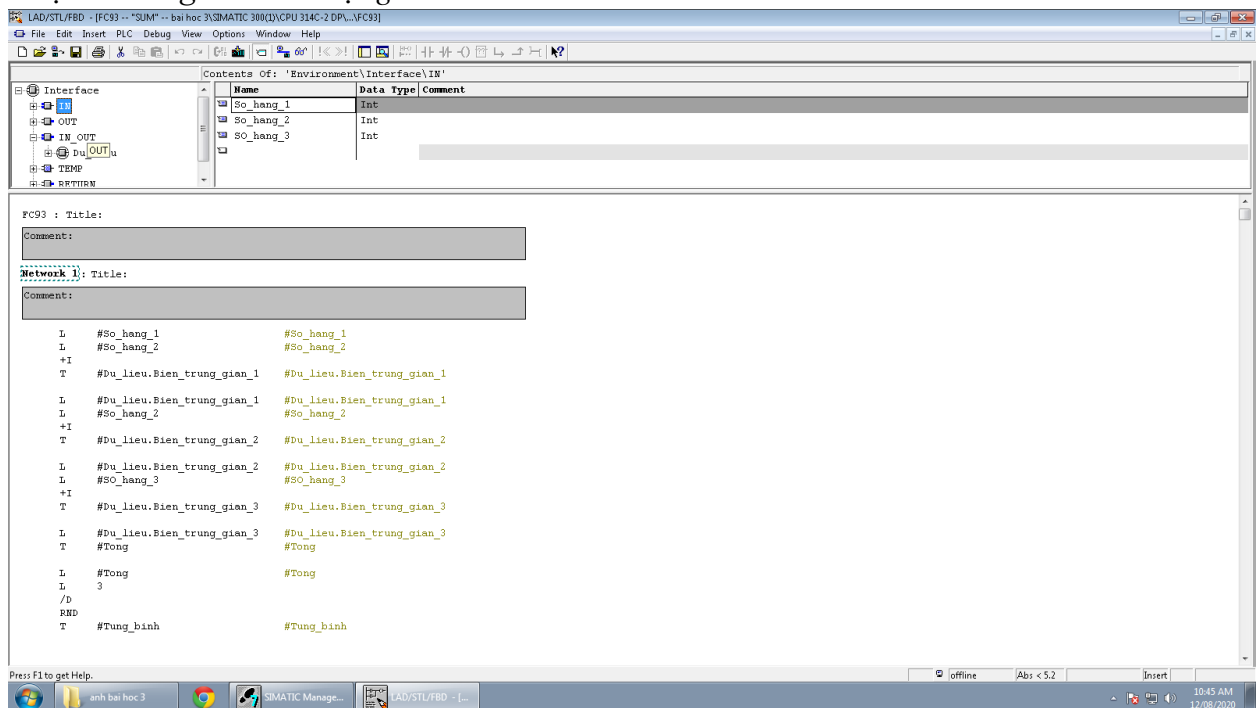
- Hàm FC1(đã được khởi tạo thành 1 hàm chương trình con sẵn) đã được gọi từ khối OB1.
- Sau khi CALL chương trình sẽ hiện lên những biến hình thức do ta qui định trong khối FC và yêu cầu người dùng nhập dữ liệu.

- Chương trình sẽ quét trong khối OB1. Đến lệnh CALL FC93 như ví dụ chương trình sẽ đưa các biến hình thức người dùng vừa nhập quét ngược lại và thực thi trong khối FC93. Sau đó sẽ trả ngược lại dữ liệu về chương trình đã gọi ra nó (ở đây là OB1). Nói đơn giản là nếu chương trình gặp lệnh CALL FC93 nó sẽ quay ra hàm FC93 để thực thi xong sẽ trả lại dữ liệu về chương trình mẹ đã gọi nó.



- Các biến hình thức được khởi tạo trong mục IN, OUT, IN-OUT sẽ được xuất hiện trong hàm mẹ để người dùng nhập dữ liệu
- Các biến hình thức trong TEMP sử dụng làm trung gian để tính toán trong hàm sẽ không được gọi ra trong chương trình mẹ.

