NGUYỄN ĐOÀN PHƯỚC
PHAN XUÂN MINH
VŨ VĂN HÀ

TỰ ĐỘNG HOÁ VỚI Simatic S7-300

NHÀ XUẤT BẢN
KHOA HỌC VÀ KỸ THUẬT

Nguyễn Doãn Phước, Phan Xuân Minh, Vũ Vân Hà

TỰ ĐỘNG HÓA
VỚI
SIMATIC S7-300

In lần thứ 3 có sửa chữa

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2004

Mã số: 6T6.5 / KHKT - 04 1104-81-04

Lời nói đầu

Trung tâm đào tạo Siemens Tự động hóa tại Trường Đại Học Bách Khoa Hà Nội là mô hình hợp tác hoàn toàn mới giữa hãng Siemens và Trường Đại Học Bách Khoa Hà Nội. Sự hợp tác giữa hai mục đích mới nghe tưởng như đầy mâu thuẫn, một bên hầu như chỉ quan tâm đến doanh số bán hàng và một bên quan tâm đến tri thức của học trò. Song cuối cùng, cả hai đã tìm được tiếng nói chung: đó là đào tạo những con người có khả năng sử dụng các thiết bị tự động hóa của hãng và như vậy Trường Đại học Bách khoa Hà Nội sẽ có các kỹ sư Điều khiển Tự động sử dụng thành thạo thiết bị tự động của Siemens– một hãng sản xuất thiết bị công nghiệp hàng đầu thế giới, cũng như hãng Siemens sẽ có những khách hàng trong tương lai luôn sẵn sàng chọn giải pháp kỹ thuật của Siemens .

*Những người đã xây dựng ý tưởng, đã hòa hai mục đích thành một để tạo dựng ra Trung tâm là PGS. Phạm Minh Hà, khi đó chị là Phó hiệu trưởng của Trường và Kỹ sư Nguyễn Thái Hưng, lúc đó anh là đại diện của hãng Siemens ở Việt Nam về lĩnh vực Tự động hóa. Còn những con người mang lại sức sống và sự trưởng thành cho Trung tâm chính là những cán bộ của Bộ môn Điều Khiển Tự Động, những con người đại diện cho Trường thực hiện sự hợp tác và đó cũng chính là các tác giả của cuốn sách "**Tự động hóa với SIMATIC S7-300**".*

Trung tâm đào tạo Siemens Tự động hóa được thành lập ngày 7 tháng 11 năm 1996. Gần bốn năm đã trôi qua, một chặng đường đủ dài để có thể đánh giá và nhìn lại mình. Niềm tự hào của Trung tâm là được hãng đánh giá cao những cống hiến trên lĩnh vực đào tạo. Những kỹ sư trẻ do Trung tâm đào tạo đều tự khẳng định mình trong công tác, nhiều em đã trở thành những cán bộ kỹ thuật chủ chốt của các công ty. Trung tâm cũng đã trở thành địa chỉ đào tạo tin cậy và là nơi tư vấn về các giải pháp kỹ thuật cho nhiều nhà máy, xí nghiệp, của các viện, trường đại học kỹ thuật .... Những công trình nghiên cứu của Trung tâm đã được công bố ở nhiều hội nghị khoa học. Những hệ thống được tích hợp tại Trung tâm bằng thiết bị của hãng được ứng dụng trong công nghiệp và được khách hàng chấp nhận về chất lượng cũng như giá thành.

*Cuốn sách "**Tự động hóa với SIMATIC S7-300**" được hoàn thành với nỗ lực không mệt mỏi của các cán bộ Trung tâm. Ngoài những giờ lên lớp, ngoài những giờ làm thực tế, đào tạo tại hiện trường, khoảng thời gian ngắn ngủi còn lại được các tác giả dành cho cuốn sách. Những kinh nghiệm giảng dạy, những công trình thực tế đã được đúc kết lại để xây dựng cuốn sách. Bên cạnh đó là sự động viên của các em sinh viên, sự cổ vũ của các kỹ sư hiện trường, những con người công việc quá bộn bề, không có những khoảng thời gian để tham dự những khóa đào tạo **nhưng rất ham** mê với các kỹ thuật mới.*

*"**Tự động hoá với SIMATIC S7-300**" được viết với mong muốn sẽ là một tài liệu tham khảo cần thiết cho các kỹ sư tích hợp hệ thống, là một giáo trình tự học tốt cho sinh viên, kỹ sư, học viên cao học và nghiên cứu sinh chuyên ngành Điều khiển Tự động, Tự động hóa, Đo lường và Tin học công nghiệp cũng như các ngành kỹ thuật khác. Cuốn sách được ra đời nhằm phục vụ bạn đọc, nên các tác giả cũng rất mong nhận được những đóng góp và phê bình từ bạn đọc. Mọi ý kiến xin gửi về:*

Trung tâm Đào tạo Siemens Tự động hoá tại Trường ĐHBK Hà Nội

Số 1 Đường Đại Cồ Việt Hà Nội

Tel. 04-8680451 Fax. 04-8680452

hoặc

Nhà xuất bản Khoa học và Kỹ thuật

70 Trần Hưng Đạo Hà Nội

Hà Nội, ngày 9 tháng 4 năm 2000

Các tác giả

Mục lục

1 NHẬP MÔN

Chương mở đầu này có nhiệm vụ giới thiệu những khái niệm nhập môn, các phương pháp cơ bản mô tả hệ thống điều khiển số có sử dụng thiết bị logic khả trình. Sự cần thiết phải nắm vững các phương pháp cơ bản này ta sẽ thấy rõ khi phải thiết kế những hệ thống điều khiển số có độ phức tạp cao.

1.1 Đại số Boole

Nền tảng cơ sở của lý thuyết điều khiển số là *đại số Boole*. Trong một số tài liệu khác nhau, đại số Boole còn được gọi là đại số đóng cắt (*Switching Algebra*).

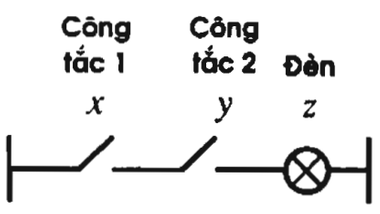
1.1.1 Biến và hàm số hai giá trị

Khi mô tả một đối tượng điều khiển bằng mô hình toán học ta thường phải biểu diễn các đại lượng vào/ra của đối tượng dưới dạng những hàm số phụ thuộc thời gian. Ví dụ điện áp trong dải $-10V \div 10V$ là đại lượng đầu vào của một mạch điện. Khi mô tả mạch điện ta xem điện áp như là một hàm số $u(t)$. Giá trị của hàm số $u(t)$ tại một thời điểm t_0 sẽ mang thông tin về giá trị điện áp đầu vào của mạch điện tại đúng thời điểm đó. Tập tất cả các giá trị của hàm số $u(t)$ gọi là *miền giá trị*. Như vậy, miền giá trị của $u(t)$ biểu diễn điện áp ở ví dụ trên phải thuộc trường số thực $\mathbb{R}$ và nằm trong khoảng $[-10\ ,\ 10]$.

Biến hai trị, hay còn gọi biến Boole là loại hàm số mà miền giá trị của nó chỉ có hai phần tử. Ta sẽ ký hiệu các biến hai trị bằng những chữ nhỏ in nghiêng như x, y, u, v ... và phần tử của chúng là 0 và 1. Ví dụ

- Công tắc là một biến Boole với hai giá trị: đóng (ký hiệu là 1) và mở (ký hiệu là 0).
- Đèn hiệu cũng là một biến Boole với hai trạng thái: sáng (ký hiệu là 1) và tắt (ký hiệu là 0).

Hai biến Boole được gọi là *độc lập với nhau* nếu sự thay đổi giá trị của biến số này không ảnh hưởng tới giá trị của biến số kia. Ví dụ hai công tắc trong hình 1.1 là hai biến Boole độc lập với nhau.



Hình 1.1. Minh họa biến độc lập và biến phụ thuộc.

Ngược lại, nếu giá trị của một biến số y phụ thuộc vào giá trị của biến số x thì biến y được gọi là *biến phụ thuộc* của biến x. Ví dụ trong hình 1.1 thì

đèn là biến phụ thuộc vào hai biến công tắc. Đèn sẽ sáng nếu cả hai biến công tắc có giá trị 1 và sẽ tắt khi một trong hai biến công tắc có giá trị 0.

Hàm hai trị là mô hình toán học mô tả sự phụ thuộc của một biến Boole vào các biến Boole khác. Chẳng hạn như để biểu diễn sự phụ thuộc của đèn, ký hiệu là z, vào hai biến công tắc, ký hiệu là x và y, ta viết

$$z = f(x,y). \tag{1.1}$$

Một cách tổng quát, hàm hai trị mô tả sự phụ thuộc của biến số y vào n biến $x_1, x_2, \cdots, x_n$ có dạng

$$y = f(x_1, x_2, \cdots, x_n).$$

Việc mô tả sự phụ thuộc của một biến Boole này vào các biến Boole khác thành hàm hai trị dựa vào ba phép tính cơ bản. Đó là các phép tính **và** (ký hiệu là $\wedge$), **hoặc** (ký hiệu là $\vee$), **phủ định** (ký hiệu là $^-$) được định nghĩa như sau:

Phép tính $\wedge$			Phép tính $\vee$			Phép phủ định	
x	y	$x \wedge y$	x	y	$x \vee y$	x	$\bar{x}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Ví dụ, hàm $f(x,y)$ biểu diễn biến đèn z phụ thuộc vào hai biến công tắc x, y sẽ là:

$$z = f(x,y) = x \wedge y, \tag{1.2}$$

trong đó phép tính $x \wedge y$ đôi khi còn được viết đơn giản thành $x \cdot y$ hay xy.

1.1.2 Định nghĩa và tính chất

Xét tập hợp $\mathbf{B}$ tất cả các biến hai trị với ba phép tính và ($\wedge$), hoặc ($\vee$), phủ định ($^-$) định nghĩa theo bảng trên. Biến hai trị trong $\mathbf{B}$ luôn có giá trị 1 sẽ là phần tử đơn vị đối với phép tính $\wedge$, tương tự biến luôn có giá trị 0 là phần tử đơn vị của phép tính $\vee$. Ta sẽ lấy ngay giá trị của nó để ký hiệu các biến đơn vị này. Vậy thì

$$x \wedge 1 = 1 \wedge x = x, \quad \text{với mọi } x \in \mathbf{B}. \tag{1.3}$$

$$x \vee 0 = 0 \vee x = x, \quad \text{với mọi } x \in \mathbf{B}. \tag{1.4}$$

Định nghĩa: Không bị giới hạn bởi quy định của bảng chân lý về các phép tính $\wedge$, $\vee$, $^-$, nếu trên $\mathbf{B}$ ta xác định được ba phép tính $\wedge$, $\vee$, $^-$, bất kỳ nào đó thỏa mãn:

1) $x \vee y = y \vee x$ (tính giao hoán) (1.5)

2) $x \vee (y \wedge z) = (x \vee y) \wedge z$ (tính kết hợp) (1.6)

3) $(x \wedge y) \vee (x \vee \bar{y}) = x$ (1.7)

với mọi $x, y \in \mathbf{B}$ thì tập $\mathbf{B}$ cùng ba phép tính đó sẽ được gọi là *đại số Boole*.

Một đại số Boole $\mathbf{B}$ với ba phép tính $\wedge$ (hay $\cdot$), $\vee$, $^-$ có các tính chất sau:

a) $\bar{\bar{x}} = x, \quad \forall x \in \mathbf{B}.$ (1.8)

b) $x = x \cdot x = x \vee x, \quad \forall x \in \mathbf{B}.$ (1.9)

c) $\bar{x} \cdot x = 0, \quad \forall x \in \mathbf{B}.$ (1.10)

d) $1 \vee x = 1, \quad \forall x \in \mathbf{B}.$ (1.11)

e) $0 \vee x = x, \quad \forall x \in \mathbf{B}.$ (1.12)

f) $\bar{x} \vee x = 1, \quad \forall x \in \mathbf{B}.$ (1.13)

g) $\overline{x \cdot y} = \bar{x} \vee \bar{y}, \quad \forall x, y \in \mathbf{B}.$ (công thức De Morgan thứ nhất) (1.14)

h) $\overline{x \vee y} = \bar{x} \cdot \bar{y}, \quad \forall x, y \in \mathbf{B}.$ (công thức De Morgan thứ hai) (1.15)

i) $(x \vee y) \cdot z = (x \cdot z) \vee (y \cdot z), \quad \forall x, y, z \in \mathbf{B}.$ (1.16)

j) $(x \cdot y) \vee z = (x \vee z) \cdot (y \vee z), \quad \forall x, y, z \in \mathbf{B}.$ (1.17)

k) $(x \vee \bar{z}) \cdot (y \vee z) = (x \cdot z) \vee (y \cdot \bar{z})$ (1.18)

l) $(x \cdot z) \vee (y \cdot \bar{z}) \vee (x \cdot y) = (x \cdot z) \vee (y \cdot \bar{z})$ (1.19)

m) $f(x_1, x_2, \cdots, x_n) = [\bar{x}_1 \cdot f(0, x_2, \cdots, x_n)] \vee [x_1 \cdot f(1, x_2, \cdots, x_n)].$ (1.20)

Gọi $y = f(x_1, x_2, \cdots, x_n)$ là một hàm hai trị của n biến $x_1, x_2, \cdots, x_n$ định nghĩa trên $\mathbf{B}$. Nếu ta thay trong $y = f(x_1, x_2, \cdots, x_n)$ tất cả các biến x_i bằng $\bar{x}_i$, ngược lại $\bar{x}_i$ được thay bằng x_i, phép tính $\vee$ thay bởi $\wedge$, đổi $\wedge$ thành $\vee$ thì hàm $\tilde{f}(x_1, x_2, \cdots, x_n)$ nhận được có tên là hàm đối ngẫu của $f(x_1, x_2, \cdots, x_n)$ và

n) $\bar{y} = \tilde{f}(x_1, x_2, \cdots, x_n).$ (1.21)

Việc chứng minh các tính chất (1.8)÷(1.21) dành cho bạn đọc như những bài tập ôn luyện. Các tính chất này rất có ích trong việc rút gọn công thức biểu diễn hàm hai trị $f(x_1, x_2, \cdots, x_n)$ và qua đó làm giảm đáng kể linh kiện, câu lệnh khi phải thực hiện việc cài đặt chúng thành thiết bị.

Khi rút gọn, biến đổi hay tính giá trị của hàm hai trị, các biểu thức trong ngoặc phải được thực hiện trước, tiếp theo là phép tính $\wedge$ (giống như tính nhân) rồi sau cùng mới là phép tính $\vee$.

Ta xét việc rút gọn công thức

$$f(x,y,z,v) = \overline{\overline{x \cdot \bar{y}} \cdot [(\bar{1} \vee \bar{y}) \cdot v \vee ((z \cdot \bar{v}) \vee (\bar{z} \cdot v))]}$$

làm một ví dụ minh họa.

Từ $\overline{1} = 0$ và (1.6) có $\overline{1} \vee \overline{y} = \overline{y}$ nên

$$f(x,y,z,v) = \overline{\overline{x \cdot \overline{y}} \cdot [\overline{y} \cdot v \vee ((z \cdot \overline{v}) \vee (\overline{z} \cdot v))]}.$$

Áp dụng công thức De Morgan và (1.8) được

$$\begin{aligned} f(x,y,z,v) &= \overline{\overline{x \cdot \overline{y}}} \vee \overline{[\overline{y} \cdot v \vee ((z \cdot \overline{v}) \vee (\overline{z} \cdot v))]} \\ &= x \cdot \overline{y} \vee \overline{[\overline{y} \cdot v \vee ((z \cdot \overline{v}) \vee (\overline{z} \cdot v))]}. \end{aligned}$$

Vì phép tính $\wedge$ phải được thực hiện trước $\vee$ nên ta có thể viết thành

$$f(x,y,z,v) = (x \cdot \overline{y}) \vee \overline{[(\overline{y} \cdot v) \vee ((z \cdot \overline{v}) \vee (\overline{z} \cdot v))]}.$$

Nếu áp dụng tiếp công thức De Morgan cho thành phần thứ hai thì

$$\begin{aligned} f(x,y,z,v) &= (x \cdot \overline{y}) \vee [\overline{(\overline{y} \cdot v)} \cdot \overline{((z \cdot \overline{v}) \vee (\overline{z} \cdot v))}] \\ &= (x \cdot \overline{y}) \vee [(y \vee \overline{v}) \cdot (\overline{(z \cdot \overline{v})} \cdot \overline{(\overline{z} \cdot v)})] \\ &= (x \cdot \overline{y}) \vee [(y \vee \overline{v}) \cdot ((\overline{z} \vee v) \cdot (z \vee \overline{v}))] \end{aligned}$$

và cuối cùng ta thu được công thức đơn giản

$$f(x,y,z,v) = x\,\overline{y} \vee \overline{y}\,\overline{v} \vee yzv. \qquad \square$$

Từ nay về sau, để thuận tiện trong việc trình bày, ta sẽ sử dụng ký hiệu $\underline{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ để biểu diễn hàm hai trị có nhiều biến $f(x_1, x_2, \cdots, x_n)$ thành $f(\underline{x})$.

Quá trình biến đổi và rút gọn biểu thức, được minh họa ở ví dụ trên, có thể gồm nhiều bước, song mỗi bước biến đổi bao giờ cũng thỏa mãn:

a) Cho hai hàm tương đương $f(\underline{x})$, $g(\underline{x})$, tức là $f(\underline{x})=g(\underline{x})$. Tính tương đương của hai hàm sẽ không thay đổi nếu trong cả hai hàm đó ta cùng thay biến số x_k bằng một biểu thức $x_k = h(\underline{x})$. Ví dụ, từ $\overline{x_1} \vee x_2 = x_1 \vee x_2$ ta thay $x_1 = x_2 x_3$ vào cả hai vế thì vẫn giữ được sự tương đương của nó $\overline{\overline{x_2 x_3}} \vee x_2 = x_2 x_3 \vee x_2$.

b) Nếu biến số y phụ thuộc $x_1, x_2, \cdots, x_n$ biểu diễn nhờ hàm $y = f(\underline{x})$ và hàm $f(\underline{x})$ có chứa một biểu thức con $g(\underline{x})$, tức là $y=f(\underline{x}, g(\underline{x}))$, thì khi thay biểu thức con đó bằng một biểu thức con tương đương khác $h(\underline{x})=g(\underline{x})$ ta vẫn có $y=f(\underline{x}, h(\underline{x}))$.

Ví dụ nếu trong $y = \overline{x_1 x_2} \vee x_3$

ta thay $\overline{x_1 x_2} = \overline{x}_1 \vee \overline{x}_2$

thì vẫn có $y = \overline{x}_1 \vee \overline{x}_2 \vee x_3$. $\square$

1.1.3 Xác định công thức hàm hai trị từ bảng chân lý

Một trong những cách đơn giản nhất để mô tả hàm hai trị $f(\underline{x})$ là biểu diễn chúng dưới dạng bảng, được gọi là bảng chân lý. Hình 1.2 là một ví dụ về bảng chân lý của hàm

$$f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3. \tag{1.22}$$

Bảng được xây dựng bằng cách liệt kê tất cả các trường hợp có thể có khi mà n biến $x_1, x_2, \cdots, x_n$ nhận những giá trị khác nhau thành từng hàng riêng biệt. Do tất cả các biến là hai trị nên bảng chân lý của hàm với n biến sẽ chỉ có hữu hạn 2^n hàng. Tiếp theo, tại cuối mỗi hàng ta gán giá trị của hàm, được xác định bằng cách thay những giá trị $x_1, x_2, \cdots, x_n$ tương ứng trong hàng đó vào công thức của hàm số. Ví dụ ở hình bên thì tại hàng thứ 3, khi thay $x_1=0, x_2=1$ và $x_3=0$ vào công thức của hàm sẽ có

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Hình 1.2. Bảng giá trị chân lý.

$$f(0, 1, 1) = 0\bar{1}0 \vee \bar{0}1\bar{0} = 1.$$

Nhờ có cách biểu diễn hàm hai trị dưới dạng bảng chân lý như vậy mà ta có thể dễ dàng kiểm chứng được rằng với n biến $x_1, x_2, \cdots, x_n$ chỉ có thể có nhiều nhất 2^{2^n} hàm hai trị $f(\underline{x})$ khác nhau. Do đó trong số $2^{2^n}+1$ hàm hai trị $f_k(\underline{x})$, $k=0,1, \ldots, 2^{2^n}$ của n biến chắc chắn phải có ít nhất 2 hàm là tương đương.

Sau đây ta sẽ xét bài toán ngược là tìm công thức biểu diễn hàm $f(\underline{x})$ từ bảng giá trị chân lý đã biết của hàm đó. Công việc này là cần thiết vì trong thực tế nhiều bài toán tổng hợp bộ điều khiển được bắt đầu từ bảng chân lý.

Trước hết hãy làm quen với hai khái niệm mới là *biểu thức nguyên tố tổng* và *biểu thức nguyên tố tích*. Cho n biến hai trị $x_1, x_2, \cdots, x_n$. Một biểu thức $T(\underline{x})$ của n biến đó được gọi là *nguyên tố* nếu trong $T(\underline{x})$:

- có mặt tất cả các biến số x_k, $k=1,2, \cdots, n$ và mỗi biến số chỉ xuất hiện một lần,
- được cấu thành chỉ bởi hai phép tính $\wedge, \bar{}$ hoặc $\vee, \bar{}$.

Ví dụ:

$$T_1(x_1,x_2,x_3) = \bar{x}_1x_2x_3 \qquad \text{(tạo bởi hai phép tính } \wedge, \bar{}),$$

$$T_2(x_1,x_2,x_3) = \bar{x}_1 \vee x_2 \vee \bar{x}_3 \qquad \text{(tạo bởi hai phép tính } \vee, \bar{})$$

là các biểu thức nguyên tố. ❑

Biểu thức nguyên tố với hai phép tính $\wedge$, $^-$ được gọi là *biểu thức nguyên tố tích* còn biểu thức nguyên tố với $\vee$, $^-$ gọi là *biểu thức nguyên tố tổng*. Trong ví dụ trên $T_1(x_1,x_2,x_3)$ là biểu thức nguyên tố tích còn $T_2(x_1,x_2,x_3)$ là biểu thức nguyên tố tổng. Để tiện cho việc trình bày, ta quy ước

$$\bar{x}_k = x_k^0 \qquad \text{và} \qquad x_k = x_k^1 \tag{1.23}$$

vậy thì một biểu thức nguyên tố tích $T_N(\underline{x})$ với n biến hai trị $x_1, x_2, \cdots, x_n$ có dạng

$$T_N(\underline{x}) = x_1^{q1} x_2^{q2} \cdots x_n^{qn} \underset{def.}{=} \prod_{k=1}^{n} x_k^{qk} \tag{1.24}$$

và một biểu thức nguyên tố tổng $T_C(\underline{x})$ với n biến hai trị $x_1, x_2, \cdots, x_n$ biểu diễn thành

$$T_C(\underline{x}) = x_1^{q1} \vee x_2^{q2} \vee \cdots \vee x_n^{qn} \underset{def.}{=} \sum_{k=1}^{n} x_k^{qk}, \tag{1.25}$$

trong đó

$$qk = \begin{cases} 0 & \text{nếu biến } x_k \text{ xuất hiện dưới dạng phủ định} \\ 1 & \text{nếu biến } x_k \text{ xuất hiện dưới dạng không phủ định} \end{cases} \tag{1.26}$$

Từ hai công thức định nghĩa (1.24), (1.25) thấy ngay được rằng các biểu thức nguyên tố đều có đặc điểm:

a) Biểu thức nguyên tố tích $T_N(\underline{x})$ theo công thức (1.24) có giá trị 1 khi và chỉ khi tất cả thừa số x_k^{qk}, $k=1,2, \cdots, n$ cùng có giá trị 1. Như vậy, nếu x_k xuất hiện trong biểu thức dạng phủ định (qk=0) thì x_k phải có giá trị 0, ngược lại khi qk=1 thì x_k phải có giá trị 1.

b) Biểu thức nguyên tố tổng $T_C(\underline{x})$ theo công thức (1.25) có giá trị 0 khi và chỉ khi tất cả thừa số x_k^{qk}, $k=1,2, \cdots, n$ cùng có giá trị 0. Do đó nếu x_k xuất hiện trong biểu thức dạng phủ định (qk=0) thì x_k phải có giá trị 1, ngược lại khi qk=1 thì x_k phải có giá trị 0.

Bây giờ ta quay lại công việc chính là xác định công thức biểu diễn hàm hai trị $f(\underline{x})$ từ bảng chân lý của nó.

Xác định nhờ biểu thức nguyên tố tích

Bảng chân lý hàm $f(x)$ của n biến $x_1, x_2, \cdots, x_n$ gồm có 2^n hàng. Giả sử rằng từ bảng chân lý ta xác định được hàm $f(\underline{x})$ có giá trị 1 ở hàng thứ i. Theo tính chất vừa nêu trên của biểu thức nguyên tố tích thì hàm $f(\underline{x})$ khi đó sẽ có giá trị đúng bằng giá trị của biểu thức nguyên tố tích

$$T_N^i(\underline{x}) = x_1^{q1} x_2^{q2} \cdots x_n^{qn},$$

trong đó các giá trị qk phải được chọn theo quy luật:

$$qk = \begin{cases} 0 & \text{nếu biến } x_k \text{ có giá trị 0 tại hàng thứ } i \\ 1 & \text{nếu biến } x_k \text{ có giá trị 1 tại hàng thứ } i \end{cases} \quad (1.27)$$

Bởi vậy hàm $f(\underline{x})$ sẽ tương đương với kết quả phép HOẶC của tất cả các biểu thức nguyên tố tích $T_N^i(\underline{x})$ của các hàng i mà tại đó $f(\underline{x})$ có giá trị 1. Nếu gọi các hàng trong bảng chân lý mà tại đó $f(\underline{x})$=1 lần lượt là $i1, i2, \ldots$ và $T_N^{i1}(\underline{x}), T_N^{i2}(\underline{x}), \ldots$ là những biểu thức nguyên tố tích của các hàng đó thì

$$f(\underline{x}) = T_N^{i1}(\underline{x}) \vee T_N^{i1}(\underline{x}) \vee \cdots$$

Ta sẽ minh họa quy tắc trên bằng một ví dụ cụ thể. Hình 1.3 là bảng chân lý của hàm gồm 4 biến x_1, x_2, x_3, và x_4. Trước hết ta đánh dấu những hàng mà tại đó hàm $f(\underline{x})$ có giá trị 1. Trong hình, những hàng này được đã được đánh dấu bằng một khung chữ nhật. Tiếp theo, tại các hàng đã đánh dấu ta xác định biểu thức nguyên tố tích $T_N^i(\underline{x})$ theo quy tắc nêu trong (1.27). Các biểu thức này bao gồm

$$T_N^1(\underline{x}) = \bar{x}_1\bar{x}_2x_3\bar{x}_4$$

$$T_N^2(\underline{x}) = \bar{x}_1x_2\bar{x}_3x_4$$

$$T_N^3(\underline{x}) = x_1\bar{x}_2\bar{x}_3x_4$$

$$T_N^4(\underline{x}) = x_1x_2\bar{x}_3x_4$$

x_1	x_2	x_3	x_4	$f(\underline{x})$	$T_N^i(\underline{x})$
0	0	0	0	0	
0	0	0	1	0	
0	**0**	**1**	**0**	**1**	$\bar{x}_1\bar{x}_2x_3\bar{x}_4$
0	0	1	1	0	
0	1	0	0	0	
0	**1**	**0**	**1**	**1**	$\bar{x}_1x_2\bar{x}_3x_4$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	**0**	**0**	**1**	**1**	$x_1\bar{x}_2\bar{x}_3x_4$
1	0	1	1	0	
1	1	0	0	0	
1	**1**	**0**	**1**	**1**	$x_1x_2\bar{x}_3x_4$
1	1	1	0	0	
1	1	1	1	0	

Hình 1.3. Xác định công thức biểu diễn hàm.

Suy ra

$$f(\underline{x}) = \bar{x}_1\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1x_2\bar{x}_3x_4 .$$

Xác định nhờ biểu thức nguyên tố tổng

Việc xác định công thức mô tả hàm $f(x)$ với n biến $x_1, x_2, \cdots, x_n$ từ bảng chân lý nhờ biểu thức nguyên tố tổng cũng được thực hiện tương tự như đã làm với biểu thức nguyên tố tích và được tóm tắt thành các bước sau:

1) Trước tiên ta đánh dấu trong bảng chân lý gồm 2^n hàng của hàm $f(x)$ tất cả các hàng mà tại đó $f(\underline{x})$ có giá trị 0.
2) Giả sử tại hàng thứ i có $f(\underline{x})$=0. Tại đó ta lập biểu thức nguyên tố tổng theo quy tắc

$$T_C^i(\underline{x}) = x_1^{q1} \vee x_2^{q2} \vee \cdots \vee x_n^{qn}$$

trong đó các giá trị qk phải được chọn theo quy luật:

$$qk = \begin{cases} 0 & \text{nếu biến } x_k \text{ có giá trị 1 trong hàng thứ } i \\ 1 & \text{nếu biến } x_k \text{ có giá trị 0 trong hàng thứ } i \end{cases} \quad (1.28)$$

3) Gọi tất cả các hàng có $f(\underline{x})=0$ lần lượt là $i1, i2, \ldots$ và $T_C^{i1}(\underline{x}), T_C^{i2}(\underline{x}), \ldots$ là những biểu thức nguyên tố tổng tương ứng của các hàng đó thì

$$f(\underline{x}) = T_C^{i1}(\underline{x}) \wedge T_C^{i2}(\underline{x}) \wedge \cdots$$

Ví dụ: Hàm $f(x)$ của 3 biến x_1, x_2, và x_3. có bảng chân lý cho trong hình 1.4. Ta đánh dấu tất cả các hàng mà tại đó $f(\underline{x})=0$ và lập biểu thức nguyên tố tổng theo quy tắc đã nêu trong (1.28) cho các hàng này. Các biểu thức đó bao gồm

$$T_C^1(\underline{x}) = x_1 \vee x_2 \vee \bar{x}_3$$

$$T_C^2(\underline{x}) = \bar{x}_1 \vee x_2 \vee x_3$$

$$T_C^3(\underline{x}) = \bar{x}_1 \vee \bar{x}_2 \vee x_3$$

x_1	x_2	x_3	$f(\underline{x})$	$T_C^i(\underline{x})$
0	0	0	1	
0	0	1	0	$x_1 \vee x_2 \vee \bar{x}_3$
0	1	0	1	
0	1	1	1	
1	0	0	0	$\bar{x}_1 \vee x_2 \vee x_3$
1	0	1	1	
1	1	0	0	$\bar{x}_1 \vee \bar{x}_2 \vee x_3$
1	1	1	1	

Hình 1.4. Xác định công thức biểu diễn hàm.

Suy ra

$$f(\underline{x}) = \left(x_1 \vee x_2 \vee \bar{x}_3\right)\left(\bar{x}_1 \vee x_2 \vee x_3\right)\left(\bar{x}_1 \vee \bar{x}_2 \vee x_3\right).$$

1.2 Biểu diễn tín hiệu số

1.2.1 Tín hiệu số là gì?

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

Bộ điều khiển số là một bộ điều khiển không làm việc với tín hiệu liên tục. Dạng tín hiệu thích ứng cho bộ điều khiển số là dãy các giá trị $\{u_k\}$, $u_k=u(kT_u)$, trong đó T_u là khoảng thời gian trích mẫu. Bởi vậy trong điều khiển số người ta cần phải rời rạc hóa $u(t)$ thành $\{u_k\}$. Quá trình rời rạc hóa miền xác định của $u(t)$ để có được dãy đếm được $\{u_k\}$ gọi là *lượng tử hóa* tín hiệu theo thời gian.

Việc lượng tử hóa tín hiệu theo thời gian là cần thiết, nhưng chưa đủ vì bộ điều khiển số cũng không thể bao quát được tất cả giá trị u_k trong khoảng $-\infty < u_k < \infty$, ví dụ nó không làm việc được với số vô tỷ. Thông thường bộ điều khiển số chỉ chấp nhận tập hợp đếm được các giá trị u_k. Việc thay tập không đếm được các giá trị của $u(t)$ bằng tập đếm được các giá trị u_k gọi là quá trình rời rạc hóa miền giá trị của $u(t)$.

Tín hiệu $u(t)$ mà cả miền xác định và miền giá trị là những tập đếm được được gọi là tín hiệu số.

Như vậy, giá trị u_k của tín hiệu số là một giá trị gần đúng được chọn làm đại diện cho tất cả các giá trị của $u(t)$ trong cả hai lân cận $t=kT_u$ và $u_k=u(kT_u)$. Bởi vậy, không mất tính tổng quát, người ta có thể quy đổi để xem u_k như là một số nguyên. Chẳng hạn, nếu lân cận của $u_k=u(kT_u)$ là các số thực có cùng 3 số sau dấu phảy với nó thì sau khi nhân u_k với 10^3 ta sẽ có một số nguyên (cách biểu diễn dấu phảy tĩnh).

1.2.2 Biểu diễn số nguyên dương

Biểu diễn trong hệ cơ số 10

Một số nguyên dương u_k bất kỳ, trong hệ cơ số 10 bao giờ cũng được biểu diễn đầy đủ bằng dãy các con số nguyên từ 0 đến 9. Ví dụ $u_k=259$ được biểu diễn nhờ 3 con số 2, 5 và 9 và cách biểu diễn đó được hiểu là

$$u_k = 2\cdot 10^2+5\cdot 10^1+9\cdot 10^0.$$

Một cách tổng quát, khi biểu diễn trong hệ cơ số 10, u_k có dạng

$$u_k = a_n\cdot 10^n+a_{n-1}\cdot 10^{n-1}+ \ldots + a_1\cdot 10^1+a_0\cdot 10^0 \qquad \text{với } 0\le a_i\le 9 \qquad (1.29)$$

Như vậy việc biểu diễn u_k trong hệ cơ số 10 là sự biến đổi u_k thành tập hữu hạn $n+1$ số số nguyên a_i, $i=0,1,\ldots,n$ thỏa mãn $0\le a_i\le 9$. Nói cách khác đó là ánh xạ

$$u_k \mapsto \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix}.$$

Số các giá trị mà a_i có được do hệ cơ số biểu diễn u_k quyết định. Trong trường hợp này u_k được biểu diễn trong hệ cơ số 10 nên a_i sẽ có 10 giá trị.

Biểu diễn trong hệ cơ số 2

Cách biểu diễn u_k trong hệ cơ số 10 theo (1.29) chưa phù hợp với nguyên tắc mạch điện của bộ điều khiển số vì vector ảnh $\underline{a}=\begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix}$ của u_k có các phần tử đa trị $0\le a_i\le 9$.

Để chuyển vector ảnh $\underline{a}$ thành các phần tử hai trị, ta biến đổi (1.29) về dạng sau

$$u_k = x_n\cdot 2^n+x_{n-1}\cdot 2^{n-1}+ \ldots + x_1\cdot 2^1+x_0\cdot 2^0 \qquad \text{với } x_i\in\{0,1\} \qquad (1.30)$$

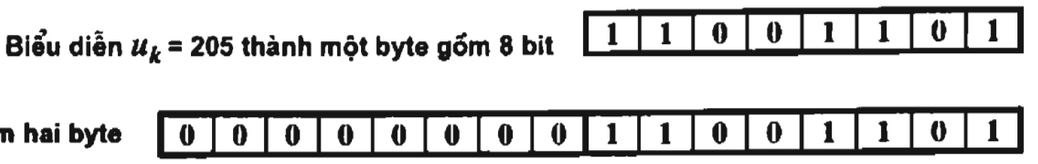
Với việc thay đổi này, các tham số x_i, $i=0,1,\ldots,n$ sẽ trở thành những đại lượng hai trị 0 hoặc 1 và (1.30) biến thành ánh xạ $u_k\to\underline{x}=\begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}$ có x_i là các biến hai trị. Nếu sử dụng ký hiệu vector hàng cho ảnh $\underline{x}$ theo cấu trúc

$u_k \rightarrow$ 

ta sẽ đi đến dạng biểu diễn thông dụng bằng mạch điện cho tín hiệu số. Mỗi ô vuông trong cách biểu diễn trên gọi là một bit và mỗi bịt là một biến hai trị.

Số các bit của vector $\underline{x}$ quyết định miền giá trị cho u_k. Với n+1 bit trong (1.30) thì miền giá trị của u_k sẽ là tập các số nguyên dương trong khoảng $0 \leq u_k \leq 2^{n+1}-1$.

Một dãy 8 bit được gọi là một byte. Hai bytes gọi là một từ (word) và hai từ sẽ được gọi là từ kép (double word). Trong kỹ thuật PLC nói riêng và điều khiển số nói chung người ta thường biểu diễn u_k bằng một byte, một từ hoặc bằng một từ kép.



Cách biểu diễn trong hệ cơ số 2 như vậy không ảnh hưởng tới thói quen tính toán của ta trong hệ thập phân như cộng, trừ. Tuy nhiên vẫn phải để ý rằng do x_i chỉ bằng 0 hoặc 1 nên khi cộng có tổng lớn hơn 1 ta phải viết x_i=0 và nhớ 1 sang hàng sau. Ví dụ khi thực hiện 53+27 trong hệ cơ số 2 sẽ có

53 =	0	0	1	1	0	1	0	1
+								
27 =	0	0	0	1	1	0	1	1
Nhớ			1	1	1	1	1	
Tổng: 80=	0	1	0	1	0	0	0	0

Mã hexadecimal của số nguyên dương

Giống như công thức (1.29), (1.30) về cách biểu diễn u_k theo hệ cơ số 2 và 10, trong hệ cơ số 16 (*hexadecimal*), số nguyên dương u_k có dạng

$$u_k = h_n \cdot 16^n + h_{n-1} \cdot 16^{n-1} + \ldots + h_1 \cdot 16^1 + h_0 \cdot 16^0, \text{ với } 0 \leq h_i \leq 15 \qquad (1.31)$$

và tham số h_i, i=0,1, ...,n là những biến 16 trị. Các trị số của h_i được ký hiệu là

$$0,1,2,\ \ldots,9,A,B,C,D,E,F.$$

trong đó các ký tự khi chuyển sang hệ thập phân sẽ tương đương với

$$A=10 \quad B=11 \quad C=12 \quad D=13 \quad E=14 \quad F=15$$

Để bộ điều khiển số hiểu được dạng biểu diễn (1.31) của u_k, người ta đã chuyển các tham số h_i, i=0,1, ..., n sang hệ cơ số 2. Do mỗi tham số có 16 giá trị nên người ta cũng chỉ cần 4 bit là đủ để biểu diễn chúng.

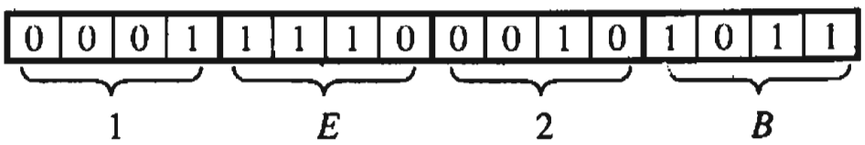
Một mảng 4 bit có tên gọi là một *nipple*.

Ví dụ: Số nguyên dương $u_k = 7723$ trong hệ cơ số 10, khi chuyển sang hệ cơ số 16 sẽ là $1E2B$ vì

$$7723 = 1\cdot 16^3 + \underbrace{14}_{E}\cdot 16^2 + 2\cdot 16 + \underbrace{11}_{B}$$

và do đó dạng hexadecimal của nó sẽ như sau:

Dạng hexadecimal của số nguyên 7723



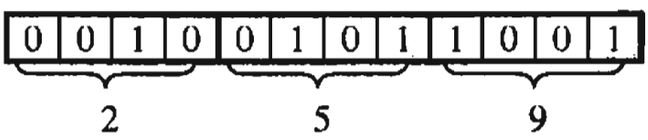
Mã BCD của số nguyên dương

Ta đã biết mã hexadecimal là kiểu sử dụng biến hai trị để thể hiện các chữ số h_i, i=0,1, …,n khi u_k được biểu diễn trong hệ cơ số 16. Hoàn toàn tương tự, mã BCD là dạng dùng biến hai trị thể hiện những chữ số $0 \le a_i \le 9$, i=0,1, …,n khi biểu diễn u_k trong hệ cơ số 10 theo công thức

$$u_k = a_n\cdot 10^n + a_{n-1}\cdot 10^{n-1} + \ \ldots\ + a_1\cdot 10^1 + a_0\cdot 10^0.$$

Ví dụ: u_k=259 được biểu diễn nhờ 3 con số 2, 5 và 9 và do đó mã BCD của nó có dạng như sau

Mã BCD của số nguyên 259



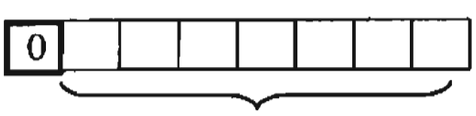
1.2.3 Biểu diễn số nguyên có dấu

Như vậy qua mục trên ta đã có các kiểu biểu diễn khác nhau cho một số nguyên dương u_k. Đối với số nguyên (âm hoặc dương) u_k, do cần phải biểu diễn dấu

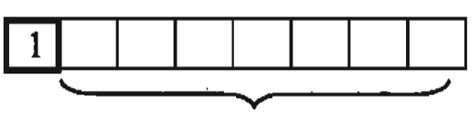
$$\text{sgn}(u_k) = \begin{cases} 0 & \text{nếu } u_k \ge 0 \\ 1 & \text{nếu } u_k < 0 \end{cases} \qquad (1.32)$$

dưới dạng một biến hai trị ta phải có thêm một bit. Bit dấu này được quy định là bit đầu tiên trong mảng bit sử dụng để biểu diễn u_k. Nói cách khác là trong mảng bit dành cho việc biểu diễn số nguyên u_k giờ đây phải bớt đi một bit cho dấu sgn(u_k).

Ví dụ để biểu diễn u_k trong hệ cơ số 2 bằng dãy 8 bit (1 byte) ta sẽ có



$0 \le u_k \le 127$



$-127 \le u_k \le -1$

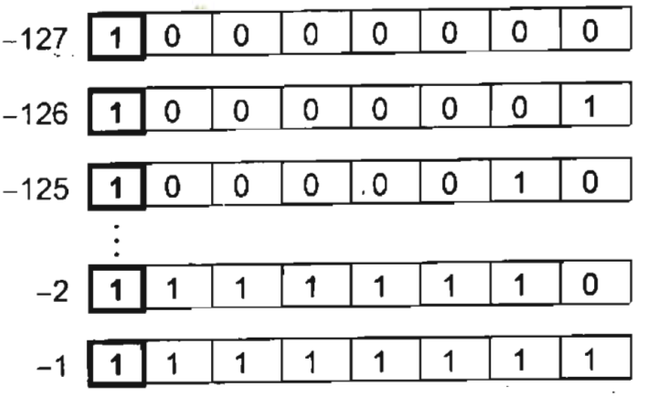
Vấn đề biểu diễn $0 \le u_k \le 127$ không có gì khác những gì đã trình bày tại mục trước vì lúc này u_k là một số nguyên dương. Riêng việc biểu diễn u_k khi $-127 \le u_k \le -1$ có hơi khác, nó không đơn thuần là biểu diễn $|u_k|$ vì nếu như vậy quy luật

$$|u_k| > |u_l| \Rightarrow -|u_k| < -|u_l| \qquad (1.33)$$

sẽ không được thỏa mãn. Người ta đã đảo lại thứ tự biểu diễn $|u_k|$ để có thể có được

$$u_k = 0 - |u_k|$$

trong dãy các bits đã dành cho việc biểu diễn và điều này dẫn tới:



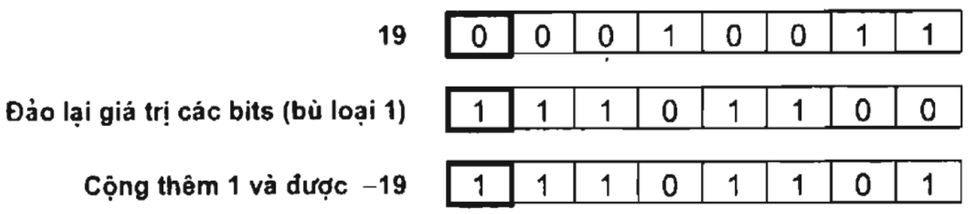
Công thức đảo lại thứ tự biểu diễn của số nguyên âm cũng dễ nhớ theo quy tắc bù loại 2 gồm các bước như sau:

a) Biểu diễn $|u_k|$ trong hệ cơ số 2 thành dãy các bit x_k, $k=1,2,\cdots,n$

b) Đảo giá trị từng bit x_k, $k=1,2,\cdots,n$ thành $\overline{x}_k$ (hay còn gọi bù loại 1)

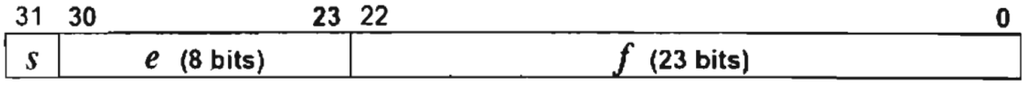
c) Cộng thêm 1

Ví dụ để biểu diễn $u_k = -19$ trong hệ cơ số 2 với độ dài 8 bit, ta thực hiện lần lượt 3 bước trên sẽ được



1.2.4 Số thực dấu phảy động

Khác với việc biểu diễn số nguyên là kích thước mảng bits có thể là 16 hoặc 32, một số thực dấu phảy động bao giờ cũng được biểu diễn thành dãy 32 bits (một từ kép). Dạng cấu trúc dấu phảy động của một số thực u_k như sau

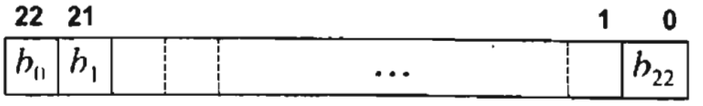


trong đó

a) s là bit dấu,

b) e là chỉ số mũ gồm 8 bits (exponent),

c) f là phần hệ số (mantissa) gồm 23 bits với cấu trúc

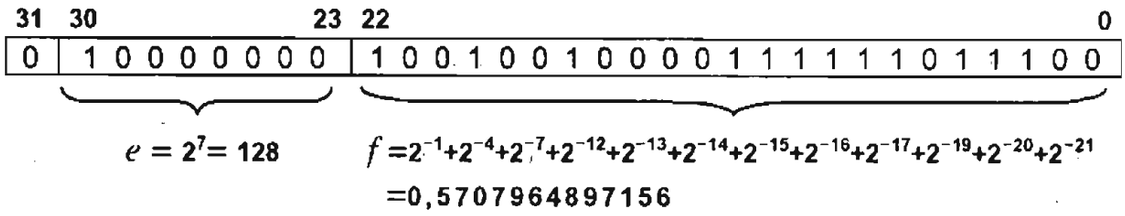


và $$f = b_0 2^{-1} + b_1 2^{-2} + \cdots + b_{22} 2^{-23},$$

Số thực u_k được tính như sau:

- nếu $0<e<255$ thì $u_k=(-1)^s\ 2^{e-127}(1+f)$
- nếu $e=0$ và $f\neq 0$ thì $u_k=(-1)^s\ 2^{-126}f$
- nếu $e=0$ và $f=0$ thì $u_k=0$
- nếu $e=255$ và $f=0$ thì $u_k=(-1)^s\infty$
- các trường hợp khác được xem không phải là số thực (NaN).

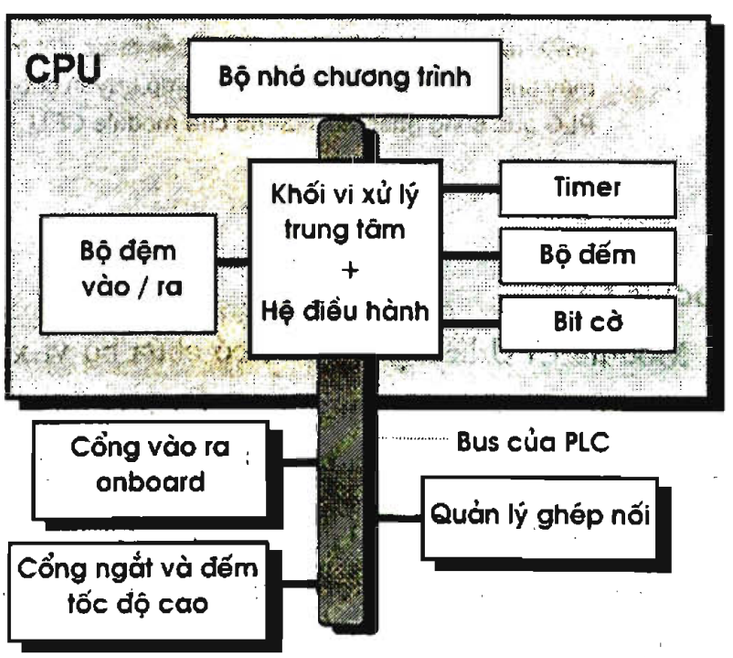
Ví dụ dạng dấu phảy động của số thực $u_k = 3{,}141592979431$ là



$$\Rightarrow \quad u_k=(-1)^s 2^{e-127}(1+f) = 2\cdot 1{,}5707964897156$$
$$= 3{,}141592979431.$$

1.3 Thiết bị điều khiển logic khả trình

Thiết bị điều khiển logic khả trình (*Programmable Logic Control*), viết tắt thành 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 một bộ điều khiển số 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 điều khiển được lưu nhớ trong bộ nhớ của PLC dưới dạng các khối chương trình (khối OB, FC hoặc FB) và được thực hiện lặp theo chu kỳ của vòng quét (scan).

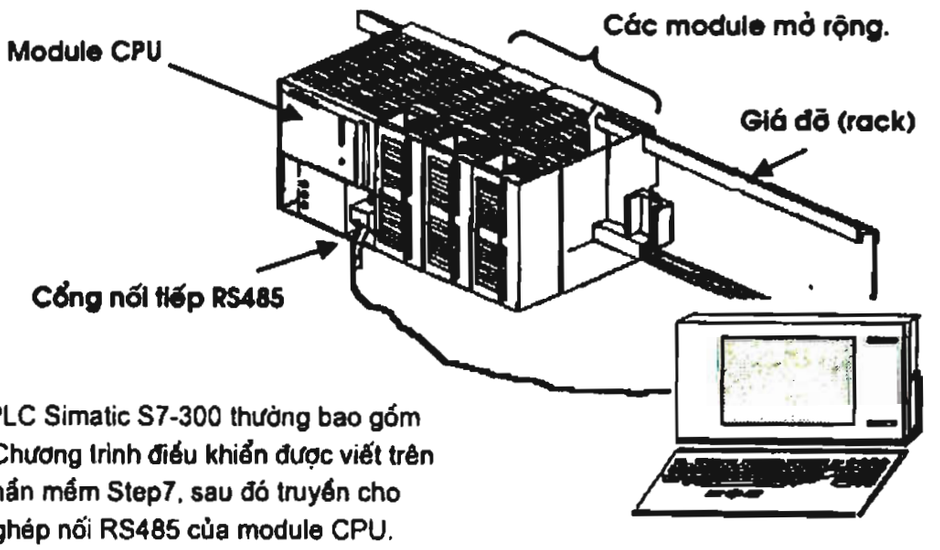


Hình 1.5. Nguyên lý chung về cấu trúc của một bộ điều khiển logic khả trình (PLC)

Để có thể thực hiện được một chương trình điều khiển, tất nhiên PLC phải có tính năng như một máy tính, nghĩa là phải có một bộ vi xử lý (CPU), một hệ điều hành, bộ nhớ để lưu chương trình điều khiển, dữ liệu và tất nhiên là phải có các cổng vào/ra để giao tiếp được với đối tượng điều khiển và để trao đổi thông tin với môi trường xung quanh [13]. Bên cạnh đó, nhằm phục vụ bài toán điều khiển số, PLC còn cần phải có thêm các khối chức năng đặc biệt khác như bộ đếm (Counter), bộ thời gian (Timer) ... và những khối hàm chuyên dụng (hình 1.5).

1.3.1 Các module 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 hóa về cấu hình. Chúng được chia nhỏ thành các module. Số các số module được sử dụng nhiều hay ít tùy theo từng bài toán, song tối thiểu bao giờ cũng phải có một module chính là module CPU. Các module còn lại là những module nhận/truyền tín hiệu với đối tượng điều khiển, các module chức năng chuyên dụng như PID, điều khiển động cơ .... Chúng được gọi chung là module mở rộng. Tất cả các module được gá trên những thanh ray (Rack).



Hình 1.6. Một thiết bị PLC Simatic S7-300 thường bao gồm nhiều module. Chương trình điều khiển được viết trên máy tính nhờ phần mềm Step7, sau đó truyền cho PLC qua cổng ghép nối RS485 của module CPU.

Module CPU

Module CPU là loại module có chứa bộ vi xử lý, 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 có trên module CPU được gọi là cổng vào ra *onboard*.

Trong họ PLC S7–300 có nhiều loại module CPU khác nhau. Nói chung chúng được đặt tên theo bộ vi xử lý có trong nó như module CPU312, module CPU314, module CPU315 ...

Những module cùng sử dụng một loại bộ vi xử lý, nhưng khác nhau về cổng vào/ra onboard cũng như các khối hàm đặc biệt được tích 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 trong tên gọi bằng thêm cụm chữ cái IFM (viết tắt của *Intergrated Function Module*). Ví dụ module CPU312 IFM, module CPU314 IFM ...

Ngoài ra còn có các loại module CPU với hai cổng truyền thông, trong đó cổng truyền thông thứ hai có chức năng chính là phục vụ việc nối mạng phân tán. Tất nhiên kèm theo cổng truyền thông thứ hai này là những phần mềm tiện dụng thích hợp cũng đã được cài sẵn trong hệ điều hành. Các loại module CPU được phân biệt với những module CPU khác bằng thêm cụm từ DP (*Distributed Port*) trong tên gọi. Ví dụ module CPU315–DP,

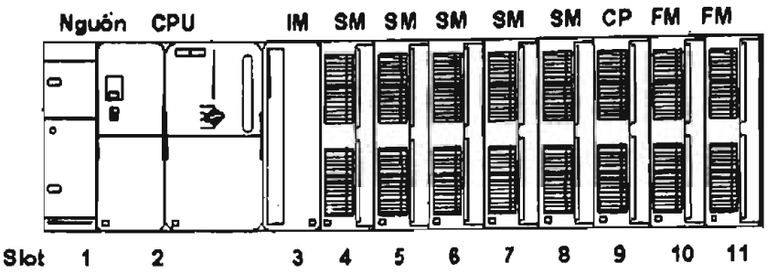
Module mở rộng

Các module mở rộng được chia thành 5 loại chính:

1) PS (*Power supply*): Module nguồn nuôi. Có 3 loại 2A, 5A và 10A.
2) SM (*Signal module*): Module mở rộng cổng tín hiệu vào/ra, bao gồm:
   a) DI (*Digital input*): Module mở rộng các cổng vào số. Số các cổng vào số mở rộng có thể là 8, 16 hoặc 32 tùy thuộc vào từng loại module.
   b) DO (*Digital output*): Module mở rộng các cổng ra số. Số các cổng ra số mở rộng có thể là 8, 16 hoặc 32 tùy thuộc vào từng loại module.
   c) DI/DO: (*Digital input/Digital output*): Module mở rộng các cổng vào/ra số. Số các cổng vào/ra số mở rộng có thể là 8 vào/8 ra hoặc 16 vào/16 ra tùy thuộc vào từng loại module.
   d) AI (*Analog input*): Module mở rộng các cổng vào tương tự. Về bản chất chúng chính là những bộ chuyển đổi tương tự số 12 bits (AD), tức là mỗi tín hiệu tượng tự được chuyển thành một tín hiệu số (nguyên) có độ dài 12 bits. Số các cổng vào tương tự có thể là 2, 4 hoặc 8 tùy từng loại module.
   e) AO(*Analog output*): Module mở rộng các cổng ra tương tự. Chúng chính là những bộ chuyển đổi số tương tự (DA). Số các cổng ra tương tự có thể là 2 hoặc 4 tùy từng loại module.
   f) AI/AO (*Analog input /Analog output*): Module mở rộng các cổng vào/ra tương tự. Số các cổng ra tương tự có thể là 4 vào/2 ra, hoặc 4 vào/4 ra tùy từng loại module.
3) IM (*Interface module*): Module ghép nối. Đây là loại module chuyên dụng có nhiệm vụ nối từng nhóm các module mở rộng lại với nhau thành một khối và được quản lý chung bởi một module CPU. Thông thường các module 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 S7–300 có thể làm việc trực tiếp được với nhiều nhất 4 racks và các racks này phải được nối với nhau bằng module IM.

4) FM (*Function module*): Module có chức năng điều khiển riêng, ví dụ như module điều khiển động cơ bước, module điều khiển động cơ servo, module PID, module điều khiển vòng kín, ...

5) CP (*Communication module*): Module phục vụ truyền thông trong mạng giữa các PLC với nhau hoặc giữa PLC với máy tính.



Hình 1.7. Cấu hình một thanh rack của trạm PLC S7-300.