Đoạn chương trình sử dụng các biến hình thức



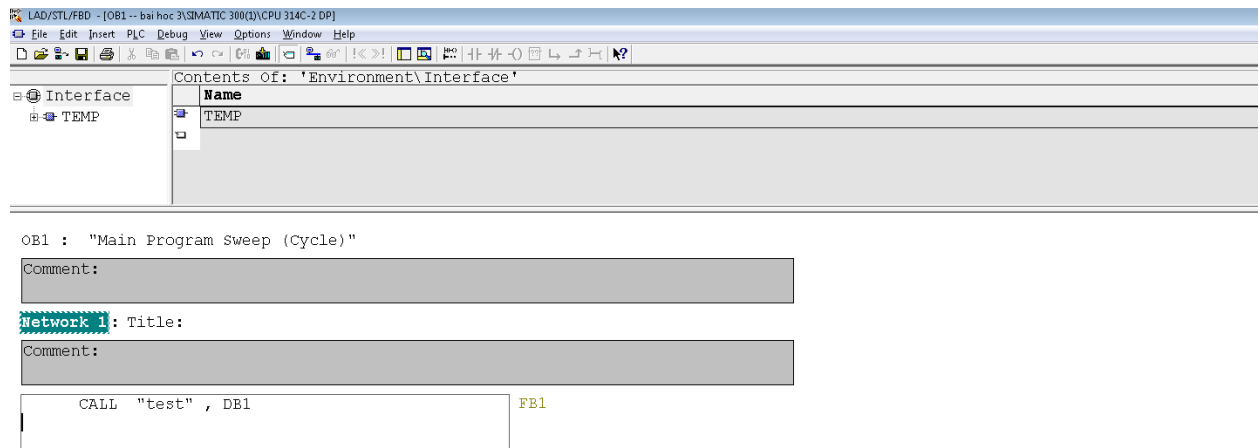
- Chương trình được thực thi trong chương trình con sẽ toàn là thực thi dưới dạng các biến hình thức. Dữ liệu của biến hình thức sẽ được nhập từ chương trình mẹ đã gọi nó. Sau khi thực hiện xong chương trình con dữ liệu lại được trả ngược về chương trình mẹ đã gọi nó.
- Với cách này chúng ta hoàn toàn có thể dễ dàng thay đổi dữ liệu các biến hình thức và thực thi với nhiều lần gọi, gọi từ nhiều khối.

➔ Bởi vì là một hàm chương trình con không có vùng nhớ để lưu giá trị của các biến hình thức. Muốn lưu được giá trị của các biến hình thức trong FC thì ta có 2 cách tạo cho nó 1 vùng dữ liệu để lưu tương tự như 1 DATA BLOCK đi kèm với 1 FB như mục 3. Vùng dữ liệu này sẽ là một UDT hoặc một DATA BLOCK(Share Data block) ở mục 4 và mục 5.

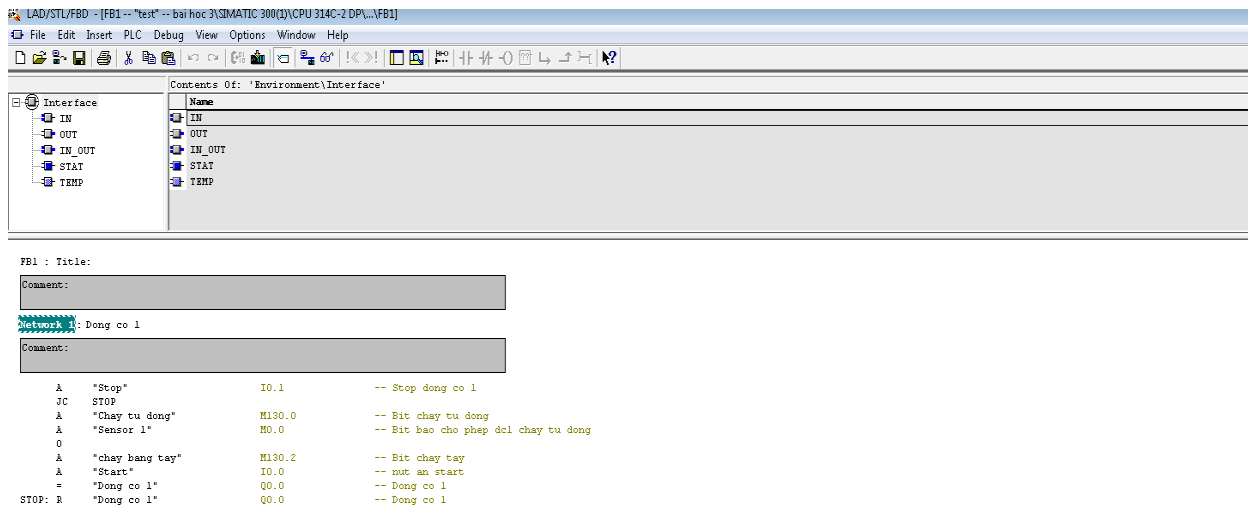
3. Khối hàm FB (Function Block)

3.1 Khối FB

- Là loại khối FC đặc biệt có khả năng trao đổi 1 lượng dữ liệu lớn với các khối chương trình khác nhau.
- Khối FB chỉ được gọi khi đi kèm với nó là 1 khối dữ liệu DATA BLOCK.
- Cũng tương tự khối FC khối FB có thể được gọi ra từ các khối như OB, FB, FC.
- Khối FB do có thể trao đổi một lượng dữ liệu lớn và cũng có thể lưu trữ được lượng dữ liệu rất lớn nên thông thường sẽ làm chương trình thực thi khác với khối FC(hay được sử dụng làm một hàm chương trình con viết sẵn)



- Cũng tương tự với khối FC. Nếu chương trình mẹ gọi khối FB(ở đây là khối OB1) chạy đến gặp lệnh CALL FB1 thì chương trình sẽ chạy vào khối FB1 và thực thi ở trong khối FB1.