1.3.2 Kiểu dữ liệu và phân chia bộ nhớ

Kiểu dữ liệu

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

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

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

```
L     B#16#14   // Nạp số nguyên 14 viết theo hệ cơ số 16 độ dài 1 byte vào ACCU1.
```

3) **WORD**: gồm 2 bytes, để biểu diễn một số nguyên dương từ 0 đến 65535. Ví dụ

```
L     930
L     W#16#3A2.
```

4) **INT**: cũng có dung lượng là 2 bytes, dùng để biểu diễn một số nguyên trong khoảng -32768 ÷ 32767. Ví dụ

```
L     930
L     W#16#3A2.
```

5) **DINT**: gồm 4 bytes, dùng để biểu diễn một số nguyên từ -2147483648 đến 2147483647. Ví dụ

```
L     L#930
L     DW#16#3A2.
```

6) **REAL**: gồm 4 bytes, dùng để biểu diễn một số thực dấu phảy động. Ví dụ

```
L     1.234567e+13
L     930.0
```

7) **S5T** (hay **S5TIME**): khoảng thời gian, được tính theo giờ/phút/giây/mili giây. Ví dụ

```
L     S5T#2h_1m_0s_5ms.
```

là lệnh tạo khoảng thời gian là hai tiếng một phút và 5 mili giây.

8) **TOD**: biểu diễn giá trị thời gian tính theo giờ/phút/giây. Ví dụ

```
L     TOD#5:45:00.
```

là lệnh khai báo giá trị thời gian trong ngày là 6 giờ kém 15.

9) **DATE**: biểu diễn giá trị thời gian tính theo năm/tháng/ngày. Ví dụ

```
L     DATE#1999-12-8.
```

là lệnh khai báo ngày mùng 8 tháng 12 năm 1999.

10) **CHAR**: biểu diễn một hoặc nhiều ký tự (nhiều nhất là 4 ký tự). Ví dụ

```
L     'ABCD'.
L     'E'
```


Cấu trúc bộ nhớ của CPU (xem thêm chương 3, đặc biệt mục 3.1.4)

Bộ nhớ của S7–300 được chia làm ba vùng chính:

1) Vùng chứa chương trình ứng dụng. Vùng nhớ chương trình được chia thành 3 miền:
   a) OB (*Organisation block*): Miền chứa chương trình tổ chức.
   b) FC (*Function*): Miền chứa chương trình con được tổ chức thành hàm có biến hình thức để trao đổi dữ liệu với chương trình đã gọi nó.
   c) FB (*Function block*): Miền chứa chương trình con, được tổ chức thành hàm và có khả năng trao đổi dữ liệu với bất cứ một khối chương trình nào khác. Các dữ liệu này phải được xây dựng thành một khối dữ liệu riêng (gọi là DB – *Data block*).
2) Vùng chứa tham số của hệ điều hành và chương trình ứng dụng, được phân chia thành 7 miền khác nhau, bao gồm:
   a) I (*Process image input*): Miền bộ đệm các dữ liệu 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.
   b) 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 vào bộ đệm Q.
   c) M: Miền các biến cờ. Chương trình ứng dụng sử dụng vùng nhớ này để lưu giữ các tham số cần thiết và có thể truy nhập nó theo bit (M), byte (MB), từ (MW) hay từ kép (MD).
   d) T: Miền nhớ phục vụ bộ thời gian (*Timer*) bao gồm việc lưu giữ 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.
   e) C: Miền nhớ phục vụ bộ đếm (*Counter*) bao gồm việc lưu giữ giá trị đặt trước (PV – *Preset value*), giá trị đếm tức thời (CV – *Current value*) và giá trị logic đầu ra của bộ đếm.

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

g) 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 tương 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).

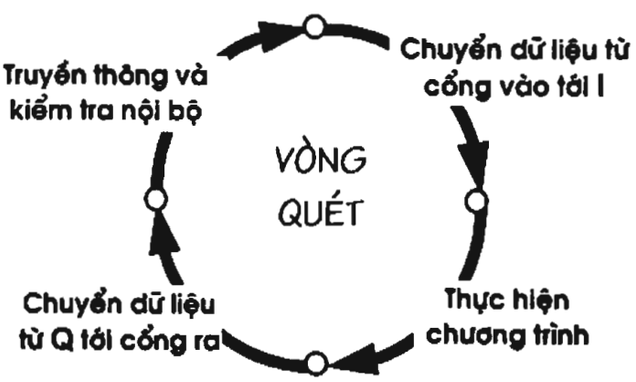
3) Vùng chứa các khối dữ liệu, được chia thành 2 loại:

a) DB (*Data block*): Miền chứa các 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).

b) 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 đã gọi nó. Nội dung của một số dữ liệu trong miền nhớ này sẽ bị xóa 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).

1.3.3 Vòng quét chương trình

PLC thực hiện chương trình theo chu trình 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 được thực hiện từ lệnh đầu tiên đến lệnh kết thúc của khối OB1 (Block End). Sau giai đoạn thực hiện chương trình là giai đoạn chuyển các nội dung của bộ đệm ảo Q tới các cổng ra số. Vòng quét được kết thúc bằng giai đoạn truyền thông nội bộ và kiểm lỗi (hình 1.8).



Hình 1.8. Vòng quét chương trình.

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

Thời gian cần thiết để PLC thực hiện được một 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 tùy thuộc vào số lệnh trong chương trình được thực hiện, vào khối lượng dữ liệu được truyền thông ... trong vòng quét đó.

Như vậy giữa việc đọc dữ liệu từ đối tượng để xử lý, tính toán và việc gửi tín hiệu điều khiển tới 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.

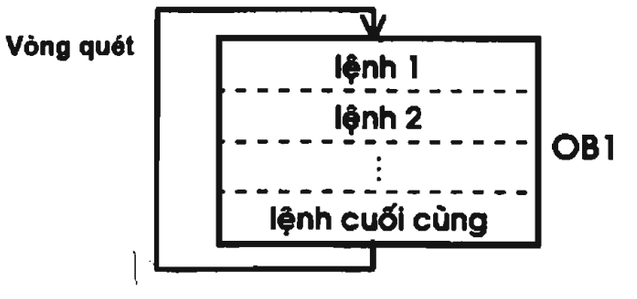
Nếu sử dụng các khối chương trình đặc biệt có chế độ ngắt, ví dụ như khối OB40, OB80 ..., chương trình của các khối đó sẽ được thực hiện trong vòng quét khi xuất hiện tín hiệu báo ngắt cùng chủng loại. Các khối chương trình này có thể được thực hiện tại mọi điểm trong vòng quét chứ không bị gò ép là phải ở trong giai đoạn thực hiện chương trình. Chẳng hạn nếu một tín hiệu báo ngắt xuất hiện khi PLC đang ở giai đoạn truyền thông và kiểm tra nội bộ, PLC sẽ tạm dừng công việc truyền thông, kiểm tra, để thực hiện khối chương trình tương ứng với tín hiệu báo ngắt đó. Với hình thức xử lý 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 lý ngắt quá dài hoặc quá lạm dụng việc sử dụng chế độ ngắt trong chương trình điều khiển.

Tại thời điểm thực hiện lệnh vào/ra, thông thường lệnh không làm việc trực tiếp với cổng vào/ra mà chỉ thông qua bộ 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 lý. Ở một số module CPU, khi gặp *lệnh vào/ra ngay lập tức*, hệ thống sẽ cho dừng mọi công việc khác, ngay cả chương trình xử lý ngắt, để thực hiện lệnh trực tiếp với cổng vào/ra.

1.3.4 Cấu trúc chương trình (xem thêm mục 3.1.1 của chương 3)

Chương trình cho S7–300 được lưu trong bộ nhớ của PLC ở vùng dành riêng cho chương trình và có thể được lập với hai dạng cấu trúc khác nhau:

1) *Lập trình tuyến tính*: Toàn bộ chương trình điều khiển 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 nó thường xuyên, từ lệnh đầu tiên đến lệnh cuối cùng và quay lại lệnh đầu tiên (hình 1.9).

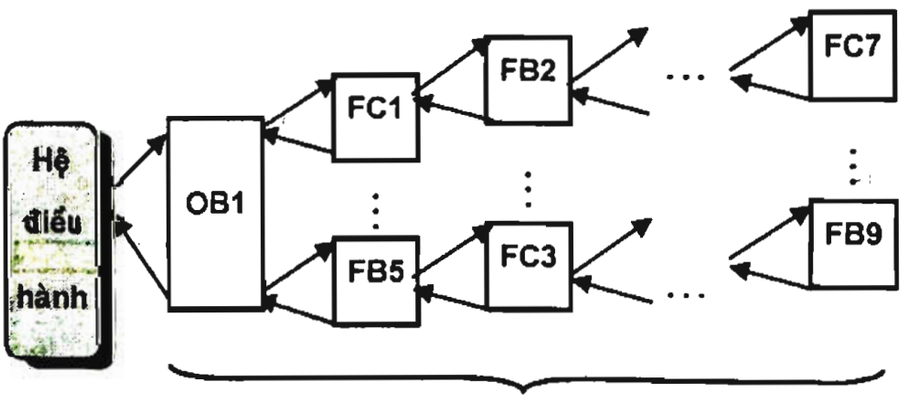


Hình 1.9. Lập trình tuyến tính.

2) *Lập trình có cấu trúc*: Chương trình được chia thành những phần nhỏ với từng nhiệm vụ riêng và các 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 bốn loại khối cơ bản:

   - Loại khối OB (*Organization block*): Khối tổ chức và quản lý 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 ký tự OB, ví dụ như OB1, OB35, OB40, OB80, ...
   - Loại khối FC (*Program block*): Khối chương trình với những chức năng riêng giống như một 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 ký tự FC. Chẳng hạn như FC1, FC2, ...
   - Loại khối FB (*Function block*): Là loại khối FC đặc biệt có khả năng trao đổi một lượng dữ liệu lớn với các khối chương trình khác. Các dữ liệu này phải 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 ký 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 ký tự DB. Ví dụ DB1, DB2, ...

Chương trình trong các khối được liên kết với nhau bằng các lệnh 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 từ chương trình con này gọi một chương trình con khác và từ 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 sử dụng. Ví dụ như đối với module CPU314 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 sang chế độ STOP và đặt cờ báo lỗi.



Hình 1.9. Lập trình có cấu trúc.

Khối OB1 luôn được PLC quét và thực hiện các lệnh từ lệnh đầu tiên đến lệnh cuối cùng và quay lại lệnh đầu tiên như đã trình bày ở hình 1.8.

1.3.5 Những khối OB đặc biệt

Trong khi khối OB1 được thực hiện đều đặn ở từng vòng quét trong giai đoạn thực hiện chương trình (giai đoạn 2) 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 lý tín hiệu ngắt (event). Chúng bao gồm:

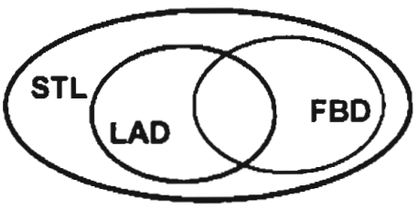
1) OB10 (*Time of Day Interrupt*): Chương trình trong khối OB10 sẽ được thực hiện khi giá trị của đồng hồ thời gian thực nằm trong một khoảng thời gian đã được quy định. OB10 có thể được gọi một lần, nhiều lần cách đều nhau từng phút, từng giờ, từng ngày, ... Việc quy định khoảng thời gian hay số lần gọi OB10 được thực hiện nhờ chương trình hệ thống SFC28 hoặc trong bảng tham số của module CPU nhờ phần mềm STEP 7.
2) OB20 (*Time Delay Interrupt*): Chương trình trong khối OB20 sẽ được thực hiện sau một khoảng thời gian trễ đặt trước kể từ khi gọi chương trình hệ thống SFC32 để đặt thời gian trễ.
3) OB35 (*Cyclic Interrupt*): Chương trình trong OB35 sẽ được thực hiện cách đều nhau một khoảng thời gian cố định. Mặc định, khoảng thời gian này sẽ là 100ms, song ta có thể thay đổi nó trong bảng tham số của module CPU nhờ phần mềm STEP 7.
4) OB40 (*Hardware Interrupt*): Chương trình trong OB40 sẽ được thực hiện khi xuất hiện một 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.
5) OB80 (*Cyle 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 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, 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.
6) OB81 (*Power Supply Fault*): Module CPU sẽ gọi chương trình trong khối OB81 khi phát hiện tháy có lỗi về nguồi nuôi.
7) OB82 (*Diagnostic Interrupt*): Chương trình trong OB82 được gọi khi CPU phát hiện có sự cố từ các module vào/ra mở rộng. Các module mở rộng này phải là những module có khả năng tự kiểm tra mình (diagnostic cabilities).
8) OB85 (*Not Load Fault*): CPU sẽ gọi khối OB85 khi phát hiện thấy chương trình ứng dụng có sử dụng chế độ ngắt nhưng chương trình xử lý tín hiệu ngắt lại không có trong khối OB tương úng.
9) 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 đối tác.

10) OB100 (*Start Up Information*): Khối OB100 sẽ được thực hiện một lần khi CPU chuyển trạng thái từ STOP (dừng) sang RUN (chạy).
11) OB101 (*Cold Start Up Information* – Chỉ có với S7–400): Khối OB101 sẽ được thực hiện một lần khi công tắc nguồn của CPU chuyển trạng thái từ OFF (tắt) sang ON (mở).
12) OB121 (*Synchronous error*): Khối OB121 sẽ được thực hiện khi CPU phát hiện thấy lỗi logic trong chương trình như đổi sai kiểu dữ liệu hoặc lỗi truy nhập khối DB, FC, FB không có trong bộ nhớ của CPU.
13) OB122 (*Synchronous error*): Khối OB122 sẽ được thực hiện khi CPU phát hiện thấy lỗi truy nhập module trong chương trình, ví dụ chương trình có lệnh truy nhập module vào ra mở rộng nhưng lại không tìm thấy module này.

2 NGÔN NGỮ LẬP TRÌNH STL

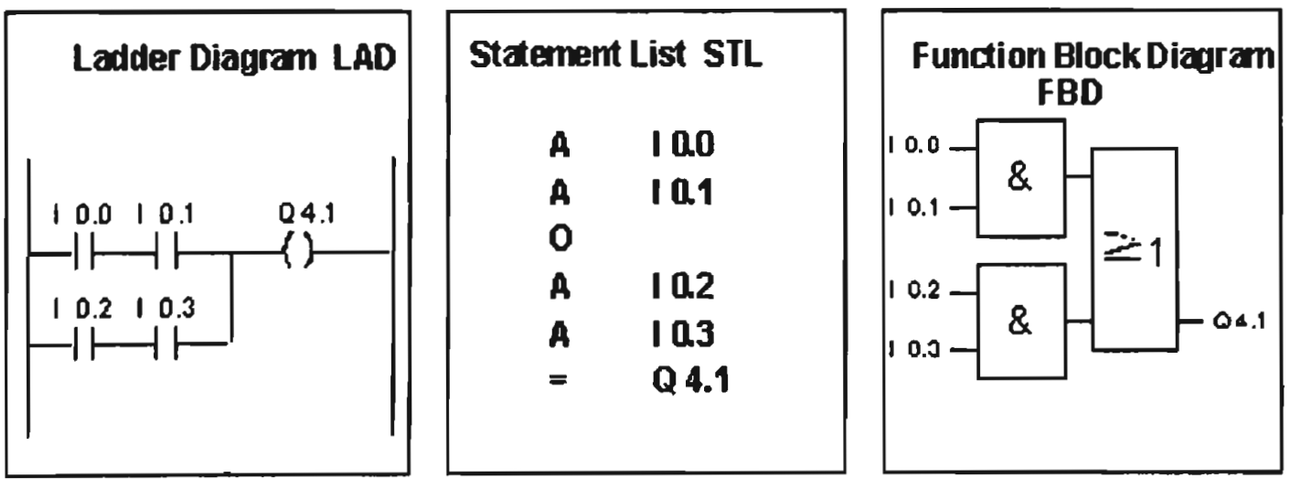
Các loại PLC nói chung thường có nhiều ngôn ngữ lập trình nhằm phục vụ các đối tượng sử dụng khác nhau. PLC S7–300 có ba ngôn ngữ lập trình cơ bản. Đó là:

- Ngôn ngữ "liệt kê lệnh", ký hiệu là STL (*Statement list*). Đây là dạng ngôn ngữ lập trình thông thường của máy tính. Một chương trình được ghép bởi nhiều câu lệnh theo một thuật toán nhất định, mỗi lệnh chiếm một hàng và đều có cấu trúc chung "tên lệnh" + "toán hạng".
- Ngôn ngữ "hình thang", ký hiệu là LAD (*Ladder logic*). Đây là dạng ngôn ngữ đồ họa thích hợp với những người quen thiết kế mạch điều khiển logic.



Hình 2.1. STL là ngôn ngữ mạnh nhất trong ba loại ngôn ngữ lập trình cho S7-300.

- Ngôn ngữ "hình khối", ký hiệu là FBD (*Function block diagram*). Đây cũng là kiểu ngôn ngữ đồ họa dành cho người có thói quen thiết kế mạch điều khiển số.



Hình 2.2. Ba kiểu ngôn ngữ lập trình chính cho S7-300

Một chương trình viết trên LAD hoặc FBD có thể chuyển sang được dạng STL, nhưng ngược lại thì không. Trong STL có nhiều lệnh không có trong LAD hay FBD (hình 2.1). Cũng chính vì lý do đó, trong tài liệu này chúng tôi chọn STL làm ngôn ngữ chính để lập trình minh họa.

2.1 Cấu trúc lệnh và trạng thái kết quả

Như đã nói, cấu trúc của một lệnh STL có dạng: "tên lệnh" + "toán hạng". Ví dụ

```
Nhãn: L      PIW304    // Đọc nội dung cổng vào của module analog
      ↑      ↑
   tên lệnh  toán hạng
```

trong đó toán hạng có thể là một dữ liệu hoặc một địa chỉ ô nhớ.

2.1.1 Toán hạng là dữ liệu

1) Dữ liệu logic TRUE (1) và FALSE (0) có độ dài 1 bit. Ví dụ

```
CALL  FC1
   In_Bit_1:   = TRUE      // Giá trị logic 1 được gán cho biến hình thức In_Bit_1
   In_Bit_2:   = FAULSE    // Giá trị logic 0 được gán cho biến hình thức In_Bit_2
   Ret_val:    = MW0       // Giá trị trả về
```

2) Số nhị phân. Ví dụ

```
L     2#110011   // Nạp số nhị phân 110011 vào thanh ghi ACCU1.
```

3) Số hexadecimal *x* có độ dài 1 byte (B#16#*x*), 1 từ (W#16#*x*) hoặc 1 từ kép (DW#16#*x*) . Ví dụ

```
L     B#16#1E          // Nạp số 1E vào byte thấp của thanh ghi ACCU1
L     W#16#3A2         // Nạp số 3A2 vào 2 byte thấp của thanh ghi ACCU1
L     DW#16#D3A2E      // Nạp số D3A2E vào thanh ghi ACCU1
```

4) Số nguyên *x* với độ dài 2 bytes cho biến kiểu INT. Ví dụ

```
L     930
L     -1025
```

5) Số nguyên *x* với độ dài 4 bytes dạng L#*x* cho biến kiểu DINT. Ví dụ

```
L     L#930
L     L#-2047
```

6) Số thực *x* cho biến kiểu REAL Ví dụ

```
L     1.234567e+13
L     930.0
```

7) Dữ liệu thời gian cho biến kiểu S5T dạng giờ_phút_giây_mili giây. Ví dụ

```
L     S5T#2h_1m_0s_5ms.
```

8) Dữ liệu thời gian cho biến kiểu TOD dạng giờ:phút:giây. Ví dụ

```
L     TOD#5:45:00.
```

9) DATE: biểu diễn giá trị thời gian tính theo năm/tháng/ngày. Ví dụ

```
L     DATE#1999-12-8.
```

10) C: biểu diễn giá trị số đếm đặt trước cho bộ đếm. Ví dụ

```
L     C#20..
```

11) P: dữ liệu biểu diễn địa chỉ của một bit ô nhớ. Ví dụ

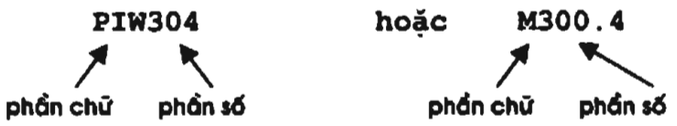
```
L     P#Q0.0.
```

12) Dữ liệu 'ký tự'. Ví dụ

```
L     'ABCD'.
L     'E'
```

2.1.2 Toán hạng là địa chỉ

Địa chỉ ô nhớ trong S7–300 gồm hai phần: phần chữ và phần số. Ví dụ



1) *Phần chữ* chỉ vị trí và kích thước của ô nhớ. Chúng có thể là:
   - M: chỉ ô nhớ trong miền các biến cờ có kích thước là 1 bit.
   - MB: chỉ ô nhớ trong miền các biến cờ có kích thước là 1 byte (8 bit)
   - MW: chỉ ô nhớ trong miền các biến cờ có kích thước là 2 bytes (16 bits)
   - MD: chỉ ô nhớ trong miền các biến cờ có kích thước là 4 bytes (32 bits).
   - I: chỉ ô nhớ có kích thước là 1 bit trong miền bộ đệm cổng vào số.
   - IB: chỉ ô nhớ có kích thước là 1 byte trong miền bộ đệm cổng vào số.
   - IW: chỉ ô nhớ có kích thước là 1 từ trong miền bộ đệm cổng vào số.
   - ID: chỉ ô nhớ có kích thước là 2 từ trong miền bộ đệm cổng vào số.
   - Q: chỉ ô nhớ có kích thước là 1 bit trong miền bộ đệm cổng ra số.
   - QB: chỉ ô nhớ có kích thước là 1 byte trong miền bộ đệm cổng ra số.
   - QW: chỉ ô nhớ có kích thước là 1 từ trong miền bộ đệm cổng ra số.
   - QD: chỉ ô nhớ có kích thước là 2 từ trong miền bộ đệm cổng ra số.
   - T: chỉ ô nhớ trong miền nhớ của bộ thời gian (*Timer*). Mặc dù cùng tên song nó có thể là địa chỉ của bit đầu ra bộ timer hay địa chỉ của thanh ghi đếm tức thời CV. Tùy vào lệnh mà địa chỉ này được hiểu là địa chỉ của bit đầu ra hay của thanh ghi CV. Ví dụ

```
A     T1     // T1 là địa chỉ đầu ra (bit) của bộ đếm.
L     T1     // T1 là địa chỉ của thanh ghi 16 bits CV.
```

   - C: chỉ ô nhớ trong miền nhớ của bộ đếm (*Counter*). Mặc dù cùng tên song nó có thể là địa chỉ của bit đầu ra bộ đếm hay địa chỉ của thanh ghi đếm tức thời CV. Tùy vào lệnh mà địa chỉ này được hiểu là địa chỉ của bit đầu ra hay của thanh ghi CV. Ví dụ

```
A     C1     // C1 là địa chỉ đầu ra (bit) của bộ đếm.
L     C1     // C1 là địa chỉ của thanh ghi 16 bits CV.
```

   - PIB: chỉ ô nhớ có kích thước 1 byte thuộc vùng peripheral input. Thường là địa chỉ cổng vào của các module tương tự (I/O *external input*).

- PIW: chỉ ô nhớ có kích thước 1 từ (2 bytes) thuộc vùng peripheral input. Thường là địa chỉ cổng vào của các module tương tự (I/O *external input*).
- PID: chỉ ô nhớ có kích thước 2 từ (4 bytes) thuộc vùng peripheral input. Thường là địa chỉ cổng vào của các module tương tự (I/O *external input*).
- PQB: chỉ ô nhớ có kích thước 1 byte thuộc vùng peripheral output. Thường là địa chỉ cổng ra của các module tương tự (I/O *external output*).
- PQW: chỉ ô nhớ có kích thước 1 từ (2 bytes) thuộc vùng peripheral output. Thường là địa chỉ cổng ra của các module tương tự (I/O *external output*).
- PQD: chỉ ô nhớ có kích thước 2 từ (4 bytes) thuộc vùng peripheral output. Thường là địa chỉ cổng ra của các module tương tự (I/O *external output*).
- DBX: chỉ ô nhớ có kích thước 1 bit trong khối dữ liệu DB được mở bằng lệnh **OPN DB** (*Open data block*).
- DBB: chỉ ô nhớ có kích thước 1 byte trong khối dữ liệu DB được mở bằng lệnh **OPN DB** (*Open data block*).
- DBW: chỉ ô nhớ có kích thước 1 từ trong khối dữ liệu DB được mở bằng lệnh **OPN DB** (*Open data block*).
- DBD: chỉ ô nhớ có kích thước 2 từ trong khối dữ liệu DB được mở bằng lệnh **OPN DB** (*Open data block*).
- DBx.DBX: chỉ trực tiếp ô nhớ có kích thước 1 bit trong khối dữ liệu DBx, với x là chỉ số của khối DB. Ví dụ **DB5.DBX 1.6**.
- DBx.DBB: chỉ trực tiếp ô nhớ có kích thước 1 byte trong khối dữ liệu DBx, trong đó x là chỉ số của khối DB. Ví dụ **DB5.DBB 1**.
- DBx.DBW: chỉ trực tiếp ô nhớ có kích thước 1 từ trong khối dữ liệu DBx với x là chỉ số khối DB. Ví dụ **DB5.DBW 1**.
- DBx.DBD: chỉ trực tiếp ô nhớ có kích thước 2 từ trong khối dữ liệu DBx, trong đó x là chỉ số khối DB. Ví dụ **DB5.DBD 1**.
- DIX: chỉ ô nhớ có kích thước 1 bit trong khối dữ liệu DB được mở bằng lệnh **OPN DI** (*Open instance data block*).
- DIB: chỉ ô nhớ có kích thước 1 byte trong khối dữ liệu DB được mở bằng lệnh **OPN DI** (*Open instance data block*).
- DIW: chỉ ô nhớ có kích thước 1 từ trong khối dữ liệu DB được mở bằng lệnh **OPN DI** (*Open instance data block*).
- DID: chỉ ô nhớ có kích thước 2 từ trong khối dữ liệu DB được mở bằng lệnh **OPN DI** (*Open instance data block*).
- L: chỉ ô nhớ có kích thước 1 bit trong miền dữ liệu địa phương (local block) của các khối chương trình OB, FC, FB.
- LB: chỉ ô nhớ có kích thước 1 byte trong miền dữ liệu địa phương (local block) của các khối chương trình OB, FC, FB.

- LW: chỉ ô nhớ có kích thước 1 từ trong miền dữ liệu địa phương (local block) của các khối chương trình OB, FC, FB.
- LD: chỉ ô nhớ có kích thước 2 từ trong miền dữ liệu địa phương (local block) của các khối chương trình OB, FC, FB.

2) *Phần số* chỉ địa chỉ của byte hoặc bit trong miền nhớ đã xác định. Nếu ô nhớ đã được xác định thông qua phần chữ là có kích thước 1 bit thì phần số sẽ gồm địa chỉ của byte và số thứ tự của bit trong byte đó được tách với nhau bằng dấu chấm. Ví dụ

```
I     1.3       // Chỉ bit thứ 3 trong byte 1 của miền nhớ bộ đệm cổng vào số PII.
M     101.5     // Chỉ bit thứ 5 trong byte 101 của miền các biến cờ M.
Q     4.5       // Chỉ bit thứ 5 trong byte thứ 4 của miền bộ đệm cổng ra số PIQ.
```

Trong trường hợp ô nhớ đã được xác định là byte, từ hoặc từ kép thì phần số sẽ là địa chỉ byte đầu tiên trong mảng byte của ô nhớ đó. Ví dụ

```
DIB   15        // Chỉ ô nhớ có kích thước 1 byte (byte 15) trong khối DB đã được mở bằng lệnh
                   OPN DI.
DBW   18        // Chỉ ô nhớ có kích thước 1 từ gồm 2 bytes 18 và 19 trong khối DB đã được mở
                   bằng lệnh OPN DB.
DB2.DBW 15      // Chỉ ô nhớ có kích thước 2 byte 15 và 16 trong khối dữ liệu DB2.
MD    105       // Chỉ ô nhớ có kích thước 2 từ gồm 4 bytes 105, 106, 107 và 108 trong miền nhớ
                   các biến cờ M.
```

2.1.3 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:

8	7	6	5	4	3	2	1	0
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 ∧, ∨ và nghịch đảo, bit FC có giá trị bằng 1. Nói cách khác, FC=0 khi dãy lệnh logic tiếp điểm vừa được kết thúc. Ví dụ

```
A     I0.3      // FC = 1
AN    I0.3      // FC = 1
=     Q4.0      // FC = 0
```

- RLO (*Result of logic operation*): Kết quả tức thời của phép tính logic vừa được thực hiện.. Ví dụ lệnh

```
A     I0.3
```

a) nếu trước khi thực hiện bit FC=0 thì có tác dụng chuyển nội dung của cổng vào số I 0.3 vào bit trạng thái RLO,

b) nếu trước khi thực hiện bit FC=1 thì có tác dụng thực hiện phép tính ∧ giữa RLO và giá trị logic cổng vào I 0.3. Kết quả của phép tính được ghi lại vào bit trạng thái RLO.

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

```
A     I0.3
AN    I0.3
```

đều gán cho bit STA cùng một giá trị là nội dung của cổng vào số I 0.3.

- OR: Ghi lại giá trị của phép tính logic ∧ cuối cùng được thực hiện để phụ giúp cho việc thực hiện phép toán ∨ sau đó. Điều này là cần thiết vì trong một biểu thức hàm hai trị, phép tính ∧ bao giờ cũng phải được thực hiện trước các phép tính ∨.
- OS (*Stored overflow bit*): Ghi lại giá trị bit bị tràn ra ngoài mảng ô nhớ.
- OV (*Overflow bit*): Bit báo 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 (sẽ giới thiệu sau). Cụ thể là:

a) Khi thực hiện lệnh toán học như cộng, trừ nhân chia với số nguyên hoặc số thực.

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).

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

CC1	CC0	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).

c) Khi thực hiện lệnh toán học với số thực nhưng kết quả bị tràn ô nhớ.

CC1	CC0	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.

d) Khi thực hiện lệnh dịch chuyển.

CC1	CC0	Ý nghĩa
0	0	Giá trị của bit bị đẩy ra bằng 0.
1	0	Giá trị của bit bị đẩy ra bằng 1.

e) Khi thực hiện lệnh logic trong ACCU.

CC1	CC0	Ý nghĩa
0	0	Kết quả bằng 0 (=0).
1	0	Kết quả khác 0 (≠0).

- 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 tạo ra được mối liên kết đó, ta cần phải kết thúc chương trình trong FB, FC bằng lệnh ghi
  a) 1 vào BR, nếu chương trình chạy không có lỗi
  b) 0 vào BR, 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
  a) 1, nếu SFC hay SFB thực hiện không có lỗi
  b) 0 , nếu có lỗi khi thực hiện SFC hay SFB.

Chú ý: Một chương trình viết trên STL (tùy thuộc vào từng người lập trình) có thể gồm nhiều Network. Mỗi một Network chứa một đoạn chương trình phục vụ một công đoạn cụ thể. Ở mỗi đầu Network, thanh ghi trạng thái nhận giá trị 0. Chỉ sau lệnh đầu tiên của Network, các bit trạng thái mới thay đổi theo kết quả phép tính.

```
Network 1
      :        } Đoạn chương trình 1
Network 2
      :        } Đoạn chương trình 2
Network 3
      :        } Đoạn chương trình 3
```

2.2 Các lệnh cơ bản

2.2.1 Nhóm lệnh logic tiếp điểm

1) 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.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	–	1

Ví dụ: Thực hiện Q4.0 = I0.3

```
Network 1
A     I0.3     // Đọc nội dung của I0.3 vào RLO
=     Q4.0     // Đưa kết quả ra cổng Q4.0
```

2) Lệnh thực hiện phép tính ∧

Cú pháp A <toán hạng>

Toán hạng là dữ 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 ∧ giữa RLO với toán hạng và ghi lại kết quả vào RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	x	x	x	1

Ví dụ: Thực hiện Q4.0 = I0.3 ∧ I0.4 (mắc nối tiếp hai công tắc)

```
Network 1
A     I0.3     // Đọc nội dung của I0.3 vào RLO
A     I0.4     // Kết hợp ∧ với nội dung cổng I0.4
=     Q4.0     // Đưa kết quả ra cổng Q4.0
```

3) Lệnh thực hiện phép tính ∧ với giá trị nghịch đảo

Cú pháp AN <toán hạng>

Toán hạng là dữ 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 ∧ giữa RLO với giá trị nghịch đảo của toán hạng và ghi lại kết quả vào RLO. Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	x	x	x	1

Ví dụ: Thực hiện Q4.0 = I0.3 ∧ $\overline{\text{I0.4}}$

```
Network 1
A     I0.3     // Đọc nội dung của I0.3 vào RLO
AN    I0.4     // Kết hợp ∧ với giá trị nghịch đảo của cổng I0.4
=     Q4.0     // Đưa kết quả ra cổng Q4.0
```

4) Lệnh thực hiện phép tính ∨

Cú pháp O <toán hạng>

Toán hạng là dữ 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 ∨ giữa RLO với toán hạng và ghi lại kết quả vào RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

Ví dụ: Thực hiện Q4.0 = I0.3 ∨ I0.4 (mắc song song hai công tắc)

```
Network 1
A     I0.3     // Đọc nội dung của I0.3 vào RLO
O     I0.4     // Kết hợp ∨ với nội dung cổng I0.4
=     Q4.0     // Đưa kết quả ra cổng Q4.0
```

5) Lệnh thực hiện phép tính ∨ với giá trị nghịch đảo

Cú pháp ON <toán hạng>

Toán hạng là dữ 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 ∨ giữa RLO với giá trị nghịch đảo của toán hạng và ghi lại kết quả vào RLO. Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

Ví dụ: Thực hiện Q4.0 = I0.3 ∨ $\overline{\text{I0.4}}$

```
Network 1
A     I0.3     // Đọc nội dung của I0.3 vào RLO
ON    I0.4     // Kết hợp ∨ với giá trị nghịch đảo của cổng I0.4
=     Q4.0     // Đưa kết quả ra cổng Q4.0
```

6) Lệnh thực hiện phép tính ∧ 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 ∧ giữa RLO với giá trị logic của biểu thức trong dấu ngoặc sau nó và ghi lại kết quả vào RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

Ví dụ: Thực hiện Q4.0 = (I0.2∨I0.3) ∧ ($\overline{I0.4}$ ∨ I0.5)

```
Network 1
A(
O     I0.2
O     I0.3
)                    // Giá trị biểu thức I0.2∨I0.3 được chuyển vào RLO.
A(
ON    I0.4
O     I0.5
)
=     Q4.0
```

7) Lệnh thực hiện phép tính ∧ với giá trị nghịch đảo của một biểu thức

Cú pháp AN(

Lệnh không có toán hạng.

Khi 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 ∧ giữa RLO với giá trị nghịch đảo logic của biểu thức trong dấu ngoặc sau nó và ghi lại kết quả vào RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

Ví dụ: Thực hiện Q4.0 = I0.2 ∧ $\overline{(\overline{I0.4} \vee I0.5)}$

```
Network 1
A     I0.2
AN(
ON    I0.4
O     I0.5
)
=     Q4.0
```

8) Lệnh thực hiện phép tính ∨ 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 ∨ giữa RLO với giá trị logic của biểu thức trong dấu ngoặc sau nó và ghi lại kết quả vào RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

Ví dụ: Thực hiện Q4.0 = I0.2 ∨ ($\overline{I0.4}$ ∧ I0.5)

```
A     I0.2
O(
AN    I0.4
A     I0.5
)
=     Q4.0
```

9) Lệnh thực hiện phép tính ∨ với giá trị nghịch đảo của một biểu thức

Cú pháp ON(

Lệnh không có toán hạng.

Khi FC=0 lệnh sẽ gán giá trị logic nghịch đảo của biểu thức trong dấu ngoặc sau nó vào RLO. Ngược lại nếu FC=1 lệnh thực hiện phép tính ∨ giữa RLO với giá trị nghịch đảo của biểu thức trong dấu ngoặc sau nó và ghi lại kết quả vào RLO. Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

10) Lệnh thực hiện phép tính exclusive or

Cú pháp X <toán hạng>

Toán hạng là dữ liệu kiểu BOOL hoặc địa chỉ bit I, Q, M, L, D, T, C.

Nếu FC=0, lệnh ghi giá trị logic của toán hạng vào RLO. Nếu 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 ghi 1 vào RLO, ngược lại thì ghi 0. Nói cách khác, lệnh sẽ đảo nội dung của RLO nếu toán hạng có giá trị 1.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

Ví dụ: Nếu $\overline{I0.4}$ ∧ I0.5 ≠ I0.2 thì Q4.0=1

```
Network 1
AN    I0.4
A     I0.5
X     I0.2      // Nghịch đảo giá trị RLO nếu I0.2=1.
=     Q4.0
```

11) Lệnh thực hiện phép tính exclusive or not

Cú pháp XN <toán hạng>

Toán hạng là dữ liệu kiểu BOOL hoặc địa chỉ bit I, Q, M, L, D, T, C.

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

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu - chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

Ví dụ: Nếu $\overline{I0.4} \wedge I0.5 = I0.2$ thì Q4.0=1

```
AN    I0.4
A     I0.5
XN    I0.2      // Nghịch đảo giá trị RLO nếu I0.2=0.
=     Q4.0
```

12) Lệnh thực hiện phép tính exclusive or với giá trị của một biểu thức

Cú pháp X(

Lệnh không có toán hạng.

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

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

13) Lệnh thực hiện phép tính exclusive or not với giá trị của một biểu thức

Cú pháp XN(

Lệnh không có toán hạng.

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

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

14) 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 tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	–	1	1	0

15) 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 tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	0	0	0

16) Lệnh đảo giá trị của RLO

Cú pháp NOT

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

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	–	1	x	–

17) 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 cho trong toán hạng.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	–	0

18) 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 cho trong toán hạng.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	–	0

19) 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

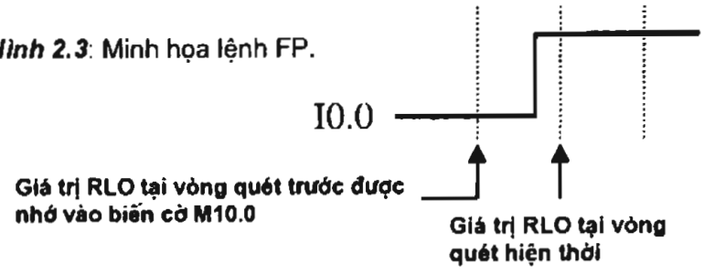
Tại mỗi vòng quét lệnh sẽ kiểm tra: nếu biến cờ (toán hạng) có giá trị 0 và RLO có giá trị 1 thì sẽ ghi 1 vào RLO, các trường hợp khác thì ghi 0, đồng thời chuyển nội dung của RLO vào lại biến cờ. Như vậy RLO sẽ có giá trị 1 trong một vòng quét khi có sườn lên trong RLO. Ví dụ: Lệnh phát hiện sườn lên

```
A     I0.0
FP    M10.0
=     Q4.5
```

sẽ tương đương với đoạn chương trình sau

```
A     I0.0
AN    M10.0
=     Q4.5
A     I0.0
=     M10.0
```

Hình 2.3: Minh họa lệnh FP.



Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

20) 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 một vòng quét khi có sườn xuống trong RLO.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	x	x	1

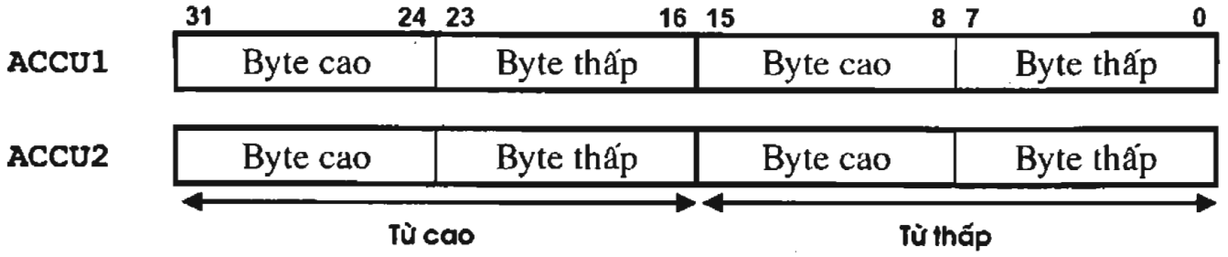
21) Lệnh chuyển giá trị của RLO vào BR

Cú pháp SAVE

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

2.2.2 Lệnh đọc, ghi và đảo vị trí bytes trong thanh ghi ACCU

Các CPU của S7-300 thường có hai thanh ghi Accumulator (ACCU) ký 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 bits ... đều được thực hiện trên hai thanh ghi này. Chúng có cấu trúc như sau:



1) 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 ÷ 65535.
- từ IW, QW, PIW, MW, LW, DBW, DIW trong khoảng 0 ÷ 65534.
- từ kép ID, QD, PID, MD, LD, DBD, DID trong khoảng 0 ÷ 65534.

Nếu là dữ liệu thì các dạng dữ liệu hợp lệ của toán hạng cho trong bảng sau:

Dữ liệu	Ví dụ	Giải thích
± ...	`L +5`	Ghi 5 vào từ thấp của ACCU1 (số nguyên 16 bits)
`B#(...,...)`	`L B#(1,8)`	Ghi 1 vào byte cao của từ thấp và 8 vào byte thấp của từ thấp trong ACCU1.
`L#...`	`L L#5`	Ghi 5 vào ACCU1 (số nguyên 32 bits)

`16#...`	`L B#16#2E`
	`L W#A2EB`
	`L DW#2C1E_A2EB`
		Dữ liệu dạng cơ số 16

`2#...`	`L 2#11001101`	Dữ liệu dạng cơ số 2
`'...'`	`L 'AB'` `L 'ABCD'`	Dữ liệu dạng ký tự
`C#...`	`L C#1000`	Dữ liệu là giá trị đặt trước cho bộ đếm (PV)
`S5TIME#...`	`L S5TIME#2S`	Dữ liệu là giá trị đặt trước cho Timer (PV)
`P#...`	`L P#M10.2`	Dữ liệu là địa chỉ ô nhớ (dùng cho con trỏ)
`D#...`	`L D#2000-6-20`	Dữ liệu là giá trị về ngày/tháng/năm (16 bits)
`T#...`	`L T#0H_1M_10S`	Dữ liệu về thời gian giờ/phút/giây (32 bits)

Lệnh `L` có tác dụng chuyển dữ liệu hoặc 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 có kích thước nhỏ hơn từ kép thì chúng sẽ được ghi

vào theo thứ tự byte thấp của từ thấp, byte cao của từ thấp, byte thấp của từ cao, byte cao của từ cao. Những bits còn trống trong ACCU1 được ghi 0. Ví dụ lệnh

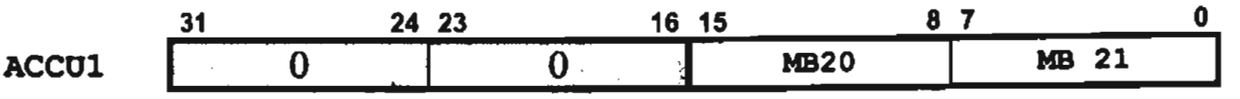
```
L     IB0
```

sẽ chuyển nội dung của IB0 vào ACCU1 như sau

và lệnh

```
L     MW20
```

sẽ chuyển nội dung của MW20 gồm 2 bytes MB20, MB21 vào ACCU1 theo thứ tự



Lệnh không sửa đổi thanh ghi trạng thái (*Status word*).

2) Lệnh chuyển nội dung của ACCU tới ô nhớ.

Cú pháp `T <toán hạng>`

Toán hạng là địa chỉ:

- byte IB, QB, PQB, MB, LB, DBB, DIB trong khoảng 0 ÷ 65535.
- từ IW, QW, PQW, MW, LW, DBW, DIW trong khoảng 0 ÷ 65534.
- từ kép ID, QD, PQD, MD, LD, DBD, DID trong khoảng 0 ÷ 65534.

Lệnh chuyển nội dung của ACCU1 vào ô nhớ có địa chỉ là toán hạng. Lệnh không thay đổi nội dung của ACCU2. Trong trường hợp ô nhớ có kích thước nhỏ hơn từ kép thì nội dung của ACCU1 được chuyển ra theo thứ tự byte thấp của từ thấp, byte cao của từ thấp, byte thấp của từ cao, byte cao của từ cao. Ví dụ lệnh

```
T     QB0
```

sẽ chỉ chuyển nội dung của byte thấp của từ thấp trong ACCU1 vào IB0 và lệnh

```
T     MW20
```

sẽ chỉ chuyển byte cao của từ thấp vào MW20, byte thấp của từ thấp vào MW21. Lệnh không sửa đổi thanh ghi trạng thái (*Status word*).

3) Lệnh đọc nội dung của thanh ghi trạng thái vào ACCU1.

Cú pháp `L STW`

Lệnh chuyển nội dung của thanh ghi trạng thái vào từ thấp của ACCU1. Lệnh không sửa đổi thanh ghi trạng thái (*Status word*).

4) 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.

5) Lệnh chuyển nội dung của ACCU2 vào ACCU1.

Cú pháp **POP**

Lệnh không có toán hạng.

Lệnh chuyển nội dung của ACCU2 vào ACCU1. Nội dung của ACCU2 và thanh ghi trạng thái không bị thay đổi.

6) Lệnh chuyển nội dung của ACCU1 vào ACCU2.

Cú pháp **PUSH**

Lệnh không có toán hạng. Lệnh ghi nội dung của ACCU1 vào ACCU2. Nội dung của ACCU1 không bị thay đổi. Lệnh cũng không thay đổi nội dung của thanh ghi trạng thái (*Status word*).

7) Lệnh đảo nội dung của hai thanh ghi ACCU1 vào ACCU2..

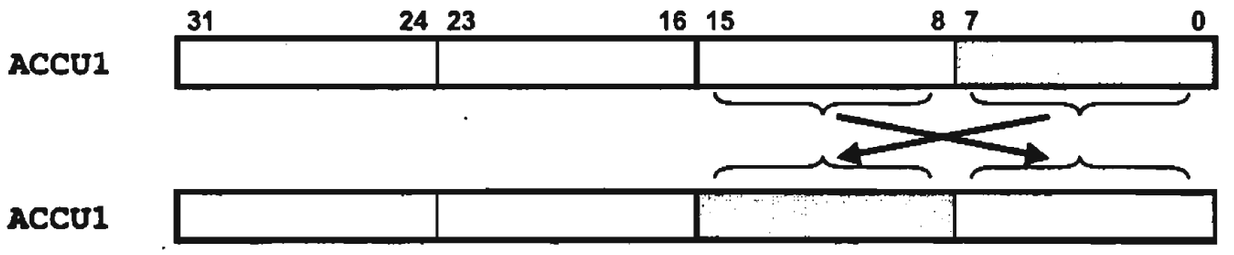
Cú pháp **TAK**

Lệnh không có toán hạng. Nội dung của ACCU1 được ghi vào ACCU2 và ngược lại nội dung của ACCU2 được ghi vào ACCU1. Lệnh này không làm thay đổi nội dung của thanh ghi trạng thái (*Status word*).

8) Lệnh đảo nội dung hai byte của từ thấp trong ACCU1

Cú pháp **CAW**

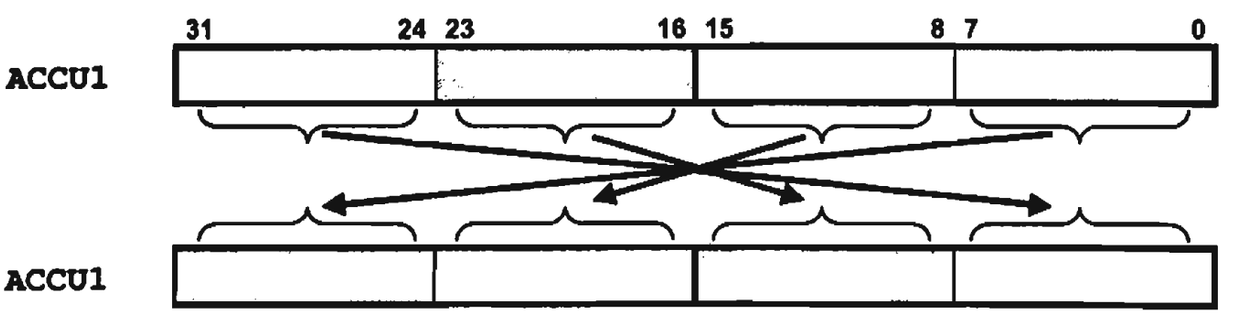
Lệnh không có toán hạng và có tác dụng đảo nội dung hai byte của từ thấp trong thanh ghi ACCU1. Lệnh không sửa đổi thanh ghi trạng thái (*Status word*).



9) Lệnh đảo nội dung các byte trong ACCU1

Cú pháp **CAD**

Lệnh không có toán hạng và có tác dụng đảo nội dung tất cả bốn byte trong thanh ghi ACCU1.



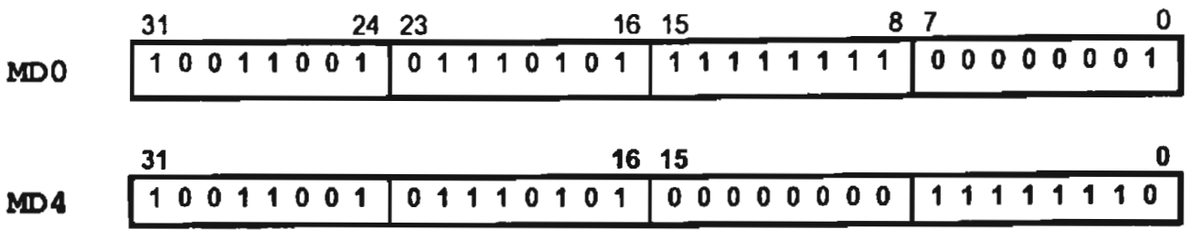
Lệnh không sửa đổi thanh ghi trạng thái (*Status word*).

10) Lệnh đảo giá trị các bits trong từ thấp của ACCU1

Cú pháp INVI

Lệnh không có toán hạng và có tác dụng đảo nội dung tất các bits trong từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái. Ví dụ

```
L     MD0
INVI
T     MD4
```

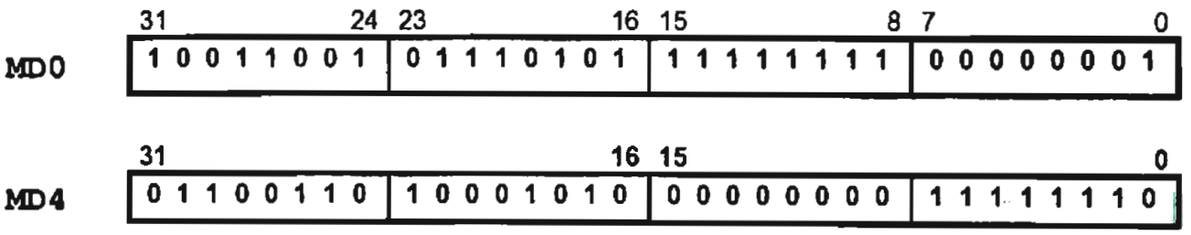


11) Lệnh đảo giá trị các bits của ACCU1

Cú pháp INVD

Lệnh không có toán hạng và có tác dụng đảo nội dung tất các bits của ACCU1. Nội dung của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái. Ví dụ

```
L     MD0
INVD
T     MD4
```



2.2.3 Các lệnh logic thực hiện trên thanh ghi ACCU

Tất cả các lệnh logic thực hiện trên thanh ghi ACCU1 được trình bày sau đây đều tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	0	0	–	–	–	–	–

trong đó ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh.

1) Lệnh thực hiện phép tính ∧ giữa các bits trong từ thấp của ACCU1, ACCU2.

Cú pháp AW [<dữ liệu hằng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu không có toán hạng, lệnh thực hiện phép tính ∧ giữa các bits thuộc từ thấp của hai thanh ghi ACCU1, ACCU2. Kết quả được ghi lại vào từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

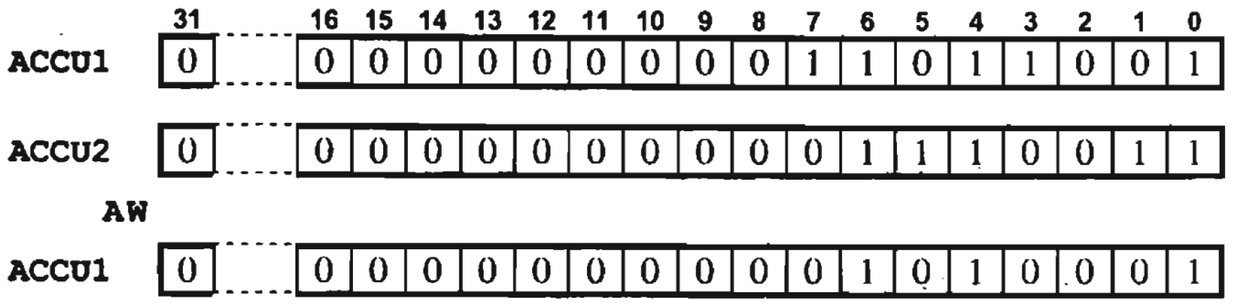
– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 16 bits. Khi đó lệnh thực hiện phép tính ∧ giữa dữ liệu với từ thấp của ACCU1. Kết quả được ghi lại vào từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

Ví dụ về dạng có toán hạng

```
L     2#11001
AD    DW#16#FF13   // Sau lệnh này ACCU1 sẽ có nội dung là 10001.
```

Ví dụ về dạng không có toán hạng:

```
L     2#11011001
L     2#1110011
AW                 // Sau lệnh này ACCU1 sẽ có nội dung là 1010001.
```



2) Lệnh thực hiện phép tính ∧ giữa các bits của hai thanh ghi ACCU1, ACCU2.

Cú pháp AD [<dữ liệu hằng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu không có toán hạng, lệnh thực hiện phép tính ∧ giữa các bits của hai thanh ghi ACCU1, ACCU2. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 32 bits. Khi đó lệnh thực hiện phép tính ∧ giữa dữ liệu với thanh ghi ACCU1. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

3) Lệnh thực hiện phép tính ∨ giữa các bits trong từ thấp của ACCU1 và ACCU2.

Cú pháp OW [<dữ liệu hằng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu không có toán hạng, lệnh thực hiện phép tính ∨ giữa các bits thuộc từ thấp của hai thanh ghi ACCU1, ACCU2. Kết quả được ghi lại vào từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 16 bits. Khi đó lệnh thực hiện phép tính v giữa toán hạng với từ thấp của ACCU1. Kết quả được ghi lại vào từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

Ví dụ:

```
L     2#11001
L     2#10011
OW                // Sau lệnh này ACCU1 sẽ có nội dung là 11011.
```

4) Lệnh thực hiện phép tính v giữa các bits của hai thanh ghi ACCU1, ACCU2.

Cú pháp OD [<dữ liệu hằng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu không có toán hạng, lệnh thực hiện phép tính v giữa tất cả 32 bits của hai thanh ghi ACCU1, ACCU2. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 32 bits. Khi đó lệnh thực hiện phép tính v giữa 32 bits của thanh ghi ACCU1 với toán hạng. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

5) Lệnh thực hiện phép tính exclusive or 16 bits

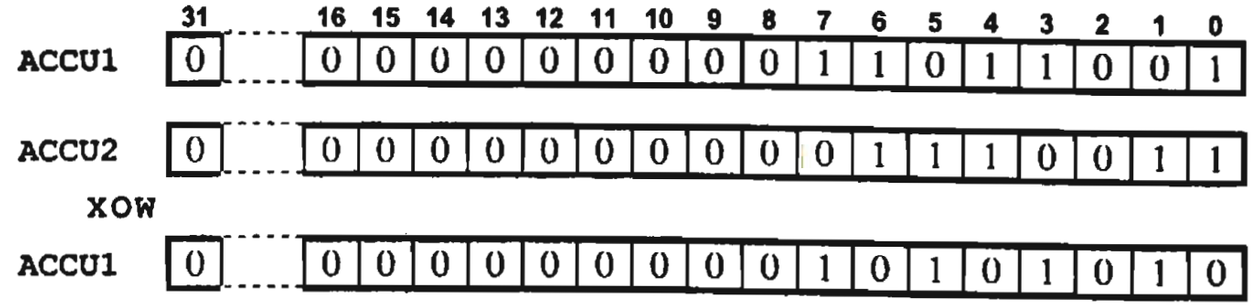
Cú pháp XOW [<dữ liệu hằng>]

Lệnh có thể có 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, ACCU2, tức là nếu hai bit không cùng giá trị thì kết quả sẽ là 1. Toàn bộ 16 bit kết quả được ghi lại vào từ thấp trong ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 32 bits. Khi đó lệnh thực hiện phép tính exclusive or giữa các bits của từ thấp trong thanh ghi ACCU1 và dữ liệu cho trong toán hạng, tức là nếu hai bit không cùng giá trị thì bit kết quả sẽ là 1. Toàn bộ 16 bit kết quả được ghi lại vào từ thấp trong ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

Ví dụ:



6) Lệnh thực hiện phép tính exclusive or 32 bits

Cú pháp XOD [<dữ liệu hằng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu không có toán hạng, lệnh thực hiện phép tính exclusive or giữa các bits của hai thanh ghi ACCU1, ACCU2, tức là nếu hai bit không cùng giá trị thì bit kết quả sẽ có giá trị là 1. Toàn bộ 32 bit kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

– Nếu có toán hạng thì toán hạng phải là một dữ liệu hằng có kích thước 32 bits. Khi đó lệnh thực hiện phép tính exclusive or giữa các bits ACCU1 và toán hạng, tức là nếu hai bit không cùng giá trị thì bit kết quả sẽ có giá trị là 1. Toàn bộ 32 bits kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

2.2.4 Nhóm lệnh tăng giảm nội dung thanh ghi ACCU

1) 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 của từ thấp trong ACCU1 với toán hạng. Kết quả được ghi lại vào byte thấp của từ thấp của ACCU1. Nội dung byte cao của từ thấp, của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

Lệnh không sửa đổi nội dung thanh ghi trạng thái (*Status word*).

2) Lệnh giảm nội dung thanh ghi ACCU1

Cú pháp DEC <toán hạng>

Toán hạng là số nguyên 8 bits.

Lệnh thực hiện phép trừ byte thấp của từ thấp trong ACCU1 cho toán hạng. Kết quả được ghi lại vào byte thấp của từ thấp của ACCU1. Nội dung byte cao của từ thấp, của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.

Lệnh không sửa đổi nội dung thanh ghi trạng thái (*Status word*).

2.2.5 Nhóm lệnh dịch chuyển nội dung thanh ghi ACCU

1) Lệnh xoay tròn các bits của ACCU1 theo chiều trái.

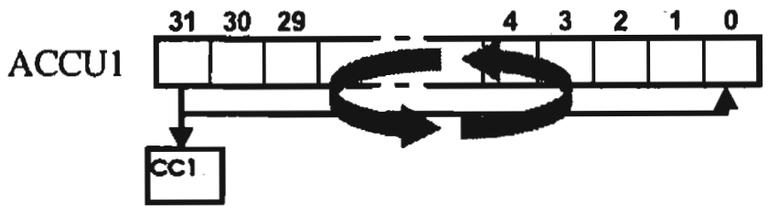
Cú pháp RLD [<toán hạng>]

Lệnh có thể có hoặc không có toán hạng.

– Nếu có toán hạng thì toán hạng là số nguyên không dấu trong khoảng 0÷32. Khi đó lệnh thực hiện phép tính xoay tròn các bits của ACCU1 theo chiều trái. Số bits được xoay được chỉ thị trong toán hạng. Tại mỗi lần xoay, bit thứ 31 (bit cuối) bị đẩy ra khỏi ACCU1 sẽ 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. Nếu toán hạng bằng 32, nội dung của ACCU1 không bị thay đổi và bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1. Hai bits CC0 và OV 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 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 0 (bit đầu tiên). Nếu byte thấp của từ thấp trong ACCU2 bằng 0, lệnh sẽ không làm gì, nếu bằng 32, nội dung của ACCU1 không bị thay đổi và bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1. Hai bits CC0 và OV 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 tác động vào thanh ghi trạng thái (*Status word*) như sau (ký hiệu – chỉ nội dung bit không bị thay đổi, x là bị thay đổi theo lệnh):

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	x	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
RLD   4
```

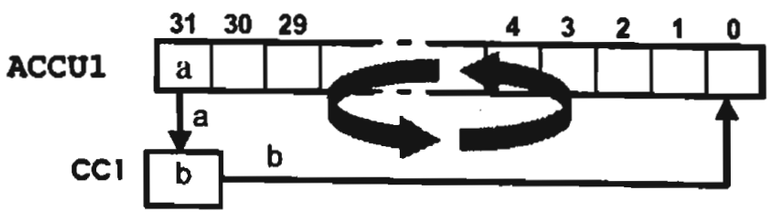
Sau lệnh trên, ACCU1 sẽ là 0101 0110 0011 1100 1110 1111 0000 1001.

2) Lệnh xoay tròn ACCU1 theo chiều trái 1 bit.

Cú pháp RLDA

Lệnh không có toán hạng.

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



Ví dụ: Nếu trước khi thực hiện lệnh có

```
ACCU1        1001 0101 0110 0011 1100 1110 1111 0000
CC1          x
```

thì sau khi lệnh RLDA được thực hiện sẽ là

```
ACCU1        0010 1010 1100 0111 1001 1101 1110 000x
CC1          1
```

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	0	0	–	–	–	–	–

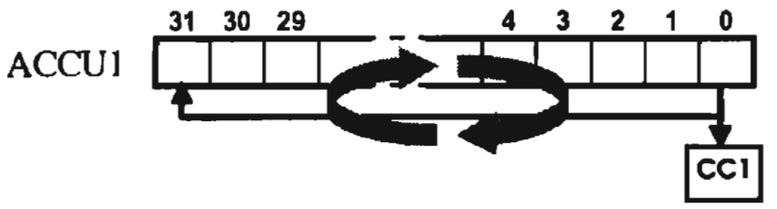
3) Lệnh xoay tròn các bits của ACCU1 theo chiều phải.

Cú pháp `RRD [<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÷32. Khi đó lệnh thực hiện phép tính xoay tròn các bits của ACCU1 theo chiều phải. Số bits được xoay được chỉ thị trong toán hạng. Tại mỗi lần xoay, bit thứ 0 (bit đầu) bị đẩy ra khỏi ACCU1 sẽ được ghi đồng thời vào CC1 và vào bit thứ 31 (bit cuối). Nếu toán hạng là một số 0, lệnh sẽ không làm gì. Hai bits CC0 và OV 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 phải. Số bits được xoay được chỉ thị trong byte thấp của từ thấp trong ACCU2. Tại mỗi lần xoay, bit thứ 0 (bit đầu tiên) bị đẩy ra khỏi ACCU1 sẽ được ghi đồng thời vào CC1 và vào bit thứ 31 (bit cuối). Nếu byte thấp của từ thấp trong ACCU2 lớn hơn 0, hai bits CC0 và OV trong thanh ghi trạng thái sẽ được ghi 0.



Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	x	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
RRD   4
```

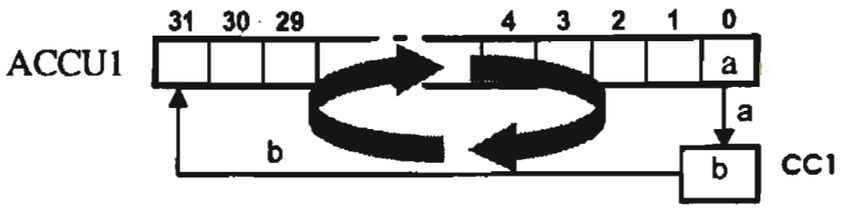
Sau lệnh trên, **ACCU1** sẽ là `0000 1001 0101 0110 0011 1100 1110 1111`.

4) Lệnh xoay tròn ACCU1 theo chiều phải 1 bit.

Cú pháp RRDA

Lệnh không có toán hạng.

Lệnh thực hiện phép tính xoay ACCU1 theo chiều phải 1 bit. Bit thứ 0 (bit đầu tiên) bị đẩy ra khỏi ACCU1 được ghi vào CC1. Nội dung của CC1 được chuyển vào bit cuối (bit thứ 31).



Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	0	0	–	–	–	–	–

Ví dụ: Trước khi thực hiện lệnh

```
ACCU1       1001 0101 0110 0011 1100 1110 1111 0000
CC1         x
```

và sau khi lệnh RRDA được thực hiện

```
ACCU1       x100 1010 1011 0001 1110 0111 0111 1000
CC1         0
```

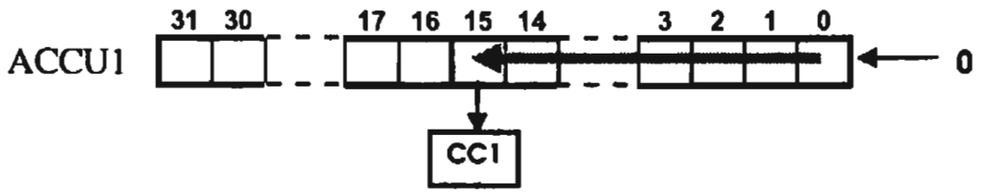
5) Lệnh dịch trái các bits của từ thấp của ACCU1.

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 phép tính dịch trái các 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 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 toán hạng là một số 0, lệnh sẽ không làm gì. Nếu toán hạng bằng 16, nội dung của ACCU1 không bị thay đổi và bit CC1 trong thanh ghi trạng thái có giá trị là bit thứ 0 của ACCU1. Hai bits CC0 và OV sẽ bằng 0 khi toán hạng là một số lớn hơn 0.

– Nếu không có toán hạng, lệnh thực hiện phép tính dịch trái các các bits trong từ thấp của ACCU1 với số bits 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 thực Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SLW   4
```

Sau lệnh trên, **ACCU1** sẽ là `1001 0101 0110 0011 1110 1111 0000 0000`.

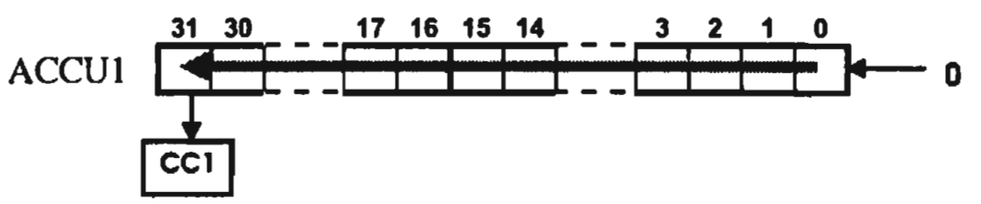
6) Lệnh dịch trái các bits của ACCU1.

Cú pháp SLD [<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÷32. Khi đó lệnh thực hiện phép tính dịch trái các các bits của ACCU1. Số bits được dịch được chỉ thị trong toán hạng. Tại mỗi lần dịch, bit thứ 31 (bit cuối) 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 lớn hơn 0, hai bits CC0 và OV sẽ bằng 0.

– Nếu không có toán hạng, lệnh thực hiện phép tính dịch trái các các bits của ACCU1 với số bits được dịch là nội dung của byte thấp trong từ thấp của ACCU2. Tại mỗi lần dịch, bit thứ 31 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 0, lệnh sẽ không làm gì. Nếu byte thấp của từ thấp trong ACCU2 là một số lớn hơn 0, hai bits CC0 và OV sẽ bằng 0.



Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SLW   4
```

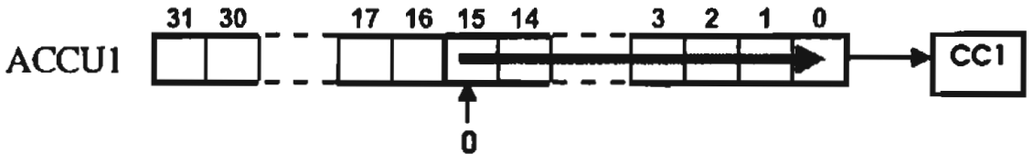
Sau lệnh trên, **ACCU1** sẽ là `0101 0110 0011 1100`

7) Lệnh dịch phải các bits của từ thấp của ACCU1.

Cú pháp SRW [<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 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 mỗi lần dịch, bit thứ 0 (bit đầu) bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1 còn bit thứ 15 được ghi 0. Nếu toán hạng là một 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 các bits trong từ thấp của ACCU1. Số bits 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 0. Nếu byte thấp của từ thấp trong ACCU2 bằng 0, lệnh sẽ không làm gì. Nếu byte thấp của từ thấp trong ACCU2 là một số lớn hơn 0, hai bits CC0 và OV sẽ bằng 0.



Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SRW   4
```

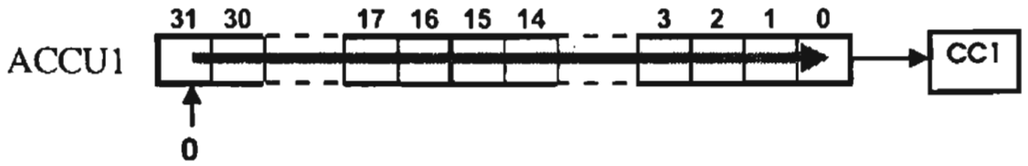
Sau lệnh trên, **ACCU1** sẽ là **1001 0101 0110 0011 0000 1100 1110 1111**.

8) Lệnh dịch trái các bits của ACCU1.

Cú pháp SRD [<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÷32. Khi đó lệnh thực hiện phép tính dịch phải các các bits của ACCU1. Số bits được dịch là toán hạng. Tại mỗi lần dịch, bit thứ 0 bị đẩy từ ACCU1 vào CC1 còn bit cuối (bit 31) được ghi 0. Nếu troán hạng là một 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 các bits của ACCU1. Số bits được dịch là nội dung của byte thấp trong từ thấp của ACCU2. Tại mỗi lần dịch, bit thứ 0 bị đẩy từ ACCU1 sang CC1 còn bit thứ 31 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 tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SRW   4
```

Sau lệnh trên, **ACCU1** sẽ là `0000 0101 0110 0011 1100 1110 1111`.

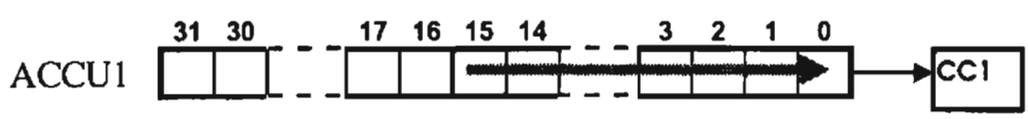
9) Lệnh dịch phải số nguyên 16 bits trong ACCU1.

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 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 CC1 còn 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 các bits trong từ thấp của ACCU1. Số bits 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 tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	–	–	–	–	–

Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SSI   4
```

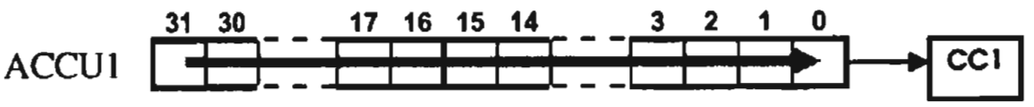
Sau lệnh trên, **ACCU1** sẽ là `1001 0101 0110 0011  1111 1100 1110 1111`.

10) Lệnh dịch phải số nguyên 32 bits trong ACCU1 có toán hạng.

Cú pháp SSD [<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÷32. Khi đó lệnh thực hiện phép tính dịch phải các các bits trong ACCU1. Số bits được dịch được chỉ thị trong toán hạng. Tại mỗi lần dịch, bit thứ 0 (bit đầu) bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1 còn bit thứ 32 được ghi lại đúng bằng giá trị cũ của nó. Nếu toán hạng là một 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 các bits trong ACCU1. Số bits được dịch là nội dung của byte thấp trong từ thấp của ACCU2. Tại mỗi lần dịch, bit thứ 0 bị đẩy ra khỏi ACCU1 sẽ được ghi vào CC1, bit thứ 32 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.



Ví dụ:

```
L     2#1001 0101 0110 0011 1100 1110 1111 0000
SSD   4
```

Sau lệnh trên, **ACCU1** sẽ là `1111 1001 0101 0110 0011 1100 1110 1111`.

Lệnh tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	x	–	–	–	–	–

2.2.6 Nhóm lệnh so sánh số nguyên 16 bits

Tất cả những lệnh so sánh hai số nguyên 16 bits nằm trong hai từ thấp của hai thanh ghi ACCU1 và ACCU2 được trình bày sau đây đều tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	0	–	0	x	x	1

trong đó hai bits trạng thái CC1 và CC0 được thay đổi theo quy tắc:

CC1	CC0	Ý nghĩa
0	0	Từ thấp ACCU2 = Từ thấp ACCU1.
0	1	Từ thấp ACCU2 < Từ thấp ACCU1.
1	0	Từ thấp ACCU2 > Từ thấp ACCU1.

1) Lệnh so sánh bằng nhau hai 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.

2) Lệnh so sánh không bằng nhau hai 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 với số nguyên trong từ thấp của ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1, ngược lại sẽ có giá trị 0.

3) Lệnh so sánh lớn hơn hai số nguyên 16 bits

Cú pháp >I

Lệnh không có toán hạng và 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 chứa 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.

4) Lệnh so sánh nhỏ hơn hai số nguyên 16 bits

Cú pháp <I

Lệnh không có toán hạng và 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 chứa 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.

5) Lệnh so sánh lớn hơn hoặc bằng hai số nguyên 16 bits

Cú pháp >=I

Lệnh không có toán hạng và 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 chứa 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.

6) Lệnh so sánh nhỏ hơn hoặc bằng hai số nguyên 16 bits

Cú pháp **<=I**

Lệnh không có toán hạng và thực hiện phép so sánh hai số nguyên 16 bits trong hai từ thấp của ACCU1 và ACCU2. Nếu số nguyên trong từ thấp của ACCU2 nhỏ hơn hoặc bằng số nguyên trong từ thấp ACCU1 thì bit RLO sẽ nhận giá trị 1.

2.2.7 Nhóm lệnh so sánh số nguyên 32 bits

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

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	0	–	0	x	x	1

trong đó hai bits trạng thái CC1 và CC0 được thay đổi theo quy tắc:

CC1	CC0	Ý nghĩa
0	0	ACCU2 = ACCU1.
0	1	ACCU2 < ACCU1.
1	0	ACCU2 > ACCU1.

1) Lệnh so sánh bằng nhau hai số nguyên 32 bits

Cú pháp **==D**

Lệnh so sánh hai số nguyên 32 bits trong hai thanh ghi ACCU1 và ACCU2. Nếu chúng bằng nhau thì RLO=1, ngược lại sẽ có giá trị 0.

2) Lệnh so sánh không bằng nhau hai số nguyên 32 bits

Cú pháp **<>D**

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

3) Lệnh so sánh lớn hơn hai số nguyên 32 bits

Cú pháp **>D**

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

4) Lệnh so sánh nhỏ hơn hai số nguyên 32 bits

Cú pháp **<D**

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

5) Lệnh so sánh lớn hơn hoặc bằng hai số nguyên 32 bits

Cú pháp >=D

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

6) Lệnh so sánh nhỏ hơn hoặc bằng hai số nguyên 32 bits

Cú pháp <=D

Lệnh không có toán hạng và thực hiện phép so sánh hai số nguyên 32 bits trong hai thanh ghi ACCU1 và ACCU2. Nếu số nguyên trong ACCU2 nhỏ hơn hoặc bằng số nguyên trong ACCU1 thì RLO sẽ nhận giá trị 1, ngược lại sẽ có giá trị 0.

2.2.8 Nhóm lệnh so sánh số thực 32 bits

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

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	x	x	x	x	0	x	x	1

trong đó hai bits trạng thái CC1 và CC0 được thay đổi theo quy tắc:

CC1	CC0	Ý nghĩa
0	0	ACCU2 = ACCU1.
0	1	ACCU2 < ACCU1.
1	0	ACCU2 > ACCU1.

1) Lệnh so sánh bằng nhau hai số thực 32 bits

Cú pháp ==R

Lệnh thực hiện phép so sánh hai số thực 32 bits trong hai thanh ghi ACCU1 và ACCU2. Nếu số thực trong ACCU1 bằng số thực trong ACCU2 thì bit trạng thái RLO sẽ nhận giá trị 1, ngược lại sẽ có giá trị 0. Ví dụ đoạn chương trình sau báo sáng đèn Q4.0 nếu số thực trong ô nhớ MD10 bằng số π

```
L     MD10
L     3.1416
==R
=     Q4.0
```

2) Lệnh so sánh không bằng nhau hai số thực 32 bits

Cú pháp <>R

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

Ví dụ nếu số thực trong ô nhớ MD10 không bằng số 0.5 thì báo sáng đèn Q4.1

```
L     MD10
L     0.5
<>R
=     Q4.0
```

3) Lệnh so sánh lớn hơn hai số thực 32 bits

Cú pháp >R

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

4) Lệnh so sánh nhỏ hơn hai số thực 32 bits

Cú pháp <R

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

5) Lệnh so sánh lớn hơn hoặc bằng hai số thực 32 bits

Cú pháp >=R

Lệnh không có toán hạng.

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

6) Lệnh so sánh nhỏ hơn hoặc bằng hai số thực 32 bits

Cú pháp <=R

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

Ví dụ nếu số thực trong ô nhớ MD10 không lớn hơn 0.5 thì báo sáng đèn Q4.1

```
L     MD10
L     0.5
<=R
=     Q4.0
```

2.3 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:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	X	X	X	X	–	–	–	–

trong đó hai bits trạng thái CC1 và CC0 được thay đổi theo quy tắc:

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).

2.3.1 Nhóm lệnh làm việc với số nguyên 16 bits

1) 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 kết quả nằm trong khoảng –32768 ÷ 32767, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Cộng hai số nguyên 16 bits chứa trong IW10, MW14 và cất kết quả vào ô nhớ DBW25 của khối dữ liệu DB1.

```
L     IW10
L     MW14
+I
T     DB1.DBW25
```

2) Lệnh trừ

Cú pháp -I

Lệnh thực hiện phép trừ số nguyên 16 bits trong từ thấp của ACCU2 cho số nguyên trong từ thấp của ACCU1. Kết quả được ghi lại vào từ thấp của ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng –32768 ÷ 32767, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép trừ IW10– MW14 và cất kết quả vào ô nhớ MW25

```
L     IW10
L     MW14
-I
T     MW25
```

3) Lệnh nhân

Cú pháp *I

Lệnh thực hiện phép nhân hai số nguyên 16 bits trong từ thấp của ACCU1,ACCU2. Kết quả là một số nguyên 32 bits sẽ được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng –32768 ÷ 32767, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép nhân IW10×MW14 và cất kết quả vào ô nhớ MW20

```
L     IW10
L     MW14
*I
T     MW25
```

4) Lệnh chia

Cú pháp /I

Lệnh thực hiện phép chia số nguyên 16 bits trong từ thấp của ACCU2 cho số nguyên 16 bits trong từ thấp của ACCU1. Kết quả là một số nguyên 16 bits sẽ được ghi lại vào từ thấp của ACCU1. Phần dư của phép chia được cất vào từ cao trong ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi.

Nếu kết quả nằm trong khoảng −32768 ÷ 32767, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

2.3.2 Nhóm lệnh làm việc với số nguyên 32 bits

1) Lệnh cộng

Cú pháp +D

Lệnh không có toán hạng. Lệnh thực hiện phép cộng hai số nguyên 32 bits nằm trong ACCU1 và ACCU2. Kết quả được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi.

Nếu kết quả nằm trong khoảng −2147483648 ÷ 2147483647, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

2) Lệnh trừ

Cú pháp -D

Lệnh thực hiện phép trừ số nguyên 32 bits trong ACCU2 cho số nguyên 32 bits trong ACCU1. Kết quả được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng −2147483648 ÷ 2147483647, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

3) Lệnh nhân

Cú pháp *D

Lệnh thực hiện phép nhân hai số nguyên 32 bits trong ACCU1,ACCU2. Kết quả là một số nguyên 32 bits sẽ được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng −2147483648 ÷ 2147483647, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép nhân MD10×MD14 và cất kết quả vào ô nhớ MD20

```
L     MD10
L     MD14
```

```
*D
T     MD20
```

4) Lệnh chia

Cú pháp `/D`

Lệnh thực hiện phép chia số nguyên 32 bits trong ACCU2 cho số nguyên 32 bits trong ACCU1. Kết quả là một số nguyên 32 bits sẽ được ghi lại vào ACCU1. Nếu kết quả nằm trong khoảng −2147483648 ÷ 2147483647, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép chia MD10: MD14 và cất kết quả vào ô nhớ MD20

```
L     MD10
L     MD14
/D
T     MD20
```

5) Lệnh lấy phần dư

Cú pháp `MOD`

Lệnh không có toán hạng và xác định phần dư của phép chia số nguyên 32 bits trong ACCU2 cho số nguyên 32 bits trong ACCU1. Kết quả là số nguyên 32 bits được ghi lại vào ACCU1. Nếu kết quả trong khoảng −2147483648 ÷ 2147483647, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

2.3.3 Nhóm lệnh làm việc với số thực

1) Lệnh cộng

Cú pháp `+R`

Lệnh không có toán hạng. Lệnh thực hiện phép cộng hai số thực dấu phảy động nằm trong ACCU1 và ACCU2. Kết quả được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi.

Nếu kết quả nằm trong khoảng −3,402823E+38 ÷ −1,175495E−38 hoặc bằng 0 hoặc nằm trong khoảng +1,175495E−38 ÷ 3,402823+38 bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

2) Lệnh trừ

Cú pháp `-R`

Lệnh thực hiện phép trừ số thực trong ACCU2 cho số thực 32 trong ACCU1. Kết quả được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả trong khoảng −3,402823E+38 ÷ −1,175495E−38 hoặc +1,175495E−38 ÷ 3,402823+38 hoặc bằng 0, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép trừ MD10– MD14 và cất kết quả vào ô nhớ MD20

```
L     MD10
L     MD14
-R
T     MD20
```

3) Lệnh nhân

Cú pháp *R

Lệnh thực hiện phép nhân hai số thực trong ACCU1,ACCU2. Kết quả được ghi lại vào ACCU1. Nội dung của thanh ghi ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng –3,402823E+38 ÷ –1,175495E–38 hoặc bằng 0 hoặc nằm trong khoảng +1,175495E–38 ÷ 3,402823+38, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép nhân MD10×π và cất kết quả vào ô nhớ MD20

```
L     MD10
L     3.1416
*R
T     MD20
```

4) Lệnh chia

Cú pháp /R

Lệnh thực hiện phép chia số thực trong ACCU2 cho số thực trong ACCU1. Kết quả được ghi lại vào ACCU1. Nếu kết quả nằm trong khoảng –3,402823E+38 ÷ –1,175495E–38 hoặc bằng 0 hoặc nằm trong khoảng +1,175495E–38 ÷ 3,402823+38, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Thực hiện phép chia MD10/π và cất kết quả vào ô nhớ MD20

```
L     MD10
L     3.1416
/R
T     MD20
```

5) 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 bits trạng thái.

6) Lệnh tính sin

Cú pháp SIN

Lệnh tính sin của số thực trong ACCU1. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng

−3,402823E+38 ÷ −1,175495E−38 hoặc bằng 0 hoặc nằm trong khoảng +1,175495E−38 ÷ 3,402823+38, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

Ví dụ: Tính sin(x) của số thực x trong MD10 và cất kết quả vào ô nhớ MD20

```
L     MD10
SIN
T     MD20
```

7) Lệnh tính cos

Cú pháp COS

Lệnh tính cos của số thực trong ACCU1. Kết quả được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Nếu kết quả nằm trong khoảng −3,402823E+38 ÷ −1,175495E−38 hoặc bằng 0 hoặc nằm trong khoảng +1,175495E−38 ÷ 3,402823+38, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

8) Lệnh tính tg

Cú pháp TAN

Lệnh tính tg của số thực trong ACCU1 và ghi lại kết quả vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Nếu kết quả bằng 0 hoặc trong khoảng −3,402823E+38 ÷ −1,175495E−38 hoặc +1,175495E−38 ÷ 3,402823+38, bit OV sẽ có giá trị 0, ngược lại hai bits OV và OS sẽ cùng nhận giá trị 1.

9) Lệnh tính arsin

Cú pháp ASIN

Lệnh tính arcsin của số thực trong ACCU1, số thực này phải nằm trong khoảng −1÷1. Kết quả là một số thực trong khoảng $-\frac{\pi}{2} \div \frac{\pi}{2}$ và được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

10) Lệnh tính arcos

Cú pháp ACOS

Lệnh tính arccos của số thực trong ACCU1, số thực này phải nằm trong khoảng −1÷1. Kết quả là một số thực trong khoảng 0 ÷ −π và được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

11) Lệnh tính artg

Cú pháp ATAN

Lệnh tính arctg của số thực trong ACCU1. Kết quả là một số thực trong khoảng $-\frac{\pi}{2} \div \frac{\pi}{2}$ và được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

12) Lệnh tính bình phương

Cú pháp SQR

Lệnh tính giá trị bình phương của số thực trong ACCU1. Kết quả là một số thực không âm và được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

Ví dụ: Tính x^2 của số thực x trong MD10 và cất kết quả vào ô nhớ MD20

```
L     MD10
SQR
T     MD20
```

13) Lệnh tính căn bậc hai

Cú pháp SQRT

Lệnh tính giá trị bình phương của số thực trong ACCU1. số thực này phải là một số thực không âm. Kết quả là một số thực không âm và được ghi lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

Ví dụ: Tính $\sqrt{x}$ của số thực x trong MD10 và cất kết quả vào ô nhớ MD20

```
L     MD10
SQRT
T     MD20
```

14) Lệnh đảo dấu

Cú pháp NERG

Lệnh không có toán hạng và có tác dụng đổi dấu số thực dấu phảy động 32 bits trong ACUU1. Kết quả lại được cất trong ACCU1. Nội dung ACCU2 không bị thay đổi. Đặc biệt lệnh này không làm thay đổi nội dung của thanh ghi trạng thái.

2.4 Lệnh logic tiếp điểm trên thanh ghi trạng thái

Do tất cả các lệnh toán học với số nguyên và số thực vừa trình bày trên đây không sửa đổi nội dung bit trạng thái RLO nên người ta có thể sử dụng chúng kết hợp với lệnh logic như lệnh **AND**, **OR**, ... dưới dạng lệnh logic tiếp điểm trên thanh ghi trạng thái.

Các lệnh này tác động vào thanh ghi trạng thái (*Status word*) như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	x	x	x	1

2.4.1 Lệnh AND trên thanh ghi trạng thái

1) Lệnh **AND** nhỏ hơn

Cú pháp A <0

Lệnh tính $\mathbf{RLO} \wedge \mathbf{CC0} \wedge \overline{\mathbf{CC1}}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn 0 hay không rồi thực hiện phép tính $\wedge$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng).

Ví dụ: Nếu số nguyên 16 bits x trong MW10 thỏa mãn $-1 \le x < 2$ thì báo sáng đèn Q4.0

```
L     -1
L     MW10
<=I             // RLO=1 nếu -1 ≤ MW10.
L     2
-I              // Nếu MW10 < 2 thì CC1=0, CC0=1 và không thay đổi RLO.
A     <0
=     Q4.0
```

2) Lệnh **AND** lớn hơn

Cú pháp A >0

Lệnh tính $\mathbf{RLO} \wedge \overline{\mathbf{CC0}} \wedge \mathbf{CC1}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn 0 hay không rồi thực hiện phép tính $\wedge$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng).

Ví dụ: Nếu số thực x trong MD10 thỏa mãn $-1{,}5 < x < 2{,}8$ thì báo sáng đèn Q4.0

```
L     2.8
L     MD10
>R              // RLO=1 nếu 2,8 > MD10.
L     -1.5
-R              // Nếu MD10 > -1,5 thì CC1=1, CC0=0 và không thay đổi RLO.
A     >0
=     Q4.0
```

3) Lệnh **AND** khác nhau

Cú pháp A <>0

Lệnh tính $\mathbf{RLO} \wedge \left[(\overline{\mathbf{CC0}} \wedge \mathbf{CC1}) \vee (\mathbf{CC0} \wedge \overline{\mathbf{CC1}})\right]$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có khác 0 hay không rồi thực hiện phép tính $\wedge$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng).

Ví dụ: Nếu số nguyên 32 bits x trong MD10 thỏa mãn $x < 2$ và $x \neq -1$ thì báo sáng đèn Q4.0

```
L     2
L     MD10
>D              // RLO=1 nếu 2 > MD10.
L     -1
-D              // Nếu MD10 ≠ -1 thì CC1≠CC0 và không thay đổi RLO.
A     <>0
=     Q4.0
```

4) Lệnh **AND** bằng nhau

Cú pháp A ==0

Lệnh tính $\mathbf{RLO} \wedge \overline{\mathbf{CC0}} \wedge \overline{\mathbf{CC1}}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có bằng 0 hay không rồi thực hiện phép tính ∧ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số thực x trong MD10 thỏa mãn x<2,8 và x=5,4 thì báo sáng đèn Q4.0

```
L     2.8
L     MD10
>R              // RLO=1 nếu 2,8 > MD10.
L     5.4
-R              // Nếu MD10 =5,4 thì CC1=CC0=0 và không thay đổi RLO.
A     ==0
=     Q4.0
```

5) Lệnh **AND** lớn hơn hoặc bằng

Cú pháp A >=0

Lệnh tính $\mathbf{RLO} \wedge \overline{\mathbf{CC0}}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn hoặc bằng 0 hay không rồi thực hiện phép tính ∧ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số thực x trong MD10 thỏa mãn $-1,5 \le x < 2,8$ thì báo sáng đèn Q4.0

```
L     2.8
L     MD10
>R              // RLO=1 nếu 2,8 > MD10.
L     -1.5
-R              // Nếu MD10 ≥-1,5 thì CC0=0 và không thay đổi RLO.
A     >=0
=     Q4.0
```

6) Lệnh **AND** nhỏ hơn hoặc bằng

Cú pháp A <=0

Lệnh tính $\mathbf{RLO} \wedge \overline{\mathbf{CC1}}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn hoặc bằng 0 hay không rồi thực hiện phép tính ∧ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng).

2.4.2 Lệnh OR trên thanh ghi trạng thái

1) Lệnh **OR** nhỏ hơn

Cú pháp **O <0**

Lệnh tính $\mathbf{RLO} \vee (\mathbf{CC0} \wedge \overline{\mathbf{CC1}})$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn 0 hay không rồi thực hiện phép tính $\vee$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số nguyên 16 bits x trong MW10 thỏa mãn $-1<x$ hoặc $x>2$ thì báo sáng đèn Q4.0

```
L     2
L     MW10
<I              // RLO=1 nếu 2 ≤ MW10.
L     -1
-I              // Nếu MW10 < -1 thì CC1=0, CC0=1 và không thay đổi RLO.
O     <0
=     Q4.0
```

2) Lệnh **OR** lớn hơn

Cú pháp **O >0**

Lệnh tính $\mathbf{RLO} \vee (\overline{\mathbf{CC0}} \wedge \mathbf{CC1})$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn 0 hay không rồi thực hiện phép tính $\vee$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số thực x trong MD10 thỏa mãn $x<-1,5$ hoặc $x>2,8$ thì báo sáng đèn Q4.0

```
L     -1.5
L     MD10
>R              // RLO=1 nếu -1,5 > MD10.
L     2.8
-R              // Nếu MD10 > 2,8 thì CC1=1, CC0=0 và không thay đổi RLO.
O     >0
=     Q4.0
```

3) Lệnh **OR** khác nhau

Cú pháp **O <>0**

Lệnh tính $\mathbf{RLO} \vee \left[(\overline{\mathbf{CC0}} \wedge \mathbf{CC1}) \vee (\mathbf{CC0} \wedge \overline{\mathbf{CC1}})\right]$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có khác 0 hay không rồi thực hiện phép tính $\vee$ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số nguyên 32 bits x trong MD10 thỏa mãn $x<2$ hoặc $x \neq -1$ thì báo sáng đèn Q4.0

```
L     2
L     MD10
>D              // RLO=1 nếu 2 > MD10.
```

```
L     -1
-D                  // Nếu MD10 ≠ -1 thì CC1≠CC0 và không thay đổi RLO.
O     <>0
=     Q4.0
```

4) Lệnh **OR** bằng nhau

Cú pháp O == 0

Lệnh tính $RLO \vee (\overline{CC0} \wedge \overline{CC1})$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có bằng 0 hay không rồi thực hiện phép tính ∨ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số thực x trong MD10 thỏa mãn x<2,8 hoặc x=5,4 thì báo sáng đèn Q4.0

```
L     2.8
L     MD10
>R                  // RLO=1 nếu 2,8 > MD10.
L     5.4
-R                  // Nếu MD10 =5,4 thì CC1=CC0=0 và không thay đổi RLO.
O     == 0
=     Q4.0
```

5) Lệnh **OR** lớn hơn hoặc bằng

Cú pháp O >=0

Lệnh tính $RLO \vee \overline{CC0}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn hoặc bằng 0 hay không rồi thực hiện phép tính ∨ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng). Ví dụ: Nếu số thực x trong MD10 thỏa mãn x≤–1,5 hoặc x≥2,8 thì báo sáng đèn Q4.0

```
L     -1.5
L     MD10
>=R                 // RLO=1 nếu -1,5 ≥ MD10.
L     2.8
-R                  // Nếu MD10 ≥2.8 thì CC0=0 và không thay đổi RLO.
O     >=0
=     Q4.0
```

6) Lệnh **OR** nhỏ hơn hoặc bằng

Cú pháp O <=0

Lệnh tính $RLO \vee \overline{CC1}$. Kết quả được ghi lại vào RLO. Nói cách khác, lệnh kiểm tra kết quả phép tính vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn hoặc bằng 0 hay không rồi thực hiện phép tính ∨ giữa RLO với kết quả của phép kiểm tra đó (0–sai, 1–đúng).

2.4.3 Lệnh EXCLUSIVE OR trên thanh ghi trạng thái

1) Lệnh exclusive or nhỏ hơn

Cú pháp X <0

Lệnh đảo nội dung của **RLO** nếu $\mathbf{CC0} \wedge \overline{\mathbf{CC1}} = 1$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: nếu số nguyên 16 bits x trong MW10 thỏa mãn x>2 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     2
L     MW10
-I                // Nếu 2 < MW10 thì CC1=0, CC0=1 và không thay đổi RLO.
X     <0
=     Q4.0
```

2) Lệnh exclusive or lớn hơn

Cú pháp X >0

Lệnh đảo nội dung của RLO nếu $\overline{\mathbf{CC0}} \wedge \mathbf{CC1} = \mathbf{1}$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: nếu số thực x trong MD10 thỏa mãn x>−1,5 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     MD10
L     2.8
-R                // Nếu MD10 > 2,8 thì CC1=1 và CC0=0.
X     >0
=     Q4.0
```

Lệnh exclusive or khác nhau

Cú pháp X <>0

Lệnh đảo nội dung của RLO nếu $\overline{\mathbf{CC0}} \wedge \mathbf{CC1} = \mathbf{1}$ hoặc $\mathbf{CC0} \wedge \overline{\mathbf{CC1}} = 1$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có khác 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: nếu số nguyên 32 bits x trong MD10 khác x≠ −1 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     MD10
L     -1
-D                // Nếu MD10 ≠ -1 thì CC1≠CC0.
X     <>0
=     Q4.0
```

4) Lệnh exclusive or bằng nhau

Cú pháp X ==0

Lệnh đảo nội dung của RLO nếu $\overline{CC0} \wedge \overline{CC1} = 1$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có bằng 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: Nếu số thực x trong MD10 thỏa mãn x=5,4 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     MD10
L     5.4
-R                // Nếu MD10 =5,4 thì CC1=CC0=0 và không thay đổi RLO.
X     ==0
=     Q4.0
```

5) Lệnh exclusive or lớn hơn hoặc bằng

Cú pháp X >=0

Lệnh đảo nội dung của RLO nếu $\overline{CC0} = 1$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có lớn hơn hoặc bằng 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: Nếu số thực x trong MD10 thỏa mãn x≥2,8 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     MD10
L     2.8
-R                // Nếu MD10 ≥2.8 thì CC0=0.
X     >=0
=     Q4.0
```

6) Lệnh exclusive or nhỏ hơn hoặc bằng

Cú pháp X <=0

Lệnh đảo nội dung của RLO nếu $\overline{CC1} = 1$. Nói cách khác lệnh sẽ kiểm tra kết quả phép tính toán học vừa thực hiện (cộng, trừ, nhân, chia ...) có nhỏ hơn hoặc bằng 0 hay không và nếu kết quả kiểm tra là đúng thì đảo nội dung bit RLO.

Ví dụ: Nếu số thực x trong MD10 thỏa mãn x≤−1,5 thì đảo trạng thái đèn Q4.0

```
A     Q4.0
L     MD10
L     -1.5
-R                // Nếu MD10 ≤-1.5 thì CC1=0.
X     <=0
=     Q4.0
```

2.5 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 phảy động
- và một số dạng dữ liệu khác.

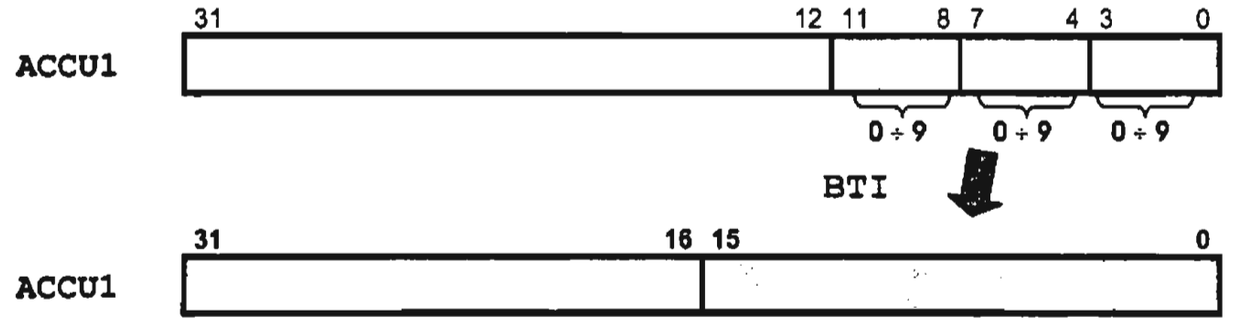
Việc làm với nhiều dạng dữ liệu khác nhau đặt ra cho ta vấn đề chuyển đổi chúng. Ví dụ khi đọc tín hiệu tương tự từ cổng tương tự ta nhận được số liệu dạng nguyên 16 bits mang giá trị tín hiệu tương tự chứ không phải bản thân giá trị đó, bởi vậy để xử lý tiếp thì cần thiết phải chuyển số nguyên đó thành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

2.5.1 Chuyển đổi số BCD thành số nguyên và ngược lại

1) Lệnh chuyển đổi BCD thành số nguyên 16 bits

Cú pháp BTI

Lệnh không có toán hạng và thực hiện việc 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. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái.



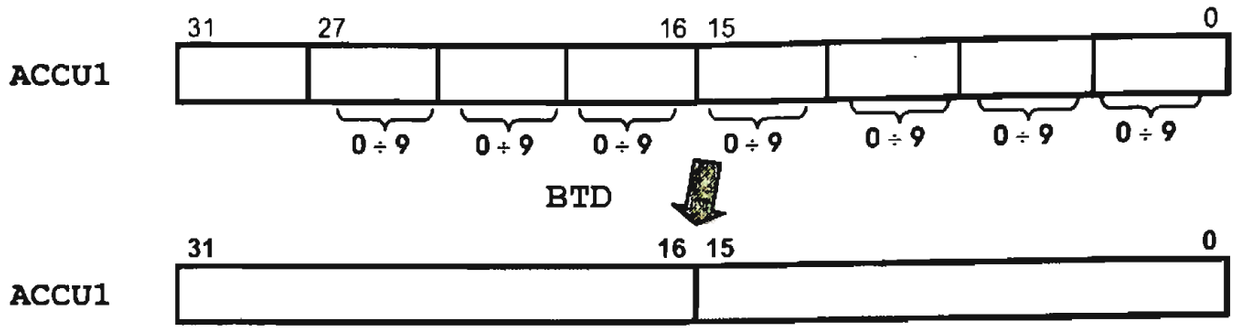
Nếu số BCD cần chuyển đổi có cấu trúc sai, ví dụ như có một chữ số (4 bits nhị phân) không nằm trong khoảng từ 0 đến 9, CPU sẽ gọi chương trình ngắt xử lý lỗi OB121 hoặc chuyển sang chế độ STOP (nếu OB121 không có chương trình).

2) Lệnh chuyển đổi BCD thành số nguyên 32 bits

Cú pháp BTD

Lệnh không có toán hạng và thực hiện việc 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 ACCU1.

Nội dung của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái.

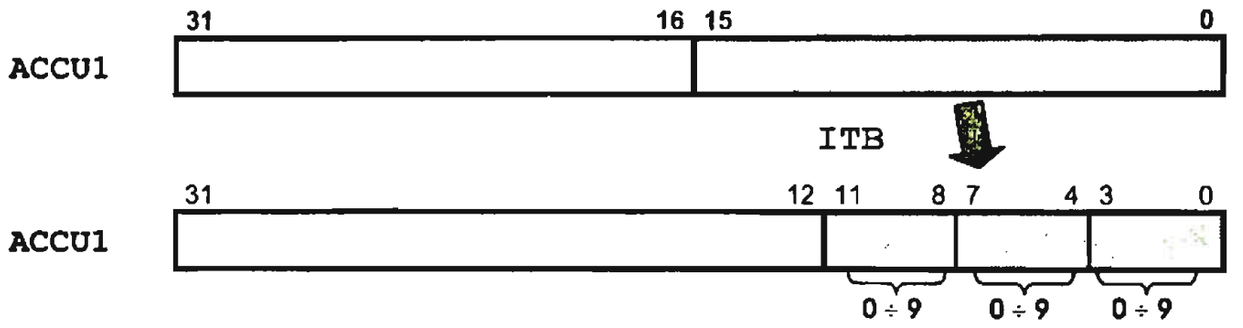


Nếu số BCD cần chuyển đổi có cấu trúc sai, ví dụ như có một chữ số (4 bits nhị phân) không nằm trong khoảng từ 0 đến 9, CPU sẽ gọi chương trình ngắt xử lý lỗi OB121 hoặc chuyển sang chế độ STOP (nếu OB121 không có chương trình).

3) Lệnh chuyển đổi số nguyên 16 bits thành BCD

Cú pháp ITB

Lệnh không có toán hạng và thực hiện việc chuyển đổi số nguyên 16 bits nằm trong từ thấp của ACCU1 thành số BCD có 3 chữ số nằm. Kết quả được cất lại vào từ thấp của ACCU1. Nội dung của từ cao trong ACCU1 và của ACCU2 không bị thay đổi.



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 làm thay đổi nội dung thanh ghi trạng thái như sau:

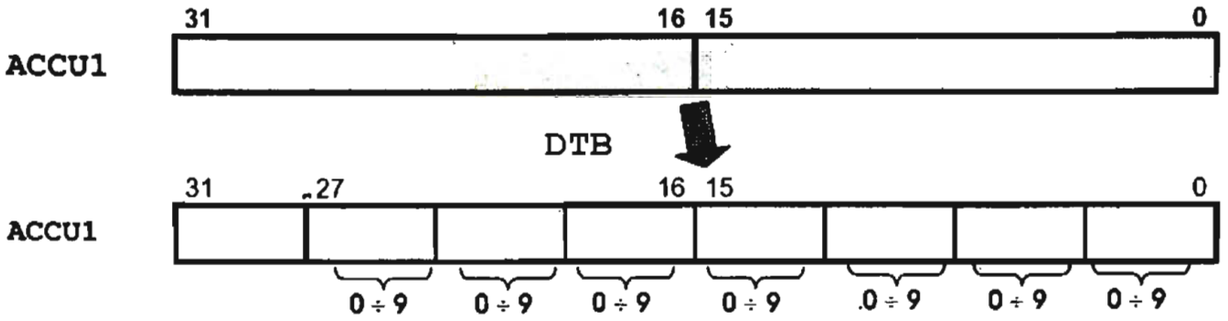
BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	X	X	–	–	–	–

4) Lệnh chuyển đổi số nguyên 32 bits thành BCD

Cú pháp DTB

Lệnh không có toán hạng và thực hiện việc chuyển đổi số nguyên 32 bits trong ACCU1 thành số BCD có 7 chữ số. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi.

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 làm thay đổi nội dung thanh ghi trạng thái giống như lệnh ITB.



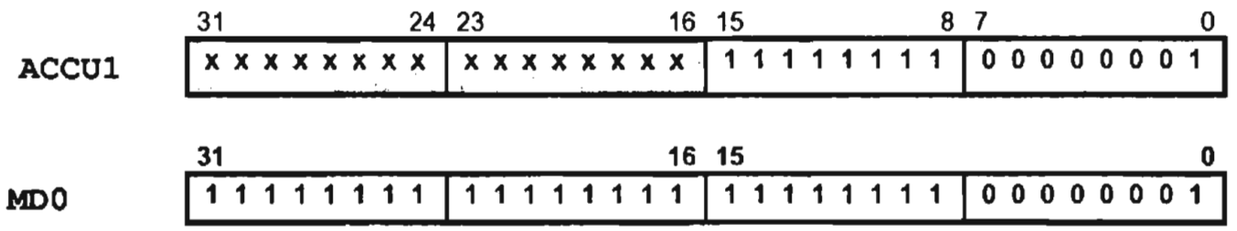
2.5.2 Chuyển đổi số nguyên 16 bits thành số nguyên 32 bits

Lệnh chuyển đổi số nguyên 32 bits thành BCD

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 lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái. Ví dụ

```
L     -127
ITD
T     MD0
```



2.5.3 Chuyển đổi số nguyên 32 bits thành số thực

Lệnh chuyển đổi số nguyên 32 bits thành BCD

Cú pháp DTR

Lệnh thực hiện việc chuyển đổi một số nguyên 32 bits trong ACCU1 thành số thực 32 bits dấu phảy động. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Lệnh cũng không làm thay đổi nội dung thanh ghi trạng thái.

Ví dụ: đổi giá trị nguyên 16 bits (−3278 ÷ 3277) đọc được từ cổng tương tự PIW304 thành một số thực có giá trị đúng bằng mức điện áp tín hiệu tại cổng (−10V ÷ 10V) và cất kết quả vào ô nhớ MD0.

```
L     PIW3047  // Số đọc được là số nguyên 16 bits
ITD            // Chuyển thành số nguyên 32 bits
DTR            // Chuyển thành số thực
L     3276.7
/R             // Tính ra đúng giá trị điện áp tại cổng
T     MD0
```

2.5.4 Chuyển đổi số thực thành số nguyên 32 bits

1) Lệnh chuyển số thực thành số nguyên gần nhất

Cú pháp RND

Lệnh không có toán hạng và thực hiện việc chuyển đổi số thực dấu phảy động trong ACCU1 thành số nguyên 32 bits có giá trị gần nhất so với số thực đã cho. Nếu số thực đã cho nằm giữa hai số nguyên, ví dụ 10,5 thì CPU sẽ lấy số chẵn. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Ví dụ

```
L     -12.7
RND
T     MD0   // MD0 chứa số nguyên 32 bits -13
```

Trường hợp số thực nằm ngoài khoảng mà một số nguyên 32 bits có thể biểu diễn được, CPU sẽ thông báo bằng kết quả tràn ở thanh ghi trạng thái. Lệnh thay đổi nội dung thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	X	X	–	–	–	–

2) Lệnh chuyển số thực thành số nguyên nhỏ nhất nhưng không nhỏ hơn số thực

Cú pháp RND+

Lệnh không có toán hạng và thực hiện việc chuyển đổi số thực dấu phảy động trong ACCU1 thành số nguyên 32 bits có giá trị nhỏ nhất nhưng không nhỏ hơn số thực đã cho. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Ví dụ

```
L     -12.7
RND+
T     MD0   // MD0 chứa số nguyên 32 bits -12.
```

Trường hợp số thực nằm ngoài khoảng mà một số nguyên 32 bits có thể biểu diễn được, CPU sẽ thông báo bằng kết quả tràn ở thanh ghi trạng thái. Lệnh thay đổi nội dung thanh ghi trạng thái giống như lệnh RND.

3) Lệnh chuyển số thực thành số nguyên lớn nhất nhưng không lớn hơn số thực

Cú pháp RND-

Lệnh không có toán hạng và thực hiện việc chuyển đổi số thực dấu phảy động trong ACCU1 thành số nguyên 32 bits có giá trị lớn nhất nhưng không lớn hơn số thực đã cho. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Ví dụ

```
L     -12.7
RND-
T     MD0   // MD0 chứa số nguyên 32 bits -13.
```

Trường hợp số thực nằm ngoài khoảng mà một số nguyên 32 bits có thể biểu diễn được, CPU sẽ thông báo bằng kết quả tràn ở thanh ghi trạng thái. Lệnh thay đổi nội dung thanh ghi trạng thái giống như lệnh RND.

4) Lệnh lấy phần nguyên

Cú pháp TRUNC

Lệnh không có toán hạng và thực hiện việc lấy phần nguyên của số thực dấu phảy động trong ACCU1. Kết quả được cất lại vào ACCU1. Nội dung của ACCU2 không bị thay đổi. Trường hợp số thực nằm ngoài khoảng mà một số nguyên 32 bits có thể biểu diễn được, CPU sẽ thông báo bằng kết quả tràn ở thanh ghi trạng thái. Lệnh thay đổi nội dung thanh ghi trạng thái giống như lệnh RND.

2.6 Các lệnh điều khiển chương trình

2.6.1 Nhóm lệnh kết thúc chương trình

S7-300 có hai lệnh kết thúc chương trình là BEC và BEU.

1) Lệnh kết thúc vô điều kiện

Cú pháp BEU

Lệnh không có toán hạng và thực hiện việc kết thúc chương trình trong khối một cách vô điều kiện. Lệnh thay đổi nội dung thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	0	0	1	–	0

2) Lệnh kết thúc có điều kiện

Cú pháp BEC

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

Lệnh thay đổi nội dung thanh ghi trạng thái như sau:

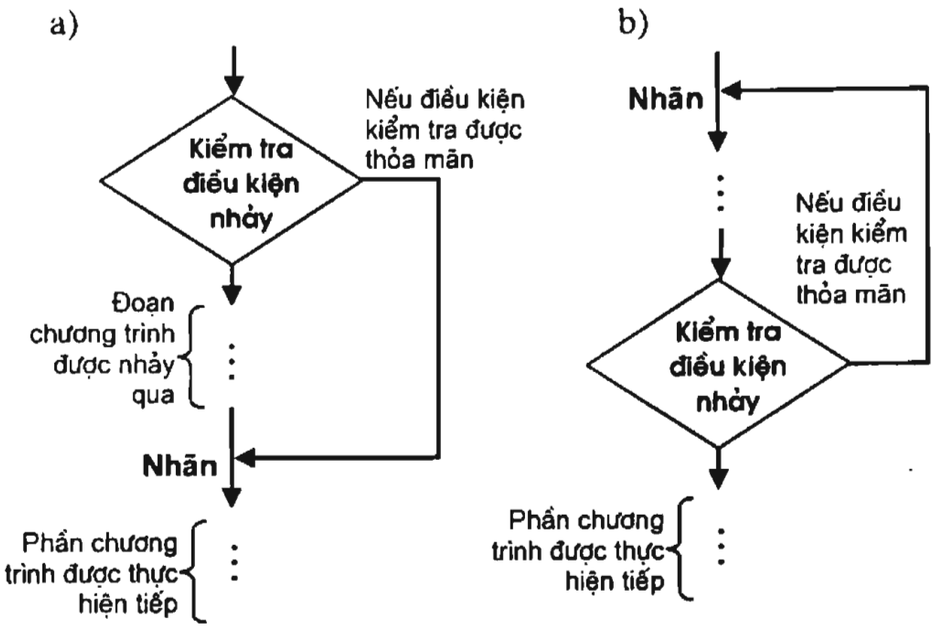
BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	x	0	1	1	0

Ví dụ chương trình sau sẽ đổi trạng thái tín hiệu cổng ra Q4.0 ở vòng quét có số thứ tự chia hết cho 1000 và không làm gì ở các vòng quét khác.

```
L     1
L     MW0       // Thanh ghi đếm số vòng quét.
+I
T     MW0
L     1000
<>I
BEC             // Kết thúc nếu chưa phải là vòng quét thứ 1000.
L     0
T     MW0       // Nạp lại số vòng quét từ đầu.
AN    Q4.0
=     Q4.0
BEU
```

2.6.2 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 thỏa mãn. Nơi lệnh nhảy tới phải thuộc cùng một khối chương trình với lệnh. Không thể nhảy từ khối chương trình này này sang một khối chương trình khác, ví dụ không thể nhảy từ FC1 sang FC10.



Hình 2.4: Lệnh rẽ nhánh theo bit trạng thái.
a) Nhảy xuôi.
b) Nhảy ngược

Nhãn là một dãy với nhiều nhất 4 ký tự hoặc số và phải được bắt đầu bằng một ký tự. Khoảng cách bước nhảy tính theo ô nhớ chứa chương trình, phải ít hơn 32767 từ. Nơi nhảy đến có thể nằm trước hoặc nằm sau lệnh nhảy. Hình 2.4 mô tả phương thức làm việc của lệnh rẽ nhánh theo bit trạng thái tới địa điểm "nhãn" trong chương trình.

1) Rẽ nhánh khi BR=1

Cú pháp `JBI <nhãn>`

Lệnh thay đổi thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	–	0

Ví dụ:

```
      JBI   dest
      :     } Đoạn chương trình được nhảy qua
dest: A     I0.0
      :     } Phần chương trình được thực hiện tiếp
```

2) Rẽ nhánh khi BR=0

Cú pháp JNBI <nhãn>

Lệnh này sẽ làm thay đổi thanh ghi trạng thái giống như lệnh JBI.