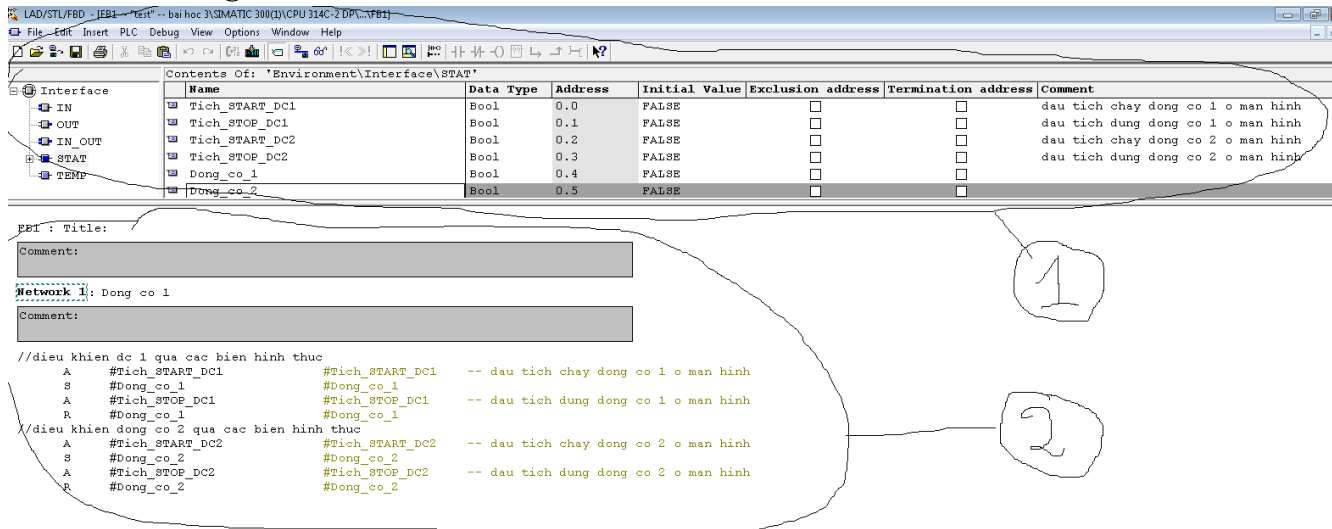


3.2 Khối dữ liệu đi kèm Instance Data Block.

H3.1: Khối Instance Data block trong Step 7 Manager

- 1: Là khối Instance Data Block
- 2: Là khối Share Data Block(có thể dùng chung cho nhiều khối OB, FB, FC)

H3.2: Chương trình được viết với các biến hình thức



- 1: Các biến hình thức được tạo ra để sử dụng điều khiển chương trình. Các biến này có thể nhập, xuất từ bên ngoài(màn hình điều khiển).
- 2: Chương trình sử dụng toàn biến hình thức để đưa ra các đầu ra.

⇒ Các biến hình thức này do có vùng lưu trữ là 1 Instance Data block nên có kích thước số lượng không giới hạn. Do đó nếu sử dụng các biến hình thức này(làm biến TAG ở màn hình hoặc biến trung gian thay cho biến M sẽ sử dụng không giới hạn). Đặc biệt biến STAT sẽ lưu được trạng thái, giá trị khi bị mất điện.

H3.3: Mở một khối Instance Data Block(khối DB1 đi kèm FB1)

Address	Declaration	Name	Type	Initial value	Actual value	Comment
1	stat	Tich_START_DC1	BOOL	FALSE	FALSE	dau tích chạy động cơ 1 o màn hình
2	stat	Tich_STOP_DC1	BOOL	FALSE	FALSE	dau tích dừng động cơ 1 o màn hình
3	stat	Tich_START_DC2	BOOL	FALSE	FALSE	dau tích chạy động cơ 2 o màn hình
4	stat	Tich_STOP_DC2	BOOL	FALSE	FALSE	dau tích dừng động cơ 2 o màn hình
5	stat	Dong_co_1	BOOL	FALSE	FALSE	
6	stat	Dong_co_2	BOOL	FALSE	FALSE	

Handwritten annotations include four vertical lines with numbers 1, 2, 3, and 4 below them, corresponding to the first four rows of the table.