Ví dụ:

```
dest: A     I0.0
      :
      JNBI  dest      // Nhảy đến dest nếu BR=0.
      :
```

3) Rẽ nhánh khi RLO=1

Cú pháp JC <nhãn>

Lệnh thay đổi thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	1	1	0

Ví dụ:

```
      JC    dest      // Nhảy đến dest nếu RLO=1.
      :   } Đoạn chương trình được nhảy qua
dest: A     I0.0
      :   } Phần chương trình được thực hiện tiếp
```

4) Rẽ nhánh khi RLO=0

Cú pháp JCN <nhãn>

Lệnh này sẽ thay đổi thanh ghi trạng thái giống như lệnh JC. Ví dụ:

```
      JCN   dest      // Nhảy đến dest nếu RLO=0.
      :   } Đoạn chương trình được nhảy qua
dest: A     I0.0
      :   } Phần chương trình được thực hiện tiếp
```

5) Rẽ nhánh khi CC1=0 và CC0=1

Cú pháp JM <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả âm. Ví dụ

```
dest: A     I0.0
      :
      JM    dest      // Nhảy đến dest nếu nội dung của ACCU1 là một số âm.
      :   } Phần chương trình được thực hiện tiếp
```

6) Rẽ nhánh khi CC1=1 và CC0=0

Cú pháp JP <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả dương. Ví dụ

```
dest: A     I0.0
      :
      JP    dest      // Nhảy đến dest nếu nội dung của ACCU1 là một số dương.
      :   } Phần chương trình được thực hiện tiếp
```

7) Rẽ nhánh khi CC1=CC0=0

Cú pháp JZ <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả bằng 0. Ví dụ

```
dest: A     I0.0
      :
      JZ    dest      // Nhảy đến dest nếu nội dung của ACCU1 là một số bằng 0.
      :   } Phần chương trình được thực hiện tiếp
```

8) Rẽ nhánh khi CC1≠ CC0

Cú pháp JN <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả khác 0.

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

Cú pháp JMZ <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả là một số không dương (âm hoặc bằng 0). Ví dụ

```
      JMZ   dest      // Nhảy đến dest nếu nội dung của ACCU1 là một số không dương.
      :   } Đoạn chương trình được nhảy qua
dest: A     I0.0
      :   } Phần chương trình được thực hiện tiếp
```

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

Cú pháp JPZ <nhãn>

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái. Nó được sử dụng để rẽ nhánh nếu như phép tính trước đó có kết quả là một số không âm (dương hoặc bằng 0). Ví dụ

```
        JPZ    dest        // Nhảy đến dest nếu nội dung của ACCU1 là một số không âm.
        :    } Đoạn chương trình được nhảy qua
dest:   A      I0.0
        :    } Phần chương trình được thực hiện tiếp
```

11) Rẽ nhánh vô điều kiện

Cú pháp **JU** **<nhãn>**

Lệnh nhảy này không làm thay đổi nội dung thanh ghi trạng thái và được thực hiện vô điều kiện, không phụ thuộc bất cứ một bit trạng thái nào.

2.6.3 Lệnh xoay vòng (LOOP)

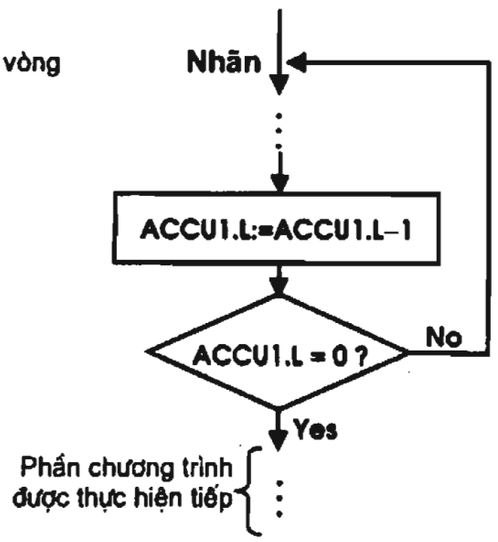
Cú pháp **LOOP <nhãn>**

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 thì CPU 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 (hình 2.5). Đoạn chương trình nằm giữa nhãn và lệnh LOOP sẽ được thực hiện cho tới khi nội dung thanh ghi ACCU1 bằng 0.

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

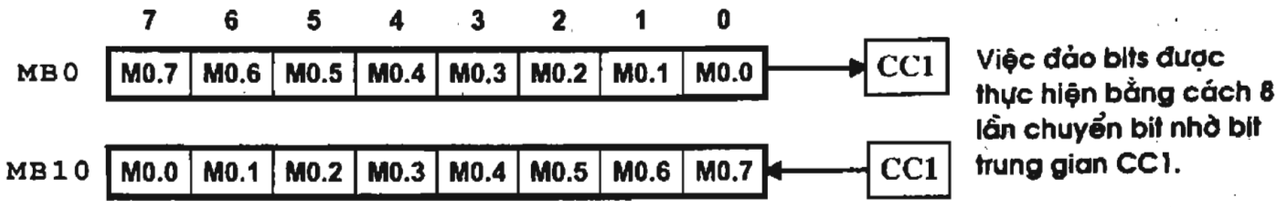
Hình 2.5: Nguyên tắc làm việc của lệnh xoay vòng (loop)



Ví dụ 1: Chương trình sau mô phỏng việc thực hiện xoay vòng 10 lần đoạn chương trình nằm giữa nhãn "dest" và lệnh LOOP.

```
      L     10
dest: T     MW0       // Từ MW0 được sử dụng làm thanh ghi đếm số lần thực hiện.
      :   } Phần chương trình được thực hiện xoay vòng
      L     MW0
      LOOP  dest      // Giảm ACCU1 đi 1 đơn vị và nhảy đến dest nếu kết quả khác 0.
      :
```

Ví dụ 2: Hãy viết chương trình đảo thứ tự các bits của byte MB0 và cất kết quả vào MB10 mỗi khi có sườn lên tại đầu vào I0.0.



Để làm việc này ta sử dụng 8 lần lệnh dịch bit phải RRCA nhằm đọc bit ngoài cùng bên phải của MB0 và ghi nó vào bit ngoài cùng bên phải của MB10 cũng bằng lệnh dịch bit nhưng theo chiều ngược lại RLCA.

Chương trình sử dụng MB1 làm byte trung gian và MB20 làm thanh ghi đếm số lần thực hiện.

```
      A     I0.0
      FP    M2.0      // Bit ghi nhớ trạng thái I0.0 trước đó.
      JCN   end       // Kết thúc chương trình nếu không có sườn lên tại I0.0.
      L     MB0
      T     MB1       // Sử dụng MB1 làm thanh ghi trung gian tức thời
      L     8
dest: T     MB20      // Từ MB20 được sử dụng làm thanh ghi đếm số lần thực hiện.
      L     MB1
      RRDA            // Lấy bit ngoài cùng bên phải của MB1 vào CC1.
      T     MB1
      L     MB10
      RLDA            // Chuyển CC1 vào bit ngoài cùng bên phải của MB10.
      T     MB10
      L     MB20
      LOOP  dest
end:  BEU
```

2.6.4 Lệnh rẽ nhánh theo danh mục (JUMP LIST)

Cú pháp JL <Nhãn>

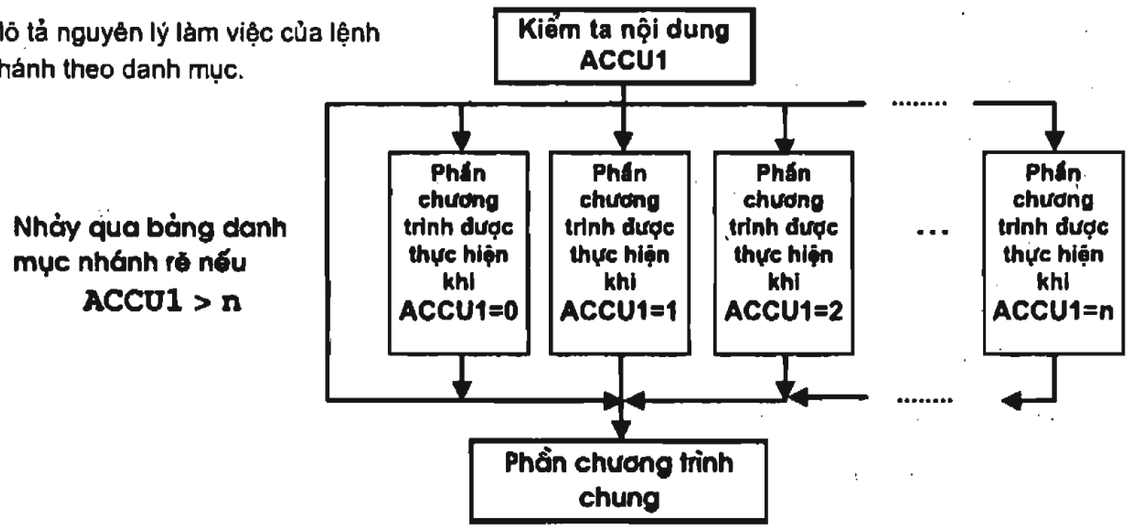
Lệnh thực hiện một loạt sự rẽ nhánh tùy theo nội dung của ACCU1. Danh mục các nhánh rẽ phải được xếp ngay sau lệnh JL dưới dạng lệnh nhảy vô điều kiện và với thứ tự từ thấp đến cao theo nội dung của ACCU1.

Số các nhánh rẽ nhiều nhất có thể là 255. Toán hạng **<nhãn>** trong lệnh chỉ phần kết thúc bảng danh mục các nhánh rẽ.

Lệnh rẽ nhánh theo danh mục không làm thay đổi nội dung thanh ghi trạng thái và có tác dụng giống như lệnh do case của Access hay dBASE.

Hình 2.6 biểu diễn nguyên tắc làm việc của lệnh rẽ nhánh theo danh mục dưới dạng sơ đồ khối.

Hình 2.6: Mô tả nguyên lý làm việc của lệnh rẽ nhánh theo danh mục.



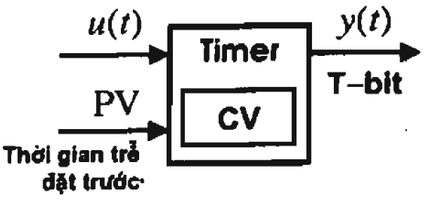
Phần chương trình dưới đây minh họa việc cài đặt sơ đồ khối đó.

```
        :
        JL      list
        JU      Ac0         // Nhảy tới Ac0 nếu ACCU1=0.
        JU      Ac1         // Nhảy tới Ac0 nếu ACCU1=1.
        :                                          } Bảng danh mục các nhánh rẽ
        JU      Acn         // Nhảy tới Acn nếu ACCU1=n.
list:   JU      Comm        // Kết thúc bảng danh mục. Nhảy tới Comm nếu ACCU1>n.
Ac0:
        :       } Phần chương trình xử lý trường hợp ACCU1=0.
        JU      Comm        // Nhảy tới phần chương trình chung Comm.
Ac1:
        :       } Phần chương trình xử lý trường hợp ACCU1=1.
        JU      Comm        // Nhảy tới phần chương trình chung Comm.
Ac2:
        :       } Phần chương trình xử lý trường hợp ACCU1=2.
        JU      Comm        // Nhảy tới phần chương trình chung Comm.
        :
        JU      Comm        // Nhảy tới phần chương trình chung Comm.
Acn:
        :       } Phần chương trình xử lý trường hợp ACCU1=n.
Comm:
        :       } Phần chương trình chung
```

2.7 Bộ thời gian (Timer)

2.7.1 Nguyên tắc làm việc

Bộ thời gian (Timer), là bộ tạo thời gian trễ τ mong muốn giữa tín hiệu logic đầu vào $u(t)$ và tín hiệu logic đầu ra $y(t)$.



Hình 2.7: Mô tả nguyên lý làm việc của Timer.

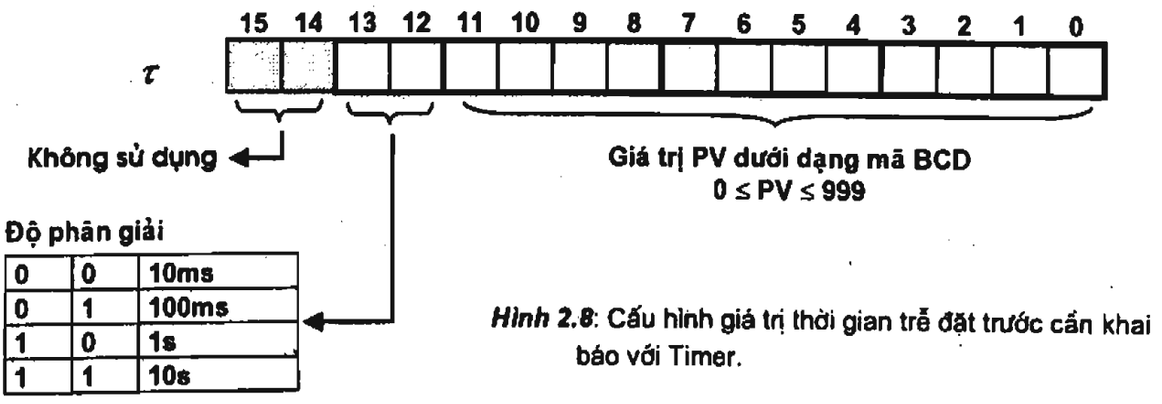
S7–300 có 5 loại Timer khác nhau. Tất cả 5 loại Timer này cùng bắt đầu tạo thời gian trễ tín hiệu kể từ thời điểm có sườn lên ở tín hiệu đầu vào, tức là khi tín hiệu đầu vào $u(t)$ chuyển trạng thái logic từ 0 lên 1, được gọi là *thời điểm* Timer *được kích*.

Thời gian trễ τ mong muốn được khai báo với Timer bằng một giá trị 16 bits (hình 2.8) bao gồm hai thành phần:

- Độ phân giải với đơn vị là ms. Timer của S7–300 có 4 loại độ phân giải khác nhau là 10ms, 100ms, 1s và 10s.
- Một số nguyên (BCD) trong khoảng 0 ÷ 999 được gọi là PV (chữ viết tắt của *Preset value* – giá trị đặt trước).

Như vậy thời gian trễ τ mong muốn sẽ chính là tích

τ = Độ phân giải × PV.

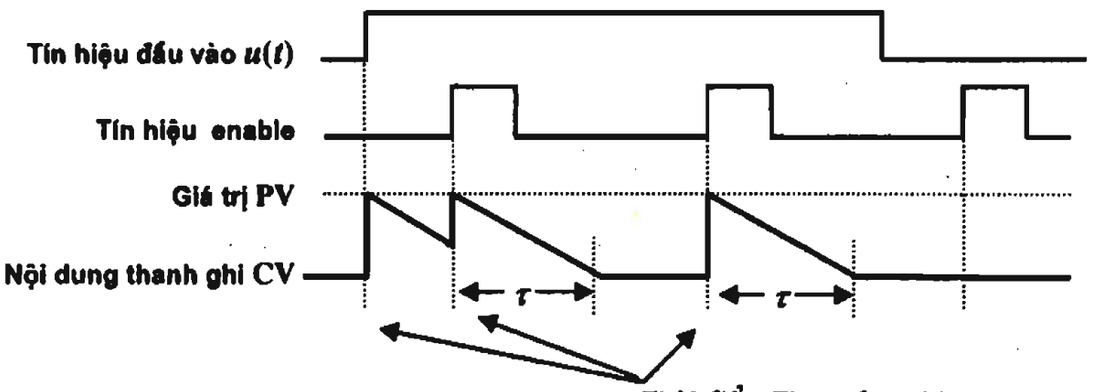


Hình 2.8: Cấu hình giá trị thời gian trễ đặt trước cần khai báo với Timer.

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

Bên cạnh sườn lên của tín hiệu đầu vào $u(t)$, Timer còn có thể được kích bằng sườn lên của tín hiệu kích chủ động có tên là tín hiệu *enable* nếu như tại thời điểm có sườn lên của tín hiệu enable, tín hiệu đầu vào $u(t)$ có giá trị logic 1.

Hình 2.9 tóm tắt lại nguyên lý làm việc chung của Timer trong S7–300/400.



Hình 2.9: Nguyên lý làm việc của Timer.

Từng loại Timer được đánh số từ 0 đến (tùy thuộc từng loại CPU) 255. Một Timer được đặt tên là Tx, trong đó x là số hiệu của Timer ($0 \le x \le 255$). Ký hiệu Tx cũng đồng thời cũng là địa chỉ hình thức của thanh ghi CV (T-word) và của đầu ra (T-bit) của Timer đó. Tuy chúng có cùng địa chỉ hình thức, song T-word và T-bit vẫn được phân biệt với nhau nhờ kiểu lệnh sử dụng với toán hạng Tx. Khi dùng lệnh làm việc với từ, Tx được hiểu là địa chỉ của T-word, ngược lại khi sử dụng lệnh làm việc với tiếp điểm, Tx được hiểu là địa chỉ của T-bit.

Một Timer đang trong chế độ làm việc (sau khi được kích) có thể được đưa lại về trạng thái chờ khởi động ban đầu, tức là chờ sườn lên tiếp theo của tín hiệu đầu vào. Công việc này gọi là *reset* Timer đó. Tín hiệu *reset* Timer được gọi là tín hiệu xoá và khi tín hiệu xóa có giá trị bằng 1 Timer sẽ không làm việc. Tại thời điểm xuất hiện sườn lên của tín hiệu xóa, T-word và T-bit của nó đồng thời được xóa về 0, tức là thanh ghi đếm tức thời CV được đặt về 0 và tín hiệu đầu ra cũng có trạng thái logic bằng 0.

2.7.2 Khai báo sử dụng

Việc khai báo sử dụng một 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 xóa Timer nếu muốn sử dụng chế độ reset chủ động.

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

1) 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. Ví dụ

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    T1       // Sử dụng cho Timer có tên là T1.
```

Lệnh FR tác động vào thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	–	–	0

2) Khai báo tín hiệu đầu vào

```
Cú pháp    A     <Địa chỉ bit>
```

"Địa chỉ bit" trong toán hạng xác định tín hiệu đầu vào $u(t)$ cho Timer. Ví dụ

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    T1       // Sử dụng cho Timer có tên là T1.
A     I2.1     // Tín hiệu tại cổng vào I2.1 là tín hiệu đầu vào của Timer.
```

3) Khai báo thời gian trễ mong muốn

```
Cú pháp    L     <hằng số>
```

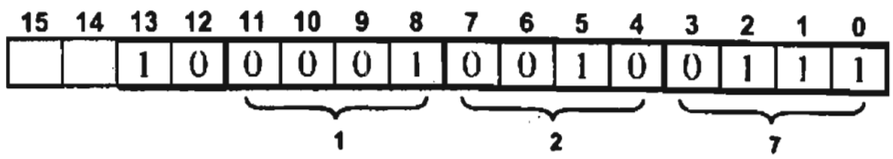
"Hằng số" trong toán hạng xác định giá trị thời gian trễ τ đặt trước cho Timer. Hằng số này có hai dạng

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

```
A     I2.0                  // Tín hiệu enable.
FR    T1
A     I2.1                  // Tín hiệu đầu vào.
L     S5T#00H02M23S00MS     // Thời gian trễ là 2 phút 23 giây.
```

- Dạng một số nguyên 16 bits có cấu trúc như hình 2.8 mô tả. Ví dụ

```
A     I2.0                  // Tín hiệu enable.
FR    T1
A     I2.1                  // Tín hiệu đầu vào.
L     W#16#2127             // Thời gian trễ là 127 giây (1 giây×127).
```



4) Khai báo loại Timer

S7–300 có 5 loại Timer được khai báo bằng các lệnh

- SD: Trễ theo sườn lên 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.

Những lệnh này tác động vào thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	–	–	0

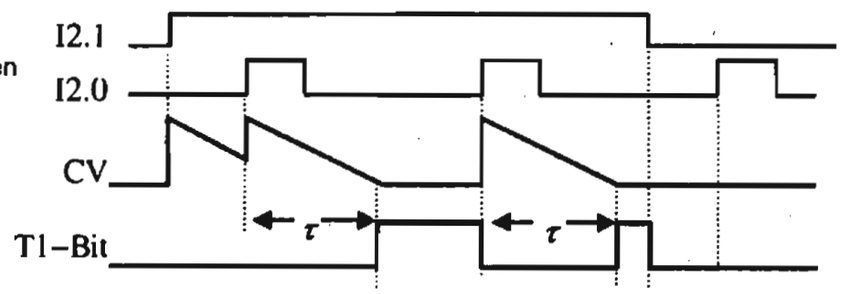
a) Trễ theo sườn lên không có nhớ (On delay timer)

Cú pháp SD <Tên Timer>

Thời gian giữ trễ được bắt đầu khi có sườn lên của tín hiệu đầu vào (hoặc khi có sườn lên của tín hiệu enable đồng thời tín hiệu vào bằng 1), tức là 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 0. Khi hết thời gian trễ T–bit có giá trị bằng 1. Như vậy T–bit có giá trị 1 khi T–Word=0.

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

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



Hình 2.10: Timer trễ theo sườn lên không có nhớ.

Ví dụ

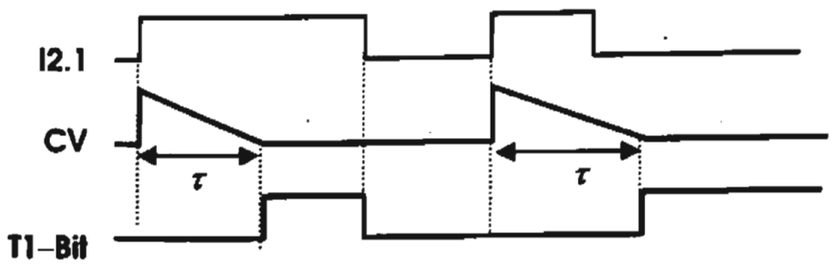
```
A     I2.0          // Tín hiệu enable.
FR    T1
A     I2.1          // Tín hiệu đầu vào.
L     W#16#2127     // Thời gian trễ là 127 giây.
SD    T1            // Loại Timer trễ theo sườn lên.
```

b) Trễ theo sườn lên có nhớ (Retentive on delay timer)

Cú pháp SS <Tên Timer>

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

Hình 2.11: Timer trễ theo sườn lên có nhớ.



Khoảng thời gian trễ chính là khoảng thời gian giữa thời điểm xuất hiện sườn lên của tín hiệu đầu 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. Ví dụ

```
A     I2.1          // Tín hiệu đầu vào.
L     W#16#2127     // Thời gian trễ là 127 giây.
SS    T1            // Loại Timer trễ theo sườn lên.
```

c) Timer tạo xung không có nhớ (Pulse timer)

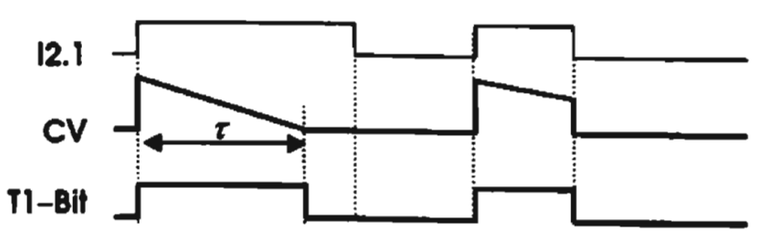
Cú pháp SP <Tên Timer>

Thời gian giữ trễ được bắt đầu tính từ khi xuất hiện sườn lên của tín hiệu đầu vào (hoặc khi có sườn lên của tín hiệu enable đồng thời tín hiệu vào bằng 1), tức là ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T–Word (CV). Trong khoảng thời gian 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. Ví dụ

```
A     I2.1          // Tín hiệu đầu vào.
L     S5T#2S25M     // Thời gian trễ là 2 giây và 20 mili giây.
SP    T1            // Loại Timer trễ theo sườn lên.
```

Hình 2.12: Timer tạo xung không có nhớ.

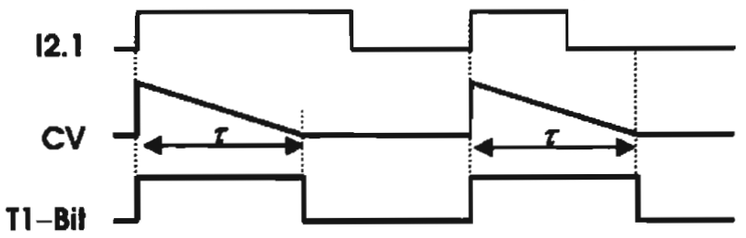


d) Timer tạo xung có nhớ (Extended pule timer)

Cú pháp SE <Tên Timer>

Thời gian giữ trễ được bắt đầu tính từ khi xuất hiện sườn lên của tín hiệu đầu vào (hoặc khi có sườn lên của tín hiệu enable đồng thời tín hiệu vào bằng 1), tức là ở ngay thời điểm đó giá trị PV được chuyển vào thanh ghi T–Word (CV).

Hình 2.13: Timer tạo xung có nhớ.



Trong khoảng thời gian 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 0 theo tín hiệu đầu vào.Ví dụ

```
A     I2.1       // Tín hiệu đầu vào.
L     S5T#2S25M  // Thời gian trễ là 2 giây và 20 mili giây.
SE    T1         // Loại Timer trễ theo sườn lên.
```

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

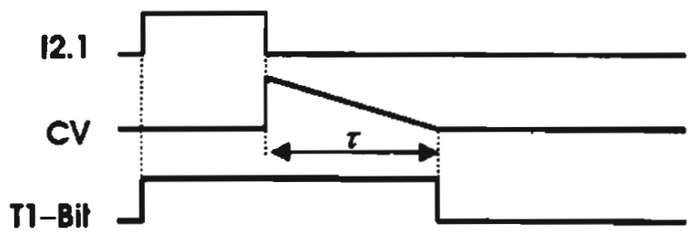
Cú pháp SF <Tên Timer>

Thời gian gữ trễ được bắt đầu tính từ 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 đó T–bit có giá trị bằng 0.

```
A     I2.1       // Tín hiệu đầu vào.
L     S5T#2S25M  // Thời gian trễ là 2 giây và 20 mili giây.
SF    T1         // Loại Timer trễ theo sườn lên.
```

Hình 2.14: Trễ theo sườn xuống.

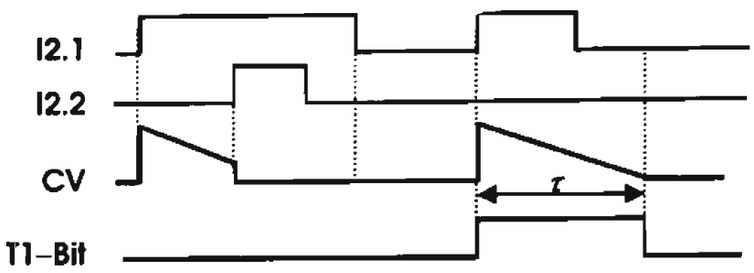


5) Khai báo tín hiệu xóa (reset)

```
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 xóa cho Timer có tên cho trong toán hạng thứ hai.

Hình 2.15: Minh họa tín hiệu xóa chủ động.



Khi tín hiệu xóa bằng 1, T–Word (thanh ghi CV) và T–Bit cùng đồng thời được đưa về 0. Nếu tín hiệu xóa về 0, Timer sẽ chờ được kích lại. Ví dụ

```
A     I2.1          // Tín hiệu đầu vào.
L     W#16#2127     // Thời gian trễ là 127 giây.
SE    T1            // Loại Timer trễ theo sườn lên.
A     I2.2          // Tín hiệu tại cổng vào I2.2 sẽ được dùng làm tín hiệu reset Timer.
R     T1
```

2.7.3 Đọc nội dung thanh ghi T–Word (CV)

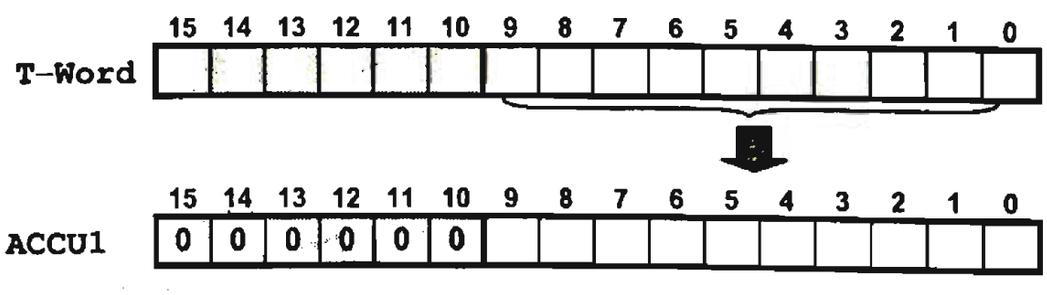
Nội dung thanh ghi T–Word là CV có thể được đọc vào ACCU1 theo hai cách:

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 tỷ số giữa khoảng thời gian kể từ khi Timer được kích, và độ phân giải.



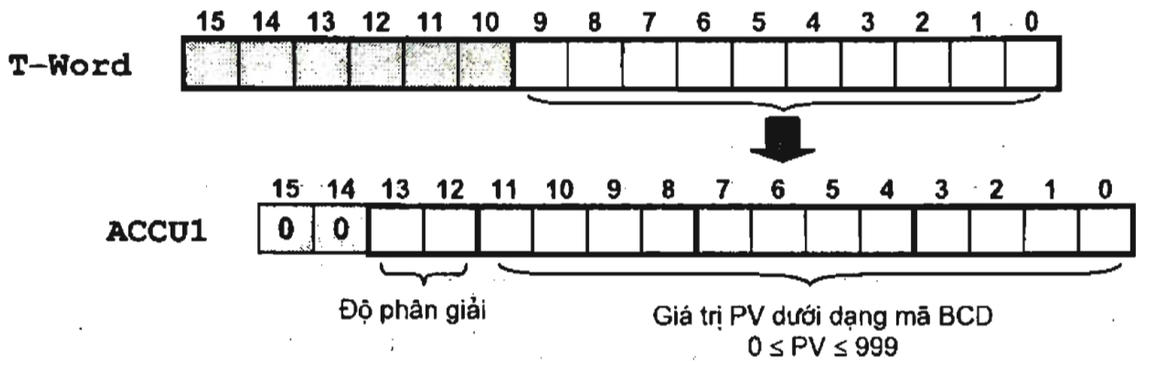
Ví dụ:

```
L     T1          // Đọc giá trị CV dạng số nguyên dương (nhị phân).
T     MW10
```

2) Đọc thời gian trễ tức thời

Cú pháp LC <Tên Timer>

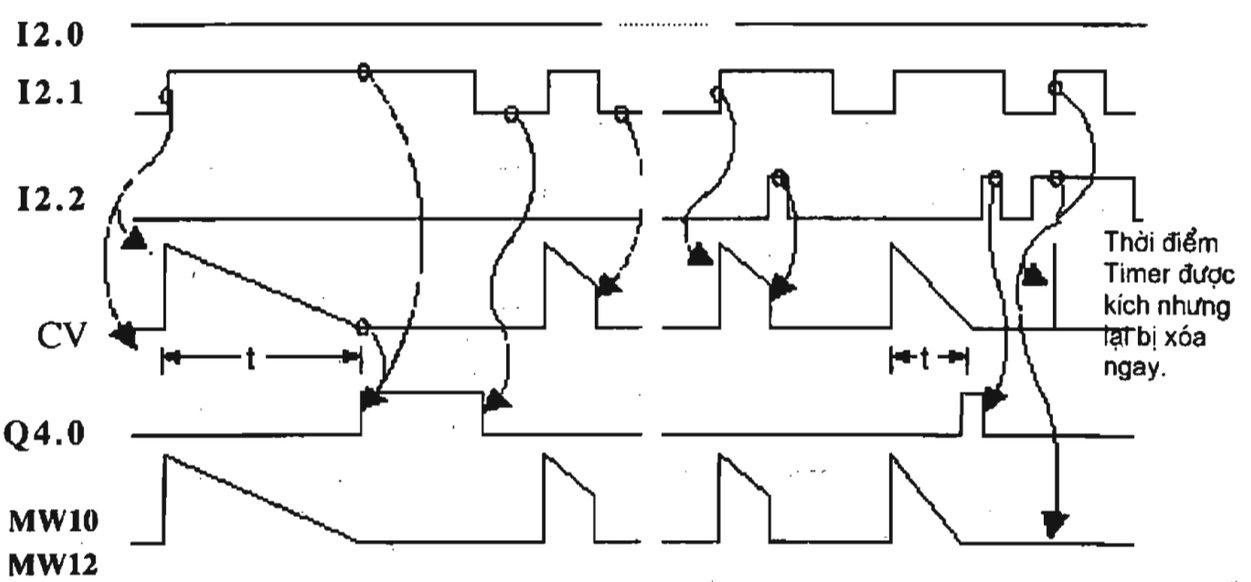
Toán hạng là tên Timer mà thanh ghi T–Word của nó sẽ được đọc vào ACCU1. Giá trị đọc được gồm hai phần: một số BCD xác định số đếm tức thời (không có thứ nguyên) và độ phân giải.



2.7.4 Ví dụ minh họa

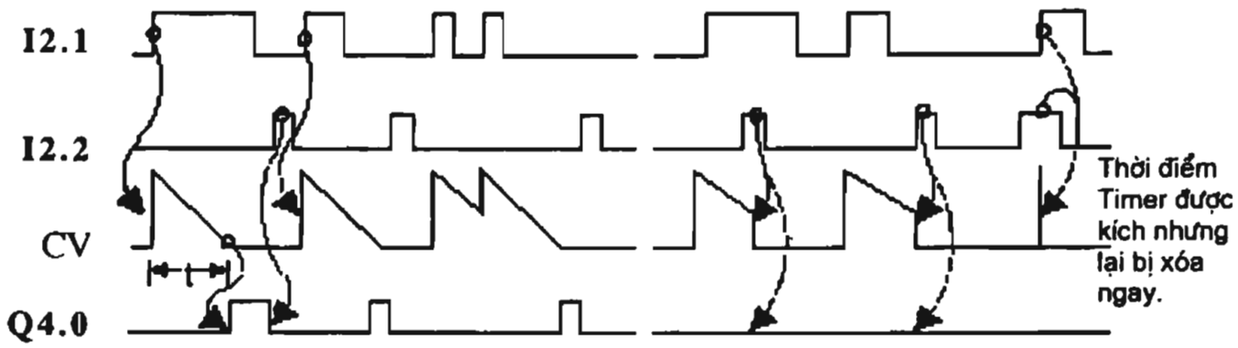
Ví dụ 1: Sử dụng Timer trễ theo sườn lên không có nhớ.

```
A     I 2.0
FR    T1        // Chủ động kích Timer T1.
A     I 2.1
L     S5T#10s   // Khai báo PV = 10 giây.
SD    T1        // Sử dụng T1 dạng on-delay.
A     I 2.2
R     T1        // Chủ động xóa Timer T1.
A     T1        // Đọc T1–Bit.
=     Q 4.0
L     T1        // Đọc CV của T1 dạng nguyên dương (nhị phân).
T     MW10
LC    T1        // Đọc CV của T1 dạng BCD.
T     MW12
```



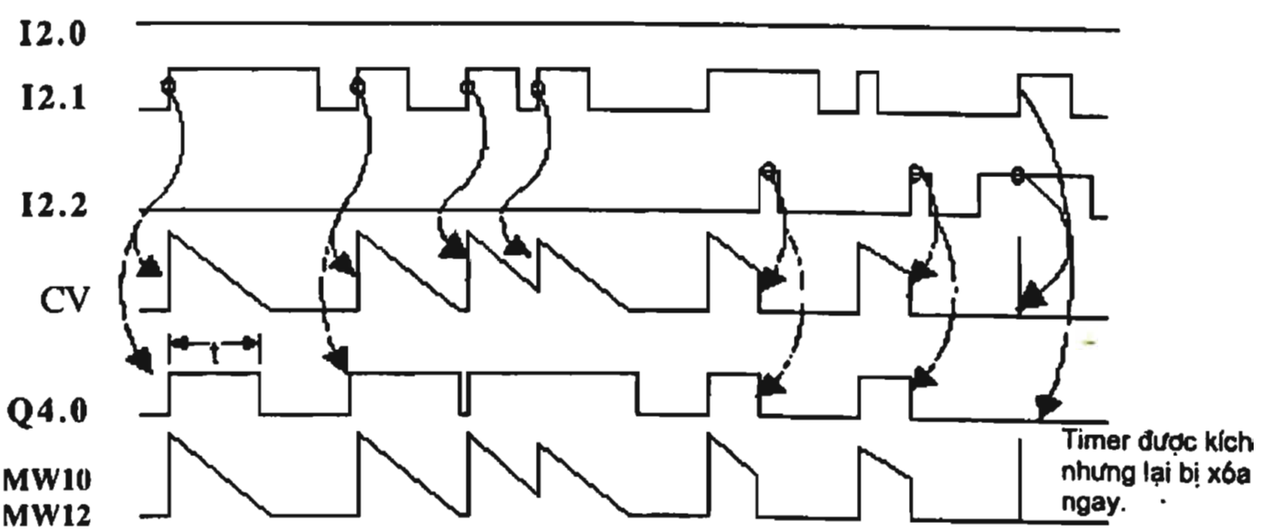
Ví dụ 2: Sử dụng Timer trễ theo sườn lên có nhớ.

```
A     I 2.1
L     S5T#10s   // Preset 10 seconds into ACCU 1.
SS    T1        // Start timer T1 as a retentive on-delay timer.
A     I 2.2
R     T1        // Reset timer T1.
A     T1        // Check signal state of timer T1.
=     Q 4.0
```



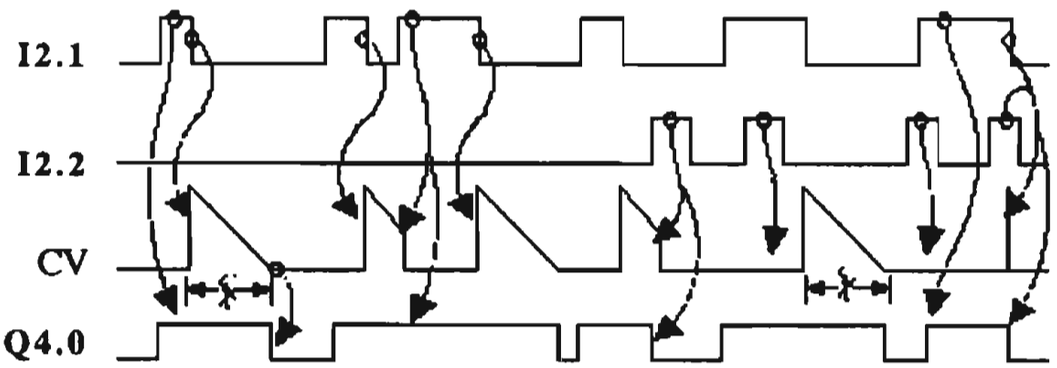
Ví dụ 3: Sử dụng Timer tạo xung có nhớ.

```
A     I 2.0
FR    T1        // Enable timer T1.
A     I 2.1
L     S5T#10s   // Preset 10 seconds into ACCU 1.
SE    T1        // Start timer T1 as an extended pulse timer.
A     I 2.2
R     T1        // Reset timer T1.
A     T1        //Check signal state of timer T1.
=     Q 4.0
L     T1        // Load current timer value of timer T1 as binary.
T     MW10
LC    T1        //Load current timer value of timer T1 as BCD.
T     MW12
```



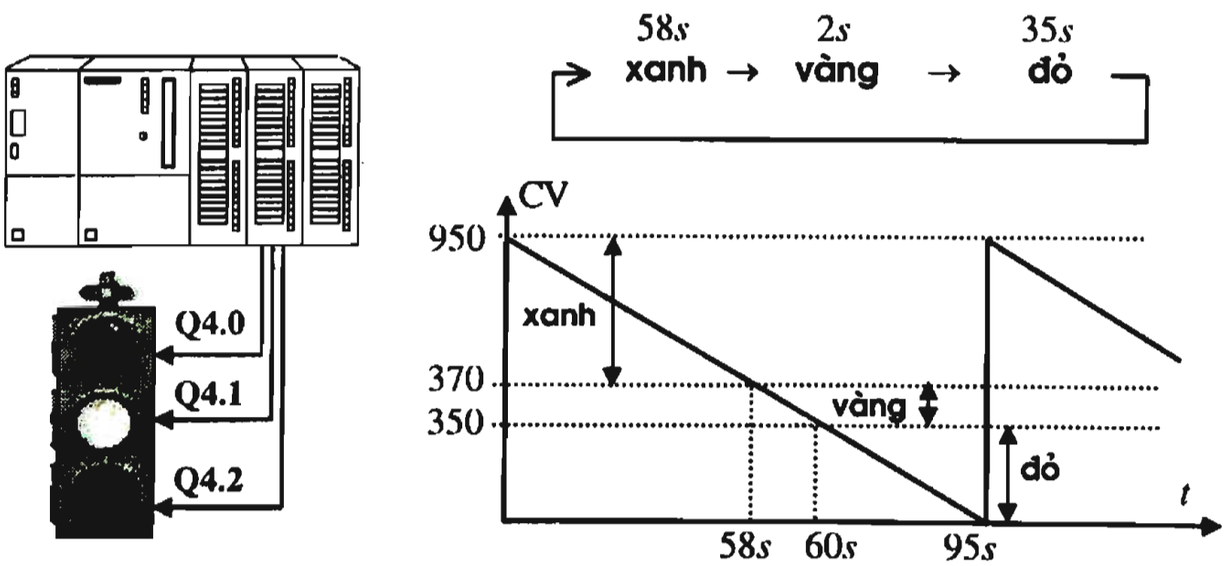
Ví dụ 4: Sử dụng Timer tạo trễ theo sườn xuống

```
A     I 2.1
L     S5T#10s   // Preset 10 seconds into ACCU 1.
SF    T1        // Start timer T1 as an off-delay timer.
A     I 2.2
R     T1        // Reset timer T1.
A     T1        // Check signal state of timer T1.
=     Q 4.0
L     T1        // Load current timer value of timer T1 as binary.
T     MW10
LC    T1        // Load current timer value of timer T1 as BCD.
T     MW12
```



Ví dụ 5:

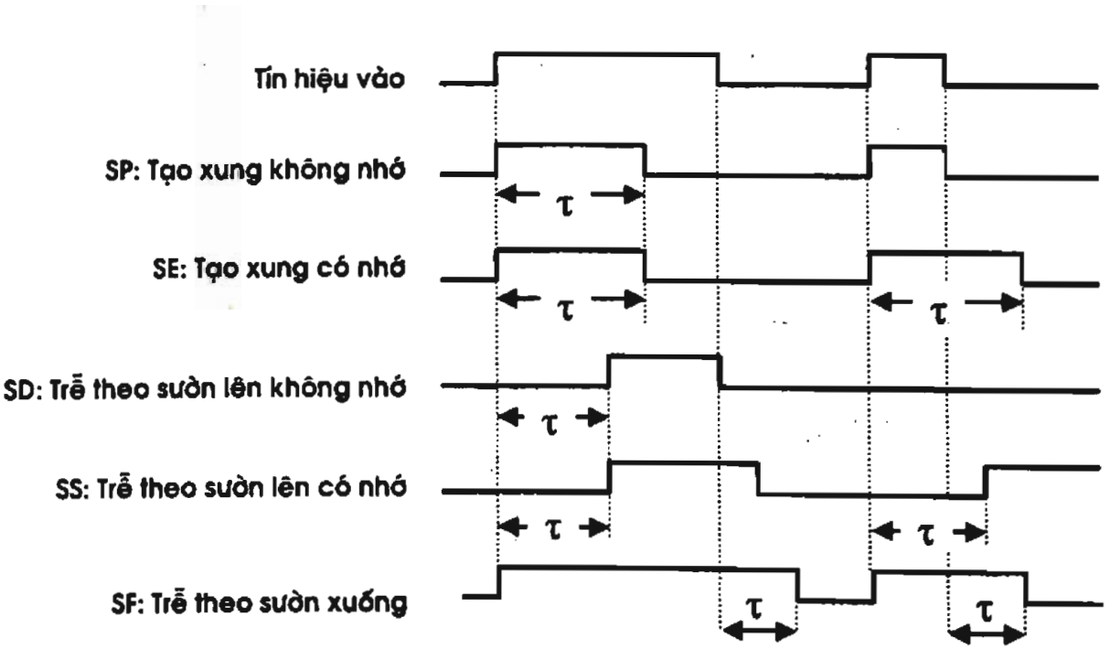
Điều khiển tự động đèn giao thông bằng S7–300. Đèn xanh được giữ trong khoảng thời gian 58*s* sau đó chuyển sang đèn vàng 2*s*, rồi đến đỏ 35*s* và quay lại đèn xanh 58*s* .... Chế độ điều khiển tự động này được tích cực khi tín hiệu I0.0 có giá trị 1.



```
      A     I 0.0          // Công tắc chuyển sang điều khiển tự động (I0.0=1)
      AN    T1             //Tạo chu kỳ 95s.
      L     W#16#1950      //Thời gian trễ là 100ms×950.
      SD    T1
      L     W#16#1370      // Đèn xanh ?.
      LC    T1             //Đọc giá trị đếm tức thời CV.
      <=I
      JC    xng
      L     W#16#1350      // Đèn vàng ?.
      >I
      JC    vng
      S     Q4.0           // Bật đèn đỏ và tắt đèn xanh, vàng.
      R     Q4.1
      R     Q4.2
      BEU
xng:  S     Q4.2           //Bật đèn xanh và tắt đèn đỏ, vàng.
      R     Q4.0
      R     Q4.1
      BEU
vng:  S     Q4.1           // Bật đèn vàng và tắt đèn xanh, đỏ.
      R     Q4.0
      R     Q4.2
      BEU
```

2.7.5 Tổng kết

Hình dưới tổng kết lại các loại Timer của S7-300 cho tiện việc tra cứu sử dụng, trong đó τ là thời gian trễ đặt trước.

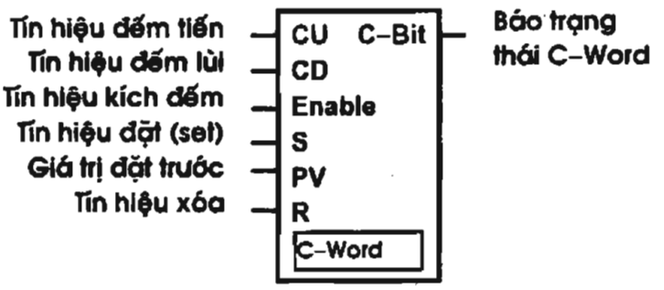


2.8 Bộ đếm (Counter)

2.8.1 Nguyên tắc làm việc

Counter là bộ đếm thực hiện chức năng đếm sườn xung của các tín hiệu đầu vào. S7–300 có tối đa 256 Counter (phụ thuộc từng CPU), ký hiệu bởi Cx, trong đó x là số nguyên trong khoảng từ 0 đến 255. Những bộ đếm của S7–300 đều có thể đồng thời đếm tiến theo sườn lên của một tín hiệu vào thứ nhất, được ký hiệu là CU (*count up*) và đếm lùi theo sườn lên của tín hiệu vào thứ hai, ký hiệu là CD (*count down*).

Thông thường bộ đếm chỉ đếm các sườn lên của tín hiệu CU và CD, song cũng có thể được mở rộng để đếm cả mức tín hiệu của chúng bằng cách sử dụng thêm tín hiệu enable (kích đếm). Nếu có tín hiệu enable, bộ đếm sẽ đếm tiến khi xuất hiện sườn lên của tín hiệu enable đồng thời tại thời điểm đó CU có mức tín hiệu là 1. Tương tự bộ đếm sẽ đếm lùi khi có sườn lên của tín hiện enable và tại thời điểm đó CD có mức tín hiệu 1.



Hình 2.16. Mô tả nguyên lý làm việc của bộ đếm.

Số sườn xung đếm được, được ghi vào thanh ghi 2 byte của bộ đếm, gọi là thanh ghi C–Word. Nội dung của C–Word được gọi là *giá trị đếm tức thời* của bộ đếm và ký hiệu bằng CV (*current value*). Bộ đếm báo trạng thái của C–Word ra ngoài qua chân C–Bit của nó. Nếu CV≠0, C–Bit có giá trị 1. Ngược lại khi CV=0, C–bit nhận giá trị logic 0. CV luôn là một giá trị không âm. Bộ đếm sẽ không đếm lùi khi CV=0.

Khác với Timer, giá trị đặt trước PV (*preset value*) của bộ đếm chỉ được chuyển vào C–Word tại thời điểm xuất hiện sườn lên của tín hiệu đặt (*set–S*).

Bộ đếm có thể được xóa chủ động bằng tín hiệu xóa (*reset*). Khi bộ đếm được xóa, cả C-word và C-bit đều nhận giá trị 0.

2.8.2 Khai báo sử dụng

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

- Khai báo tín hiệu enable nếu muốn sử dụng tín hiệu chủ động kích đếm.
- 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 xóa (reset).

trong đó ít nhất phải có một trong hai bước 2 hoặc 3 được thực hiện.

1) Khai báo tín hiệu kích đếm (enable)

```
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 kích đếm cho bộ đếm có tên cho trong toán hạng thứ hai. Tên của bộ đếm có dạng Cx với 0≤x≤255. Ví dụ

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    C1       // Sử dụng cho Counter có tên là C1.
```

Lệnh FR tác động vào thanh ghi trạng thái như sau:

BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
–	–	–	–	–	0	–	–	0

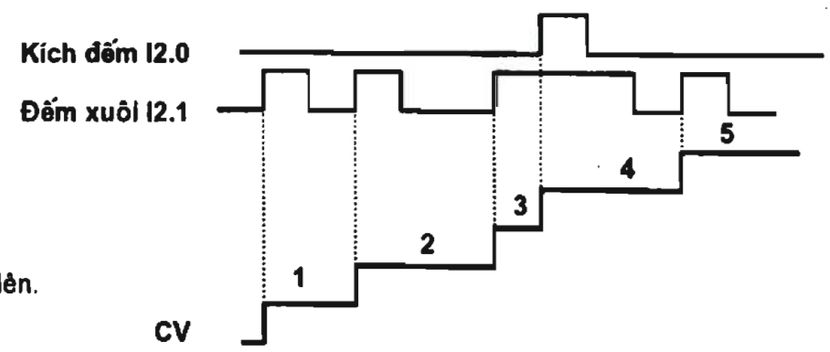
2) Khai báo tín hiệu được đếm tiến theo sườn lên

```
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 với 0≤x≤255. Mỗi khi xuất hiện một sườn lên của tín hiệu, bộ đếm sẽ tăng nội dung thanh ghi C–Word (CV) lên 1 đơn vị. Lệnh CU tác động vào thanh ghi trạng thái giống như lệnh FR .

Ví dụ

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    C1       // Sử dụng cho Counter có tên là C1.
A     I2.1     // Sườn lên của tín hiệu tại cổng vào I2.1 sẽ được C1 đếm tiến.
CU    C1
```



Hình 2.17: Bộ đếm tiến theo sườn lên.

3) Khai báo tín hiệu được đếm lùi theo sườn lên

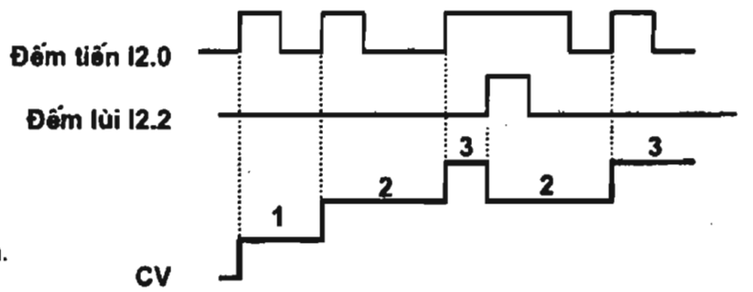
```
Cú pháp    A     <Địa chỉ bit>
           CD    <Tên Counter>
```

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 tiến. Tên của bộ đếm có dạng Cx với 0≤x≤255.

Mỗi khi xuất hiện sườn lên của tín hiệu, bộ đếm sẽ giảm nội dung thanh ghi C–Word (CV) đi 1 đơn vị nếu CV>0. Trong trường hợp CV đã bằng 0 thì nội dung C–Word không bị thay đổi. Lệnh CD tác động vào thanh ghi trạng thái giống như lệnh FR.

Ví dụ

```
A     I2.1     // Sườn lên của tín hiệu tại cổng vào I2.1 sẽ được C1 đếm tiến.
CU    C1
A     I2.2     // Sườn lên của tín hiệu tại cổng vào I2.2 sẽ được C1 đếm lùi.
CD    C1
```



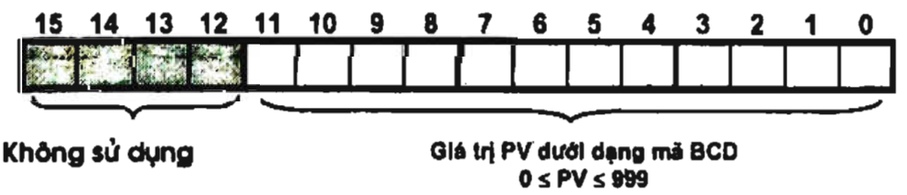
Hình 2.18: Bộ đếm tiến và lùi theo sườn lên.

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

```
Cú pháp   A    <Địa chỉ bit>
          L    C#<hằng số>
          S    <Tên Counter>
```

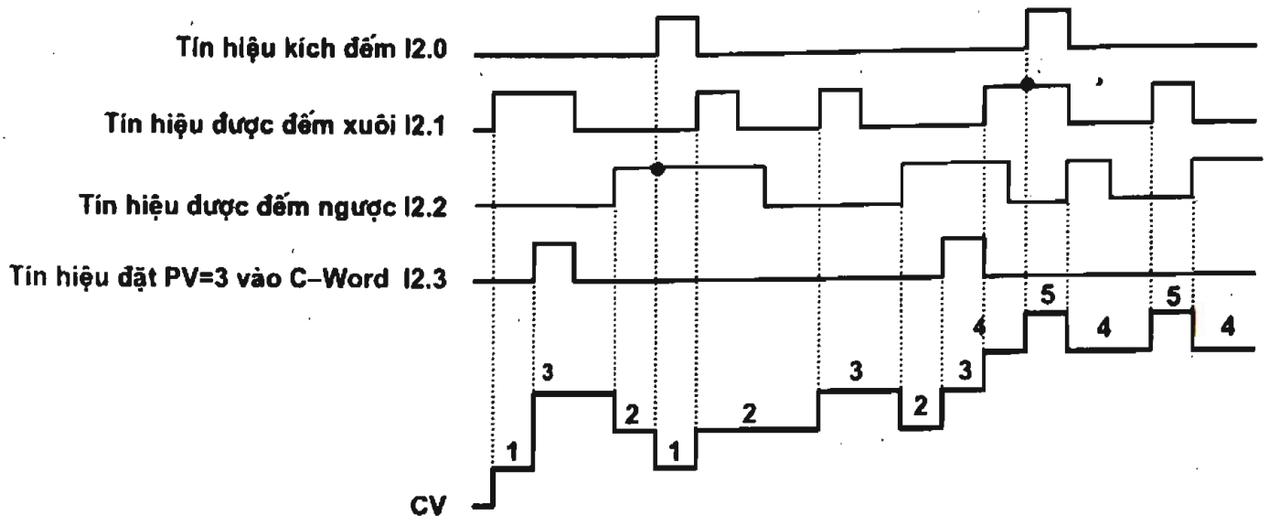
Toán hạng thứ nhất "Địa chỉ bit" xác định tín hiệu mà mỗi khi xuất hiện sườn lên của nó, hằng số PV cho trong lệnh thứ hai dưới dạng dạng BCD sẽ được chuyển vào thanh ghi C–Word của bộ đếm có tên trong toán hạng của lệnh thứ 3.

Tên của bộ đếm có dạng Cx với 0≤x≤255.



Ví dụ:

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    C1
A     I2.1     // Sườn lên của tín hiệu tại cổng vào I2.1 sẽ được C1 đếm tiến.
CU    C1
A     I2.2     // Sườn lên của tín hiệu tại cổng vào I2.2 sẽ được C1 đếm lùi.
CD    C1
A     I2.3     // Tín hiệu cổng vào I2.3 là tín hiệu đặt giá trị PV vào thanh ghi C-Wordi.
L     C#3
S     C1
```



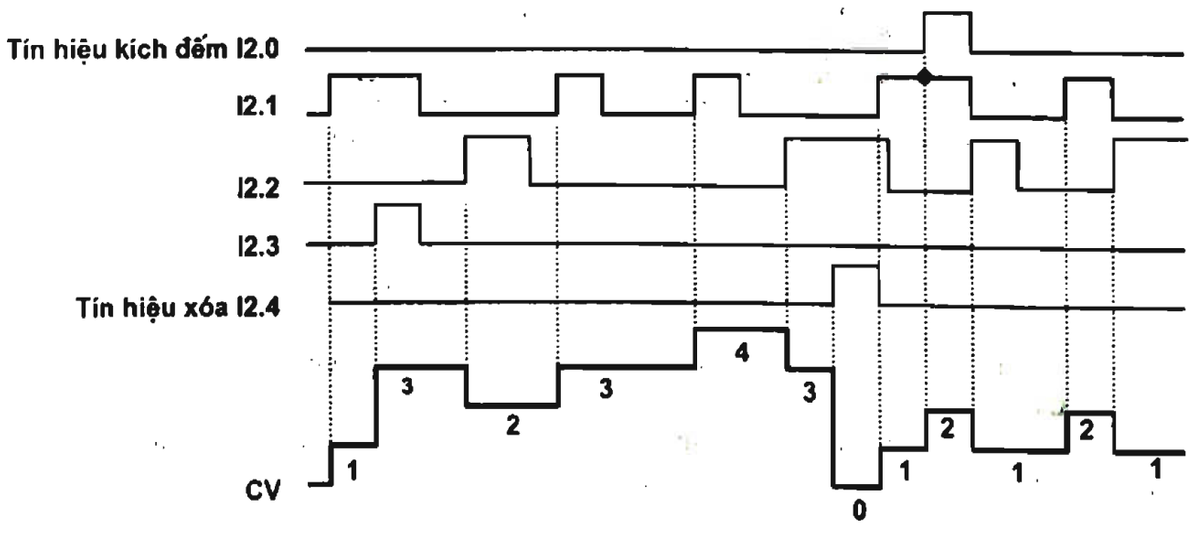
Hình 2.19: Tác dụng của tín hiệu đặt

5) Khai báo tín hiệu xóa (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 2 sẽ được xóa về 0. Tên của bộ đếm có dạng Cx với 0≤x≤255. Ví dụ

```
A     I2.0     // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    C1
A     I2.1     // Sườn lên của tín hiệu tại cổng vào I2.1 sẽ được C1 đếm tiến.
CU    C1
A     I2.2     // Sườn lên của tín hiệu tại cổng vào I2.2 sẽ được C1 đếm lùi.
CU    C1
A     I2.3     // Tín hiệu cổng vào I2.3 là tín hiệu đặt giá trị PV vào thanh ghi C–Wordi.
L     C#3
S     C1
A     I2.4     // Sườn lên của tín hiệu tại cổng vào I2.4 sẽ xóa C–Word của C1.
R     C1
```



Hình 2.20: Tác dụng của tín hiệu xóa

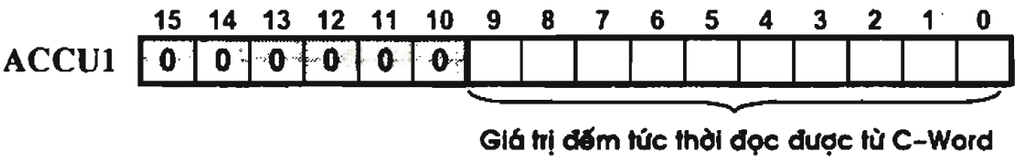
6) Đọc nội dung thanh ghi C–Word

Nội dung thanh ghi C–Word là CV, cũng giống như ở Timer, có thể được đọc vào ACCU1 theo hai cách:

a) Đọc số đếm tức thời dạng binary

Cú pháp L <Tên Counter>

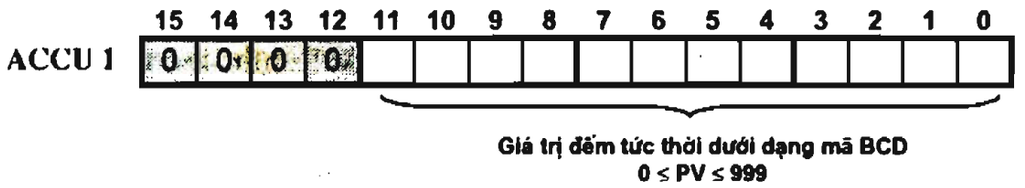
Toán hạng là tên bộ đếm mà thanh ghi C–Word của nó sẽ được đọc vào ACCU1. Giá trị đọc được là một số nguyên dương xác định số đếm tức thời. Tên của bộ đếm có dạng Cx với 0≤x≤255.



b) Đọc số đếm tức thời dạng BCD

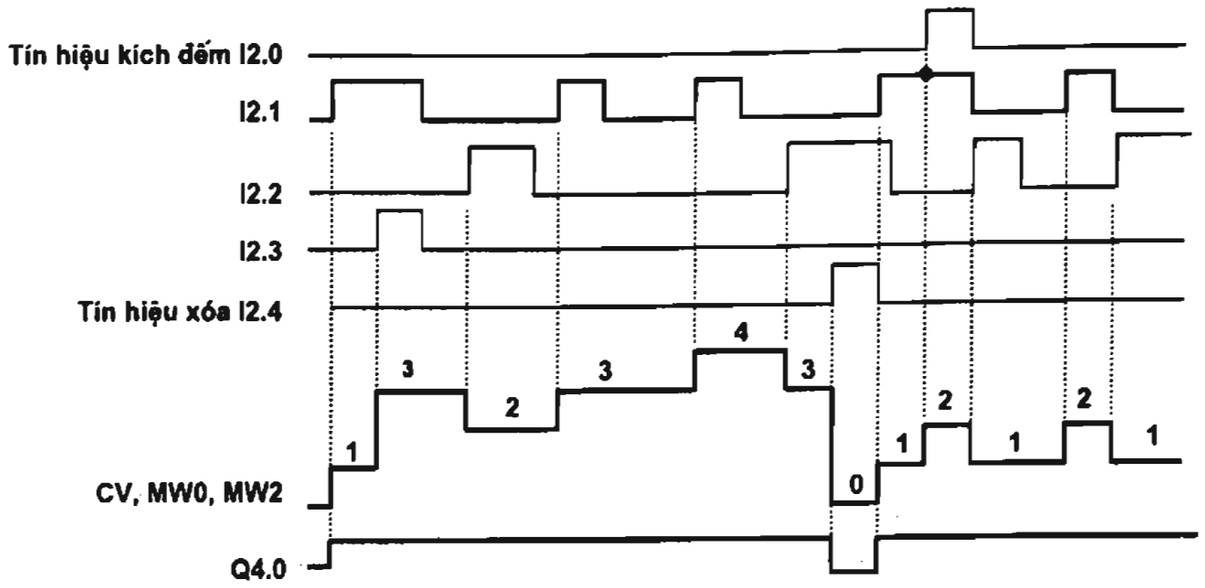
Cú pháp LC <Tên Counter>

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. Tên của bộ đếm có dạng Cx với 0≤x≤255.



2.8.3 Ví dụ minh họa

```
A     I2.0    // Tín hiệu tại cổng vào I2.0 sẽ được dùng làm tín hiệu enable.
FR    C1
A     I2.1    // Sườn lên của tín hiệu tại cổng vào I2.1 sẽ được C1 đếm tiến.
CU    C1
A     I2.2    // Sườn lên của tín hiệu tại cổng vào I2.2 sẽ được C1 đếm lùi.
CU    C1
A     I2.3    // Tín hiệu cổng vào I2.3 là tín hiệu đặt giá trị PV vào thanh ghi C-Wordi.
L     C#3
S     C1
A     I2.4    // Sườn lên của tín hiệu tại cổng vào I2.4 sẽ xóa C-Word của C1.
R     C1
LC    C1      // Đọc giá trị đếm tức thời dạng BCD.
T     MW0
L     C1      // Đọc giá trị đếm tức thời dạng binary.
T     MW2
A     C1      // Đọc C-Bit
=     Q4.0
```



Hình 2.21: Sử dụng bộ đếm C1.

2.9 Kỹ thuật sử dụng con trỏ

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 truy nhập gián tiếp. Ta hãy xét lệnh đọc nội dung ô nhớ MW0 vào ACCU1 làm ví dụ

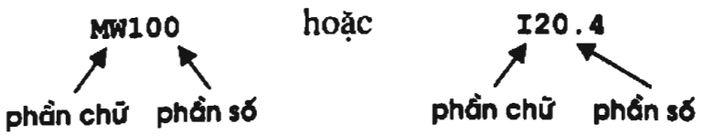
```
L     MW0        // Đọc giá trị của ô nhớ MW0 vào thanh ghi ACCU1.
```

Lệnh này là lệnh 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 lệnh 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 lệnh truy nhập gián tiếp nhờ con trỏ MD10 như sau

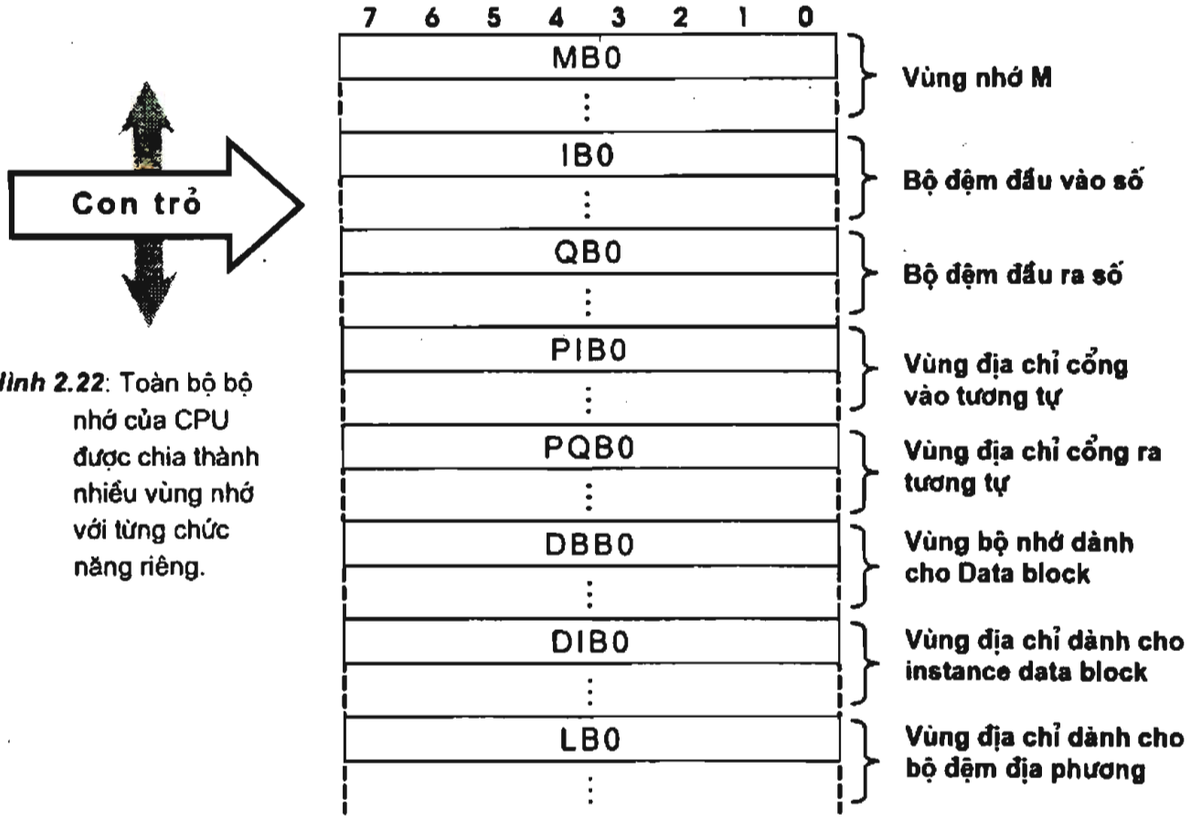
```
L     0
T     MD10
L     MW[MD10]   // Đọc giá trị của ô nhớ có địa chỉ cho trong MD10.
```

Như phần đầu chương đã mô tả, địa chỉ một ô nhớ trong S7–300 gồm hai phần: phần chữ và phần số. Ví dụ



trong đó phần chữ chỉ vị trí trong vùng, kích thước của ô nhớ và phần số chỉ địa chỉ của byte hoặc bit trong vùng nhớ đã xác định. Tương ứng với cách biểu diễn địa chỉ như vậy mà con trỏ cũng có hai dạng:

- Chỉ chứa phần số. Đây là kiểu con trỏ địa phương xác định vị trí ô nhớ trong vùng.
- Chứa cả phần số và chữ. Đây là con trỏ toàn cục xác định vị trí ô nhớ trong bộ nhớ.



Hình 2.22: Toàn bộ bộ nhớ của CPU được chia thành nhiều vùng nhớ với từng chức năng riêng.

2.9.1 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, ví dụ 20, 22, 100, ...
- địa chỉ bit, ví dụ 20.0, 22.2, 100.5, ...

nên con trỏ địa phương MW, MD cũng có hai hình thái

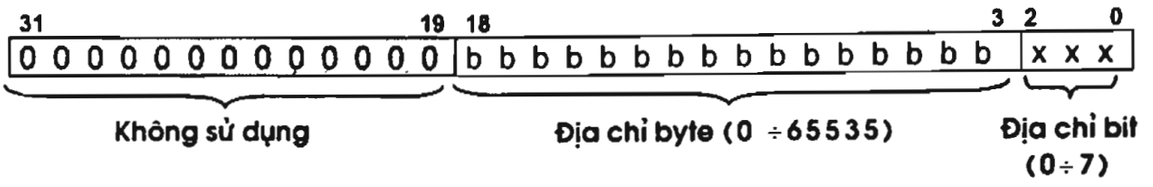
- con trỏ địa phương chỉ vị trí byte trong vùng,
- và con trỏ địa phương chỉ vị trí bit trong vùng.

1) **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 một byte (từ, từ kép hay môt dãy các bytes) thì nội dung của con trỏ là địa chỉ byte đầu tiên trong dãy các byte đó.

   Ví dụ:

```
L     20
T     MD10          // Sử dụng MD10 làm con trỏ
L     DIB[MD10]     // Đọc giá trị byte nhớ DIB20
T     MW[MD10]      // Chuyển vào MW20
```

2) **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 byte đó theo cấu trúc:



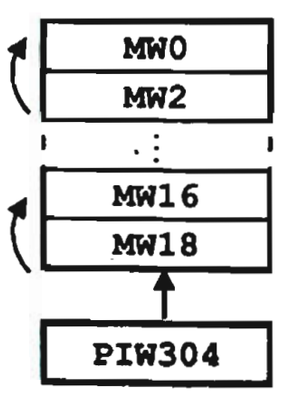
Cấu trúc dữ liệu này của con trỏ chỉ địa chỉ bit được khai báo trong S7–300 bằng toán hạng dạng:

```
P#<Địa chỉ byte> . <số thứ tự bit>
```

Ví dụ:

Chương trình sau nhập dữ liệu từ cổng tương tự PIW304 và cất vào bộ đệm. Bộ đệm là vùng nhớ gồm 10 từ MW0 ÷ MW18. Dữ liệu vừa đọc được sẽ được cất vào ô cuối cùng trong bộ đệm. Các dữ liệu đã có trong bộ đệm được chuyển dần lên. Dữ liệu đầu tiên trong bộ đệm bị đẩy ra khỏi bộ đệm. Chương trình sử dụng hai con trỏ là MD20 và MD24. Byte MB28 chứa số đếm.

```
      L     P#0.0
      T     MD20          // Địa chỉ ô nhớ đầu tiên
      L     9
next: T     MB28          // Chỉ số đếm.
      L     MD20
      L     P#2.0
      +D                  // Địa chỉ ô nhớ phía dưới
      T     MD24
      L     MW[MD24]
      T     MW[MD20]      // Chuyển giá trị.
      L     MD24
      T     MD20          // Chuyển con trỏ.
      L     MB28
      LOOP  next
      L     PIW304
      T     MW18
```



Hình 2.23: Bộ đệm chứa giá trị đọc được.

2.9.2 Sử dụng thanh ghi con trỏ AR1 và AR2

S7–300 có hai thanh ghi 32 bits 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 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 làm hai trường hợp:

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

1) **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>]`

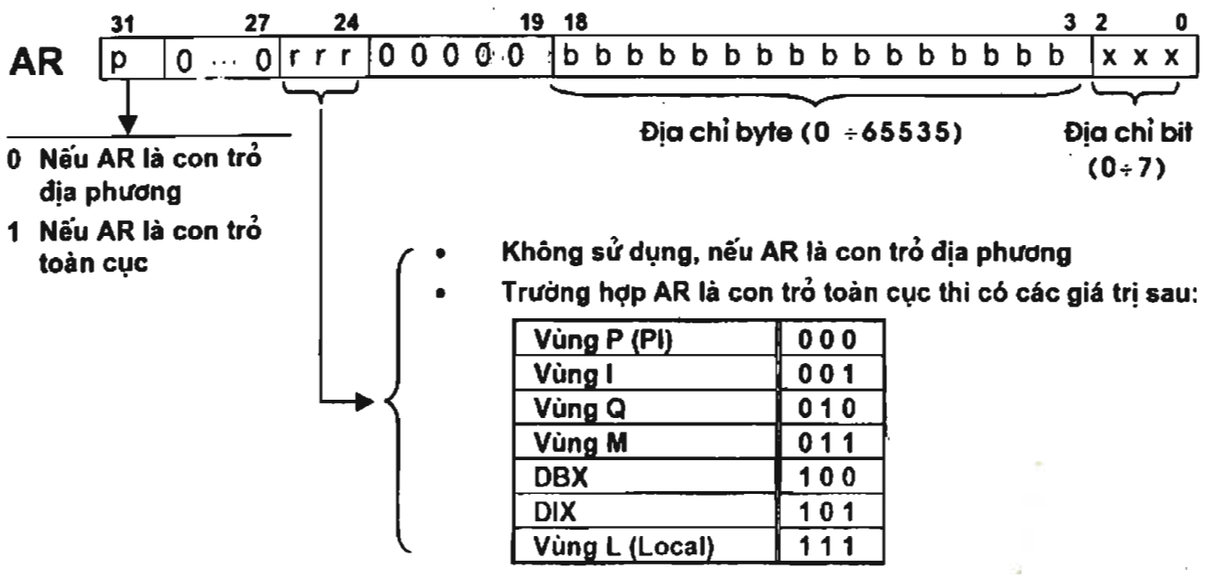
Toán hạng của lệnh gán giá trị có cấu trúc

`P#[<Tên vùng bộ nhớ>]<Địa chỉ byte>.<số thứ tự bit>`

Có thể có hoặc không. Nếu không có, con trỏ AR sẽ là con trỏ địa phương. Nếu có (M,I,Q,P, DBX hoặc DIX) thì AR sẽ là con trỏ toàn cục.

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:

Hình 2.24: Mô tả khái niệm Offset của con trỏ.

```
<Tên lệnh>  <Vùng và kích thước>[ARx,P#<Số bytes> . <Số bit>]
                                  \_____ Offset _____/
```

Trừ trường hợp truy nhập bit (`A,O,=, ...`), trong lệnh phải ghi rõ kích thước mảng bits 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 được tới 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]   // Chuyểngiá trị đọc được vào byte MB10
L      W[AR2,P#5.0]   // Đọc MW15
T      MW[AR1,P#2.0]  // Chuyểnvào MW3
L      DBD[AR1,P#9.0] // Đọc DBD10
T      D[AR2,P#20.0]  // Chuyển vào MD30
```

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 sửa 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 (hình 2.23). 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          // Chuyển con trỏ AR1 xuống ô nhớ tiếp theo
      L      MB24
      LOOP   next
      L      PIW304
      T      MW18
```

2) **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>]
          +AR2  [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
```

3) **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 có hoặc không có toán hạng. Trong trường hợp không có toán hạng, lệnh sẽ chuyển nội dung của thanh ghi AR vào ACCU1. Nếu có toán hạng, lệnh sẽ chuyển nội dung của thanh ghi AR vào từ kép có địa chỉ được chỉ thị trong toán hạng. Lệnh không làm thay đổi nội dung thanh ghi trạng thái. Ví dụ

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

4) **Đảo nội dung hai thanh ghi AR:**

```
Cú pháp:  CAR
```

Lệnh không có toán hạng và 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.. Lệnh không làm thay đổi nội dung thanh ghi trạng thái.

2.10 Khai báo và sử dụng khối dữ liệu (DB)

S7–300 có vùng M được sử dụng làm các ô nhớ lưu giữ 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 phép khai báo nhiều khối DB cùng một lúc (tối

đa là 65535), được phân biệt với nhau nhờ chỉ số khối như DB1, DB2, DB3 ..., 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 CPU314). Mọi khối DB đều có thể được truy nhập tới từng bit.

2.10.1 Khai báo một khối dữ liệu

Khối dữ liệu (DB) được khai báo nhờ phần mềm soạn thảo Step7. Việc hướng dẫn sử dụng phần mềm Step7 sẽ được trình bày chi tiết sau trong chương 4. Ở đây chỉ giới thiệu các bước khai báo một khối DB. Hình 2.25 minh họa một khối dữ liệu có tên là DB1 đã được khai báo bằng 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 ký tự), ARRAY (mảng dữ liệu) ....
- Đặt giá trị ban đầu cho biến (có thể bỏ qua).
- Chú thích (có thể bỏ qua).

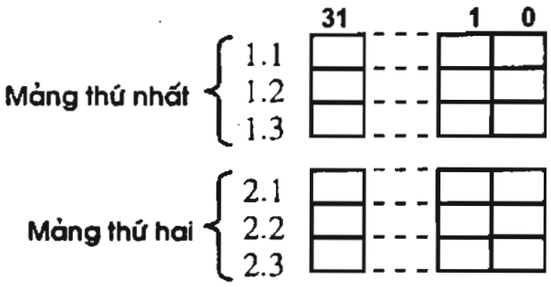
Phần mềm Step7 sẽ tự xác định địa chỉ đầu cho từng biến cũng như kích thước của toàn bộ khối dữ liệu. Trong ví dụ ở hình 2.24 về khối DB1 ta nhận ra được là khối DB1 có kích thước 46 bytes.

Address	Name	Type	Initial Value	Comment
0.0		STRUCT		
+0.0	Name1	BOOL	FALSE	
+0.1	Name2	BOOL	FALSE	
+1.0	Name3	BYTE	B#16#0	
+2.0	Name4	WORD	W#16#0	
+4.0	Name5	DWORD	DW#16#0	
+8.0	Name6	INT	0	
+10.0	Name7	REAL	0.000000e+000	
+14.0	Name8	STRING[10]	'Siemens'	
+26.0	Name9	ARRAY[1..10]		
*2.0		INT		
=46.0		END_STRUCT		

Hình 2.24: Một ví dụ về khai báo khối dữ liệu.

1) Kiểu biến **STRING**: Đây là biến có dạng một dãy ký tự. Dãy ký tự có độ dài tính theo byte là số cho trong dấu ngoặc vuông. Kích thước của biến bằng độ dài của dãy ký tự cộng thêm 2 bytes chứa mã kết thúc chuỗi ký tự đó. Trong ví dụ ở hình 2.25, biến với tên Name8 là một biến kiểu **STRING** có kích thước 12 bytes, trong đó 10 bytes dùng để chứa dãy ký tự.

2) 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 của chiều thứ nhất và y là độ dài của chiều thức hai. Tương tự, mảng 3 chiều cũng được khai báo bằng **ARRAY[1..x,1..y,1..z]**. Ở ví dụ minh họa hình 2.24, biến có tên **Name9** là mảng một chiều gồm 10 phần tử kiểu **INT**. Do mỗi phần tử kiểu **INT** chiếm 2 bytes nên toàn bộ biến **Name9** có kích thước là 10×2=20 bytes. Hình 2.25 mô tả sự phân chia ô nhớ cho một mảng 2 chiều **ARRAY[1..2,1..3]** của các phần tử kiểu **REAL** (4 bytes) và mảng này có kích thước là 2×3×4=24 bytes.

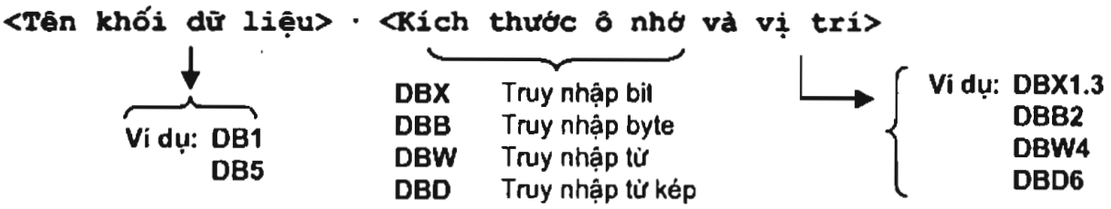


Hình 2.25: Mảng hai chiều ARRAY[1..2,1..3]

2.10.2 Truy nhập và quản lý khối dữ liệu

1) 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:



Cách truy nhập như trên còn được 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

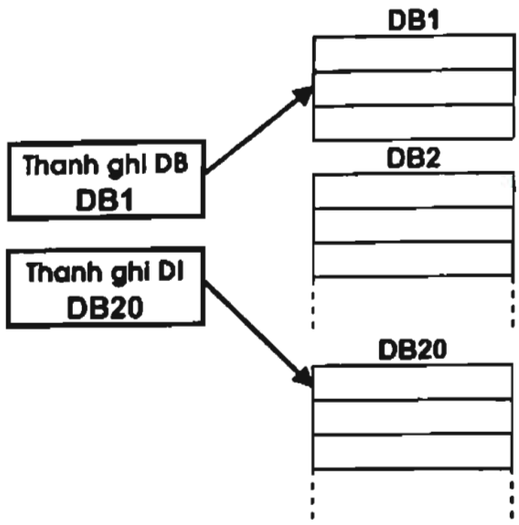
```
A     DB1.DBX1.5   // Đọc nội dung bit thứ 5 thuộc byte thứ 0 của khối DB1
A     DB5.DBX2.3   // Thực hiện phép ∧ với giá trị của bit thứ 3, byte 2 của khối DB5
=     DB10.DBX2.4  // Chuyển vào bit thứ 4 byte 2 của khối DB10.
```

2) 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 một trong hai thanh ghi chỉ khối dữ liệu (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 DI<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* (hình 2.26).



Hình 2.26: Truy nhập gần thông qua hai thanh ghi chỉ khối dữ liệu.

Lệnh mở khối dữ liệu không làm thay đổi nội dung thanh ghi trạng thái.

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 (Chuyển tên khối DB20 vào thanh ghi DI).
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          // Địa chỉ ô nhớ đầu tiên
L     W[AR1,P#0.0]
T     W[AR1,P#2.0]
```

Cũng ở chế độ truy nhập gần ta còn sử dụng được các lệnh quản lý:

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 ACCU1 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 bytes 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 thanh ghi trạng thái. Nội dung cũ của ACCU1 được chuyển vào ACCU2.

Ví dụ: Nếu độ dài của DB2 lớn hơn DB3 thì báo sáng đèn Q4.0

```
OPN    DB2
OPN    DI3
L      DBLG      // Độ dài tính theo byte của DB2
L      DILG      // Độ dài tính theo byte của DB3
>D
=      Q4.0
```

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

2.10.3 Ví dụ minh họa về truy nhập khối dữ liệu

Giả sử có khối dữ liệu DB2 chứa dãy số thực x_i , i=0,1, ... với độ dài không biết trước, trong đó x_0 nằm trong DB2.DBD0, x_1 trong DB2.DBD4, ... (mỗi số thực chiếm 4 bytes). Chương trình sau sẽ tìm x_i từ chỉ số i cho trước theo các bước:

- Trước khi tìm x_i cần phải kiểm tra xem chỉ số i cho trước đó có vượt ra ngoài dãy hay không. Nếu i nằm ngoài dãy đã cho thì dương cờ báo lỗi Q4.0.
- Dùng thanh ghi con trỏ AR1 chỉ vào ô nhớ chứa x_i . Để làm được việc này ta cho AR1 chỉ vào ô nhớ đầu tiên của mảng, sau đó từ chỉ số i xác định ra khoảng cách (tính theo byte) ô nhớ chứa x_i so với byte đầu tiên của mảng và suy ra offset theo công thức: offset = $4i$ (byte) × 8 (bit)

```
OPN    DB2
LAR1   p#DBX0.0          // Địa chỉ byte đầu tiên trong DB2
L      DBLG              // Đọc độ dài khối dữ liệu DB2
L      MW0               // i
SLW    2                 // 4i
>=I
JCN    end               // Chỉ số nằm ngoài DB2
SLW    3                 // Offset của con trỏ
+AR1                     // Con trỏ ô nhớ chứa x_i trong DB2
```

```
      L     D[AR1,p#0.0]      // Đọc nội dung ô nhớ (x_i)
      T     MD2
      SET                     // Xóa cờ báo lỗi
      R     Q4.0
      BEU
end:  SET                     // Cờ báo lỗi
      S     Q4.0
      BEU
```

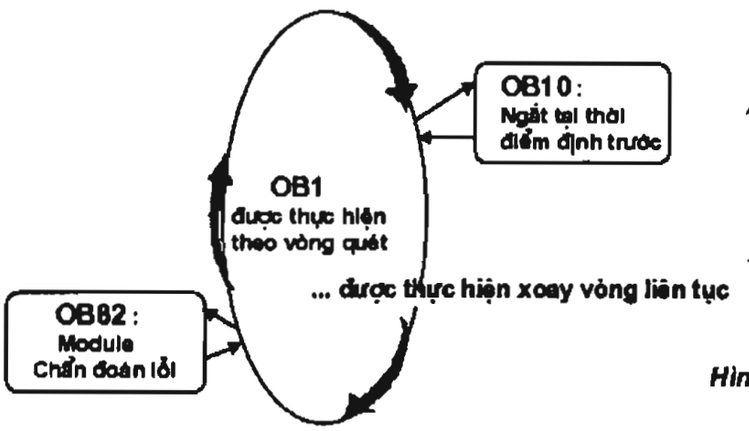
3 KỸ THUẬT LẬP TRÌNH

3.1 Giới thiệu chung

3.1.1 Lập trình tuyến tính và lập trình có cấu trúc

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

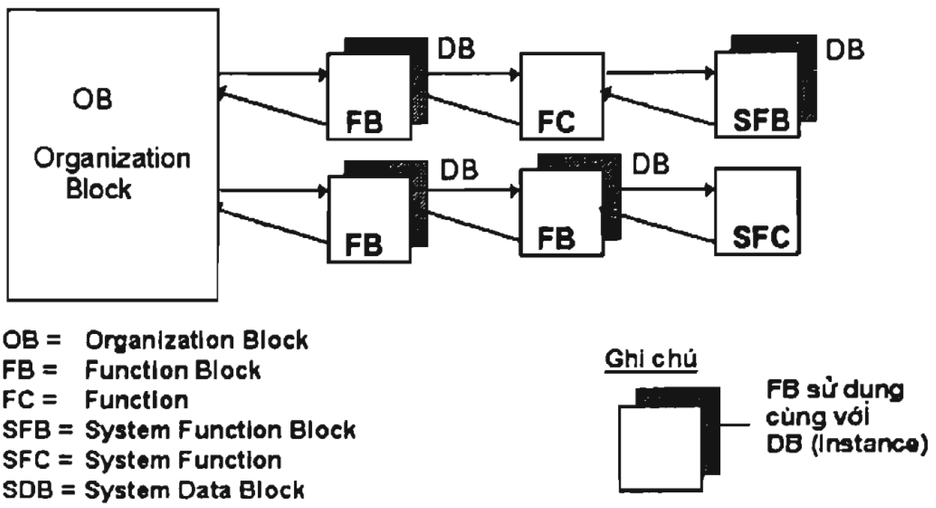
Với hình thức tổ chức như vậy thì phần chương trình trong khối OB1 có đầy đủ điều kiện của một chương trình điều khiển thời gian thực và toàn bộ chương trình ứng dụng có thể chỉ cần được viết trong OB1 là đủ (hình 3.1). Cách tổ chức chương trình với chỉ một khối OB1 duy nhất như vậy được gọi là *lập trình tuyến tính* (linear programming).



Hình 3.1. Khối OB1 được hệ thống gọi xoay vòng liên tục theo vòng quét.

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

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



Hình 3.2. Cấu trúc một chương trình (có cấu trúc).

Khác với kiểu lập trình tuyến tính, *kỹ thuật lập trình có cấu trúc* (structure programming) là phương pháp lập trình mà ở đó toàn bộ chương trình điều khiển được chia nhỏ thành các khối FC hay FB mang một nhiệm vụ cụ thể riêng và được quản lý chung từ những khối OB (hình 3.2). Kiểu lập trình này rất phù hợp cho bài toán điều khiển phức tạp, nhiều nhiệm vụ cũng như cho việc sửa chữa, rõ rối sau này.

3.1.2 Tổ chức bộ nhớ CPU

Hình 3.3 trình bày tổng quan về cách phân chia bộ nhớ cho các vùng nhớ khác nhau, bao gồm:

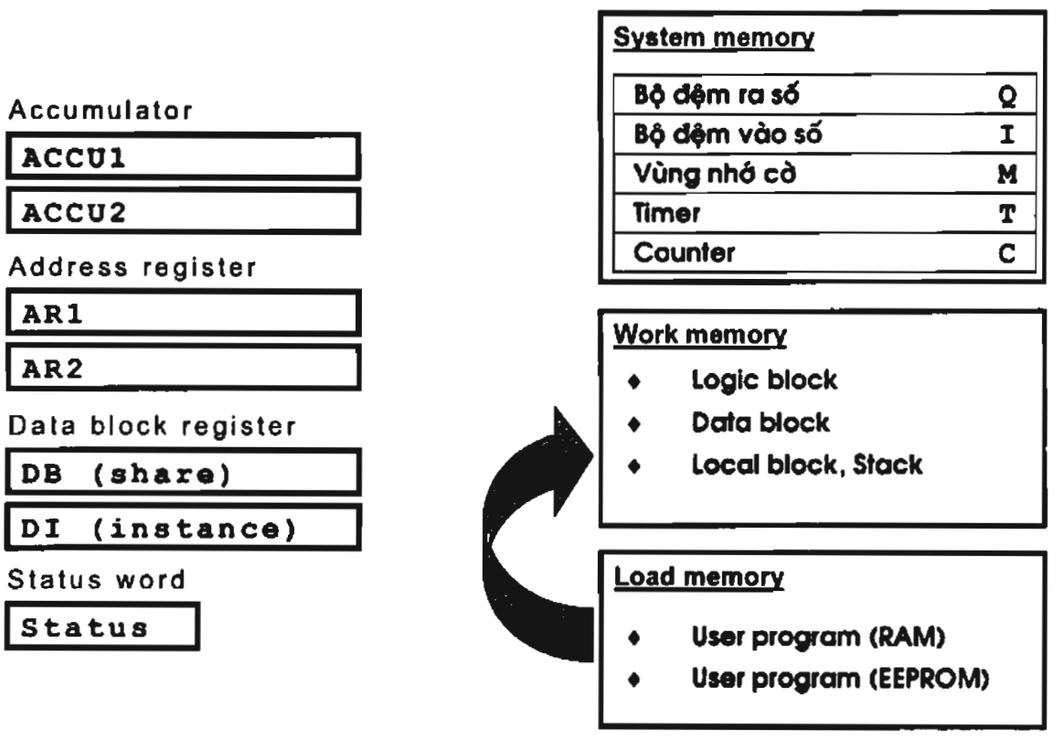
- vùng nhớ chứa các thanh ghi,
- vùng System memory,
- vùng Load memory,
- vùng Work memory.

Kích thước của các vùng nhớ này phụ thuộc vào chủng loại của từng module CPU.

Load memory: Là vùng nhớ chứa chương trình ứng dụng (do người sử dụng viết) bao gồm tất cả các khối chương trình ứng dụng OB, FC, FB, các khối chương trình trong thư viện hệ thống được sử dụng (SFC, SFB) và các khối dữ liệu DB. Vùng nhớ này được tạo bởi một phần bộ nhớ RAM của CPU và EEPROM (nếu có EEPROM). Khi thực hiện

động tác xóa bộ nhớ (MRES) toàn bộ các khối chương trình và khối dữ liệu nằm trong RAM sẽ bị xóa. Cũng như vậy, khi chương trình hay khối dữ liệu được đổ (down load), từ thiết bị lập trình (PG, máy tính) vào module CPU, chúng sẽ được ghi lên phần RAM của vùng nhớ Load memory.

Work memory: Là vùng nhớ chứa các khối DB đang được mở, khối chương trình (OB, FC, FB, SFC hoặc SFB) đang được CPU thực hiện và phân bộ nhớ cấp phát cho những tham số hình thức để các khối chương trình này trao đổi tham trị với hệ điều hành và với các khối chương trình khác (local block). Tại một thời điểm nhất định vùng Work memory chỉ chứa một khối chương trình. Sau khi khối chương trình đó được thực hiện xong thì hệ điều hành sẽ xóa nó khỏi Work memory và nạp vào đó khối chương trình kế tiếp đến lượt được thực hiện.



Hình 3.3. Phân chia các vùng ô nhớ trong CPU.

System memory: Là vùng nhớ chứa các bộ đệm vào/ra số (Q, I) , các biến cờ (M), thanh ghi C–Word, PV, T–bit của Timer, thanh ghi C–Word, PV, C–bit của Counter. Việc truy cập, sửa đổi dữ liệu những ô nhớ thuộc vùng nhớ này được phân chia hoặc bởi hệ điều hành của CPU hoặc do chương trình ứng dụng.

Có thể thấy rằng trong các vùng nhớ được trình bày trên không có vùng nhớ nào được dùng làm bộ đệm cho các cổng vào/ra tương tự. Nói cách khác các cổng vào/ra tương tự không có bộ đệm và như vậy mỗi lệnh truy nhập module tương tự (đọc hoặc gửi giá trị) đều có tác dụng trực tiếp tới cổng vật lý của module.

Bảng sau trình bày chi tiết hơn về ý nghĩa các vùng nhớ (kích thước phụ thuộc vào chủng loại CPU)

Tên gọi	Kích thước truy nhập	Kích thước tối đa (phụ thuộc CPU)	Ý nghĩa
Process-image input (I) **Bộ đệm vào số.**	I IB IW ID	0.0 ÷ 127.7 0 ÷ 127 0 ÷ 126 0 ÷ 124	Đầu mỗi vòng quét, hệ điều hành sẽ ghi vào phần nhớ này các giá trị được lấy từ cổng vào số (digitale inputs) vật lý của module mở rộng.
Process-image output (Q) **Bộ đệm ra số.**	Q QB QW QD	0.0 ÷ 127.7 0 ÷ 127 0 ÷ 126 0 ÷ 124	Cuối mỗi vòng quét, hệ điều hành sẽ đọc nội dung của miền nhớ này và chuyển ra cổng ra số (digitale output) của các module mở rộng.
Bit memory (M) **Vùng nhớ cờ.**	M MB MW MD	0.0 ÷ 255.7 0 ÷ 255 0 ÷ 254 0 ÷ 252	Được sử dụng như một miền các biến cờ cho chương trình ứng dụng.
Timer (T)	T0 ÷ T255		Miền nhớ lưu giữ các giá trị PV, CV và T-Bit của Timer. Được truy nhập để sửa đổi bởi hệ điều hành và chương trình ứng dụng.
Counter (C)	C0 ÷ C255		Miền nhớ lưu giữ các giá trị PV, CV và C-Bit của Counter. Được truy nhập để sửa đổi bởi hệ điều hành và chương trình ứng dụng.
Data block (DB) **Khối dữ liệu share.**	DBX DBB DBW DBD	0.0 ÷ 65535.7 0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Được mở bằng lệnh **"OPN DB"**
Data block (DI) **Khối dữ liệu instance.**	DIX DIB DIW DID	0.0 ÷ 65535.7 0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Là khối DB nhưng được mở bằng lệnh **"OPN DI"**
Local block (L) Miền nhớ địa phương cho các tham số hình thức.	L LB LW LD	0.0 ÷ 65535.7 0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Miền nhớ được cấp phát cho các khối **OB, FC, FB** mỗi khi khối này được gọi để thực hiện. Miền nhớ này cũng sẽ được giải phóng khi thực hiện xong các khối chương trình đó.
Peripheral input (PI)	PIB PIW PID	0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Chỉ có địa chỉ truy cập để đọc. Không có phần bộ nhớ thực sự.
Peripheral output (PQ)	PQB PQW PQD	0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Chỉ có địa chỉ truy cập để ghi. Không có phần bộ nhớ thực sự.

Trừ phần bộ nhớ EEPROM thuộc vùng Load memory và một phần RAM tự nuôi đặc biệt (non-volatile) dùng để lưu giữ tham số cấu hình trạm PLC như địa chỉ trạm (MPI address), tên các module mở rộng, tất cả các phần bộ nhớ còn lại ở chế độ mặc định không có khả năng tự nhớ (non-retentive). Khi mất nguồn nuôi hoặc khi thực hiện công

việc xóa bộ nhớ (MRES), toàn bộ nội dung của phần bộ nhớ non–retentive sẽ bị mất. Tuy nhiên ta có thể sử dụng phần mềm Step7 để chuyển những khối DB chứa những dữ liệu quan trọng, cũng như các dữ liệu của Timer, Counter vào phần bộ nhớ RAM tự nuôi khi mất điện (gọi là phần non–volatile hay retentive).

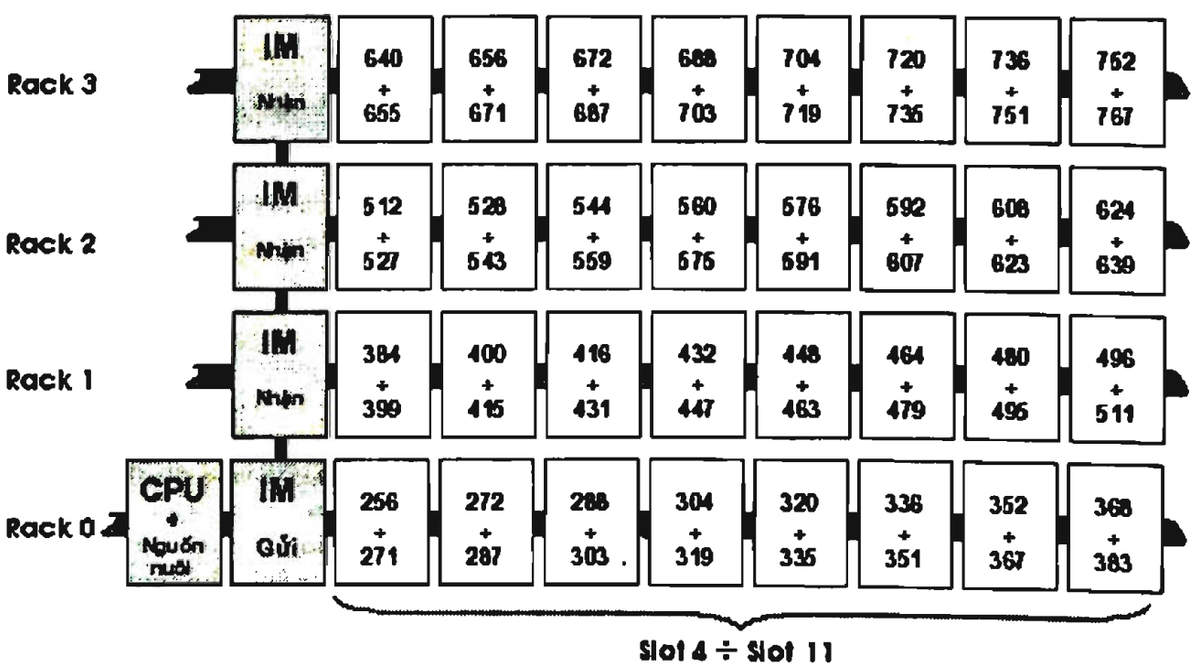
Bảng sau trình bày những dữ liệu có thể được chuyển vào phần bộ nhớ non–volatile của CPU314 nhờ Step7.

Dữ liệu thuộc miền	Phần có thể chuyển	Mặc định của Step7
Vùng nhớ cờ (M)	0 ÷ 256 (byte)	16 (số các bytes)
Timer	0 ÷ 128	0 (số các Timer)
Counter	0 ÷ 64	8 (số các Counter)
Các khối DB	0 ÷ 127	1. Có thể quy định từng phần chứ không cần phải toàn bộ khối DB.

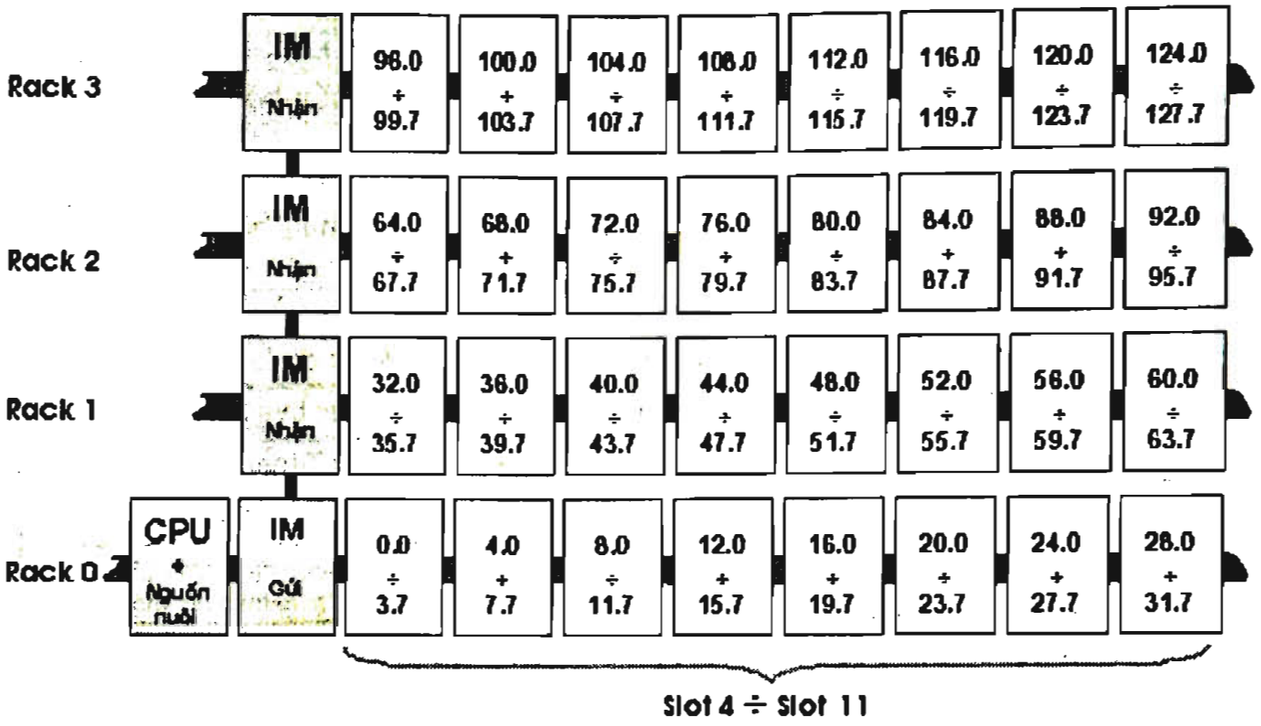
3.1.3 Xác định địa chỉ cho module mở rộng

Một trạm PLC được hiểu là một module CPU ghép nối cùng với các module mở rộng khác (module DI, DO, AI, AO, CP, FM) trên những thanh rack (giá đỡ), trong đó việc truy nhập của CPU vào các module mở rộng được thực hiện thông qua địa chỉ của chúng. Một module CPU có khả năng quản lý được 4 thanh rack với tối đa 8 module mở rộng trên mỗi thanh.

Tùy vào vị trí lắp đặt của module mở rộng trên những thanh rack mà các module có những địa chỉ khác nhau. Hình 3.4 và 3.5 trình bày quy tắc xác định địa chỉ cho module mở rộng phụ thuộc vào vị trí lắp đặt của nó.



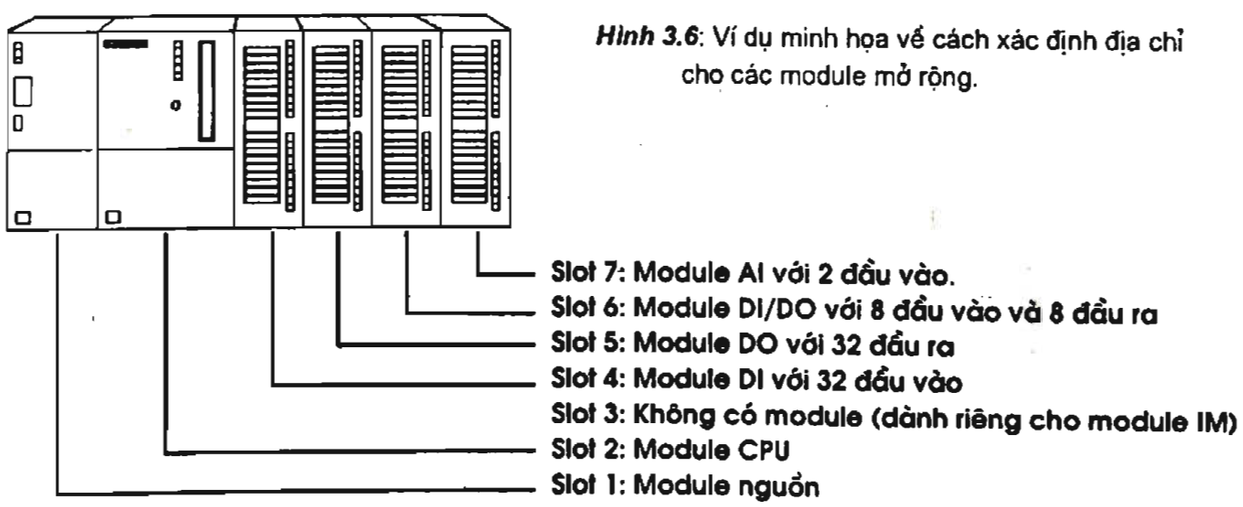
Hình 3.4. Quy tắc xác định địa chỉ cho các module tương tự.



Hình 3.5: Quy tắc xác định địa chỉ cho các module số.

Để minh họa cho việc xác định địa chỉ, ta lấy một ví dụ về cấu hình cứng của trạm PLC cho trong hình 3.6. Vậy thì:

- Slot 4 có module DI với 32 đầu vào số. Địa chỉ các đầu vào này sẽ là I0.0 ÷ I3.7.
- Slot 5 có module DO với 32 đầu ra số. Địa chỉ các đầu ra này sẽ là Q4.0 ÷ Q7.7.
- Slot 6 có module DI/DO với 8 đầu vào số và 8 đầu ra số. Địa chỉ các đầu vào/ra này là I8.0 ÷ I8.7 và Q8.0 ÷ Q8.7.
- Slot 7 có module AI với 2 đầu vào tương tự. Địa chỉ các đầu vào này là PIW304 và PIW306.



Hình 3.6: Ví dụ minh họa về cách xác định địa chỉ cho các module mở rộng.

3.1.4 Trao đổi dữ liệu giữa CPU và các module mở rộng

Trong trạm PLC luôn có sự trao đổi dữ liệu giữa CPU với các module mở rộng thông qua bus nội bộ. Ngay tại đầu vòng quét, các dữ liệu tại cổng vào của các module số (DI) đã được CPU chuyển tới bộ đệm vào số (process image input table – I). Cuối mỗi vòng quét nội dung của bộ đệm ra số (process image output table – Q) lại được CPU chuyển tới cổng ra của các module ra số (DO). Việc thay đổi nội dung hai bộ đệm này được thực hiện bởi chương trình ứng dụng (user program). Điều này cho thấy nếu trong chương trình ứng dụng có nhiều lệnh đọc giá trị cổng vào số thì cho dù giá trị logic thực có của cổng vào này có thể đã bị thay đổi trong quá trình thực hiện vòng quét, chương trình sẽ vẫn luôn đọc được cùng một giá trị từ I và giá trị đó chính là giá trị của cổng vào có tại thời điểm đầu vòng quét. Cũng như vậy, nếu chương trình ứng dụng nhiều lần thay đổi giá trị cho một cổng ra số thì do nó chỉ thay đổi nội dung bit nhớ tương ứng trong Q nên chỉ có giá trị ở lần thay đổi cuối cùng mới thực sự được đưa tới cổng ra vật lý của module DO.

Khác hẳn với việc đọc/ghi cổng số, việc truy nhập cổng vào/ra tương tự lại được CPU thực hiện trực tiếp với module mở rộng (AI/AO). Như vậy mỗi lệnh đọc giá trị từ địa chỉ thuộc vùng PI (peripheral input) sẽ thu được một giá trị đúng bằng giá trị thực có ở cổng tại thời điểm thực hiện lệnh. Tương tự khi thực hiện lệnh gửi một giá trị (số nguyên 16 bits) tới địa chỉ của vùng PQ (peripheral output), giá trị đó sẽ được gửi ngay tới cổng ra tương tự của module.

Sở dĩ có sự khác nhau như vậy là do đặc thù về sự tổ chức bộ nhớ và phân chia địa chỉ của S7-300. Chỉ có các module vào/ra số mới có bộ đệm còn các module vào/ra tương tự thì không, chúng chỉ được cung cấp địa chỉ để truy nhập (địa chỉ PI và PQ).

Tuy nhiên miền địa chỉ PI và PQ lại được cung cấp nhiều hơn là số các cổng vào ra tương tự có thể có của một trạm. Chẳng hạn, thực chất các cổng vào tương tự chỉ có thể có là từ địa chỉ PIB256 đến địa chỉ PIB767 nhưng miền địa chỉ của PI và PQ lại là từ 0 đến 65535. Điều này tạo ra khả năng kết nối các cổng vào/ra số với những địa chỉ dôi ra đó trong PI/PQ giúp chương trình ứng dụng có thể truy nhập trực tiếp các module DI/DO mở rộng để có được giá trị tức thời tại cổng mà không cần thông qua bộ đệm I và Q. Ví dụ ta có thể thay lệnh đọc đồng thời 8 cổng vào số thông qua bộ đệm I

```
L     IB0
```

bằng lệnh đọc trực tiếp từ module DI

```
L     PIB0
```

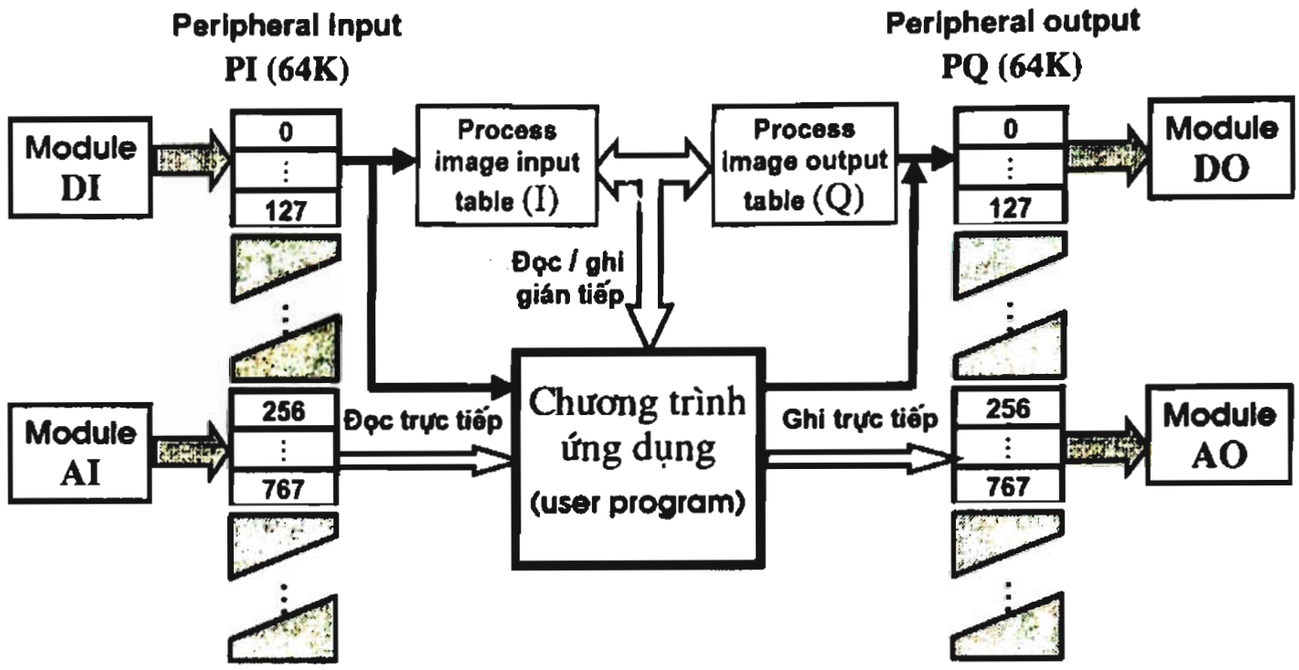
Hoặc lệnh ghi ra 16 cổng ra số thông qua bộ đệm Q

```
T     QW4
```

có thể được thay bằng lệnh ghi trực tiếp tới module DO

```
T     PQW4
```

Khả năng kết nối trực tiếp chương trình ứng dụng với module DI/DO mở rộng được trình bày trong hình 3.7.

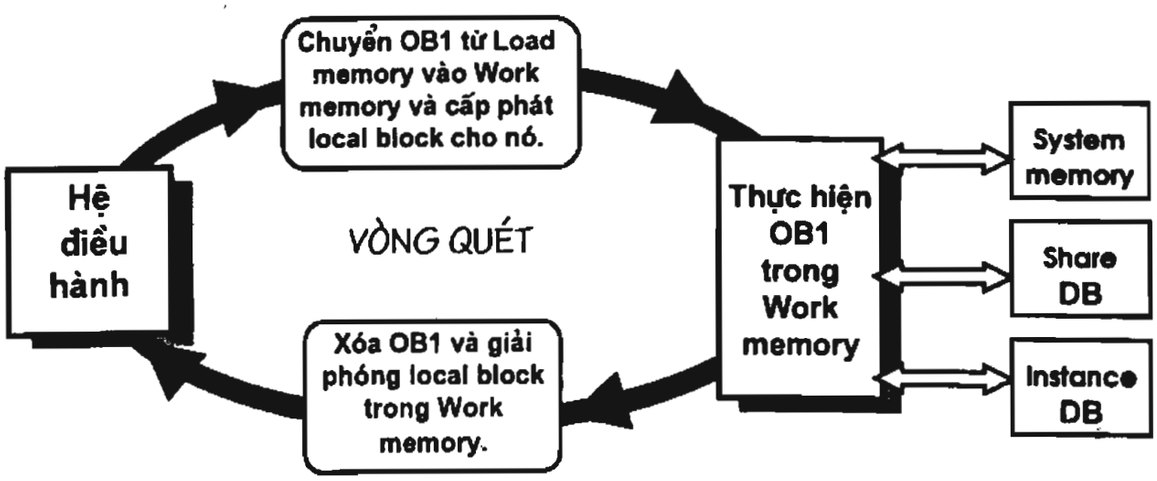


Hình 3.7. Nguyên lý trao đổi dữ liệu giữa CPU và các module mở rộng.