- 1: Địa chỉ Bits, Byte của biến hình thức được lưu trong DATA BLOCK. Trên hình 3.1 là địa chỉ của các biến hình thức từ 0.0 đến 0.6
- + Truy nhập đến địa chỉ. DB1.DBX0.0
- 2: Tên các biến hình thức qui định trong FB sẽ được lưu ở đây.
- 3: Kiểu dữ liệu của các biến hình thức
- 4: Comment của các biến hình thức.

4. Khối Share Data Block

H4.1: Khởi tạo một khối Share Data Block(ví dụ DB100)

DB93	DATI_SUM	DB	42	Data Block
DB94	DATI_position convayer	DB	78	Data Block
DB100	DATA_dung_chung	DB	42	Data Block

- Đây chính là 1 khối Data Block có thể dùng cho nhiều khối OB, FB, FC.

H4.2: Cấu trúc trong khối DB100

The screenshot shows the 'Global Variables' table in the SIMATIC Manager. The table has five columns: Address, Name, Type, Initial value, and Comment. The 'Address' column is highlighted in blue. The variables are defined as follows:

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	xung_thuc_te	INT	0	Xung encoder thuc te
+2.0	Hang_so_K	INT	0	Hang so K=mm/pulse
+4.0	start_DC1	BOOL	FALSE	
+4.1	stop_DC1	BOOL	FALSE	
+4.2	start_DC2	BOOL	FALSE	
+4.3	Stop_DC2	BOOL	FALSE	
=6.0		END_STRUCT		

Địa chỉ Tên biến hình
thức Kiểu dữ liệu

4.1 Cách truy nhập dữ liệu từ Share Data Block

4.1.1 Cách truy nhập xa

Cú pháp **DB<tên khối DB>.DB<tên địa chỉ ô nhớ và vị trí>**

Trong đó:

- DBX là truy nhập Bits
- DBB là truy nhập Byte
- DBW là truy nhập một Word(=2Byte)
- DBD là truy nhập Double Word(=4Byte)

Ví dụ như trên hình 4.2:

Theo thứ tự địa chỉ lần lượt là

- DB100.DBW0 //kiểu Interger(kiểu Work) với địa chỉ 0 trong khối DB100
- DB100.DBW2 //kiểu Interger(kiểu Work) với địa chỉ 0 trong khối DB100
- DB100.DBX4.0 //Kiểu Bits(kiểu Bool) với địa chỉ 4.0 trong khối DB100
- DB100.DBX4.1 //Kiểu Bits(kiểu Bool) với địa chỉ 4.1 trong khối DB100
- DB100.DBX4.2 //Kiểu Bits(kiểu Bool) với địa chỉ 4.2 trong khối DB100
- DB100.DBX4.3 //Kiểu Bits(kiểu Bool) với địa chỉ 4.3 trong khối DB100

H4.1.1: Cách truy nhập xa trong phần mềm

The screenshot shows the SIMATIC Manager interface. The top menu bar includes File, Edit, Insert, PLC, Debug, View, Options, Window, and Help. The main window is divided into two panes. The left pane shows a project tree with 'Interface' expanded, listing variables: IN, OUT, IN_OUT, STAT, and TEMP. The right pane, titled 'Contents Of: 'Environment\Interface'', shows a table of these variables.

Name
IN
OUT
IN_OUT
STAT
TEMP

Below the panes, the Ladder Logic (LAD) editor is visible. It shows two networks. Network 1 is titled 'Network 1' and contains a variable declaration table.

FB1 : Title:

Comment:

Network 1: Title:

Comment:

Variable	Declaration	Address
A	"DATA_dung_chung".start_DC1 //truy nhập bit 4.0 trong DB100	DB100.DBX4.0
S	Q 0.0	
A	"DATA_dung_chung".stop_DC1 //truy nhập bit 4.1 trong DB100	DB100.DBX4.1
R	Q 0.0	
//diou khien dc 2 su dung cac bien hinh thuc trong DATA BLOCK		
A	"DATA_dung_chung".start_DC2 //truy nhập bit 4.2 trong DB100	DB100.DBX4.2
S	Q 0.1	
A	"DATA_dung_chung".Stop_DC2 //truy nhập bit 4.3 trong DB100	DB100.DBX4.3
R	Q 0.1	

→Chú ý

- Các biến hình thức tạo trong DB(vị dụ như trên là DB100.DBX4.0) có thể được khởi tạo không giới hạn
- Các biến này có thể sử dụng để thay thế hoàn toàn cho biến trung gian M.
- Các biến này hoàn toàn có thể lưu được giá trị khi PLC mất điện.