3.2 Lập trình tuyến tính

Kỹ thuật lập trình tuyến tính là phương pháp lập trình mà toàn bộ chương trình ứng dụng sẽ chỉ nằm trong một khối OB1. Kỹ thuật này có ưu điểm là gọn, rất phù hợp với những bài toán điều khiển đơn giản, ít nhiệm vụ.

Do toàn bộ chương trình điều khiển chỉ nằm trong khối OB1 nên khối OB1 sẽ gần như là được thường trực trong vùng nhớ Work memory, trừ trường hợp khi hệ thống phải phải xử lý các tín hiệu báo ngắt. Ngoài khối OB1, trong vùng Work memory còn có miền nhớ địa phương (local block) cấp phát cho OB1 và những khối DB được OB1 sử dụng. Hình dưới mô tả quy trình thực hiện chương trình điều khiển tuyến tính.



Hình 3.8. Thực hiện một chương trình tuyến tính.

3.2.1 Local block của OB1

Khi thực hiện khối OB1, hệ điều hành luôn cấp một local block có kích thước mặc định là 20 bytes trong Work memory để OB1 có thể lấy được những dữ liệu từ hệ điều hành. Các dữ liệu này gồm:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB1_EV_CLASS	Byte	Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1 = Vòng quét đầu, 3 = Từ vòng quét thứ 2
OB1_PRIORITY	Byte	Mức ưu tiên 1 (Mức ưu tiên thấp nhất)
OB1_OB_NUMBR	Byte	1 = Chỉ số của khối OB
OB1_RESERVED_1	Byte	Dự trữ (của hệ điều hành)
OB1_RESERVED_2	Byte	Dự trữ (của hệ điều hành)
OB1_PREV_CYCLE	Int	Thời gian vòng quét trước (milliseconds)
OB1_MIN_CYCLE	Int	Thời gian vòng quét ngắn nhất đã có (milliseconds)
OB1_MAX_CYCLE	Int	Thời gian vòng quét lớn nhất đã có (milliseconds)
OB1_DATE_TIME	Date_And_Time	Thời điểm OB1 bắt đầu được thực hiện.

Mặc dù kích thước chỉ là 20 bytes mặc định, nhưng người sử dụng có thể mở rộng local block để sử dụng thêm các biến nhớ cho chương trình (hình dưới). Tuy nhiên phải để ý rằng do local block được giải phóng ở cuối mỗi vòng quét và được cấp lại ở đầu vòng quét sau nên các giá trị có trong local block của vòng quét trước cũng bị mất khi bắt đầu vòng quét mới. Do đó, tốt nhất chỉ nên sử dụng local block cho việc lưu giữ các biến nháp tạm thời trong tính toán của một vòng quét.

Address	Decl.	Name	Type	
0.0	temp	OB1_EV_CLASS	BYTE	Người sử dụng không được thay đổi các biến này.
1.0	temp	OB1_SCAN_1	BYTE	
2.0	temp	OB1_PRIORITY	BYTE	
3.0	temp	OB1_OB_NUMBR	BYTE	
4.0	temp	OB1_RESERVED_1	BYTE	
5.0	temp	OB1_RESERVED_2	BYTE	
6.0	temp	OB1_PREV_CYCLE	INT	
8.0	temp	OB1_MIN_CYCLE	INT	
10.0	temp	OB1_MAX_CYCLE	INT	
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME	
20.0	temp	Temp1	DINT	Phần người sử dụng định nghĩa thêm để sử dụng.
24.0	temp	Temp2	DWORD	
28.0	temp	Temp3	INT	
30.0	temp	Temp4	BYTE	
31.0	temp	Temp5	BOOL	
31.1	temp	Temp6	BOOL	

Còn lại, cách sử dụng local block cũng không khác gì như sử dụng vùng biến cờ M (Bit memory). Chẳng hạn, để đọc khoảng thời gian thực hiện vòng quét trước đã được hệ điều hành chuyển vào ô nhớ 2 bytes gồm byte 6 và byte 7 trong local block dưới dạng số nguyên 16 bits, ta dùng lệnh

```
L     LW6        // Đọc nội dung 2 bytes kể từ địa chỉ 6 của local block vào ACCU1
```

Bên cạnh việc truy nhập theo địa chỉ ô nhớ như đã làm, ta còn có thể sử dụng tên biến hình thức **OB1_PREV_CYCLE** đã có của ô nhớ **LW6** như sau

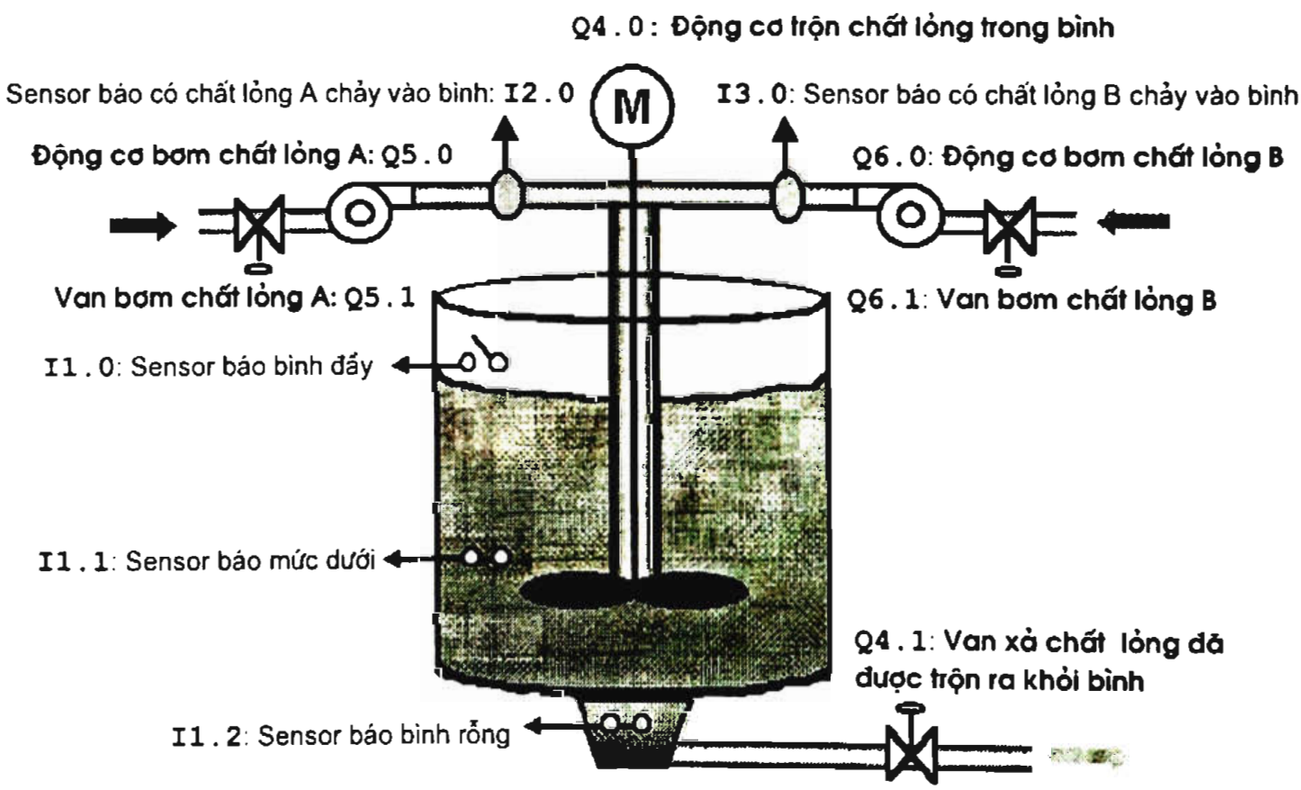
```
L     #OB1_PREV_CYCLE
```

Ví dụ, chương trình sau viết cho OB1 sẽ báo sáng đèn Q4.0 nếu đã có một vòng quét được thực hiện lâu hơn 20ms và đã có một vòng quét thực hiện nhanh hơn 15ms.

```
L     #OB1_MAX_CYCLE
L     20
>I
L     #OB1_MIN_CYCLE
L     15
-I
A     <0
S     Q4.0
```

3.2.2 Điều khiển bình trộn

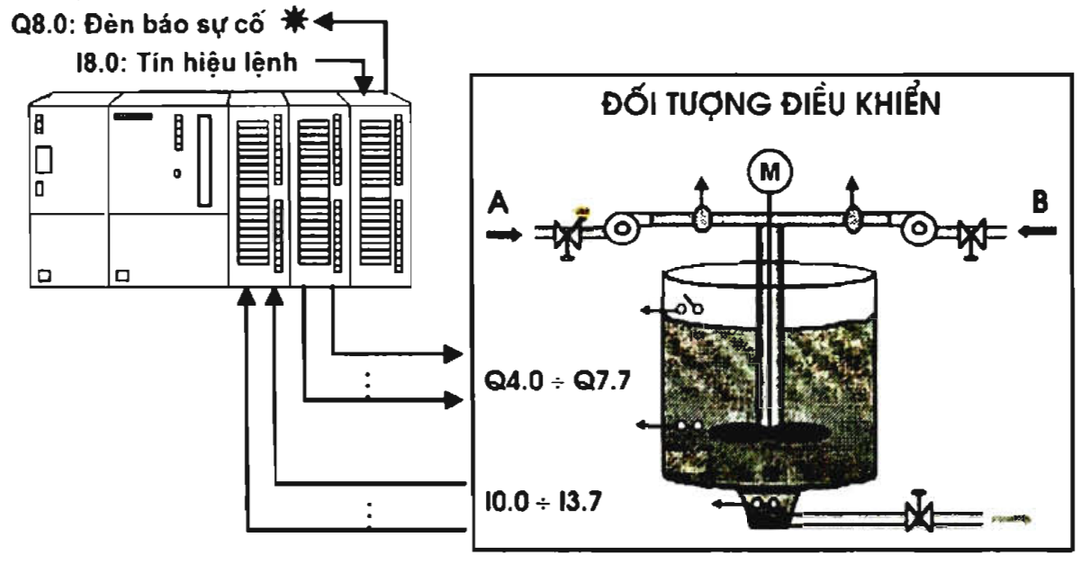
Để minh họa cho việc thiết kế một chương trình điều khiển tuyến tính, ta xét bài toán điều khiển bình trộn dạng đơn giản. Hình 3.9 mô tả bình trộn hai chất lỏng A và B.



Hình 3.9. Bài toán điều khiển bình trộn.

Các địa chỉ của tín hiệu sensors, điều khiển van và bơm trong hình 3.9 được xác định trên cơ sở cấu hình cứng đã chọn cho trạm PLC S7-300 gồm (hình 3.10):

- một module nguồn nuôi 5A,
- một module CPU314,
- một module DI 32 bits,
- một module DO 32 bits và
- một module DI/DO 8 vào/8 ra.



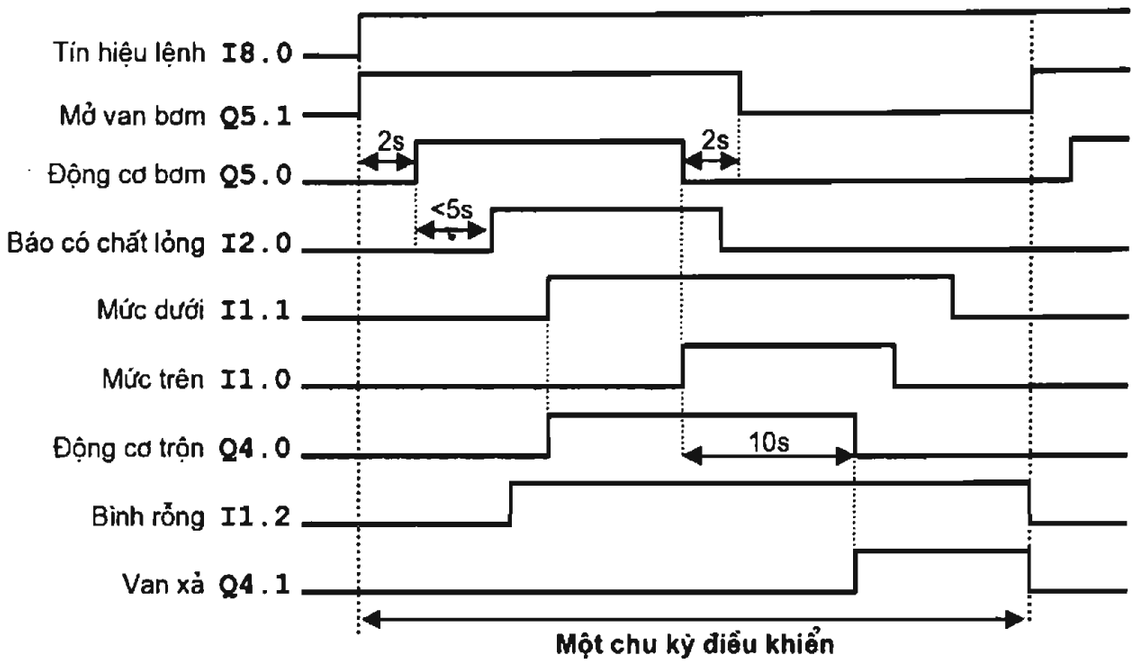
Hình 3.10. Cấu hình trạm PLC điều khiển bình trộn.

Bình trộn được điều khiển theo quy trình như sau:

1) Quá trình trộn chất lỏng tự động chỉ được thực hiện khi có tín hiệu lệnh I8.0=1.
2) Hai chất lỏng A và B cùng được bơm vào một bình để trộn nhờ các động cơ bơm. Động cơ bơm chỉ được khởi động (Q5.0=1, Q6.0=1) sau khi đã mở van (Q5.1=1, Q6.1=1) được 2 giây.
3) Nếu sau khi đã khởi động động cơ bơm được 5 giây mà vẫn không có tín hiệu báo đã có chất lỏng chảy vào bình (I2.0=1, I3.0=1) thì dừng tất cả các quá trình lại, đồng thời thông báo sự cố ra ngoài bằng đèn báo sự cố bơm (Q8.0).
4) Khi bình đã được bơm đầy (I1.0=1) thì dừng cả hai động cơ bơm. Quá trình dừng bơm được thực hiện theo thứ tự dừng máy bơm trước (Q5.0=0, Q6.0=0), sau đó 2 giây thì khóa van bơm (Q5.1=0, Q6.1=0).
5) Chất lỏng trong bình được quấy trộn đều bằng động cơ trộn (Q4.0=1). Quá trình trộn được bắt đầu khi đã có tín hiệu báo trong bình đủ chất lỏng (I1.1=1) và kết thúc sau 10 giây khi có tín hiệu báo bình đã đầy (I1.0=1).
6) Sau khi đã được trộn đều thì chất lỏng được tháo ra khỏi bình nhờ van xả (Q4.1=1). Chất lỏng trong bình được xả ra ngoài cho tới khi có tín hiệu báo đã xả hết chất lỏng trong bình (I1.2=0).

7) Khi đã xả xong và vẫn còn tín hiệu lệnh (I8.0=1) thì quay lại bước 2. Nếu trong quá trình thực hiện việc trộn tự động (từ bước 2 đến bước 6) mà tín hiệu lệnh không còn nữa (I8.0=0) thì vẫn thực hiện tiếp cho tới cuối chu kỳ trộn, tức là khi xong bước 6, rồi mới dừng máy.

Hình dưới biểu diễn giản đồ thời gian các tín hiệu **I8.0**, **Q5.1**, **Q5.0**, **I2.0**, **I1.1**, **I1.0**, **I1.2**, **Q4.0**, **Q4.1**, cho việc điều khiển bơm chất lỏng A, trộn và xả hỗn hợp trong một chu kỳ trộn. Giản đồ quy trình điều khiển bơm chất lỏng B cũng có dạng tương tự.




```
Network 1    // Nếu có sự cố máy bơm thì ngừng chương trình.
      A      Q8.0
      BEC

Network 2    // Mở van bơm. Van bơm được mở khi có sườn lên của tín hiệu lệnh và bình rỗng, hoặc khi có
               sườn xuống của van xả và vẫn còn tín hiệu lệnh.
      A      I8.0
      FP     M8.0      // Có sườn lên của tín hiệu lệnh.
      AN     I1.2      // Và bình rỗng.
      O(               // Hoặc có sườn xuống của van xả và vẫn còn tín hiệu lệnh.
      A      Q4.1
      FN     M4.1
      A      I8.0
      )
      S      Q5.1      // Mở van bơm chất lỏng A.
      S      Q6.1      // Mở van bơm chất lỏng B.

Network 3    // Điều khiển mở động cơ bơm chất lỏng. Động cơ được mở 2 giây sau van bơm.
      A      Q5.1
      L      S5T#2s
      SD     T50
      A      T50
      FP     M5.0
      S      Q5.0      // Mở động cơ bơm A
      S      Q6.0      // Mở động cơ bơm B
```

```
Network 4    // Phát hiện sự cố máy bơm.
      A      Q5.0
      L      S5T#5s
      SD     T20         // Tạo trễ 5 giây sau sườn lên của tín hiệu máy bơm.
      A      T20         // Nếu 5 giây sau khi bơm mà không có tín hiệu báo có chất lỏng thì dừng.
      A(
      ON     I2.0
      ON     I3.0
      )
      JC     stop

Network 5    // Khởi động động cơ trộn ngay khi có tín hiệu báo mức dưới.
      A      I1.1        // Đã có đủ chất lỏng trong bình.
      FP     M1.1
      S      Q4.0

Network 6    // Dừng máy bơm khi có tín hiệu báo bình đã đầy.
      A      I1.0        // Đã có đủ chất lỏng trong bình.
      FP     M1.0
      R      Q5.0        // Dừng bơm A.
      R      Q6.0        // Dừng bơm B.

Network 7    // Khóa van bơm 2 giây sau khi tắt động cơ bơm.
      A      I5.0
      L      S5T#2s
      SF     T51         // Tạo trễ theo sườn xuống.
      A      T51
      FN     M5.1
      R      Q5.1        // Khóa van A.
      R      Q6.1        // Khóa van B.

Network 8    // Tắt động cơ trộn và mở van xả.
      A      I1.0
      L      S5T#10s
      SD     T40
      A      T40         // Bình đã đầy được 10 giây.
      FP     M4.0
      R      Q4.0        // Tắt động cơ trộn.
      S      Q4.1        // Mở van xả.

Network 9    // Khóa van xả khi có sườn xuống của tín hiệu báo bình rỗng.
      A      I1.2
      FN     M1.2
      R      Q4.1
      BEU                // Hết một chu kỳ.

Network 10   // Xử lý sự cố máy bơm.
stop: R      Q5.0        // Dừng máy bơm A.
      R      Q5.1        // Khóa van A.
      R      Q6.0        // Dừng máy bơm B.
      R      Q6.1        // Khóa van B.
      R      Q4.0        // Dừng động cơ trộn.
      S      Q8.0        // Bật đèn báo sự cố.
      BE
```

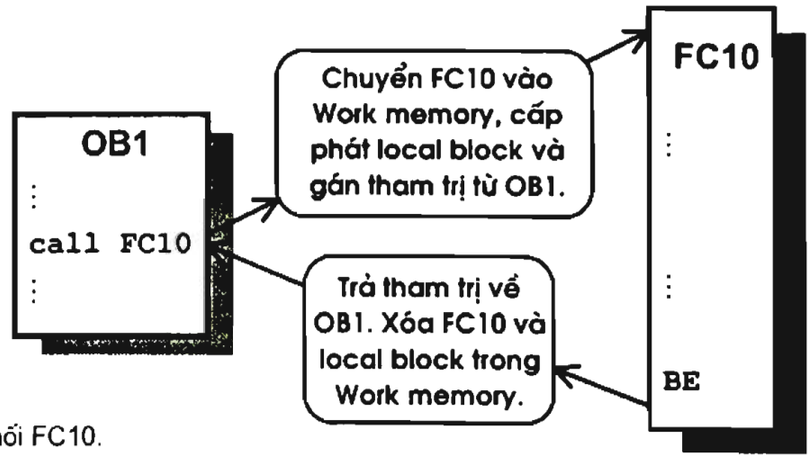
3.3 Lập trình có cấu trúc

Lập trình có cấu trúc (structure programming) là kỹ thuật cài đặt thuật toán điều khiển bằng cách chia nhỏ thành các khối chương trình con FC hay FB với mỗi khối thực hiện một nhiệm vụ cụ thể của bài toán điều khiển chung và toàn bộ các khối chương trình này lại được quản lý một cách thống nhất bởi khối OB1. Trong OB1 có các lệnh gọi những khối chương trình con theo thứ tự phù hợp với bài toán điều khiển đặt ra.

Hoàn toàn tương tự, một nhiệm vụ điều khiển con có thể còn được chia nhỏ thành nhiều nhiệm vụ nhỏ và cụ thể hơn nữa, do đó một khối chương trình con cũng có thể được gọi từ một khối chương trình con khác. Duy có điều cấm kỵ ta cần phải tránh là không bao giờ một khối chương trình con lại gọi đến chính nó. Ngoài ra, do có sự hạn chế về ngăn xếp của các module CPU nên không được tổ chức chương trình con gọi lồng nhau quá số lần mà module CPU được sử dụng cho phép.

Để đơn giản trong trình bày, khi một khối chương trình con này gọi một khối chương trình con khác, ta sẽ ký hiệu khối chứa lệnh gọi là khối mẹ và khối được gọi là khối con. Hình 3.11 mô tả quy trình thực hiện việc gọi một khối con FC10 từ khối mẹ OB1.

Giữa khối mẹ và khối con có sự liên kết thể hiện qua việc trao đổi các giá trị. Khi gọi khối con, khối mẹ cần cho những sơ kiện thông qua các tham trị đầu vào để khối con thực hiện nhiệm vụ. Sau khi thực hiện xong nhiệm vụ, khối con phải trả lại cho khối mẹ kết quả bằng những tham trị đầu ra. Hệ điều hành của CPU tổ chức việc truyền tham trị thông qua local block của từng khối con.



Hình 3.11. Thực hiện gọi khối FC10.

Như vậy, khi thực hiện lệnh gọi một khối con, hệ điều hành sẽ:

1) Chuyển khối con được gọi từ vùng Load memory vào vùng Work memory.
2) Cấp phát cho khối con một phần bộ nhớ trong Work memory để làm local block. Cấu trúc local block được quy định khi soạn thảo các khối.
3) Truyền các tham trị từ khối mẹ cho biến hình thức **IN**, **IN–OUT** của local block.

4) Sau khi khối con thực hiện xong nhiệm vụ và ghi kết quả dưới dạng tham trị đầu ra cho biến **OUT**, **IN–OUT** của local block, hệ điều hành sẽ chuyển các tham trị này cho khối mẹ và giải phóng khối con cùng local block ra khỏi vùng Work memory.

3.3.1 Khai báo local block cho FC

Local block của khối con được chia thành hai phần:

- Phần các biến hình thức để khối con nhận và truyền tham trị với khối mẹ. Biến hình thức trong local block của khối FC có ba loại cho trong bảng dưới:

Loại biến hình thức	Ý nghĩa
IN	Biến hình thức nhận tham trị từ khối mẹ làm sơ kiện cho chương trình trong khối con
OUT	Biến hình thức truyền tham trị từ khối con về khối mẹ.
IN–OUT	Biến hình thức vừa có khả năng nhận vừa có khả năng truyền tham trị giữa khối con với khối mẹ.

- Phần chứa các biến tạm thời được ký hiệu là **TEMP** (chữ viết tắt của Temporary) chứa các giá trị tính toán tức thời. Do local block sẽ được giải phóng khi kết thúc chương trình, giá trị các biến tạm thời này cũng sẽ bị mất theo ngay sau khi chương trình trong khối con được thực hiện xong.

Việc khai báo local block đồng nghĩa với việc đặt tên biến, định nghĩa loại biến (biến hình thức hay biến tạm thời) và kiểu dữ liệu (nguyên, thực, ký tự ...) cho từng biến, trong đó tên biến là những dãy ký tự hoặc số và không thuộc nhóm ký tự khóa (đã được dùng bởi hệ điều hành).

Chương trình truy nhập local block thông qua các tên biến dưới dạng toán hạng của lệnh theo cấu trúc:

```
#<tên biến>
```

Ví dụ:

```
L     #receive     // Đọc nội dung của ô nhớ có tên là receive trong local block vào ACCU1
T     #transmit    // Chuyển ACCU1 tới ô nhớ có tên là transmit trong local block
```

Chý ý : Một điều cần phải được đặc biệt chú ý là bắt đầu từ miền các biến tạm thời **TEMP**, địa chỉ được đánh lại từ đầu. Lý do là chỉ miền này mới được thực sự cấp bộ nhớ trong Work memory. Miền biến hình thức không được cấp ô nhớ mà chỉ có con trỏ địa chỉ. Do đó nếu trong chương trình, toán hạng của những lệnh truy nhập ô nhớ của local block có cấu trúc

```
L#<địa chỉ>
```

thì đó sẽ là ô nhớ thuộc miền các biến **TEMP**.

Những kiểu dữ liệu hợp lệ cho tất cả các loại biến (kể cả biến hình thức và biến tạm thời) được tổng kết trong bảng sau:

Kiểu dữ liệu	Kích thước (bit)	Tham trị thích hợp
BOOL	1	Kiểu biến logic với hai giá trị 0 hoặc 1. Tham trị có thể là một giá trị logic (TRUE / FALSE) hoặc là nội dung của một bit.
BYTE	8	Tham trị phải là nội dung của một byte.
WORD	16	Tham trị phải là nội dung của một từ (2 bytes).
DWORD	32	Tham trị phải là nội dung của một từ kép (4 bytes).
CHAR	8	Tham trị được truyền có thể là một mã ASCII hoặc nội dung của một byte.
INT	16	Tham trị được truyền vào có thể là nội dung của một từ (2 bytes) hoặc là một số nguyên trong khoảng –32768 ÷ 32767.
DINT	32	Tham trị được truyền vào có thể là nội dung của một từ kép (4 bytes) hoặc là một số nguyên trong khoảng $-2^{31} \div 2^{31}-1$
REAL	32	Tham trị được truyền vào có thể là nội dung của một từ kép (4 bytes) hoặc là một số thực dấu phảy động. Ví dụ 3.1416.
TIME	32	Tham trị được truyền vào có thể là nội dung của một từ kép hoặc là một số đo khoảng thời gian dạng T# ngàyD_giờH_phútM_giâyS_mili giâyMS.
DATE	32	Tham trị được truyền vào có thể là nội dung của một từ kép (4 bytes) hoặc là một giiá trị ngày tháng dạng D#năm–tháng–ngày.
TOD	32	Tham trị được truyền vào có thể là nội dung của một từ kép (4 bytes) hoặc là một giá trị thời gian dạng TOD# ngàyD_giờH_phútM_giâyS_mili giâyMS.
S5TIME	32	Tham trị được truyền vào có thể là nội dung của một từ kép (4 bytes) hoặc là một số đo thời gian dạng S5T#ngàyD_giờH_phútM_giâyS_mili giâyMS.
DT Date_And_Time	64	Tham trị được truyền vào có thể là nội dung của ô nhớ có kiểu Date_And_Time (DT) hoặc là một giá trị dạng DT#năm–tháng–ngày–giờ:phút:giây:mili giây.
ANY	80	Đây là kiểu biến tổng quát, thay thế được cho các kiểu ở trên. Ngoài ra tham trị của kiểu biến này còn có thể là thanh ghi CV, T–Bit, C–Bit, tên của Timer; tên của Counter; tên các logic block như FB10, FC2 ... , tên biến hình thức.

Để minh họa cho việc khai báo và sử dụng local block ta làm một số ví dụ.

Ví dụ 1 : Viết chương trình con cho khối FC1 để đọc số nguyên 16 bits trong khoảng –32767 ÷ 32767 và biến đổi nó thành số thực trong khoảng –10.0 ÷ 10.0. Chương trình này có thể được sử dụng để đọc giá trị của cổng tương tự. Cổng tương tự được áp dụng sẽ là cổng mà địa chỉ của nó là tham trị truyền vào cho FC1.

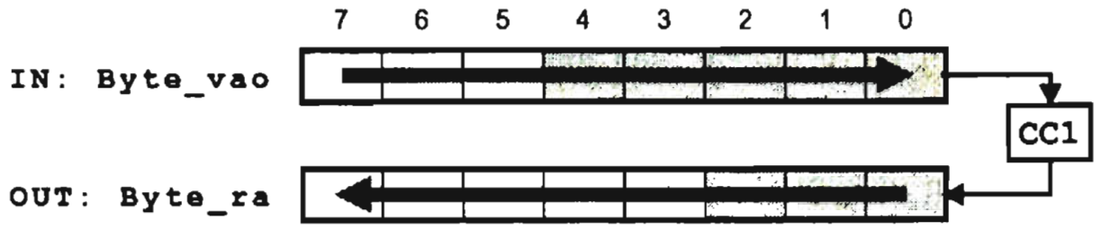
Local block của FC1

Địa chỉ	Loại	Tên	Kiểu	Giá trị đặt trước
0.0	IN	gia_tri_vao	INT	
2.0	OUT	gia_tri_ra	REAL	
	IN–OUT			
0.0	TEMP			

Chương trình

```
L     #gia_tri_vao      // Lấy số nguyên 16 bits từ khối mẹ (tham trị truyền vào)
ITD                     // Biến đổi thành số nguyên 32 bits
DTR                     // Biến đổi thành số thực
L     3.276700e+003     // Chia cho số thực 3276.7
/R
T     #gia_tri_ra       // Truyền kết quả về cho khối mẹ
BE
```

Ví dụ 2 : Quay lại bài toán đảo thứ tự các bits đã viết cho ví dụ 2 của mục 2.6.3 thuộc chương 2. Ở đây ta mở rộng ra là việc đảo 8 bits đó không cố định là chỉ thực hiện với riêng byte MB0 và cất kết quả vào MB10 mà có thể được sử dụng với mọi byte trong bộ nhớ, kể cả các byte thuộc bộ đệm vào ra I, Q. Muốn vậy bắt buộc ta phải viết lại dưới dạng khối chương trình, chẳng hạn khối FC10.



Ta sẽ sử dụng 8 lần lệnh dịch bit phải **RRCA** nhằm đọc bit ngoài cùng bên phải của **Byte_vao** và ghi nó vào bit ngoài cùng bên phải của **Byte_ra** cũng bằng lệnh dịch bit nhưng theo chiều ngược lại **RLCA**. Chương trình sử dụng các biến nháp có tên là **Dem** làm thanh ghi đếm số lần thực hiện việc dịch chuyển bit và hai byte **Vao_tam**, **Ra_tam** làm các byte nhớ trung gian.

Local block của FC10

Địa chỉ	Loại	Tên	Kiểu	Giá trị đặt trước
0.0	IN	Byte_vao	BYTE	
2.0	OUT	Byte_ra	BYTE	
	IN-OUT			
0.0	TEMP	Dem	INT	
2.0	TEMP	Vao_tam	BYTE	
3.0	TEMP	Ra_tam	BYTE	

Chương trình

```
      L     #Byte_vao
      T     #Vao_tam      // Biến trung gian tức thời cho tham trị vào
      L     8
dest: T     #Dem          // Thanh ghi đếm số lần thực hiện lệnh dịch chuyển bit.
      L     #Vao_tam
      RRDA                // Lấy bit ngoài cùng bên phải của Vao_tam vào CC1.
      T     #Vao_tam
      L     #Ra_tam       // Sử dụng biến trung gian tức thời
      RLDA                // Chuyển CC1 vào bit ngoài cùng bên phải của Ra_tam.
      T     #Ra_tam
      L     #Dem
      LOOP  dest
      L     #Ra_tam
      T     #Byte_ra      // Chuyển Ra_tam thành tham trị đầu ra.
      BE
```

Do toàn bộ các lệnh của khối FC10 được xử lý xong trong một vòng quét nên cũng trong một vòng quét nội dung các bits của **Byte_vao** sẽ được đảo xong thứ tự và chuyển ra **Byte_ra**. Bởi vậy nội dung các biến **TEMP** của ví dụ trên sẽ không bị mất trong quá trình thực hiện FC10.

3.3.2 Gọi khối FC và thủ tục truyền tham trị

Lệnh gọi một khối con và truyền tham trị cho nó từ khối mẹ có dạng

Cú pháp **CALL FCx**

trong đó FCx là tên khối con được gọi.

Ngay khi gặp lệnh gọi một khối con, chương trình soạn thảo Step7 sẽ căn cứ vào cấu trúc của local block, cụ thể là những biến hình thức của khối con (biến **IN**, **OUT**, **IN–OUT**), mà cho hiện lại những biến này chờ người sử dụng khai báo tham trị.

Kiểu tham trị truyền từ khối mẹ vào khối con thông qua biến hình thức **IN** hay **IN–OUT** phụ thuộc vào kiểu đã gán. Cụ thể là:

- Nếu biến được khai báo một trong các kiểu **BOOL**, **CHAR**, **INT**, **DINT**, **TIME**, **BOOL**, **DATE**, **TOD**, **S5TIME** thì tham trị truyền có thể là một giá trị cụ thể hoặc là nội dung của một ô nhớ có kích thước tương ứng.
- Nếu biến được khai báo kiểu **BYTE**, **WORD**, **DWORD**, **DINT** thì bắt buộc tham trị phải là nội dung của ô nhớ có kích thước phù hợp.

Riêng đối với tham trị được khối con trả về cho khối mẹ qua biến hình thức **OUT** hay **IN–OUT** thì luôn phải là một ô nhớ có cùng kích thước với biến.

Ví dụ, ngay sau khi ta viết lệnh gọi khối FC1 đã soạn thảo ở ví dụ 1 trong mục 3.3.1 để biến đổi số nguyên 16 bits thành số thực trong khoảng $-10.0 \div 10.0$, chương trình soạn thảo sẽ cho hiện tên các biến hình thức IN và OUT của FC1 chờ được gán tham trị:

```
CALL  FC1
   gia_tri_vao: =
   gia_tri_ra:  =
```

Do biến **IN** của FC1 có kiểu **INT** nên cả hai cách khai báo tham trị sau đều hợp lệ

```
CALL  FC1
   gia_tri_vao: = PIW304        // Tham trị là nội dung của ô nhớ 2 bytes PIW304
   gia_tri_ra:  = MD0
```

và

```
CALL  FC1
   gia_tri_vao: = 5000          // Tham trị là số nguyên 5000 có kích thước 16 bits
   gia_tri_ra:  = MD0
```

Ở cách gán tham trị thứ nhất, giá trị trả về của FC1 sẽ là

$$\text{MD0} = \frac{\text{PIW304}}{32767} \times 10$$

còn ở kiểu gán thứ hai thì FC1 trả về giá trị

$$\text{MD0} = \frac{5000}{32767} \times 10 = 1.52593$$

Một ví dụ khác với FC10 đã soạn thảo ở ví dụ 2 trong mục 3.3.1. Với khối FC10 này thì chương trình đảo thứ tự các bits MB0 thành MB10 viết trong OB1 tại mục 2.6.3 được sửa đổi lại như sau:

```
      A     I0.0
      FP    M0.0              // Bit ghi nhớ trạng thái I0.0 trước đó.
      JCN   end               // Kết thúc chương trình nếu không có sườn lên tại I0.0.
      CALL  FC10
         Byte_vao:   = MB0
         Byte_ra:    = MB10
end:  BEU
```

Ngoài lệnh **CALL FCx** S7–300 còn cung cấp hai lệnh gọi khối khác là

```
Cú pháp    UC    FCx
           CC    FCx
```

trong đó **UC** (unconditional call) là lệnh gọi vô điều kiện và **CC** (conditional call) là lệnh gọi có điều kiện (khi **RLO=1**). Song cả hai lệnh này không có khả năng tạo giao tiếp giữa khối mẹ và khối con qua tham biến hình thức (**IN**, **OUT**, **IN–OUT**). Hơn thế nữa nếu khối **FCx** được gọi có sử dụng tham biến hình thức, hệ điều hành sẽ xem đó là lỗi lập trình và phát tín hiệu báo ngắt dừng hệ thống.

Như vậy, để sử dụng được với hai lệnh trên, khối **FCx** có thể có các biến **TEMP** trong local block, nhưng tuyệt đối không được có và sử dụng các tham biến hình thức.

Ví dụ nếu khối chương trình điều khiển bình trộn OB1, đã trình bày trong mục 3.2.2, nay được viết lại cho khối FC2 gồm các Network từ 2 đến 10, trong đó khối FC2 không sử dụng các tham biến hình thức, thì trong OB1 ta chỉ cần viết lệnh gọi có điều kiện (không có tín hiệu sự cố máy bơm) như sau:

```
Network 1
      AN    Q8.0
      CC    FC2         // Chỉ thực hiện FC2 khi RLO=1, tức là khi Q8.0=0.
      BE
```

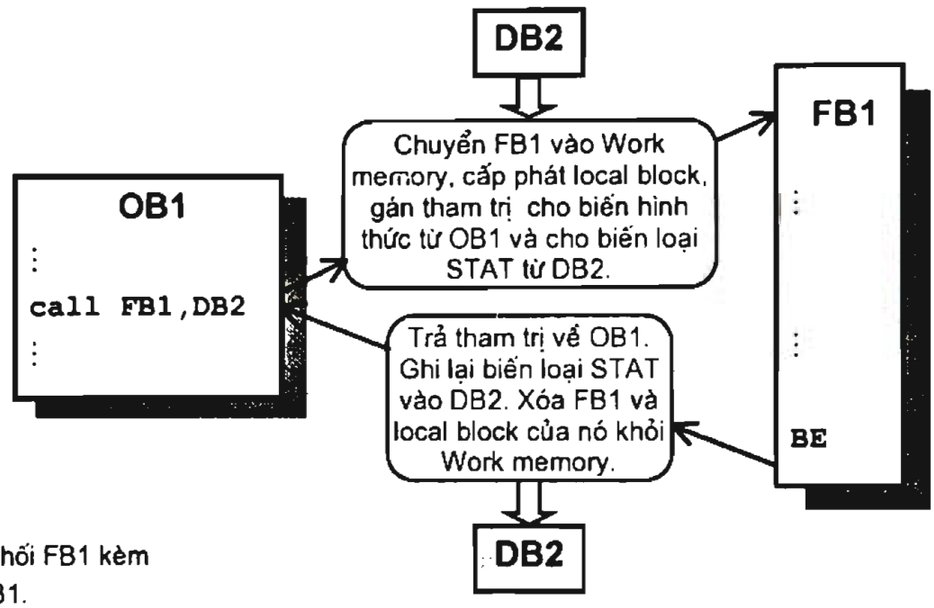
3.3.3 Local block của FB

Nhược điểm của kiểu khối FC là nội dung các biến tạm thời kiểu **TEMP** không được lưu giữ lại cho những vòng quét sau. Điều này bắt buộc những khối FC có sử dụng biến kiểu **TEMP** trong local block phải được thực hiện xong trong một vòng quét và do đó hạn chế miền sử dụng của chúng.

Khắc phục nhược điểm trên, S7–300/400 cung cấp một loại khối có tính năng tương tự như khối FC nhưng lại có khả năng lưu giữ lại được nội dung các biến tạm thời cho các vòng quét kế tiếp, được gọi là khối hàm FB. Loại biến tạm thời có nội dung được lưu giữ lại này có tên là **STAT** (viết tắt của static).

Phương thức lưu giữ lại nội dung các biến loại **STAT** được hệ điều hành thực hiện nhờ một khối dữ liệu như sau (hình 3.12):

- khi thực hiện lệnh gọi, hệ điều hành chuyển khối FB được gọi vào Work memory, cấp phát cho nó trong Work memory một local block như yêu cầu. Ghi các tham trị từ khối mẹ vào các tham biến hình thức loại **IN**, **IN–OUT** và nội dung các ô nhớ tương ứng trong DB kèm theo vào biến loại **STAT** trong local block.
- khi chương trình trong khối FB kết thúc, hệ điều hành chuyển nội dung của biến hình thức loại **OUT**, **IN–OUT** về cho khối mẹ và ghi lại các giá trị của biến thuộc loại **STAT** trong local block vào khối dữ liệu kèm theo. Sau đó giải phóng local block cùng khối FB ra khỏi Work memory.



Hình 3.12. Thực hiện gọi khối FB1 kèm cùng với DB2 từ OB1.

Về cơ bản, local block của khối FB cũng giống như của khối FC, nhưng có thêm biến loại **STAT**. Các loại biến của khối FB cho trong bảng dưới:

Loại biến	Ý nghĩa
IN	Biến hình thức sử dụng để nhận tham trị từ khối mẹ làm sơ kiện cho chương trình trong khối con
OUT	Biến hình thức dùng để trả tham trị từ khối con về khối mẹ.
IN–OUT	Biến hình thức. Loại biến này vừa có khả năng nhận, vừa có khả năng trả tham trị cho khối mẹ.
STAT	Nội dung của biến loại này có khả năng được lưu giữ lại khi kết thúc chương trình trong FB.
TEMP	Biến tạm thời. Nội dung sẽ bị mất khi chương trình trong FB kết thúc.

Việc khai báo local block cho FB cũng hoàn toàn tương tự như cho FC gồm: đặt tên biến, xác định loại biến (biến hình thức, **STAT** hay **TEMP**) và kiểu dữ liệu (nguyên, thực, ký tự ...) cho từng biến.

Tên biến phải là những dãy ký tự hoặc số và không thuộc nhóm ký tự khóa (đã được dùng bởi hệ điều hành).

Kiểu dữ liệu hợp lệ cho tất cả các loại biến (kể cả biến hình thức, biến **STAT** và **TEMP**) đã cho trong bảng ở mục 3.3.1. Riêng đối với biến **STAT** ta còn sử dụng được kiểu dữ liệu **ARRAY**, **STRING** (xem mục 2.10.1 của chương 2).

Xét lại ví dụ 2 đã làm ở mục 2.9.2 của chương 2 về việc nhập dữ liệu từ cổng tương tự nhưng được mở rộng hơn là không cố định từ cổng PIW304 mà có thể sử dụng cho các cổng tương tự khác nhau, số giá trị được lưu giữ lại là số thực và cũng không chỉ là 10 mà là 50 dữ liệu. Thêm vào đó, kết quả trả về là giá trị trung bình của 50 dữ liệu đó. Giá trị trung bình này vẫn thường được xem như là giá trị tín hiệu tương tự đã được khử nhiễu. Ta thấy với yêu cầu như vậy thì không thể chỉ viết chương trình cho OB1 vì cổng truy nhập không được xác định trước và cũng khó có thể sử dụng miền nhớ cờ (M) làm bộ đệm vì miền này tối đa chỉ có 256 bytes.

Ta sẽ cài đặt chương trình trong khối FB1, trong đó bộ đệm chứa 50 giá trị đọc được từ cổng tương tự được lưu giữ nhờ biến loại **STAT** trong local block. Chương trình sử dụng:

- một biến hình thức thuộc loại **IN** có tên **Cong** để xác định địa chỉ cổng tương tự được truy nhập,
- một biến hình thức thuộc loại **OUT** có tên **Ket_qua** để FB1 trả về giá trị trung bình của 50 dữ liệu gần nhất vừa nhận được,
- 50 biến thuộc loại **STAT** làm bộ đệm, có tên là **Bo_dem**.
- một biến thuộc loại **STAT** có tên **So_gia_tri** để ghi nhớ số các dữ liệu hiện có trong bộ đệm (nhiều nhất là 50),
- một biến loại **TEMP** có tên **Dem** để đếm số vòng lặp khi thực hiện việc dồn bộ đệm,
- một biến thuộc loại **TEMP** có tên **Tong_tam** để lưu giá trị tổng tạm thời.

Local block của FB1

Địa chỉ	Loại	Tên	Kiểu	Ý nghĩa
0.0	IN	Cong	INT	Biến nhận tham trị từ khối mẹ
2.0	OUT	Ket_qua	REAL	Biến trả kết quả về cho khối mẹ
	IN-OUT			
6.0	STAT	Bo_dem	ARRAY[1..50]	Bộ đệm gồm 50 dữ liệu là số thực
*4.0			REAL	
206.0	STAT	So_gia_tri	INT	Số các dữ liệu hiện có trong bộ đệm
0.0	TEMP	Dem	BYTE	
2.0	TEMP	Tong_tam	REAL	

Chương trình

```
      LAR1   p##Bo_dem          // Con trỏ chỉ ô nhớ đầu tiên trong bộ đệm
      L      49                 // Số các ô nhớ phải được dồn lên trong bộ đệm
next: T      #Dem               // Dồn ô nhớ
      L      D[AR1,P#4.0]       // Đọc nội dung ô nhớ kế tiếp phía dưới
```

```
      T     D[AR1,P#0.0]          // Chuyển lên ô nhớ phía trên
      +AR1  P#4.0                 // Chuyển con trỏ xuống ô dưới
      L     #Dem
      LOOP  next
      L     #Cong                 // Nhập dữ liệu mới
      ITD
      DTR                         // Chuyển thành số thực
      T     #Bo_dem[50]           // Ghi vào ô nhớ cuối cùng của bộ đệm
      L     50
      L     #So_gia_tri           // Số các dữ liệu có trong bộ đệm
      <=I                         // Bộ đệm chưa đầy
      JC    sum
      L     1
      +I
      T     #So_gia_tri
sum:  L     0.00000e+000          // Tính tổng
      T     #Tong_tam
      LAR1  p##Bo_dem             // Con trỏ chỉ ô nhớ đầu tiên trong bộ đệm
      L     50                    // Số các số hạng của tổng
con:  T     #Dem
      L     #Tong_tam
      L     D[AR1,p#0.0]
      +R
      T     #Tong_tam
      +AR1  p#4.0
      L     #Dem
      LOOP  con
      L     #Tong_tam             // Tính giá trị trung bình
      L     #So_gia_tri
      ITD
      DTR                         // Chuyển thành số thực
      /R
      T     #Ket_qua              // Tham trị trả về
      BE
```

Cũng giống như local block của FC, trong ví dụ trên ta thấy bắt đầu từ miền các biến tạm thời **TEMP**, địa chỉ được đánh lại từ đầu. Do đó nếu trong chương trình, toán hạng của những lệnh truy nhập ô nhớ của local block với cấu trúc

L#<địa chỉ>

thì đó sẽ là ô nhớ thuộc miền các biến **TEMP**.

3.3.4 Instance block và thủ tục gọi khối FB

Khác với khối FC, khối hàm FB bao giờ cũng làm việc cùng với một khối dữ liệu DB dùng để lưu giữ nội dung các biến kiểu **STAT** của local block (hình 3.12). Khối DB này có tên là khối dữ liệu instance. Lý do là khi thực hiện lệnh gọi khối hàm FB, hệ điều hành cũng mở luôn khối dữ liệu này bằng lệnh "**OPN DI**".

Như vậy, kèm với lệnh gọi khối FB ta phải chỉ thị luôn cả tên khối dữ liệu DB tương ứng. Lệnh gọi khối hàm FB có cấu trúc như sau:

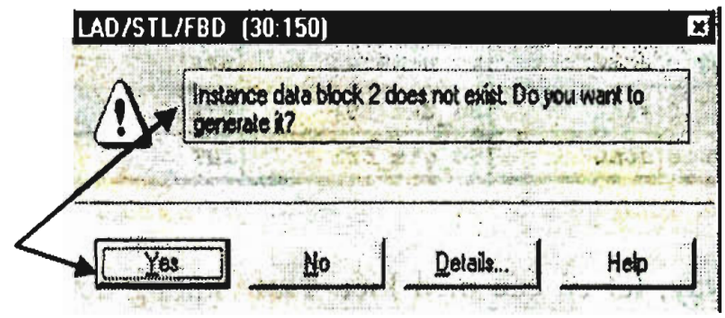
Cú pháp `CALL FBx , DBy`

trong đó FBx là tên khối hàm được gọi và DBy là tên khối dữ liệu kèm theo. Khối dữ liệu DBy phải có cấu trúc phù hợp với local block của FBx đã được soạn thảo.

Phần mềm Step7 hỗ trợ người soạn thảo việc tạo lập khối dữ liệu DB có cấu trúc phù hợp với local block của khối hàm FB được gọi. Ngay sau khi viết lệnh gọi một khối hàm FB và nếu khối DB kèm theo chưa được soạn thảo trước, Step7 sẽ tạo lập một DB mới có cấu trúc phù hợp với local block của khối hàm FB đó. Chẳng hạn nếu trong khối OB1 ta viết lệnh gọi khối hàm FB1 đã soạn thảo ở mục 3.3.3 cùng với khối dữ liệu có tên DB2

```
CALL  FB1,DB2
```

mà DB2 lại chưa có, Step7 sẽ cho hiện thông báo



Khối DB instance chưa có. Ấn phím **Yes** để tạo khối DB này.

Khi ấn phím **Yes** để chấp nhận tạo DB2, chương trình soạn thảo Step7 sẽ tạo DB2 theo cấu trúc của local block của FB1 và dựa vào đó để cho hiện lại những biến hình thức **IN**, **OUT**, **IN–OUT** chờ người sử dụng khai báo tham trị:

```
CALL  FB      1 , DB2
      Cong    : =
      Ket_qua: =
```

Để sử dụng FB1 nhằm lọc nhiễu tín hiệu tương tự tại cổng PIW304 và chuyển kết quả vào ô nhớ kiểu **TEMP** trong local block của khối mẹ OB1 có tên **analog_value**, với việc lọc nhiễu được thực hiện bằng cách tính giá trị trung bình của 50 dữ liệu gần nhất, ta khai báo local block cho OB1 và soạn thảo chương trình chương trình như sau:

Local block của OB1

Địa chỉ	Loại	Tên	Kiểu
0.0	TEMP	OB1_EV_CLASS	Byte
1.0	TEMP	OB1_SCAN_1	Byte
⋮	⋮	⋮	⋮
10.0	TEMP	OB1_MAX_CYCLE	Int
12.0	TEMP	OB1_DATE_TIME	DATE_AND_TIME
20.0	TEMP	analog_value	REAL

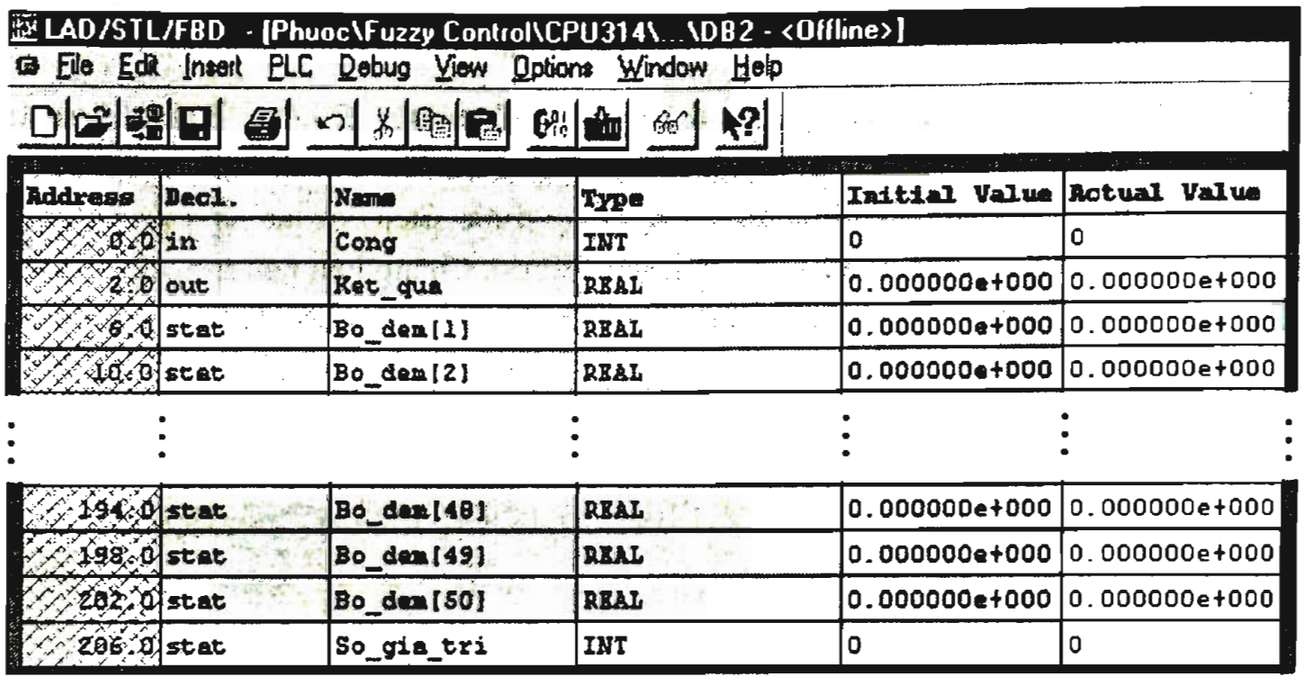
Phần có sẵn và chỉ được sử dụng, không được sửa đổi.

Người sử dụng khai báo thêm.

Chương trình

```
CALL  FB      1 , DB2
      Cong    : =PIW304
      Ket_qua: =#analog_value
```

Khối DB2 vừa được tạo lập theo cấu trúc local block của FB1 có dạng như sau:



Address	Decl.	Name	Type	Initial Value	Actual Value
0.0	in	Cong	INT	0	0
2.0	out	Ket_qua	REAL	0.000000e+000	0.000000e+000
6.0	stat	Bo_dem[1]	REAL	0.000000e+000	0.000000e+000
10.0	stat	Bo_dem[2]	REAL	0.000000e+000	0.000000e+000
⋮	⋮	⋮	⋮	⋮	⋮
194.0	stat	Bo_dem[48]	REAL	0.000000e+000	0.000000e+000
198.0	stat	Bo_dem[49]	REAL	0.000000e+000	0.000000e+000
202.0	stat	Bo_dem[50]	REAL	0.000000e+000	0.000000e+000
206.0	stat	So_gia_tri	INT	0	0

Như vậy, tất cả các biến kiểu **TEMP** không được chuyển vào DB2. Các tham biến hình thức thuộc loại **IN**, **OUT**, **IN–OUT** xuất hiện trong DB2, song thực chất chỉ là địa chỉ và không được lưu giữ tham trị. Chỉ riêng giá trị của các biến kiểu **STAT** là được chuyển vào DB2.

Khối dữ liệu DB2 nói riêng và các khối instance của FB nói chung đều có thể được sử dụng như một khối dữ liệu thông thường, tức là ta có thể dùng các lệnh mở khối để mở chúng, truy nhập các ô nhớ của chúng.

Ví dụ đoạn chương trình sau sẽ:

- Cho sáng đèn Q4.0 nếu giá trị thực có tại cổng tương tự PIW304 vừa đọc được nhỏ hơn giá trị trung bình của 50 giá trị đọc được mới nhất và giá trị này được xem như là giá trị có tại cổng cũng ở thời điểm đó nhưng đã được lọc nhiễu tức thời.
- Cho sáng đèn Q4.1 nếu số các giá trị có trong bộ đệm chưa đủ 50.

```
CALL  FB     1 , DB2
      Cong   : =PIW304
      Ket_qua: =#analog_value
OPN   DB2
L     DBD202              // Đọc nội dung của ô nhớ Bo_dem[50]
L     #analog_value
<R                        // So sánh với giá trị trung bình
=     Q4.0
L     DBW206              // Số các giá trị hiện có trong bộ đệm
L     50
<I
=     Q4.1
```

3.3.5 Ngăn xếp B và ngăn xếp L (B–Stack, L–Stack)

Chương trình trong OB1, tùy theo thiết kế của chương trình điều khiển, có thể gọi nhiều khối FC và FB khác nhau. Bản thân các khối FC hay FB con này của OB1 cũng có thể gọi tiếp các khối FC, FB khác, tức là chúng cũng có thể là khối mẹ của các khối FB, FC con khác .... Cứ tiếp tục như vậy, trong chương trình ứng dụng sẽ có nhiều lệnh **CALL** tạo ra một xâu quan hệ khối mẹ–khối con. Độ dài tối đa cho phép của xâu quan hệ mẹ–con này giữa các khối chương trình phụ thuộc vào chủng loại module CPU đang sử dụng, chẳng hạn với CPU314 thì độ dài tối đa cho phép là 8 khối.

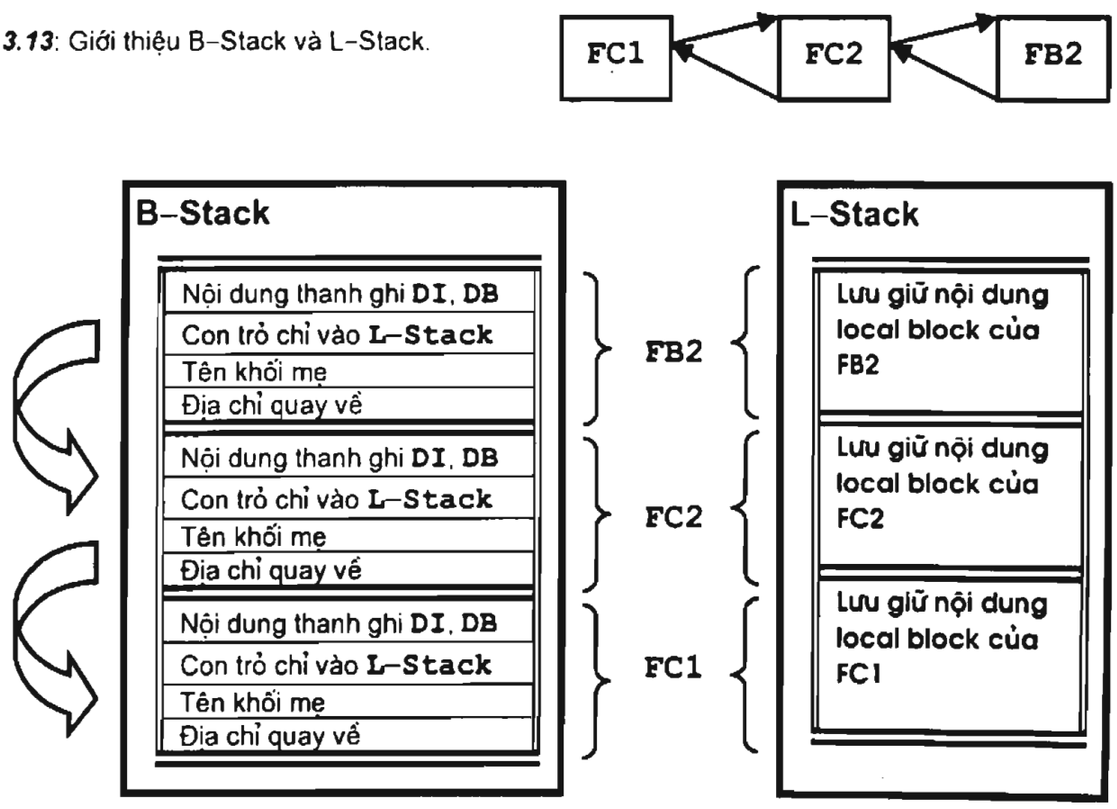
Khi gọi một khối con, ngoài việc tổ chức cấp phát local block, chuyển khối con vào Work memory ... hệ điều hành còn cần phải ghi nhớ lại vị trí lệnh gọi cũng như dữ liệu của công việc đang được thực hiện dở dang trong khối mẹ vào ngăn xếp B và ngăn xếp L (B–Stack, L–Stack) để sau đó còn biết chỗ quay về khi kết thúc chương trình trong khối con và tiếp tục thực hiện được công việc bị ngắt nửa chừng của khối mẹ (hình 3.13).

Như vậy, để ghi lại vị trí quay về và công việc phải làm tiếp, hệ điều hành sẽ:

- cất nội dung các ô nhớ của local block của khối mẹ vào L–Stack.
- cất vào B–Stack nội dung các thanh ghi DB, DI, tên khối mẹ, địa chỉ câu lệnh tiếp ngay sau lệnh gọi trong khối mẹ và con trỏ chỉ vùng dữ liệu trong L–Stack.

Thứ tự cất vào B–Stack và L–Stack là từ trên xuống. Các dữ liệu mới được ghi vào đầu ngăn xếp, những nội dung cũ được dồn xuống. Độ sâu ngăn xếp quyết định độ dài xâu quan hệ khối mẹ–khối con, tức là quyết định số các lệnh **CALL** được gọi lồng nhau.

Hình 3.13: Giới thiệu B–Stack và L–Stack.



3.4 Sử dụng các khối OB

Các khối OB có thể được xếp theo loại công dụng thành 3 nhóm:

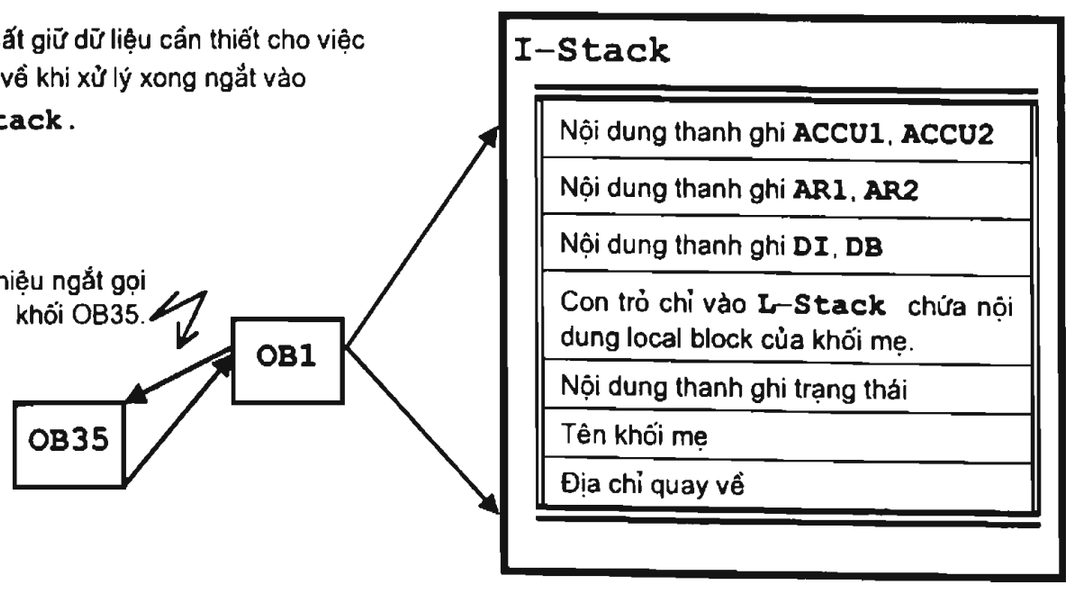
- nhóm các khối OB chứa chương trình ứng dụng xử lý ngắt,
- nhóm các khối OB chứa chương trình khởi động và
- nhóm các khối OB xử lý lỗi trong hệ thống.

Cũng như FC, FB, khối OB là khối chứa chương trình, do đó cũng là một logic block. Điểm khác biệt cơ bản giữa OB với các khối khác thuộc logic block là OB không được gọi chủ động để thực hiện (ví dụ như bằng lệnh **CALL**) mà bị động bởi các tín hiệu ngắt. Khái niệm "gọi bị động" được hiểu là vị trí cũng như thời điểm phát lệnh gọi không được lập trình từ trước mà hoàn toàn mang tính ngẫu nhiên. Chương trình trong các khối OB này cũng có thể có các lệnh gọi khối FC hoặc FB nhưng tất nhiên không thể gọi một khối OB khác.

Mỗi khối OB được gọi bằng một loại tín hiệu ngắt. Vậy nếu xảy ra hiện tượng xuất hiện cùng một lúc nhiều tín hiệu ngắt thì sao?. Trong trường hợp như vậy, khối OB nào có thứ tự ưu tiên cao hơn sẽ được xử lý trước và chương trình trong khối OB có thứ tự ưu tiên thấp hơn phải chờ cho tới khi tất cả các khối có ưu tiên cao hơn đã được xử lý xong mới đến lượt được thực hiện. Khối OB1 là khối có mức ưu tiên thấp nhất và do đó mọi tín hiệu ngắt đều ngắt được quá trình thực hiện chương trình của khối OB1

3.4.1 Ngăn xếp I (I-Stack)

Do cũng được gọi (bị động bằng tín hiệu ngắt) nên giống như việc xử lý lệnh **CALL**, hệ điều hành cần phải cất giữ vị trí quay về, các dữ liệu cần thiết để tiếp tục công việc trong khối mẹ. Ngăn xếp cất những dữ liệu này có tên là I-Stack (hình 3.14). Độ sâu của ngăn xếp I quyết định số các khối chương trình xử lý ngắt (OB) được ngắt lồng nhau và độ sâu này phụ thuộc vào chủng loại của từng module CPU.



Hình 3.14. Cất giữ dữ liệu cần thiết cho việc quay về khi xử lý xong ngắt vào **I-stack**.

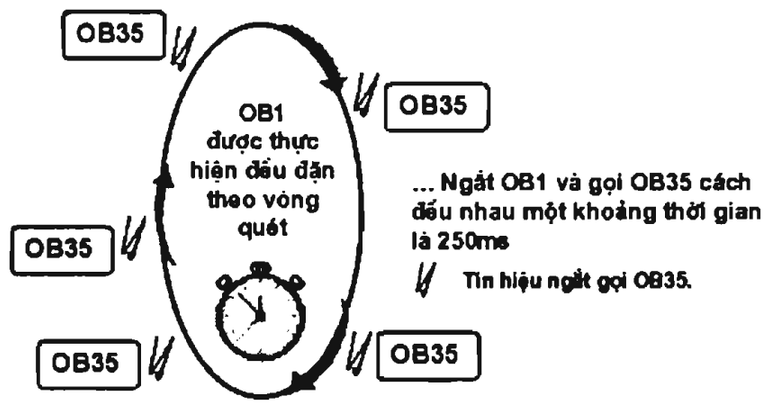
3.4.2 Chương trình ứng dụng xử lý ngắt

Chương trình ứng dụng xử lý ngắt được hiểu là loại chương trình viết cho các khối OB và được gọi bởi các tín hiệu báo ngắt thuộc loại:

- được phát ra đều đặn cách đều nhau một khoảng thời gian định trước,
- được phát ra tại một điểm thời gian định trước,
- được phát ra từ các module (ngắt cứng).

Ngắt tuần tự theo thời gian (OB30 ÷ OB38)

Ngay khi nhận thấy trong chương trình ứng dụng có một trong các khối OB30 ÷ OB38, hệ thống sẽ tự động tích cực chế độ phát tín hiệu báo ngắt gọi các khối này với khoảng thời gian cách đều nhau. Giá trị mặc định cho chu kỳ phát tín hiệu báo ngắt này là 100ms. Nói cách khác, cứ 100ms thì các khối OB này được gọi và thực hiện một lần. Hình 3.15 minh họa cho chế độ ngắt tuần tự theo thời gian.



Hình 3.15. Xử lý tín hiệu ngắt theo chu kỳ thời gian.

Tổng quát thì tất cả các khối trong khoảng OB30 ÷ OB38 đều thuộc nhóm khối chương trình xử lý ngắt theo chu kỳ thời gian. Song không phải module CPU nào cũng cho phép sử dụng tất cả các khối OB đó, chẳng hạn CPU314 chỉ cho phép sử dụng OB35.

Trường hợp có nhiều khối OB cùng xử lý một tín hiệu báo ngắt thì ta có thể phân biệt chúng với nhau theo thứ tự ưu tiên. Chỉ số thứ tự ưu tiên được gán cho từng khối nhờ phần mềm Step7. Ta cũng có thể sử dụng Step7 để thay đổi chu kỳ phát tín hiệu báo ngắt (xem chương 4).

Ví dụ: Để trích mẫu tín hiệu tương tự $u(t)$ có tại cổng PIW304 với chu kỳ lấy mẫu là T_a=250ms và cất giữ giá trị $u_k=u(kT_a)$ trích được vào ô nhớ MW0, trước hết ta cần phải khai báo chu kỳ phát tín hiệu báo ngắt T_a =250ms (nhờ Step7) cho module CPU và tiếp theo viết các lệnh sau vào khối OB35:

```
L     PIW304
T     MW0
```

Local block của các khối OB30 ÷ OB38 có dạng chung giống như của OB35 cho trong bảng sau:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB35_EV_CLASS	Byte	Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB35_SCAN_1	Byte	Báo OB35 đã được thực hiện bằng giá trị 16#36
OB35_PRIORITY	Byte	Có giá trị là 11 (thứ tự ưu tiên)
OB35_OB_NUMBR	Byte	35. Là chỉ số của khối OB35
OB35_RESERVED_1	Byte	Dự trữ (của hệ điều hành)
OB35_RESERVED_2	Byte	Dự trữ (của hệ điều hành)
OB35_PHASE_OFFSET	Word	Thời gian trễ (milliseconds)
OB35_RESERVED_3	Int	Dự trữ (của hệ điều hành)
OB35_EXC_FREQ	Int	Chu kỳ thời gian thực hiện (milliseconds)
OB35_DATE_TIME	Date_And_Time	Thời điểm OB35 bắt đầu được thực hiện.

So với các biến trong local block của OB1 đã được giải thích tại mục 3.2.1 thì ở đây, có biến **OB35_EXC_FREQ** và **OB35_PHASE_OFFSET** là hơi khác về mặt ý nghĩa sử dụng và cần phải được giải thích rõ thêm:

- **OB35_EXC_FREQ** chứa chu kỳ phát tín hiệu ngắt (mặc định là 100ms hoặc đã được quy định lại thành T_u nhờ Step7).
- **OB35_PHASE_OFFSET** chứa khoảng thời gian trễ kể từ khi xuất hiện tín hiệu báo ngắt cho tới khi OB35 được gọi. Thông thường ô nhớ này có nội dung bằng 0, song trong một số trường hợp ứng dụng người ta vẫn phải gán cho nó một giá trị dương khác 0 nhằm tránh nguy cơ nhiều khối OB30 ÷ OB38 cùng được thực hiện một lúc dễ gây ra lỗi về thời gian cho hệ thống.

Như đã nói, ngay khi phát hiện thấy một trong các khối OB30 ÷ OB38 có trong Load Memory, hệ thống sẽ tự động tích cực chế độ phát tín hiệu báo ngắt theo chu kỳ 100ms. Chu kỳ T_u =100ms mặc định này có thể sửa lại được nhờ công cụ phần mềm **Simatic Manager** nhưng giá trị sửa lại đó là cố định trong suốt quá trình thực hiện chương trình ứng dụng sau này, tức là ta chỉ có thể sửa lại chu kỳ T_u phát tín hiệu ngắt khi CPU ở chế độ **STOP** và phải sử dụng **Simatic Manager** để nạp các tham số mới cho module CPU.

Linh hoạt hơn so với việc sửa đổi lại T_u, ta có thể tích cực hoặc hủy bỏ chế độ ngắt theo chu kỳ bằng những hàm có sẵn trong hệ điều hành và do đó không cần phải chuyển CPU về trạng thái **STOP**. Cụ thể là:

- Hàm SFC39 (tên hình thức **DIS_IRT**) có tác dụng che ngắt.
- Hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- Hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

– Hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.1.

Ngắt tại một thời điểm định trước (OB10 ÷ OB17) – xem thêm 3.5.3

Khối OB10 nói riêng (ví dụ cho module CPU314) và các khối OB10÷OB17 nói chung (phụ thuộc chủng loại của module CPU) sẽ được hệ điều hành gọi một lần tại một thời điểm định trước hoặc nhiều lần kể từ thời điểm đã cho. Khi được gọi nhiều lần kể từ thời điểm đã được xác định, ta có thể quy định:

– mỗi phút một lần,
– mỗi tiếng một lần,
– mỗi ngày một lần,
– mỗi tuần một lần,
– mỗi tháng một lần và
– mỗi năm một lần.

Khối OB10 có local block như sau (tương tự cũng cho cả các khối OB11÷OB17):

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB10_EV_CLASS	Byte	B#16#11 = Ngắt đang được tích cực
OB10_STRT_INFO	Byte	B#16#11 = OB10 đã được gọi và thực hiện
OB10_PRIORITY	Byte	Có giá trị là 2 (thứ tự ưu tiên)
OB10_OB_NUMBR	Byte	10. Là chỉ số của khối OB10
OB10_RESERVED_1	Byte	Dự trữ (của hệ điều hành)
OB10_RESERVED_2	Byte	Dự trữ (của hệ điều hành)
OB10_PERIOD_EXE	Word	Mã quy định chế độ thực hiện xử lý ngắt. W#16#0000: Một lần W#16#0201: Mỗi phút một lần W#16#0401: Mỗi giờ một lần W#16#1001: Mỗi ngày một lần W#16#1201: Mỗi tuần một lần W#16#1401: Mỗi tháng một lần W#16#1801: Mỗi năm một lần
OB10_RESERVED_3	Int	Dự trữ (của hệ điều hành)
OB10_RESERVED_4	Int	Dự trữ (của hệ điều hành)
OB10_DATE_TIME	Date_And_Time	Thời điểm OB10 bắt đầu được thực hiện.

Có hai cách để định nghĩa thời điểm phát tín hiệu ngắt và quy định chế độ làm việc (một lần hay nhiều lần) cho OB10÷OB17. Cách thứ nhất là sử dụng công cụ phần mềm **Simatic Manager** và cách thứ hai là sử dụng hàm SFC28 có tên hình thức **SET_TINT** của hệ thống.

Sau khi định nghĩa thời điểm gọi OB10, bản thân khối OB10 cũng cần phải được tích cực. Ta cũng có hai cách để tích cực khối OB10 hoặc bằng công cụ phần mềm **Simatic Manager** hoặc nhờ hàm SFC30 (có tên hình thức **ACT_TINT**) của hệ thống.

Để hủy bỏ trạng thái tích cực của khối OB10 ta sử dụng hoặc dụng công cụ phần mềm **Simatic Manager** hoặc nhờ hàm SFC29 (có tên hình thức **CAN_TINT**).

Tín hiệu báo ngắt tại thời điểm định trước này có thể được che nhờ hàm SFC39 (có tên hình thức **DIS_IRT**) hay bóc mặt nạ che nhờ hàm SFC40 (có tên hình thức **EN_IRT**). Ngoài ra ta cũng có thể sử dụng hàm SFC41 (có tên hình thức **DIS_AIRT**) để che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý hoặc hàm SFC42 (có tên hình thức **EN_AIRT**) để bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý. Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.1

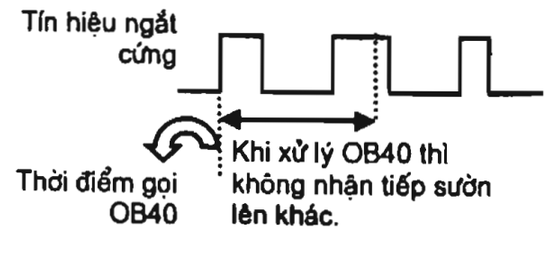
Ngắt cứng (OB40 ÷ OB47) – xem thêm 3.5.4

Đây là loại tín hiệu báo ngắt được phát từ module mở rộng (module I/O, AI/AO, CP hay FM) hoặc từ các cổng vào ra số onboard (của module CPU IFM). Chế độ báo ngắt cứng này thường được sử dụng trong các chương trình điều khiển mà ở đó đòi hỏi phải có sự đáp ứng nhanh với tín hiệu từ ngoài đưa vào.

Không phải mọi loại module I/O, AI/AO mở rộng đều có khả năng phát tín hiệu báo ngắt. Thậm chí đối với ngay cả những module đặc biệt có khả năng phát tín hiệu ngắt cứng thì ta cũng phải đặt tham số chế độ làm việc ngắt cứng cho nó:

- hoặc bằng công cụ phần mềm **Simatic Manager** (cho module I/O, AI/AO) hay bằng các phần mềm kèm theo của module (CP, FM),
- hoặc nhờ các hàm của hệ thống như:
  - SFC55 (tên hình thức **WR_PARM**) để ghi tham số đặt cấu hình cho module,
  - SFC56 (tên hình thức **WR_DPARM**) để sửa đổi một vài tham số cấu hình của module trong chế độ **RUN**,
  - SFC57 (tên hình thức **PARM_MOD**) để sửa đổi toàn bộ tham số cấu hình của module.

Một điều đặc biệt của chế độ ngắt cứng là trong khoảng thời gian thực hiện chương trình của OB40 (hoặc của các khối OB41÷OB47) hệ thống sẽ không nhận và không xử lý bất cứ một tín hiệu ngắt cứng nào khác (hình 3.16).



Hình 3.16: Xử lý tín hiệu ngắt cứng.

Giống như mọi tín hiệu ngắt khác, tín hiệu ngắt cứng cũng có thể được:

- che ngắt nhờ SFC39 (tên hình thức **DIS_IRT**) hay
- bóc mặt nạ che nhờ SFC40 (tên hình thức **EN_IRT**).

Ngoài ra ta cũng có thể sử dụng hàm SFC41 (tên hình thức **DIS_AIRT**) để che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý hoặc SFC42 (tên hình thức **EN_AIRT**) để bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

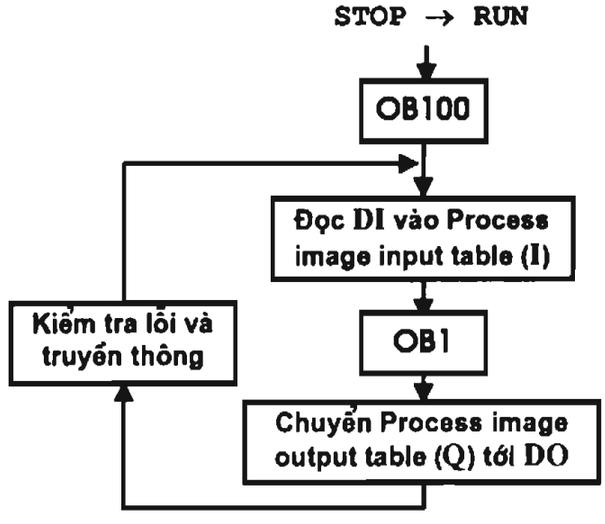
Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.

Khối OB40 có local block như sau (tương tự cũng cho cả các khối OB41÷OB47):

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB40_EV_CLASS	Byte	B#16#11 = Ngắt đang được tích cực.
OB40_STRT_INFO	Byte	B#16#41 = OB40 đã được gọi và thực hiện.
OB40_PRIORITY	Byte	Có giá trị là 16 (thứ tự ưu tiên mặc định).
OB40_OB_NUMBR	Byte	40. Là chỉ số của khối OB10.
OB40_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB40_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB40_MDL_EXE	Int	Địa chỉ module I/O số, module AI/AO, CP hay FM phát tín hiệu báo ngắt.
OB40_POINT_ADDR	Dword	Nếu là module I/O số thì nội dung ô nhớ này là địa chỉ cổng số phát tín hiệu báo ngắt. Nếu là module AI/AO, CP hay FM thì nội dung ô nhớ là trạng thái ngắt của module.
OB40_DATE_TIME	Date_And_Time	Thời điểm OB40 bắt đầu được thực hiện.

3.4.3 Chương trình khởi động (Initialization)

Khi cài đặt chương trình ứng dụng, rất nhiều thuật toán đòi hỏi phải có những giá trị, tham số hay sơ kiện ... ban đầu. Những giá trị, tham số ... này không thể được khai báo trong khối OB1 vì như vậy cứ đầu vòng quét là chương trình điều khiển sẽ lại trở về trạng thái khởi động.



Hình 3.17: Khởi động chương trình điều khiển với khối OB100.

Hệ điều hành của CPU S7–300 cung cấp khối OB100 cho phép ta thực hiện được các công việc khởi động cho chương trình điều khiển. Khi chuyển CPU từ trạng thái **STOP** sang **RUN** để thực hiện chương trình điều khiển, hệ điều hành bao giờ cũng gọi và

thực hiện chương trình trong khối OB100 trước, sau đó mới thực sự bắt đầu vòng quét với việc gọi OB1. Hình 3.17 mô tả quá trình khởi động của S7–300.

Mỗi khi CPU chuyển trạng thái từ **STOP** sang **RUN** là OB100 được gọi và thực hiện, không phân biệt việc chuyển trạng thái đó được tiến hành trực tiếp bằng khoá chuyển công tắc trên module CPU hay nhờ phần mềm **Simatic Manager**.

Khối OB100 có local block như sau:

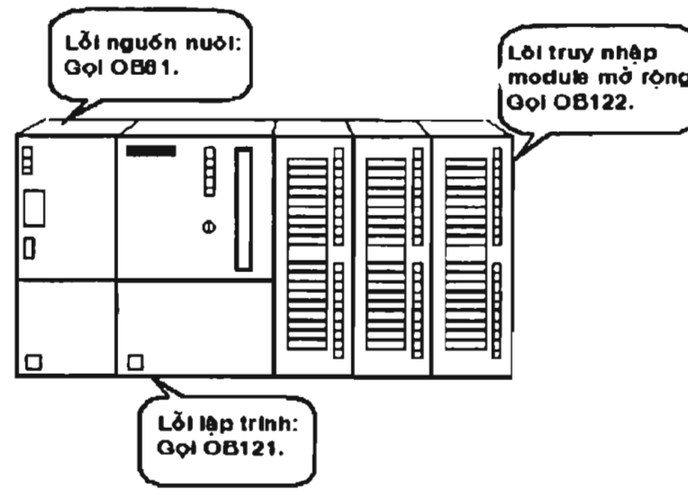
Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB100_EV_CLASS	Byte	B#16#13.
OB100_STRTUP	Byte	Chế độ gọi OB100: ▪ B#16#81 = Được gọi khi chuyển từ STOP sang RUN. ▪ B#16#82 = Được gọi tự động khi chuyển từ STOP sang RUN hoặc khi chuyển nguồn từ OFF sang ON và CPU vẫn đang ở trạng thái RUN.
OB100_PRIORITY	Byte	27 (thứ tự ưu tiên).
OB100_OB_NUMBR	Byte	100. Là chỉ số của khối OB100.
OB100_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB100_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB100_STOP	Word	Mã hiệu ngắt làm cho CPU chuyển về trạng thái STOP.
OB100_STRT_INFO	Dword	Thông tin về việc thực hiện chế độ khởi động.
OB100_DATE_TIME	Date_And_Time	Thời điểm OB100 bắt đầu được thực hiện.

3.4.4 Xử lý lỗi hệ thống

Lỗi hệ thống có hai loại:

- Lỗi asynchronous (lỗi không đồng bộ), bao gồm:
  + lỗi vượt quá thời gian xoay vòng cho phép – OB80,
  + lỗi sự cố nguồn nuôi (ví dụ không có pin) – OB81,
  + lỗi sự cố module (ví dụ chập mạch trên module vào) – OB82,
  + lỗi thiếu khối OB chứa chương trình xử lý ngắt – OB85,
  + lỗi truyền thông – OB87.
- Lỗi synchronous (lỗi đồng bộ), bao gồm:
  + lỗi lập trình (ví dụ thiếu khối DB, FC hoặc FB) – OB121,
  + lỗi truy nhập module (ví dụ có lệnh truy nhập module mở rộng nhưng lại không tìm thấy module đó) – OB122.

Hình 3.18 minh họa việc sử dụng một số khối OB (khối OB81, OB121, OB122) để cài đặt chương trình xử lý lỗi hệ thống.



Hình 3.18. Các tín hiệu ngắt xử lý lỗi hệ thống.

Khi gặp lỗi không đồng bộ, hệ thống sẽ chuyển CPU về trạng thái **STOP**. Tất cả các tín hiệu ngắt báo lỗi không đồng bộ đều có thể được che hoặc bỏ mặt nạ che nhờ sử dụng:

- hàm SFC39 (với tên hình thức **DIS_IRT**) có tác dụng che ngắt.
- hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.
- hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.1.

Xử lý lỗi về thời gian thực hiện chương trình (OB80)

Khối OB80 sẽ được hệ thống gọi khi:

a) Thời gian thực hiện chương trình vượt quá thời gian vòng quét cực đại cho phép. Mặc định, mỗi vòng quét được quy định là phải thực hiện không quá 150ms. Sự quy định này là cần thiết để có thể đảm bảo được tính thời gian thực của chương trình điều khiển. Mặc dù ta có thể sử dụng phần mềm **Simatic Manager** để tăng khoảng thời gian vòng quét cực đại cho phép, song điều này là hoàn toàn không nên, nhất là khi phải điều khiển đối tượng biến đổi nhanh.

b) Theo thiết kế, một trong số OB10÷OB17 đáng ra phải được gọi tại một thời điểm định trước, song vì một lý do nào đó, ví dụ như do đồng hồ thời gian thực của CPU đã bị chỉnh lại, mà điều đó không được thực hiện (mục 3.5.3).

c) Hệ thống đang phải xử lý một tín hiệu ngắt chưa xong mà đã gặp phải tín hiệu báo ngắt cùng loại. Ví dụ nếu thời gian cần thiết để thực hiện OB35 lại lâu hơn chu kỳ phát tín hiệu báo ngắt T_a đã khai báo thì sẽ xảy ra trường hợp OB35 chưa được xử lý xong hệ thống đã lại phải gọi OB35 để xử lý cho lần tiếp theo (hiện tượng đệ quy).

d) Gặp phải lỗi trong chương trình của một khối OB, chẳng hạn như lỗi logic, lỗi thuật toán ....

Ngay cả trong trường hợp gặp một tín hiệu ngắt cứng hoặc ngắt theo chu kỳ thời gian nhưng lại không có khối OB tương ứng (OB40 hay OB35) của tín hiệu ngắt đó, hệ thống cũng chuyển sang gọi khối OB80 đồng thời đưa CPU về trạng thái **STOP**.

Như vậy sẽ có nhiều loại tín hiệu ngắt báo lỗi khác nhau cùng gọi đến OB80. Chúng sẽ được OB80 phân biệt với nhau trong quá trình xử lý bằng mã nhận biết kiểu lỗi. Tương ứng với những mã nhận biết kiểu lỗi khác nhau, khối OB sẽ tự tổ chức cho mình các local block khác nhau.

Local block của khối OB80 cho trường hợp a) có cấu trúc như sau:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB80_EV_CLASS	Byte	B#16#35. Mã nhận biết thứ nhất
OB80_FLT_ID	Byte	B#16#01. Mã nhận biết thứ hai
OB80_PRIORITY	Byte	26 (thứ tự ưu tiên).
OB80_OB_NUMBR	Byte	80. Là chỉ số của khối OB80.
OB80_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB80_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB80_LAST_CYL	Word	Thời gian vòng quét vừa thực hiện.
OB80_MIN_CYL	Word	Thời gian vòng quét ngắn nhất đã thực hiện.
OB80_MAX_CYL	Word	Thời gian vòng quét lâu nhất đã thực hiện.
OB80_DATE_TIME	Date_And_Time	Thời điểm OB80 bắt đầu được thực hiện.

Local block của khối OB80 cho trường hợp b), c) và d) có cấu trúc như sau:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB80_EV_CLASS	Byte	B#16#35. Mã nhận biết thứ nhất
OB80_FLT_ID	Byte	Mã nhận biết thứ hai B#16#02. B#16#03. B#16#07.
OB80_PRIORITY	Byte	26 (thứ tự ưu tiên).
OB80_OB_NUMBR	Byte	80. Là chỉ số của khối OB80.
OB80_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB80_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB80_ERR_INFO	Word	Kiểu lỗi phát hiện được.
OB80_ERR_EV_CLA	Byte	Kiểu tín hiệu ngắt.
OB80_ERR_EV_NUM	Byte	Số hiệu tín hiệu ngắt.
OB80_OB_PRIORIT	Byte	Thứ tự ưu tiên của khối OB đang được thực hiện thì xuất hiện tín hiệu báo ngắt.
OB80_OB_NUM	Byte	Tên khối OB đang được thực hiện thì xuất hiện tín hiệu báo ngắt.
OB80_DATE_TIME	Date_And_Time	Thời điểm OB80 bắt đầu được thực hiện.

Xử lý lỗi nguồn nuôi (OB81)

Hệ thống sẽ gọi khối OB81 khi phát hiện có lỗi không đồng bộ báo sự cố về nguồn nuôi hoặc pin nuôi miền nhớ non-volatile. Khác với việc xử lý những lỗi không đồng bộ khác, khi xử lý sự cố nguồn nuôi hệ thống sẽ không chuyển CPU về trạng thái **STOP**.

Tín hiệu ngắt báo lỗi nguồn nuôi có thể được che hoặc bỏ mặt nạ che nhờ:

- hàm SFC39 (với tên hình thức **DIS_IRT**) có tác dụng che ngắt.
- hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.
- hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.1.

Local block của khối OB81 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB81_EV_CLASS	Byte	B#16#39.
OB81_FLT_ID	Byte	B#16#22. Mã nhận biết lỗi
OB81_PRIORITY	Byte	26/28 (thứ tự ưu tiên).
OB81_OB_NUMBR	Byte	81. Là chỉ số của khối OB81.
OB81_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB81_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB81_MDL_ADDR	Int	Kiểu lỗi phát hiện được.
OB81_RESERVED_3	Byte	Kiểu tín hiệu ngắt.
OB81_ERR_EV_NUM	Byte	Số hiệu tín hiệu ngắt.
OB81_RESERVED_4	Byte	Dự trữ (của hệ điều hành).
OB81_RESERVED_5	Byte	Dự trữ (của hệ điều hành).
OB81_RESERVED_6	Byte	Dự trữ (của hệ điều hành).
OB81_DATE_TIME	Date_And_Time	Thời điểm OB81 bắt đầu được thực hiện.

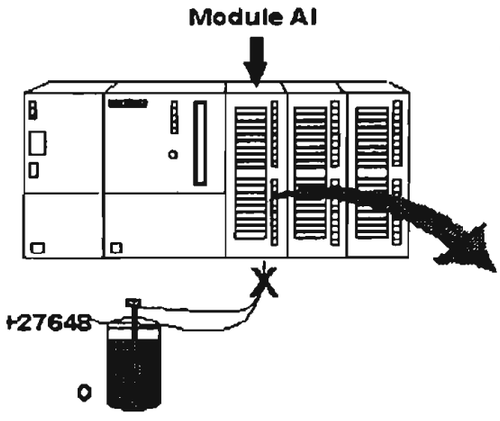
Xử lý lỗi module (OB82) – xem mục 3.5.4

Hệ thống sẽ gọi khối OB82 khi phát hiện lỗi không đồng bộ về sự cố trên module mở rộng (hình 3.19). Những module này phải là các module có khả năng tự kiểm tra lỗi (diagnostic cabilities). Nếu không tìm thấy OB82, hệ thống sẽ chuyển CPU về trạng thái **STOP**. Tín hiệu ngắt báo lỗi modulei có thể được che hoặc bỏ mặt nạ che nhờ:

- hàm SFC39 (tên hình thức **DIS_IRT**) có tác dụng che ngắt.
- hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

– hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

Chi tiết về cách gọi và khai báo tham trị cho những hàm hệ thống trên sẽ được trình bày sau trong mục 3.5.1.



Hình 3.19: Tín hiệu ngắt báo sự cố module tương tự sẽ được phát nếu có hiện tượng chập mạch tín hiệu đầu vào.

Nếu module tương tự này đã được đặt chế độ tự chẩn đoán lỗi thì khi phát hiện thấy lỗi chập mạch tại tín hiệu đầu vào, hệ thống sẽ gọi OB82.

Local block của khối OB82 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB82_EV_CLASS	Byte	B#16#38 hoặc B#16#39.
OB82_FLT_ID	Byte	B#16#42. Mã nhận biết có lỗi module
OB82_PRIORITY	Byte	26/28 (thứ tự ưu tiên).
OB82_OB_NUMBR	Byte	82. Là chỉ số của khối OB82.
OB82_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB82_IO_FLAG	Byte	01010100 Module vào 01010101 Module ra.
OB82_MDL_ADDR	Int	Địa chỉ module có lỗi.
OB82_MDL_DEFECT	Bool	Trạng thái module: Lỗi.
OB82_INT_FAULT	Bool	Lỗi bên trong module.
OB82_EXT_INFO	Bool	Lỗi bên ngoài module.
OB82_FLD_CONNCT	Bool	Lỗi nối với bus dẫn nội.
OB82_NO_CONFIG	Bool	Module không có tham số cấu hình.
OB82_CONFIG_ERR	Bool	Tham số cấu hình của module sai.
OB82_MDL_TYPE	Byte	Kiểu module.
OB82_SUB_MDL_ER	Bool	Phần phụ của module bị lỗi hoặc không có.
OB82_COMM_FAULT	Bool	Lỗi truyền thông với module.
OB82_MDL_STOP	Bool	Module ở trạng thái STOP
OB82_WTCH_DOG_F	Bool	Module dừng bởi Watch-dog-Timer.
OB82_INT_PS_FLT	Bool	Lỗi do nguồn trong.
OB82_PRIM_BATT	Bool	Lỗi do pin nuôi chính.
OB82_BCKUP_BATT	Bool	Lỗi do pin nuôi phụ.
OB82_RESERVED_2	Bool	Dự trữ (của hệ điều hành).
OB82_RACK_FLT	Bool	Lỗi bởi thanh ray.

OB82_PROC_FLT	Bool	Lỗi bộ vi xử lý của module.
OB82_EPROM_FLT	Bool	Lỗi bộ nhớ EPROM của module.
OB82_RAM_FLT	Bool	Lỗi bộ nhớ RAM của module.
OB82_ADU_FLT	Bool	Lỗi ADU của module.
OB82_FUSE_FLT	Bool	Lỗi cầu chì của module.
OB82_HW_INTR_FL	Bool	Lỗi tín hiệu ngắt cứng đầu vào.
OB82_RESERVED_3	Bool	Không sử dụng.
OB82_DATE_TIME	Date_And_Time	Thời điểm OB82 bắt đầu được thực hiện.

Xử lý lỗi thiếu khối OB (OB85)

Hệ thống sẽ gọi khối OB85 khi gặp lỗi đồng bộ cũng như tín hiệu báo ngắt tại thời điểm định trước nhưng lại không tìm thấy các khối OB tương ứng để xử lý những tín hiệu ngắt này (ví dụ thiếu khối OB10, OB82). Khối OB85 cũng được hệ thống gọi nếu trong chương trình ứng dụng có sử dụng các hàm chuẩn của hệ thống (SFC) mà những hàm này lại truy nhập đến những khối OB không có trong bộ nhớ, ví dụ chương trình sử dụng hàm SFC30 để tích cực OB10 nhưng không tìm thấy OB10 trong bộ nhớ. Nếu không tìm thấy OB85, hệ thống sẽ chuyển CPU về trạng thái **STOP**.

Tín hiệu ngắt báo lỗi thiếu khối OB này có thể được che nhờ hàm SFC39 (tên hình thức **DIS_IRT**) hoặc bỏ mặt nạ che nhờ hàm SFC40 (tên hình thức **EN_IRT**). Ta cũng có thể sử dụng hàm SFC41 (tên hình thức **DIS_AIRT**) để che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý hoặc hàm SFC42 (tên hình thức **EN_AIRT**) để bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý. Chi tiết về những hàm chuẩn này sẽ được trình bày sau trong mục 3.5.1.

Local block của khối OB85 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB85_EV_CLASS	Byte	B#16#35.
OB85_FLT_ID	Byte	Mã báo kiểu lỗi B#16#A1: Chương trình SFC không tìm thấy khối OB B#16#A2: Hệ điều hành không tìm thấy khối OB (ví dụ như khi đặt tham số cấu hình).
OB85_PRIORITY	Byte	26 (thứ tự ưu tiên).
OB85_OB_NUMBR	Byte	85. Là chỉ số của khối OB85.
OB85_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB85_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB85_RESERVED_3	Word	Dự trữ (của hệ điều hành).
OB85_ERR_EV_CLA	Byte	Dạng tín hiệu báo lỗi.
OB85_ERREV_NUM	Byte	Chỉ số tín hiệu báo lỗi.
OB85_OB_PRIORIT	Byte	Thứ tự ưu tiên của khối OB đang được thực hiện khi có lỗi.
OB85_OB_NUM	Byte	Tên khối OB đang được thực hiện khi xuất hiện lỗi.
OB85_DATE_TIME	Date_And_Time	Thời điểm OB85 bắt đầu được thực hiện.

Xử lý lỗi truyền thông (OB87)

Hệ thống sẽ gọi khối OB87 khi có lỗi truyền thông, ví dụ như lỗi time out. Nếu không tìm thấy OB87 để xử lý lỗi, hệ thống sẽ chuyển CPU về trạng thái **STOP**.

Tín hiệu ngắt báo lỗi truyền thông có thể được che hoặc bỏ mặt nạ che nhờ:

- hàm SFC39 (với tên hình thức **DIS_IRT**) có tác dụng che ngắt.
- hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.
- hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn tín hiệu ngắt đang được xử lý.

Chi tiết về cách gọi và khai báo tham trị cho những hàm chuẩn trên sẽ được trình bày sau trong mục 3.5.

Local block của khối OB87 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB87_EV_CLASS	Byte	B#16#35.
OB87_FLT_ID	Byte	Mã báo kiểu lỗi B#16#D2: Không phát được tín hiệu yêu cầu kiểm tra (diagnostic). B#16#E1: Sai cấu trúc ID khi truyền thông với global data (GD). B#16#E2: Không đưa được thông tin trạng thái của GD vào DB. B#16#E6: Nhóm thông tin trạng thái của GD không chuyển được vào DB.
OB87_PRIORITY	Byte	26 (thứ tự ưu tiên).
OB87_OB_NUMBR	Byte	87. Là chỉ số của khối OB87.
OB87_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB87_RESERVED_2	Byte	Dự trữ (của hệ điều hành).
OB87_RESERVED_3	Word	Dự trữ (của hệ điều hành).
OB87_RESERVED_4	Dword	Dự trữ (của hệ điều hành).
OB87_DATE_TIME	Date_And_Time	Thời điểm OB87 bắt đầu được thực hiện.

Xử lý lỗi lập trình (OB121) – xem thêm mục 3.5.2

Khối OB121 sẽ được gọi bằng tín hiệu báo lỗi lập trình, ví dụ như lỗi chuyển đổi sai kiểu dữ liệu BCD, lỗi truy nhập khối DB hay gọi một khối FC không có trong bộ nhớ ....

Khối OB121 có cùng thứ tự ưu tiên giống như khối OB bị ngắt (thường là OB1). Điều này giúp cho chương trình trong OB121 sử dụng được nội dung cũ có trong các thanh ghi trước khi xuất hiện tín hiệu báo lỗi. Tương tự khi thực hiện xong chương trình trong OB121 và trở về khối OB bị ngắt, chương trình ứng dụng trong khối OB này sử dụng được nội dung các thanh ghi đã được sửa đổi bởi OB121.

Tín hiệu ngắt báo lỗi lập trình có thể được che hoặc bỏ mặt nạ che nhờ:

- hàm SFC36 (tên hình thức **MSK_FLT**) có tác dụng che ngắt.
- hàm SFC37 (tên hình thức **DMSK_FLT**) có tác dụng bỏ mặt nạ che ngắt.

Chương trình ứng dụng cũng có thể sử dụng hàm SFC38 (với tên hình thức **READ_ERR**) để đọc nội dung thanh ghi báo kiểu lỗi lập trình gặp phải. Chi tiết về cách gọi và khai báo tham trị cho những hàm chuẩn trên sẽ được trình bày sau trong mục 3.5.

Local block của khối OB121 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB121_EV_CLASS	Byte	B#16#25.
OB121_SW_FLT	Byte	Mã báo kiểu lỗi (xem thêm mục 3.5.2) B#16#21: Lỗi biến đổi BCD. B#16#22: Lỗi đọc ô nhớ ngoài miền cho phép. B#16#23: Lỗi ghi vào ô nhớ ngoài miền cho phép B#16#24: Lỗi đọc ô nhớ có địa chỉ không đúng cấu trúc. B#16#25: Lỗi ghi vào ô nhớ có địa chỉ sai cấu trúc. B#16#26: Lỗi sai tên Timer. B#16#27: Lỗi sai tên Counter. B#16#28: Lỗi đọc ô nhớ qua con trỏ có địa chỉ sai. B#16#29: Lỗi ghi vào ô nhớ qua con trỏ có địa chỉ sai. B#16#30: Lỗi ghi vào DB chỉ cho phép đọc. B#16#31: Lỗi ghi vào DI chỉ cho phép đọc. B#16#32: Lỗi mở một DB có tên quá lớn. B#16#33: Lỗi mở một DI có tên quá lớn. B#16#34: Lỗi gọi một khối FC có tên quá lớn. B#16#35: Lỗi gọi một khối FB có tên quá lớn. B#16#3A: Không tìm thấy DB. B#16#3C: Không tìm thấy FC. B#16#3E: Không tìm thấy FB.
OB121_PRIORITY	Byte	Thứ tự ưu tiên của khối OB bị ngắt (thường là OB1).
OB121_OB_NUMBR	Byte	121.
OB121_BLK_TYPE	Byte	Kiểu khối mà tại đó xuất hiện lỗi.
OB121_RESERVED_1	Byte	Dự trữ (của hệ điều hành).
OB121_FLT_REG	Word	Kiểu lỗi.
OB121_BLK_NUM	Word	Tên khối mà tại đó xuất hiện lỗi.
OB121_PRG_ADDR	Dword	Địa chỉ câu lệnh sinh ra lỗi.
OB121_DATE_TIME	Date_And_Time	Thời điểm OB121 bắt đầu được thực hiện.

Xử lý lỗi truy nhập module (OB122) – xem thêm mục 3.5.2

Hệ thống sẽ gọi khối OB122 khi gặp sự cố truy nhập các module mở rộng, ví dụ gặp sự cố đọc tín hiệu tại cổng của một module I/O. Khối OB122 có cùng thứ tự ưu tiên giống như khối OB bị ngắt nên chương trình trong OB122 có thể sử dụng được nội dung cũ có trong các thanh ghi trước khi xuất hiện tín hiệu báo sự cố. Tương tự khi thực hiện xong chương trình trong OB122 và trở về khối OB bị ngắt, chương trình ứng dụng trong khối OB này sử dụng được nội dung các thanh ghi đã được sửa đổi bởi OB122.

Tín hiệu ngắt báo sự cố truy nhập module có thể được nhờ hàm SFC36 (với tên hình thức **MSK_FLT**) hoặc bỏ mặt nạ che ngắt nhờ hàm SFC37 (tên hình thức **MSK_FLT**).

Chương trình ứng dụng cũng có thể sử dụng hàm SFC38 (tên hình thức **READ_ERR**) để đọc nội dung thanh ghi báo kiểu lỗi truy nhập module gặp phải.

Local block của khối OB122 có dạng:

Tên hình thức	Kiểu	Giá trị và ý nghĩa
OB122_EV_CLASS	Byte	B#16#29.
OB122_FLT_ID	Byte	Mã báo kiểu lỗi B#16#42: Lỗi đọc. B#16#43: Lỗi ghi.
OB122_PRIORITY	Byte	Thứ tự ưu tiên của khối OB bị ngắt (thường là OB1).
OB122_OB_NUMBR	Byte	122.
OB122_BLK_TYPE	Byte	Kiểu khối mà tại đó xuất hiện lỗi.
OB122_MEM_AREA	Byte	Vùng bộ nhớ mà tại đó xuất hiện lỗi.
OB122_MEM_ADDR	Word	Địa chỉ ô nhớ mà tại đó xuất hiện lỗi.
OB122_BLK_NUM	Word	Tên khối mà tại đó xuất hiện lỗi.
OB122_PRG_ADDR	Dword	Địa chỉ câu lệnh gặp phải lỗi.
OB122_DATE_TIME	Date_And_Time	Thời điểm OB122 bắt đầu được thực hiện.

3.5 Những hàm chuẩn quản lý ngắt

3.5.1 Che và bỏ mặt nạ che các tín hiệu ngắt, tín hiệu báo lỗi không đồng bộ

Mục 3.4 đã nhắc nhiều tới các hàm hệ thống dùng để che hoặc bỏ mặt nạ che các tín hiệu ngắt, các tín hiệu báo lỗi. Cụ thể là:

- Hàm SFC39 (với tên hình thức **DIS_IRT**) có tác dụng dương mặt nạ che ngắt.
- Hàm SFC40 (tên hình thức **EN_IRT**) có tác dụng bỏ mặt nạ che ngắt.
- Hàm SFC41 (tên hình thức **DIS_AIRT**) có tác dụng dương mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn mức ưu tiên của khối chương trình chứa lệnh gọi hàm.
- Hàm SFC42 (tên hình thức **EN_AIRT**) có tác dụng bỏ mặt nạ che tất cả các ngắt có mức ưu tiên cao hơn mức ưu tiên của khối chương trình chứa lệnh gọi hàm.

Các tín hiệu báo ngắt, báo lỗi được xếp theo từng nhóm như sau:

1) Nhóm tín hiệu báo ngắt tại một thời điểm định trước. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB10÷OB17.
2) Nhóm tín hiệu báo ngắt trễ so với thời điểm định trước. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB20÷OB23.
3) Nhóm tín hiệu báo ngắt theo chu kỳ thời gian T_a. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB30÷OB38.
4) Nhóm tín hiệu báo ngắt cứng từ bên ngoài. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB40÷OB47.

5) Nhóm tín hiệu báo ngắt truyền thông. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB50, OB51.
6) Nhóm tín hiệu ngắt báo lỗi không đồng bộ. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB80÷OB87.
7) Nhóm tín hiệu ngắt báo lỗi đồng bộ. Chương trình xử lý các tín hiệu ngắt này phải nằm trong các khối OB121, OB122. Đặc biệt của nhóm này là ngoài các hàm SFC39, SFC40, SFC41, SFC42, chúng còn có thể được che ngắt hoặc bỏ mặt nạ che nhờ các hàm SFC36 (tên hình thức **MSK_FLT**) và SFC37 (tên hình thức **DMSK_FLT**).

trong đó các nhóm 1), 3), 4), 6) và 7) đã được trình bày tại mục 3.3, Những nhóm còn lại, bạn đọc nào quan tâm, có thể tìm thấy trong tài liệu [12].

Hàm SFC39 (DIS_IRT)

Hàm có tác dụng dương mặt nạ che: một tín hiệu ngắt nhất định, một nhóm các tín hiệu ngắt hoặc che tất cả các tín hiệu ngắt và tín hiệu báo lỗi không đồng bộ. Khi một tín hiệu ngắt hay báo lỗi được dương mặt nạ che, hệ thống sẽ không để ý tới tín hiệu đó nữa, tức là sẽ không gọi khối OB tương ứng chứa chương trình xử lý tín hiệu ngắt, báo lỗi này cho tới khi mặt nạ che được bỏ đi.

Hàm SFC39 có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	**MODE**	**Byte**	Xác định loại tín hiệu ngắt, báo lỗi được che: B#16#0: Che tất cả các tín hiệu ngắt và tín hiệu báo lỗi không đồng bộ. Không che tín hiệu báo lỗi đồng bộ. B#16#1: Che các tín hiệu ngắt, báo lỗi thuộc một nhóm nhất định. Nhóm các tín hiệu được che được chỉ thị bởi tên khối OB đầu tiên của nhóm cho biến **OB_NR**. B#16#2: Che một tín hiệu ngắt. Tín hiệu ngắt được che được chỉ thị bởi tên khối OB tương ứng cho biến **OB_NR**.
IN	**OB_NR**	**Int**	Tên khối OB của tín hiệu ngắt, báo lỗi được che.
OUT	**RET_VAL**	**Int**	Giá trị trả về của hàm: W#16#0000: Hàm làm việc bình thường. W#16#8090: Dữ liệu cho **OB_NR** bị sai. W#16#8091: Dữ liệu cho **MODE** bị sai.

Ví dụ: Với các tham trị được gán cho lệnh gọi SFC39 trong khối OB35 như sau, tất cả các tín hiệu ngắt theo chu kỳ thời gian sẽ bị hủy (che) kể từ khi chương trình trong OB35 trích được một mẫu tín hiệu tương tự tại cổng PIW304 lớn hơn 20000.

Chương trình (OB35)

```
L     PIW304          // Trích mẫu tín hiệu tương tự
T     MW0             // Chuyển vào ô nhớ MW0
L     20000           // So sánh với 20000
<=I
BEC
```

```
CALL  SFC 39              // Huỷ ngắt
      MODE   :  =B#16#1
      OB_NR  :  =30       // OB30 là OB đầu tiên của nhóm tín hiệu ngắt theo chu kỳ thời gian.
      RET_VAL:  =MW10
BE
```

Hàm SFC40 (EN_IRT)

Hàm có tác dụng gỡ bỏ mặt nạ che

- của một tín hiệu ngắt,
- của một nhóm các tín hiệu ngắt hoặc
- của tất cả các tín hiệu ngắt và tín hiệu báo lỗi không đồng bộ.

Khi một tín hiệu ngắt hay báo lỗi không đồng bộ được gỡ bỏ mặt nạ che, hệ thống sẽ gọi khối OB tương ứng chứa chương trình xử lý mỗi khi xuất hiện tín hiệu ngắt, báo lỗi này.

Hàm SFC40 có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	MODE	Byte	Xác định loại tín hiệu ngắt, báo lỗi được bỏ mặt nạ che: B#16#0: Bỏ mặt nạ che cho tất cả các tín hiệu ngắt và tín hiệu báo lỗi không đồng bộ. B#16#1: Bỏ mặt nạ che cho các tín hiệu ngắt, báo lỗi thuộc một nhóm nhất định. Nhóm các tín hiệu nay phải được chỉ thị bởi tên khối OB đầu tiên của nhóm cho biến **OB_NR**. B#16#2: Bỏ mặt nạ che cho một tín hiệu ngắt. Tín hiệu ngắt được bỏ mặt nạ phải được chỉ thị bởi tên khối OB tương ứng cho biến **OB_NR**.
IN	OB_NR	Int	Tên khối OB của tín hiệu ngắt, báo lỗi được bỏ mặt nạ che.
OUT	RET_VAL	Int	Giá trị trả về của hàm: W#16#0000: Hàm làm việc bình thường. W#16#8090: Dữ liệu cho **OB_NR** bị sai. W#16#8091: Dữ liệu cho **MODE** bị sai.

Ví dụ: Đoạn chương trình sau trong OB1 sẽ tích cực lại tín hiệu ngắt theo chu kỳ thời gian nếu có sườn lên của I0.0 hoặc sườn xuống của I0.1.

Chương trình (OB1)

```
      A     I0.0
      FP    M0.0
      O(
      A     I0.1
      FN    M0.1
      )
      JCN   end                  // Không thực hiện SFC40 nếu RLO=0
      CALL  SFC 40               // Kích ngắt
            MODE   :  =B#16#2
            OB_NR  :  =35        // OB35
            RET_VAL:  =MW10
end:  BEU
```

Hàm SFC41 (DIS_AIRT)

Hàm có tác dụng dương mặt nạ che cho tất cả các tín hiệu ngắt, tín hiệu báo lỗi không đồng bộ có thứ tự ưu tiên cao hơn độ ưu tiên của khối OB chứa lệnh gọi hàm này.

Trong một khối chương trình, hàm SFC41 có thể được gọi nhiều lần. Số lần gọi được hệ điều hành đếm và ghi nhận lại dưới dạng tham trị trả về của hàm.

Hàm chỉ có một biến hình thức với tên **RET_VAL** thuộc loại biến **OUT** và có kiểu dữ liệu là **INT**. Tham trị của hàm trả về qua biến này là số lần hàm đã được gọi.

Các tín hiệu ngắt, báo lỗi đã được che sẽ không được hệ điều hành xử lý cho tới khi chúng được tích cực lại nhờ hàm SFC42 (**EN_AIRT**) hoặc khi khối chương trình chứa lệnh gọi hàm đã được thực hiện xong (xem thêm hàm SFC42).

Ví dụ: Phần chương trình sau trong khối OB1 nằm sau lệnh gọi hàm SFC41 sẽ không bị ngắt bởi bất cứ một tín hiệu báo ngắt hoặc tín hiệu báo lỗi không đồng bộ nào.

Chương trình (OB1)

```
A     I0.0
:
:     } Phần chương trình bị ngắt bởi các tín hiệu báo
:     } ngắt và tín hiệu báo lỗi không đồng bộ.

CALL  SFC 41
      RET_VAL: =MW10          // MW10 sẽ có giá trị là 1.
:
:     } Phần chương trình không bị ngắt bởi bất cứ một tín hiệu báo
:     } ngắt hoặc một tín hiệu báo lỗi không đồng bộ.

BE
```

Hàm SFC42 (EN_AIRT)

Hàm có tác dụng bỏ mặt nạ che của tất cả các tín hiệu ngắt, tín hiệu báo lỗi không đồng bộ có thứ tự ưu tiên cao hơn độ ưu tiên của khối OB chứa lệnh gọi hàm này.

Trong một khối chương trình, hàm SFC42 có thể được gọi nhiều lần. Các tín hiệu ngắt, báo lỗi chỉ thực sự được bỏ mặt nạ che nếu số lần gọi hàm SFC42 đúng bằng số lần đã gọi hàm SFC41 trước đó. Ví dụ nếu trước đó ta đã gọi hàm SFC41 3 lần thì để bỏ được mặt nạ che ngắt cho các tín hiệu báo ngắt, báo lỗi không đồng bộ, ta cũng phải gọi hàm SFC42 đúng 3 lần.

Hàm chỉ có một biến hình thức với tên **RET_VAL** thuộc loại biến **OUT** và có kiểu dữ liệu là **INT**. Tham trị của hàm trả về qua biến này là:

- số lần hàm còn cần phải gọi để có thể thực sự bỏ được mặt nạ che của tất cả các tín hiệu ngắt, tín hiệu báo lỗi không đồng bộ,
- hoặc là giá trị **W#16#8080** nếu hàm SFC42 được gọi khi tất cả các tín hiệu ngắt và tín hiệu báo lỗi không đồng bộ đang ở trạng thái tích cực.

Hàm SFC42 không bị ngắt.

Ví dụ: Phần chương trình nằm giữa hai lệnh gọi hàm SFC41 và SFC42 sẽ không bị ngắt bởi bất cứ một tín hiệu báo ngắt hoặc tín hiệu báo lỗi không đồng bộ nào.

Chương trình (OB1)

```
A       I0.0
:   }   Phần chương trình bị ngắt bởi các tín hiệu báo ngắt và tín hiệu báo lỗi
:   }   không đồng bộ.

CALL    SFC 41
        RET_VAL: =MW10          // MW10 sẽ có giá trị là 1.
:   }   Phần chương trình không bị ngắt bởi bất cứ một tín hiệu báo
:   }   ngắt hoặc một tín hiệu báo lỗi không đồng bộ.

CALL    SFC 42
        RET_VAL: =MW12          // MW12 sẽ có giá trị là 0, trong khi MW10 vẫn bằng 1.

:   }   Phần chương trình bị ngắt bởi các tín hiệu báo ngắt và tín hiệu báo lỗi
:   }   không đồng bộ.

CALL    SFC 42
        RET_VAL: =MW14          // MW12 sẽ có giá trị báo lỗi là W#16#8080.
:
BE
```

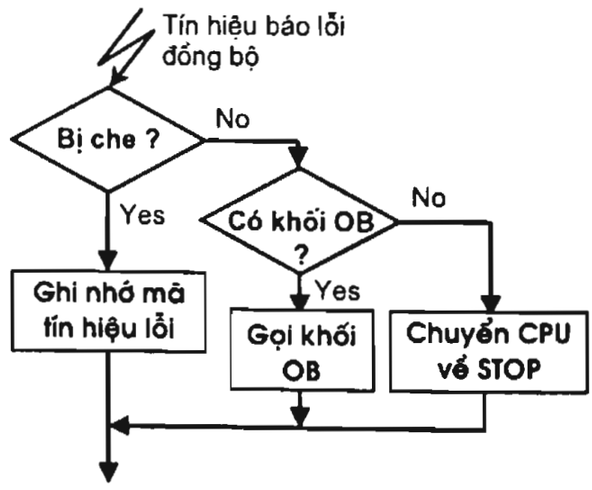
3.5.2 Che và bỏ mặt nạ che tín hiệu báo lỗi đồng bộ – xem thêm 3.4.4

Đối với các tín hiệu báo lỗi đồng bộ ta có 3 hàm hệ thống sau để quy định chế độ làm việc cho chương trình xử lý (khối OB121, OB122):

1) SFC36 (tên hình thức **MSK_FLT**) dùng để đặt mặt nạ che,
2) SFC37 (tên hình thức **DMSK_FLT**) dùng để bỏ mặt nạ che và
3) SFC38 (tên hình thức **READ_ERR**) dùng để đọc nội dung thanh ghi báo lỗi tín hiệu.

Thông thường, khi gặp một tín hiệu báo lỗi đồng bộ (lỗi lập chương trình, lỗi truy nhập module ...), hệ điều hành sẽ kiểm tra tín hiệu đó có bị che ngắt hay không:

- Nếu không bị che, hệ thống sẽ gọi khối OB chứa chương trình xử lý lỗi tương ứng. Khối này do người sử dụng viết tùy theo yêu cầu của bài toán điều khiển. Trong trường hợp không tìm thấy những khối OB này, hệ điều hành sẽ chuyển CPU về trạng thái **STOP**.
- Nếu bị che, hệ thống không gọi OB xử lý lỗi. Mã báo kiểu lỗi được hệ điều hành ghi vào thanh ghi báo lỗi đồng bộ. Nội dung thanh ghi này ta có thể đọc được nhờ hàm SFC38 (**READ_ERR**).



Hình 3.20: Các bước xử lý tín hiệu báo lỗi đồng bộ của hệ điều hành.

Hình 3.20 mô tả các bước hệ điều hành xử lý tín hiệu báo lỗi đồng bộ. Như mục 3.4.4 đã đề cập, lỗi đồng bộ có hai loại:

- loại lỗi lập trình (xử lý bởi OB121) và
- lỗi truy nhập module (xử lý bởi OB122),

trong đó mỗi loại lại có nhiều kiểu lỗi. Thiết nghĩ sẽ là cần thiết cho việc sử dụng tốt các hàm SFC36, SFC37 nếu ta có một cái nhìn tổng quát về các kiểu lỗi đó.

Các kiểu lỗi lập trình (OB121)

1) **Lỗi biến đổi BCD**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh đổi kiểu dữ liệu có cấu trúc BCD với một chữ số ngoài khoảng 0÷9. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh:

```
L     W#16#2E
BTI               // Lỗi, vì chữ số E nằm ngoài khoảng 0÷9
```

2) **Lỗi đọc ô nhớ ngoài miền cho phép**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh đọc nội dung một ô nhớ có địa chỉ nằm ngoài miền cho phép. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh

```
L     MW300       // Lỗi, vì miền biến cờ (M) chỉ có 256 bytes.
```

3) **Lỗi ghi vào ô nhớ ngoài miền cho phép**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh ghi vào một ô nhớ có địa chỉ nằm ngoài miền cho phép. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh

```
T     MW320       // Lỗi, vì miền biến cờ (M) chỉ có 256 bytes.
```

4) **Lỗi đọc ô nhớ có địa chỉ sai cấu trúc**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh đọc một ô nhớ nhưng địa chỉ lại sai cấu trúc. Thông thường việc truy nhập ô nhớ có cấu trúc địa chỉ sai mà không bị phát hiện sai cú pháp khi soạn thảo là những lệnh truy nhập thông qua con trỏ. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh:

```
LAR1  P#0.0
L     W[AR1,P#0.0]      // Lỗi, vì địa chỉ đúng phải là IW[AR1,P#0.0]
```

5) **Lỗi ghi vào ô nhớ có địa chỉ sai cấu trúc**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh ghi vào một ô nhớ có địa chỉ sai cấu trúc. Thông thường việc truy nhập ô nhớ có cấu trúc địa chỉ sai mà không bị phát hiện sai cú pháp khi soạn thảo là những lệnh truy nhập thông qua con trỏ. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh:

```
LAR1  P#4.0
T     W[AR1,P#0.0]      // Lỗi, vì địa chỉ đúng phải là QW[AR1,P#0.0]
```

6) **Lỗi sai tên Timer**: Tín hiệu báo kiểu lỗi này sẽ xuất hiện khi gặp phải lệnh làm việc với Timer có tên nằm ngoài khoảng mà CPU cho phép. Ví dụ CPU312 chỉ có 128 Timer nên hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu MW0=129 và gặp lệnh

```
SP    T[MW0]            // Lỗi, nếu MW0 có giá trị là 129.
```

7) **Lỗi sai tên Counter**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh làm việc với Counter có tên nằm ngoài khoảng mà CPU cho phép. Ví dụ CPU312 chỉ có 64 Counter nên hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu **MW0**=100 và gặp lệnh

```
CU      C[MW0]          // Lỗi, nếu MW0 có giá trị là 100.
```

8) **Lỗi đọc ô nhớ qua con trỏ có địa chỉ sai**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh đọc nội dung ô nhớ có kích thước bytes, word hay dword nhưng con trỏ lại có phần địa chỉ bit ≠0. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh

```
LAR1    P#M0.2
L       B[AR1,P#0.0]        // Lỗi, vì một bytes được tính từ bit thứ 0.
```

9) **Lỗi ghi vào ô nhớ qua con trỏ có địa chỉ sai**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh ghi vào một ô nhớ có kích thước bytes, word hay dword nhưng con trỏ lại có phần địa chỉ bit ≠0. Ví dụ hệ thống sẽ phát tín hiệu báo lỗi kiểu này nếu gặp lệnh

```
LAR1    P#M0.2
T       B[AR1,P#0.0]        // Lỗi, vì một bytes được tính từ bit thứ 0.
```

10) **Lỗi làm việc với DB**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh ghi dữ liệu vào một DB (share) chỉ cho phép đọc.

11) **Lỗi làm việc với DI**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh ghi dữ liệu vào một DI (instance) chỉ cho phép đọc.

12) **Lỗi mở một DB có tên quá lớn**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh mở một khối DB (share) có tên nằm ngoài miền mà CPU cho phép.

13) **Lỗi mở một DI có tên quá lớn**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh mở một khối DI (instance) có tên nằm ngoài miền mà CPU cho phép.

14) **Lỗi gọi một khối FC có tên quá lớn**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh gọi một khối FC có tên nằm ngoài miền mà CPU cho phép.

15) **Lỗi gọi một khối FB có tên quá lớn**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh gọi một khối FB có tên nằm ngoài miền mà CPU cho phép.

16) **Lỗi mở một khối DB không có trong bộ nhớ**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh mở một khối DB nhưng khối này lại chưa được nạp vào bộ nhớ (Load memory) của CPU.

17) **Lỗi gọi một khối FC không có trong bộ nhớ**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh gọi một khối FC nhưng khối này lại chưa được nạp vào bộ nhớ (Load memory) của CPU.

18) **Lỗi gọi một khối FB không có trong bộ nhớ**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống gặp phải lệnh gọi một khối FB nhưng khối này lại chưa được nạp vào bộ nhớ (Load memory) của CPU.

Các kiểu lỗi truy nhập module (OB122)

1) **Lỗi đọc từ module tín hiệu**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống thực hiện lệnh đọc cổng một module tín hiệu (SM) nhưng lại không tìm thấy module này hoặc CPU không thực hiện được sự kết nói với module (time out).
2) **Lỗi ghi ra module tín hiệu**: Tín hiệu báo kiểu lỗi này sẽ được phát khi hệ thống thực hiện lệnh ghi ra cổng một module tín hiệu (SM) nhưng lại không tìm thấy module này hoặc CPU không thực hiện được sự kết nói với module (time out).

Hàm SFC36 (MSK_FLT)

Hàm này có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	PRGFLT_SET_MASK	Dword	Đánh dấu kiểu lỗi lập trình sẽ được che.
IN	ACCFLT_SET_MASK	Dword	Đánh dấu kiểu lỗi truy nhập sẽ được che.
OUT	RET_VAL	Int	Giá trị trả về của hàm: W#16#0000: Không có kiểu lỗi nào vừa được che W#16#0001: Ít nhất có một kiểu lỗi vừa được che.
OUT	PRGFLT_MASKED	Dword	Mã báo các kiểu lỗi lập trình đã được che.
OUT	ACCFLT_MASKED	Dword	Mã báo các kiểu lỗi truy nhập đã được che

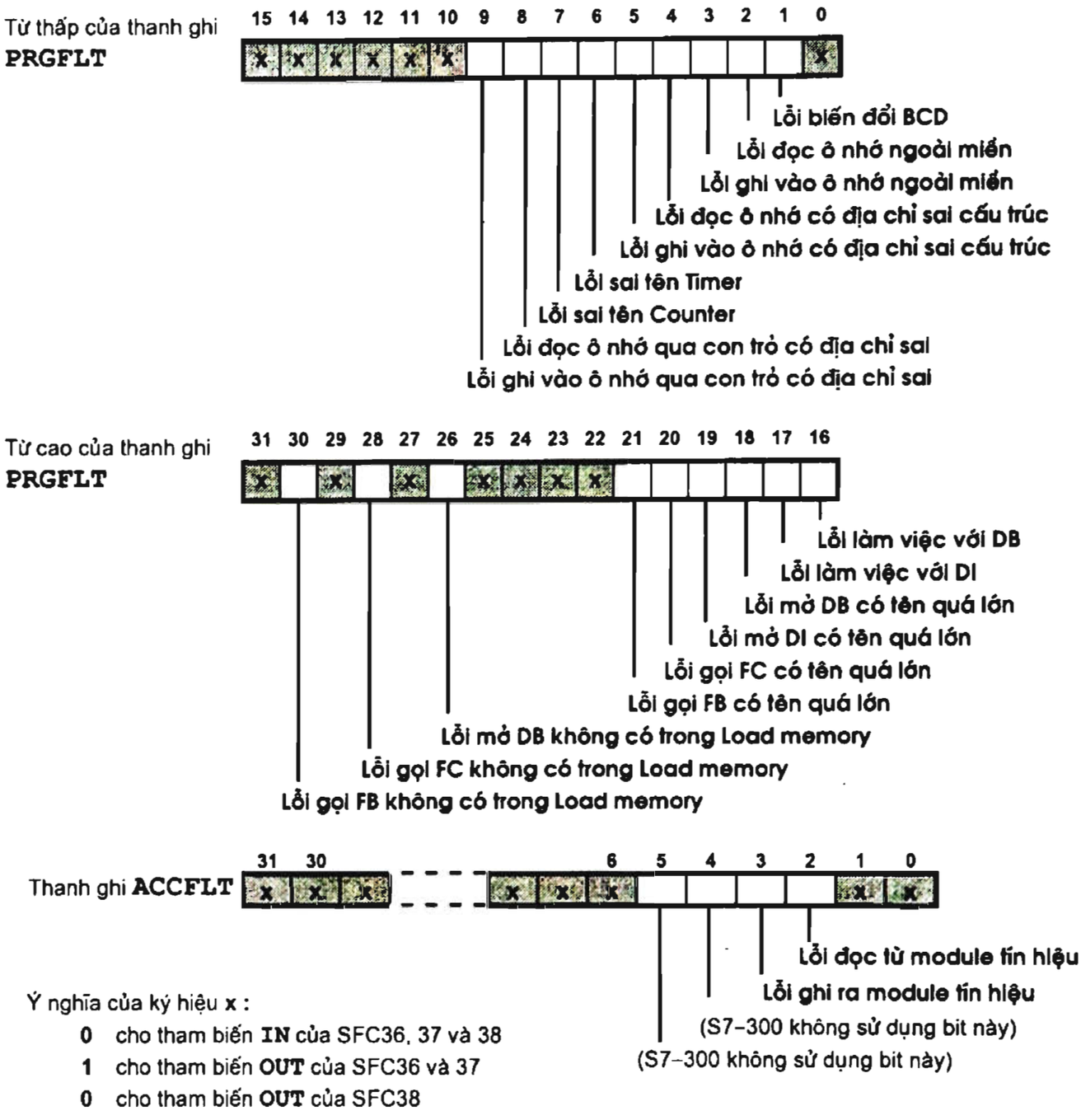
Hàm có tác dụng dương mặt nạ che một số kiểu tín hiệu báo lỗi đồng bộ. Những kiểu tín hiệu báo lỗi đồng bộ cần được dương mặt nạ che phải được đánh dấu vào bit tương ứng của hai tham biến hình thức đầu vào của hàm SFC36:

- Muốn dương mặt nạ che kiểu tín hiệu báo lỗi lập trình nào trong số 18 kiểu lỗi đã nêu thì viết 1 vào bit tương ứng của **PRGFLT_SET_MASK**. Hàm sẽ không thay đổi mặt nạ che những kiểu tín hiệu báo lỗi lập trình mà bit tương ứng có giá trị 0.
- Muốn dương mặt nạ che kiểu tín hiệu báo lỗi truy nhập nào trong số 2 kiểu lỗi đã nêu thì viết 1 vào bit tương ứng của **ACCFLT_SET_MASK**. Hàm sẽ không thay đổi mặt nạ che những kiểu tín hiệu báo lỗi truy nhập mà bit tương ứng có giá trị 0.

Thứ tự các bit tương ứng cho 18 kiểu lỗi lập trình trong thanh ghi **PRGFLT** và cho 2 kiểu lỗi truy nhập trong thanh ghi **ACCFLT** được trình bày ở hình 3.21.

Hàm có hai giá trị trả về là **PRGFLT_MASKED** và **ACCFLT_MASKED** để thông báo những kiểu lỗi đồng bộ nào đã được che mặt nạ và những kiểu tín hiệu nào chưa. Hình thức thông báo của hàm là bit tương ứng trong thanh ghi của kiểu tín kiệu báo lỗi đồng bộ đã được che mặt nạ sẽ có giá trị 1 (xem hình 3.21).

Hàm SFC36 không bị ngắt.



Hình 3.21: Vị trí bit khai báo cho các kiểu tín hiệu báo lỗi đồng bộ.

Hàm SFC37 (DMSK_FLT)

Hàm có tác dụng gỡ bỏ mặt nạ che cho một số kiểu tín hiệu báo lỗi đồng bộ. Những kiểu tín hiệu báo lỗi đồng bộ cần được gỡ bỏ mặt nạ che phải được đánh dấu bằng giá trị logic 1 vào bit tương ứng của hai tham biến hình thức đầu vào của hàm SFC37 là **PRGFLT_RESET_MASK** và **ACCFLT_RESET_MASK**. Vị trí các bits của các kiểu tín hiệu báo lỗi đồng bộ trong hai tham biến này được mô tả ở hình 3.21.

Hàm SFC37 không bị ngắt và có các tham biến hình thức vào–ra như sau:

Loại	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	PRGFLT_RESET_MASK	Dword	Đánh dấu kiểu lỗi lập trình sẽ được bỏ mặt nạ che.
IN	ACCFLT_RESET_MASK	Dword	Đánh dấu kiểu lỗi truy nhập sẽ được bỏ mặt nạ che.
OUT	RET_VAL	Int	Giá trị trả về của hàm: W#16#0000: Tất cả các kiểu lỗi đã được gỡ mặt nạ che. W#16#0001: Ít nhất có một kiểu lỗi không được gỡ bỏ mặt nạ che.
OUT	PRGFLT_MASKED	Dword	Mã báo các kiểu lỗi lập trình đã được che.
OUT	ACCFLT_MASKED	Dword	Mã báo các kiểu lỗi truy nhập đã được che.

Hàm trả về giá trị **PRGFLT_MASKED** và **ACCFLT_MASKED** thông báo những kiểu lỗi đồng bộ nào có mặt nạ che và những kiểu tín hiệu nào không. Hình thức thông báo của hàm là bit tương ứng trong thanh ghi của kiểu tín kiệu báo lỗi đồng bộ có mặt nạ che sẽ có giá trị 1 (xem hình 3.21).

Ví dụ: Để gỡ bỏ mặt nạ che cho các kiểu tín hiệu báo lỗi truy nhập module ta viết:

```
CALL  SFC 37
      PRGFLT_RESET_MASK := DW#16#0
      ACCFLT_RESET_MASK := DW#16#C
      RET_VAL: = MW10
      PRGFLT_MASKED     := MD0
      ACCFLT_MASKED     := MD4
```

Hàm SFC38 (READ_ERR)

Hệ thống có một thanh ghi báo sự xuất hiện lỗi đồng bộ. Mỗi khi xuất hiện một kiểu tín hiệu báo lỗi, cho dù tín hiệu này đã được che hay không, tức là hệ thống có xử lý tín hiệu báo lỗi kiểu đó hay không, mã báo sự xuất hiện kiểu lỗi đó vẫn được hệ điều hành ghi vào bit tương ứng trong thanh ghi báo xuất hiện lỗi. Vị trí các bit trong thanh ghi báo sự xuất hiện lỗi ứng với kiểu tín hiệu lỗi được trình bày trong hình 3.21.

Ta có thể sử dụng hàm SFC38 để đọc nội dung thanh ghi báo sự xuất hiện lỗi. Sau khi đọc nội dung của thanh ghi, hàm SFC38 sẽ xóa luôn nội dung của nó.

Hàm SFC38 không bị ngắt và có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	PRGFLT_QUERY	Dword	Đánh dấu mã báo sự xuất hiện kiểu lỗi lập trình muốn kiểm tra.
IN	ACCFLT_QUERY	Dword	Đánh dấu mã báo sự xuất hiện kiểu lỗi truy nhập muốn kiểm tra.
OUT	RET_VAL	Int	Giá trị trả về của hàm: W#16#0000: Tất cả các kiểu lỗi đã đã được che. W#16#0001: Ít nhất có một kiểu lỗi không được che.
OUT	PRGFLT_CLR	Dword	Các kiểu lỗi lập trình đã xuất hiện.
OUT	ACCFLT_CLR	Dword	Các kiểu lỗi truy nhập module đã xuất hiện.

Hàm trả về giá trị **PRGFLT_CLR** và **ACCFLT_CLR** thông báo những kiểu lỗi đồng bộ đã xuất hiện.

Ví dụ: Để kiểm tra xem module tín hiệu 16 đầu ra số ở slot 5 (địa chỉ Q4.0÷Q5.7) có truy nhập được hay không, ta viết các lệnh sau:

```
      CALL  SFC 36
            PRGFLT_SET_MASK    := DW#16#0
            ACCFLT_SET_MASK    := DW#16#8   // Che kiểu báo lỗi ghi ra module tín hiệu
            RET_VAL: = MW0
            PRGFLT_MASKED      := MD2
            ACCFLT_MASKED      := MD6
      AN    M9.3                            // Kết thúc nếu kiểu báo lỗi ghi ra vẫn chưa được
                                               che
      BEC
      L     B#16#0
      T     PQB4                            // Thử ghi ra module (địa chỉ QB4)
      CALL  SFC 38
            PRGFLT_RESET_MASK := DW#16#0
            ACCFLT_RESET_MASK := DW#16#8    // Kiểm tra xem có lỗi ghi ra module hay không
            RET_VAL: = MW10
            PRGFLT_MASKED      := MD12
            ACCFLT_MASKED      := MD16
      A     BR
      A     M19.3
      =     M255.1                          // M255.1=1 nếu có lỗi ghi ra QB4
      L     B#16#0
      T     PQB5                            // Thử ghi ra module (địa chỉ QB5)
      CALL  SFC 38
            PRGFLT_RESET_MASK := DW#16#0
            ACCFLT_RESET_MASK := DW#16#8    // Kiểm tra xem có lỗi ghi ra module hay không
            RET_VAL: = MW20
            PRGFLT_MASKED      := MD22
            ACCFLT_MASKED      := MD26
      A     BR
      A     M29.3
      =     M255.2                          // M255.2=1 nếu có lỗi ghi ra QB5
      CALL  SFC 37
            PRGFLT_RESET_MASK := DW#16#0
            ACCFLT_RESET_MASK := DW#16#8    // Bỏ che kiểu báo lỗi ghi ra module tín hiệu
            RET_VAL: = MW30
            PRGFLT_MASKED      := MD32
            ACCFLT_MASKED      := MD36
      A     M39.3
      BEC                                   // Kết thúc nếu không bỏ được mặt nạ che
      A     M255.1
      JC    err                             // Có lỗi tại QB4
      L     IB0
      T     QB4                             // Gửi ra được QB4 vì không có lỗi
err:  A     M255.2
      JC    end                             // Có lỗi tại QB5
      L     IB1
      T     QB5                             // Gửi ra được QB5 vì không có lỗi
end:  BEU
```

3.5.3 Tích cực và hủy bỏ ngắt thời điểm – xem thêm 3.4.2

Khi tín hiệu báo ngắt theo thời điểm được phát ra, hệ thống sẽ gọi khối chương trình tương ứng OB10÷OB17 để xử lý. Số các khối OB xử lý tín hiệu ngắt tại thời điểm định trước này phụ thuộc vào từng chủng loại module CPU mà ta sử dụng. Ví dụ với CPU314 ta chỉ sử dụng được OB10 có thứ tự ưu tiên là 2.

Để khai báo sử dụng OB10÷OB17, như đã đề cập trong mục 3.4.2, có các công cụ:

1) Xác định thời điểm phát tín hiệu báo ngắt bằng phần mềm **Simatic Manager** hoặc nhờ hàm hệ thống SFC28 có tên hình thức **SET_TINT**.
2) Tích cực ngắt nhờ phần mềm **Simatic Manager** hoặc nhờ hàm hệ thống SFC30 có tên hình thức **ACT_TINT**.
3) Hủy bỏ tín hiệu ngắt đang tích cực nhờ hàm SFC29 với tên hình thức **CAN_TINT**.
4) Xem trạng thái tín hiệu ngắt nhờ hàm SFC31 với tên hình thức **QRY_TINT**.

Trước khi sử dụng các hàm hệ thống trên, khối OB10÷OB17 đã phải có trong Load memory của CPU. Trong quá trình thực hiện, nếu không tìm thấy các khối OB này hệ điều hành sẽ gọi OB85 để xử lý lỗi thiếu khối OB và nếu cũng không tìm thấy khối OB85, nó sẽ chuyển CPU về trạng thái **STOP**.

Khi đã tích cực tín hiệu ngắt tại thời điểm cho trước mà vì một lý do nào đó ta lại chỉnh lại đồng hồ thời gian của CPU thì có thể sẽ gây ra nguy cơ đồng hồ được chỉnh (tiến hoặc lùi) qua thời điểm phát tín hiệu ngắt. Trong trường hợp như vậy hệ điều hành sẽ gọi khối OB80 để thực hiện chương trình xử lý lỗi không đồng bộ về thời gian và nếu cũng không tìm thấy khối OB80, nó sẽ chuyển CPU về trạng thái **STOP**.

Hàm SFC28 (SET_TINT)

Hàm khai báo thời điểm phát cũng như tần suất phát tín hiệu ngắt (một lần, nhiều lần theo từng phút/giờ ...). Hàm SFC28 không bị ngắt và có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	OB_NR	Int	Tên khối OB chứa chương trình xử lý ngắt (ví dụ OB10).
IN	SDT	Date_And_Time	Thời điểm bắt đầu phát tín hiệu ngắt.
IN	PERIOD	Word	Tần suất phát tín hiệu ngắt: W#16#0000: Một lần. W#16#0201: Mỗi phút một lần. W#16#0401: Mỗi giờ một lần. W#16#1001: Mỗi ngày một lần. W#16#1202: Mỗi tuần một lần. W#16#1401: Mỗi tháng một lần. W#16#1801: Mỗi năm một lần.
OUT	RET_VAL	Int	Mã báo trạng thái làm việc của hàm: W#16#0000: Không có lỗi. W#16#8090: Lỗi tên khối OB. W#16#8091: Lỗi về giá trị thời gian. W#16#8092: Lỗi về giá trị tần suất.

Hàm có biến hình thức **SDT** với kiểu dữ liệu đặc biệt là **DT** (**Date_And_Time**) theo cấu trúc

DT#năm-ngày-tháng-giờ:phút:giây.mili giây

Đây là dữ liệu 64 bits không khai báo được trực tiếp. Bắt buộc ta phải sử dụng hàm thư viện của Step7 để tạo ra kiểu dữ liệu này (hàm FC3 với tên **TOD and DATE to DT**), sau đó ghi vào một ô nhớ cùng kiểu trong DB hoặc trong local block của khối chương trình, ví dụ biến loại **TEMP** có tên **START_DATE_TIME** với kiểu dữ liệu **DT**. Khi đó tham trị của **START_DATE_TIME** mới truyền được cho biến **SDT** của SFC28. Ví dụ

```
CALL  SFC 28
      OB_NR:    = 10
      SDT:      = #START_DATE_TIME
      PERIODE:  = W#16#0
      RET_VAL:  = MW100
```

Hàm SFC29 (CAN_TINT)

Hàm hủy bỏ tín hiệu ngắt sẽ được phát tại thời điểm định trước. Khi gọi hàm SFC29, giá trị về thời điểm phát tín hiệu ngắt cũng sẽ bị xóa. Bởi vậy để tích cực lại ngắt ta phải gọi cả hàm SFC28 để đặt lại thời điểm phát tín hiệu ngắt.

Hàm SFC29 không bị ngắt và có các tham biến hình thức vào–ra như sau:

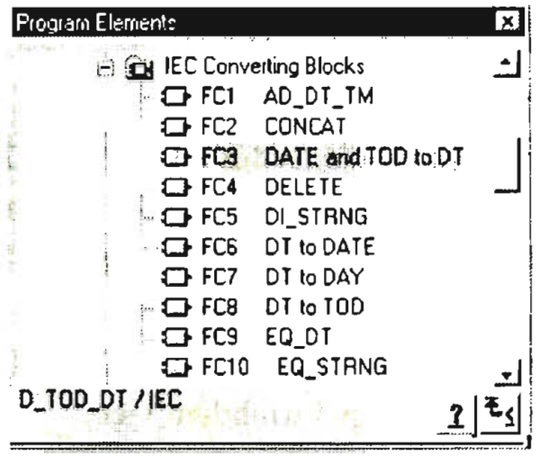
Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	OB_NR	Int	Tên khối OB chứa chương trình xử lý ngắt (ví dụ OB10).
OUT	RET_VAL	Int	Mã báo lỗi khi thực hiện hàm: W#16#0000: Không có lỗi. W#16#8090: Lỗi tên khối OB. W#16#80A0: Không tìm thấy OB.

Hàm SFC30 (ACT_TINT)

Hàm có tác dụng kích hoạt tín hiệu ngắt tại thời điểm định trước đã được khai báo. Mặc dù tín hiệu ngắt đã được khai báo bởi SFC28 nhưng chỉ thực sự được tích cực khi đã sử dụng thêm hàm này. Hàm SFC30 không thể bị ngắt và có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	OB_NR	Int	Tên khối OB chứa chương trình xử lý ngắt (ví dụ OB10).
OUT	RET_VAL	Int	Mã báo lỗi khi thực hiện hàm: W#16#0000: Không có lỗi. W#16#8090: Lỗi tên khối OB. W#16#80A0: Thời điểm phát tín hiệu ngắt chưa được khai báo. W#16#80A1: Đã qua thời điểm phát tín hiệu ngắt. Lỗi này chỉ xuất hiện nếu tín hiệu ngắt được khai báo là phát một lần.

Ví dụ: Khối chương trình FC20 sau sẽ khai báo và tích cực tín hiệu ngắt sẽ được phát một lần vào thời điểm **Start_Time** (giờ/phút/giây) và **Start_Date** (ngày/tháng/năm). Chương trình sử dụng hàm thư viện FC3 của Step7 (hình 3.22) để tạo kiểu dữ liệu hợp lệ cho biến **SDT** của SFC28 bằng cách ghép nội dung hai biến **Start_Time** có kiểu dữ liệu **TOD** và **Start_Date** có kiểu dữ liệu **DATE** lại với nhau. Chi tiết thêm về các hàm thư viện của Step7 sẽ trình bày sau trong chương 4, mục 4.2.5. Khối FC20 này có hai giá trị trả về là những giá trị **RET_VAL** của SFC28 và của SFC30.



Hình 3.22: Sử dụng hàm thư viện của Step7 để tạo dữ liệu kiểu DT.

Local block của FC20

Địa chỉ	Loại	Tên	Kiểu	Giá trị đặt trước
0.0	IN	Start_Time	TOD	
4.0	IN	Start_Date	DATE	
6.0	OUT	Status_SFC28	INT	
8.0	OUT	Status_SFC30	INT	
	IN–OUT			
0.0	TEMP	Start_Date_Time	DATE_ANDTIME	

Chương trình

```
CALL  FC 3
    IN1:      =#Start_date          // Ngày/tháng/năm định phát tín hiệu ngắt
    IN2:      =#Start_Time          // Giờ/phút/giây định phát tín hiệu ngắt
    RET_VAL:=#Start_date_time       // Dữ liệu hợp lệ cho SFC28
CALL  SFC 28                        // Đặt thời điểm phát tín hiệu ngắt
    OB_NR:    = 10
    SDT:      = #START_DATE_TIME
    PERIODE:= W#16#0                // Chỉ phát một lần
    RET_VAL:= #Status_SFC28
CALL  SFC 30                        // Tích cực OB10 xử lý ngắt
    OB_NR:    = 10
    RET_VAL:= #Status_SFC30
```

Hàm SFC31 (QRY_TINT)

Hàm SFC31 có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	OB_NR	Int	Tên khối OB chứa chương trình xử lý ngắt (ví dụ OB10).
OUT	RET_VAL		Mã báo lỗi khi thực hiện hàm: W#16#0000: Không có lỗi. W#16#8090: Không tìm thấy OB.
OUT	STATUS	Word	Thông tin về trạng thái của tín hiệu ngắt.

Hàm SFC31 có tác dụng giúp chương trình ứng dụng kiểm tra được trạng thái của các khối chương trình xử lý tín hiệu báo ngắt theo thời điểm định trước (OB10÷OB17). Kết quả kiểm tra được hàm trả về qua biến **STATUS** với ý nghĩa các bits cho trong bảng bên. Giống như hàm khai báo và tích cực ngắt theo thời điểm, hàm SFC31 không bị ngắt trong quá trình làm việc.

Bit	Giá trị	Giải thích
0	0	Tín hiệu được tích cực bởi hệ điều hành
1	0	Chấp nhận tín hiệu ngắt mới
2	1	Tín hiệu ngắt đã được tích cực
3	-	-
4	1	Đã tìm thấy OB
5	0	Tín hiệu ngắt đã bị hủy.

Ví dụ: Chương trình sau sử dụng hàm SFC31 để kiểm tra trạng thái OB10 của tín hiệu ngắt theo thời điểm. Nếu nó đã bị hủy (Bit0=1) nhưng có khối OB10 trong Load memory và hệ thống cho phép khai báo tín hiệu ngắt khác thì tiếp theo chương trình sẽ gọi FC20 đã được soạn thảo ở ví dụ trước (trang 157) để tích cực lại OB10 cùng tín hiệu ngắt một lần vào thời điểm là ngày 8/12/2000 lúc 4 giờ 30 phút.

```
      CALL  SFC 31
          OB_NR:   =10                          // Kiểm tra trạng thái khối OB10
          RET_VAL:=MW0
          STATUS:  =MW2                         // Kết quả kiểm tra
      A     M3.0                                // Tín hiệu ngắt chưa được tích cực (đã bị hủy)
      AN    M3.1                                // Cho phép khai báo tín hiệu ngắt
      A     M3.4                                // Có khối OB trong Load memory
      JNC   end                                 // Kết thúc nếu không có các điều kiện trên
      CALL  FC  20
          Start_time:    =TOD#04:30:0.0         // Giờ phát tín hiệu
          Start_date:    =D#2000-12-08          // Ngày phát tín hiệu
          Status_SFC28:  =MW4
          Status_SFC30:  =MW6
end:  BEU
```

3.5.4 Thay đổi chế độ làm việc của module mở rộng

Trong mục 3.4.2, khi nói về tín hiệu ngắt cứng với các khối chương trình xử lý ngắt OB40 ÷ OB47 cũng như mục 3.4.4 về việc tích cực chế độ tự chẩn đoán lỗi của những module tín hiệu đặc chủng (module tín hiệu có ngắt cứng, tự chẩn đoán), module CP, module FM, ... ta có đề cập đến khả năng đặt tham số chế độ làm việc từ chương trình ứng dụng thông qua một số hàm trao đổi tham số giữa CPU và module mở rộng, tức là ngay cả khi CPU đang ở trạng thái **RUN**. Với những hàm này chương trình ứng dụng có thể mềm dẻo sử dụng nhiều chế độ làm việc khác nhau của module mở rộng.

Ví dụ 1: Ta có thể lấy việc đọc cổng tương tự làm ví dụ. Chẳng hạn theo yêu cầu bài toán thì có lúc tín hiệu tương tự đầu vào có dạng là dòng, có lúc là áp. Ngay cả khi cố định dạng tín hiệu vào là áp thì có lúc nó là tín hiệu thuộc dải ±10V, có lúc lại nằm trong dải ±2.5V. Điều này chương trình ứng dụng sẽ nhận biết được thông qua tín hiệu số tại cổng I0.0. Nếu I0.0=1 thì dạng tín hiệu tương tự có tại địa chỉ PIW304 là áp trong

dải ±10V, ngược lại khi I0.0=0 thì nó sẽ là điện áp thuộc khoảng ±2.5V. Một số module tương tự của S7-300 cho phép tùy chọn chế độ làm việc với tín hiệu tương tự dạng điện áp hay dòng cũng như dải giá trị tín hiệu, ±10mA, ±5mA, ±3.2mA .... Tuy nhiên nếu chỉ sử dụng **Simatic Manager** để quy định chế độ làm việc cho module tương tự đó thì trong suốt quá trình thực hiện chương trình hoặc ta chỉ có thể làm việc hoặc với tín hiệu thuộc dải ±10V, hoặc ±2.5V mà không thay đổi theo I0.0 như yêu cầu bài toán đặt ra. Ở đây để giải quyết được yêu cầu bài toán, bắt buộc phải thay đổi chế độ làm việc của module tương tự từ chương trình ứng dụng và điều này chỉ thực hiện được nếu ta sử dụng hàm trao đổi dữ liệu cấu hình để truyền tham số đặt chế độ làm việc mới từ chương trình ứng dụng tới module tương tự. ❐

Các hàm cho phép ta từ chương trình ứng dụng truy nhập vào thanh ghi xác định chế độ làm việc của module mở rộng bao gồm:

- SFC55 (tên hình thức **WR_PARM**) để ghi tham số đặt cấu hình cho module. Tham số cấu hình này được người sử dụng tự xây dựng và tổ chức thành một khối dữ liệu (DB) có cấu trúc phù hợp với chủng loại module.
- SFC56 (tên hình thức **WR_DPARM**) để sửa đổi một vài tham số cấu hình của module. Những tham số này phải được lấy từ các khối dữ liệu đã có của hệ thống (SDB100÷SDB103).
- SFC57 (tên hình thức **PARM_MOD**) để sửa đổi toàn bộ tham số cấu hình của module. Tất cả các tham số này phải được lấy từ các khối dữ liệu đã có của hệ thống (SDB100÷SDB103).

Sau đây là một số chế độ làm việc của các module mở rộng có thể thay đổi được từ chương trình ứng dụng:

a) Phát tín hiệu ngắt cứng (OB40÷OB47) từ module tín hiệu vào/ra số. Tín hiệu ngắt cứng được phát khi xuất hiện sườn lên hoặc xuống của tín hiệu logic tại cổng vào của module.

b) Tự chẩn đoán lỗi của module vào/ra số, của module vào/ra tương tự (OB82).

c) Quy định lại dạng tín hiệu tương tự (áp hay dòng) có tại cổng vào tương tự của module.

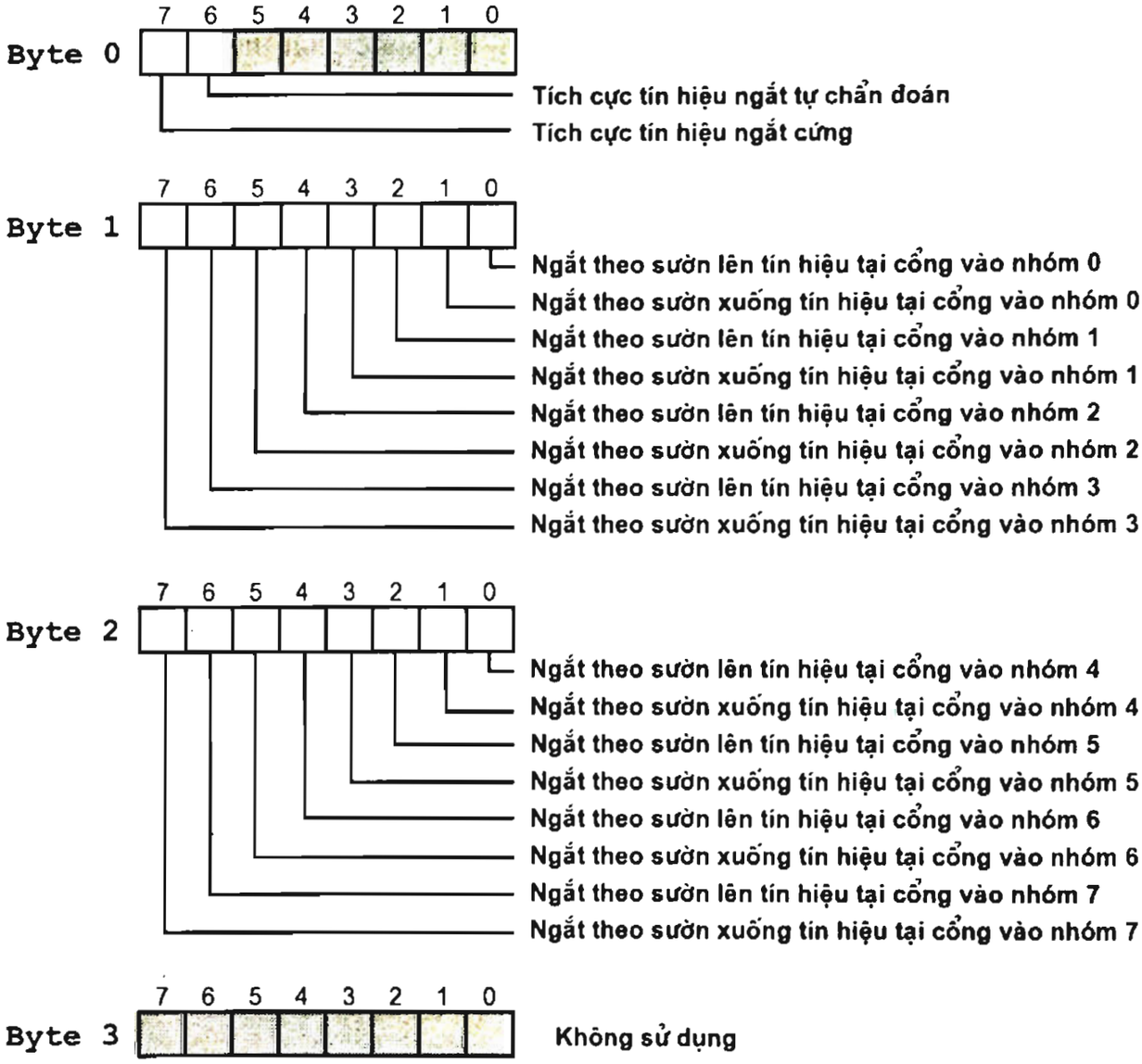
d) Xác định lại dải tín hiệu cho phép của tín hiệu tương tự.

e) Phát tín hiệu báo ngắt nếu tín hiệu tương tự đầu vào nằm ngoài khoảng đã được định nghĩa. Khoảng giá trị được định nghĩa này tất nhiên phải nằm trong dải tín hiệu cho phép, chẳng hạn dải tín hiệu là ±10V được chuyển đổi sang toàn bộ khoảng biến đổi của số nguyên 16 bits là từ –32768 đến 32767 nhưng ta có thể quy định lại là chúng chỉ được phép nằm trong khoảng từ khoảng –20000 đến 20000.

Bên cạnh những chế độ vừa nêu còn có một vài chế độ làm việc khác của module mở rộng có thể được thay đổi bởi chương trình ứng dụng. Bạn đọc có thể tham khảo thêm chúng trong tài liệu [12].

Ngoài ra, ở đây chúng tôi cũng sẽ chỉ tập trung vào các chế độ làm việc cho phép người sử dụng tự xây dựng tham số đặt cấu hình dưới dạng các khối dữ liệu DB phù hợp với module và ghi vào module nhờ hàm SFC55 (**WR_PARM**).

Cấu trúc khối DB chứa tham số đặt chế độ làm việc cho module vào số (DI) gồm 4 bytes với cấu trúc từng byte như sau:



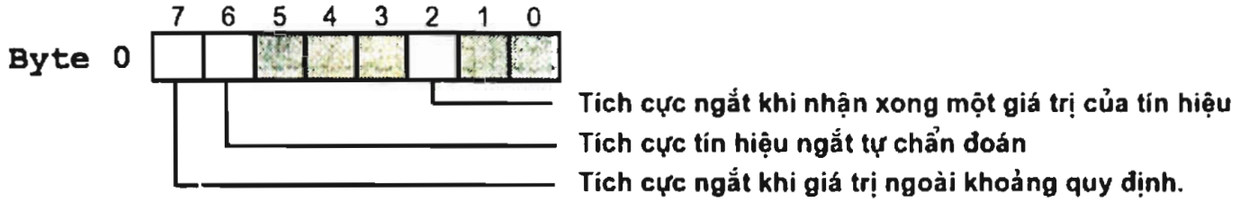
Các cổng vào/ra của module DI/DO được phân chia thành từng nhóm. Việc phân chia nhóm các cổng vào/ra phụ thuộc từng loại module được sử dụng và có thể tra cứu trong [11].

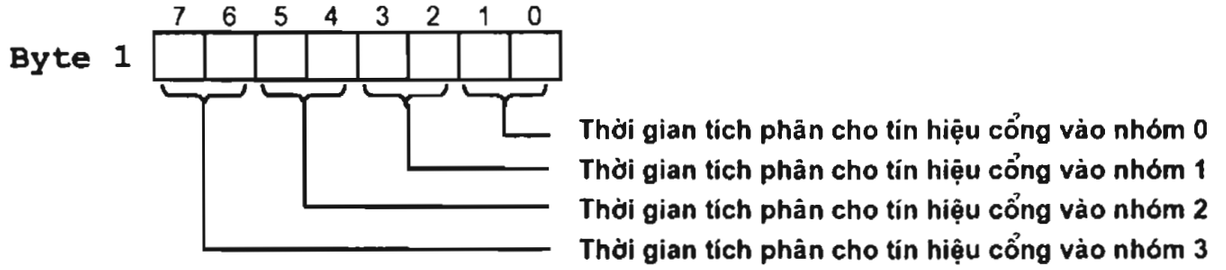
Ví dụ 2: Khối dữ liệu DB45 ở hình bên chứa tham số (data record) đặt cấu hình cho module DI nằm ở slot 4 để module này có khả năng tự chẩn đoán lỗi (OB82) và phát tín hiệu ngắt cứng (OB40) khi có sườn lên của tín hiệu số tại cổng vào I0.0 (thuộc nhóm 0).

DB45

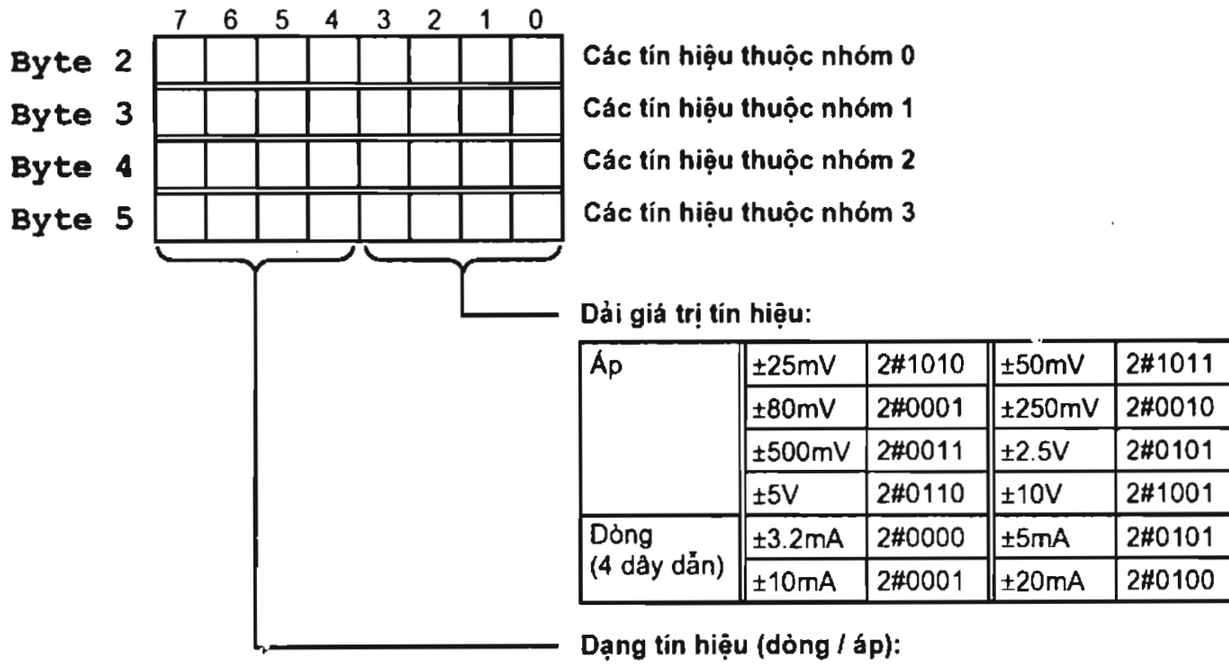
Address	Name	Type	Initial Value
0.0		STRUCT	
+0.0	En	Byte	B#16#C0
+1.0	Flag0_3	Byte	B#16#1
+2.0	Flag4_7	Byte	B#16#0
+3.0	Unused	Byte	B#16#0
=4.0		END_STRUCT	

Cấu trúc khối DB chứa tham số đặt chế độ làm việc cho module vào tương tự (AI) có khả năng tự chẩn đoán gồm 14 bytes với cấu trúc từng byte như sau:

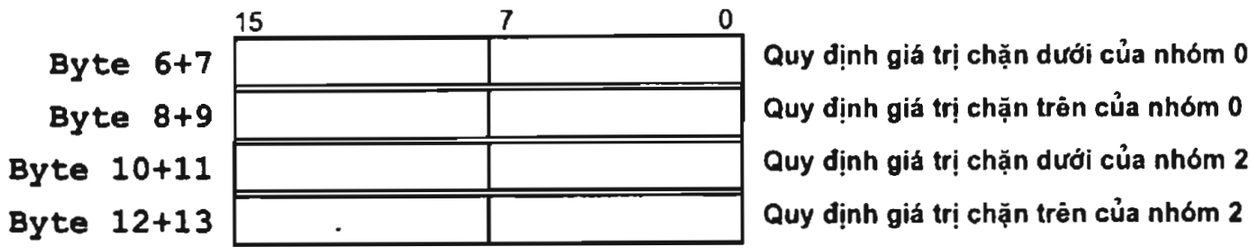




Thời gian	Mã hiệu	Thời gian	Mã hiệu
2.5ms	2#00	20ms	2#10
16.7ms	2#01	100ms	2#11



Áp	±25mV	2#1010	±50mV	2#1011
	±80mV	2#0001	±250mV	2#0010
	±500mV	2#0011	±2.5V	2#0101
	±5V	2#0110	±10V	2#1001
Dòng (4 dây dẫn)	±3.2mA	2#0000	±5mA	2#0101
	±10mA	2#0001	±20mA	2#0100



Các cổng vào của module AI được phân chia thành từng nhóm. Việc phân chia nhóm các cổng vào phụ thuộc từng loại module và có thể tra cứu trong [11]. Ngoài ra còn có nhiều loại module AI có khả năng làm việc với những dạng tín hiệu khác nữa như tín hiệu dòng 2 dây dẫn, tín hiệu điện trở 4 dây/2 dây, tín hiệu điện trở nhiệt .... Bạn đọc có thể tham khảo các mã hiệu cho những dạng tín hiệu này trong tài liệu [12].

Ví dụ 3: Khối dữ liệu DB55 ở hình bên chứa tham số (data record) đặt cấu hình cho module AI (SM331) với 8 đầu vào nằm ở slot 7 (địa chỉ bắt đầu là 304) để quy định cho module làm việc với chế độ:

DB55

Address	Name	Type	Initial Value
0.0		STRUCT	
+0.0	En	Byte	B#16#64
+1.0	Int_Time	Byte	B#16#FF
+2.0	Input0	Byte	B#16#19
+3.0	Input1	Byte	B#16#19
+4.0	Input2	Byte	B#16#16
+5.0	Input3	Byte	B#16#16
+6.0	ULV_0	Word	W#16#7FFF
+8.0	LLV_0	Word	W#16#8000
+10.0	ULV_2	Word	W#16#7FFF
+12.0	LLV_2	Word	W#16#8000
=14.0		END_STRUCT	

- Dạng tín hiệu vào tương tự thuộc nhóm 0 và nhóm 1 là điện áp trong dải ±10V.
- Dạng tín hiệu vào tương tự thuộc nhóm 2 và nhóm 3 là điện áp trong dải ±5V.
- Giá trị nhận được tại cổng vào được quy định là một số nguyên 16 bits, tức là toàn bộ khoảng từ –32768 đến 32767.
- Thời gian tích phân để xác định một giá trị tín hiệu được quy định là 100ms cho tất cả 4 nhóm (8 đầu vào).
- Phát tín hiệu ngắt tự chẩn đoán khi có lỗi của module (OB82).

Hàm SFC55 (WR_PARM)

Hàm SFC55 có tác dụng truyền tham số đặt cấu hình cho module mở rộng. Tham số này đã phải được tổ chức thành những khối dữ liệu (DB), chẳng hạn như các khối DB45, DB55 vừa được giới thiệu trong ví dụ 2 và ví dụ 3. Quá trình truyền tham số này là không đồng bộ, nói cách khác trong thời gian truyền tham số, module không có khả năng giao tiếp với ngoại vi, bởi vậy chương trình ứng dụng cần phải đợi cho tới khi quá trình truyền tham số được hoàn tất mới có thể sử dụng được module và để làm được điều này, hàm SFC55 có biến **BUSY** thông báo lúc nào hoàn thành quá trình truyền tham số.

Hàm SFC55 có các tham biến hình thức vào–ra như sau:

Loại biến	Tên biến	Kiểu dữ liệu	Ý nghĩa
IN	REQ	Bool	Tên khối OB chứa chương trình xử lý ngắt (ví dụ OB10).
IN	IOID	Byte	Vùng truy nhập: B#16#54: Vào. B#16#55: Ra.
IN	LADDR	Word	Địa chỉ của module
IN	RECNUM	Byte	Chỉ số của data record. Ở đây, nếu tham số cấu hình được người sử dụng tự tổ chức thì RECNUM phải là 1.
IN	RECORD	Any	Tên khối dữ liệu chứa tham số và độ dài (tính theo bytes).
OUT	RET_VAL		Mã báo lỗi khi thực hiện hàm (xem bảng dưới).
OUT	BUSY	Bool	Báo tham số đang được truyền.

REQ: Yêu cầu gửi.

Hàm chỉ thực sự bắt đầu công việc gửi tham số tới module nếu **REQ**=1. Tuy nhiên, hàm cũng vẫn sẽ hoàn thành nốt công việc nếu đang trong quá trình gửi mà tín hiệu yêu cầu gửi **REQ** đã trở về 0.

BUSY: Mã hiệu báo đang bận truyền.

Khi hàm chưa hoàn thành xong việc gửi tham số tới module, **BUSY** vẫn có giá trị 1. Việc gửi tham số có thể phải kéo dài vài vòng quét và **BUSY=0** chỉ khi công việc gửi tham số thực sự đã hoàn tất.

RET_VAL: Mã hiệu báo lỗi của hàm

Mã hiệu	Ý nghĩa
W#16#8090	Địa chỉ module nằm ngoài miền định nghĩa
W#16#8091	Địa chỉ module sai cấu trúc
W#16#8092	Kiểu dữ liệu của tham số trong khối DB không phù hợp với loại biến (byte, word, dword).
W#16#8093	Sai giá trị của **IOID**.
W#16#80A1	Không truyền được tham số (đường truyền không thông).
W#16#80B1	Độ dài của **DB** chứa tham số không đúng
W#16#80B3	Module tìm thấy ở slot đã chỉ thị lại khác kiểu.
W#16#80D2	Tham số không đúng cho module.

RECORD: Tên khối DB chứa tham số (data record) và độ dài của nó có cấu trúc

P#DB<chỉ số>.DBX<địa chỉ bit đầu tiên> Byte<độ dài>

Ví dụ: **55** Ví dụ: **DBX0.0** Ví dụ: **14**

Ví dụ 4: Quay lại bài toán đã nêu ở ví dụ 1. Module AI với 8 cổng vào tương tự nằm tại slot 7 có địa chỉ module là 304 (địa chỉ các cổng vào là PIW304÷PIW319). Bài toán đặt ra là phụ thuộc vào mức giá trị logic của I0.0, dải các tín hiệu tương tự dạng điện áp tại cổng thuộc nhóm 0 sẽ có lúc là ±10V (I0.0=1) hoặc ±2.5V (I0.0=0). Các tín hiệu ở cổng thuộc nhóm 1 luôn có dạng điện áp với dải biến đổi cố định là ±10V và dạng tín hiệu của các cổng thuộc nhóm 2, 3 cũng luôn cố định là áp với dải biến đổi ±5V. Ngoài ra, thời gian tích phân cho việc nhận giá trị của tất cả các cổng được chọn là 100ms, chế độ phát tín hiệu ngắt tự chẩn đoán module phải được tích cực và giá trị tương tự của tín hiệu luôn được biến đổi thành số nguyên 16 bits trong toàn bộ khoảng từ –32768 đến 32767.

Trong ví dụ 3 ta đã xây dựng khối dữ liệu DB55 chứa tham số đặt cấu hình cho module. Theo yêu cầu bài toán, giá trị của ô nhớ **INPUT0** sẽ không cố định là **B#16#19** mà thay đổi theo mức của I0.0. Cụ thể là khi I0.0=1 thì **INPUT0=B#16#19** và khi I0.0=0 thì **INPUT0=B#16#15**.

Hàm SFC55 được sử dụng để truyền tham số trong DB55 tới module AI. Hàm được gọi khi xuất hiện sườn của I0.0 và quá trình truyền tham số trước đó đã kết thúc.

```
      A     M0.0              // M0.0=1: quá trình truyền tham số cũ chưa kết thúc.
      JC    tran
      A     I0.0
      FP    M0.1              // Có sườn lên của tín hiệu tại cổng I0.0.
      =     M0.3              // Điều kiện để thực hiện hàm SFC55 là M0.3=1.
      JCN   next
      L     B#16#19           // Thay đổi giá trị ô nhớ INPUT0 về dải ±10V
      JU    writ
next: A     I0.0
      FN    M0.2              // Có sườn xuống của tín hiệu tại cổng I0.0.
      =     M0.3              // Điều kiện M0.3=1 cho hàm SFC55.
      JCN   end
      L     B#16#15           // Thay đổi giá trị ô nhớ INPUT0 về dải ±2.5V
writ: T     DB55.DBB2
tran: CALL  SFC 55
         REQ:      = M0.3          // Điều kiện để hàm SFC55 bắt đầu thực hiện công việc là M0.3=1.
         IOID:     = B#16#54       // Truy nhập module vào tương tự (AI)
         LADDR:    = 304           // Địa chỉ module
         RECNUM:   = 1
         RECORD:   = P#DB55.DBX0.0 BYTE14
         RET_VAL:= MW2
         BUSY:     = M0.0
end:  BEU
```

4 HƯỚNG DẪN SỬ DỤNG STEP7

4.1 Cài đặt Step7 và chọn chế độ làm việc

Step7 là một phần mềm hỗ trợ:

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

Ngoài ra Step7 còn có cả một thư viện đầy đủ với các hàm chuẩn hữu ích, phần trợ giúp online rất mạnh có khả năng trả lời mọi câu hỏi của người sử dụng về cách sử dụng Step7, về cú pháp lệnh trong lập trình, về xây dựng cấu hình cứng của một trạm cũng như của một mạng gồm nhiều trạm PLC ...

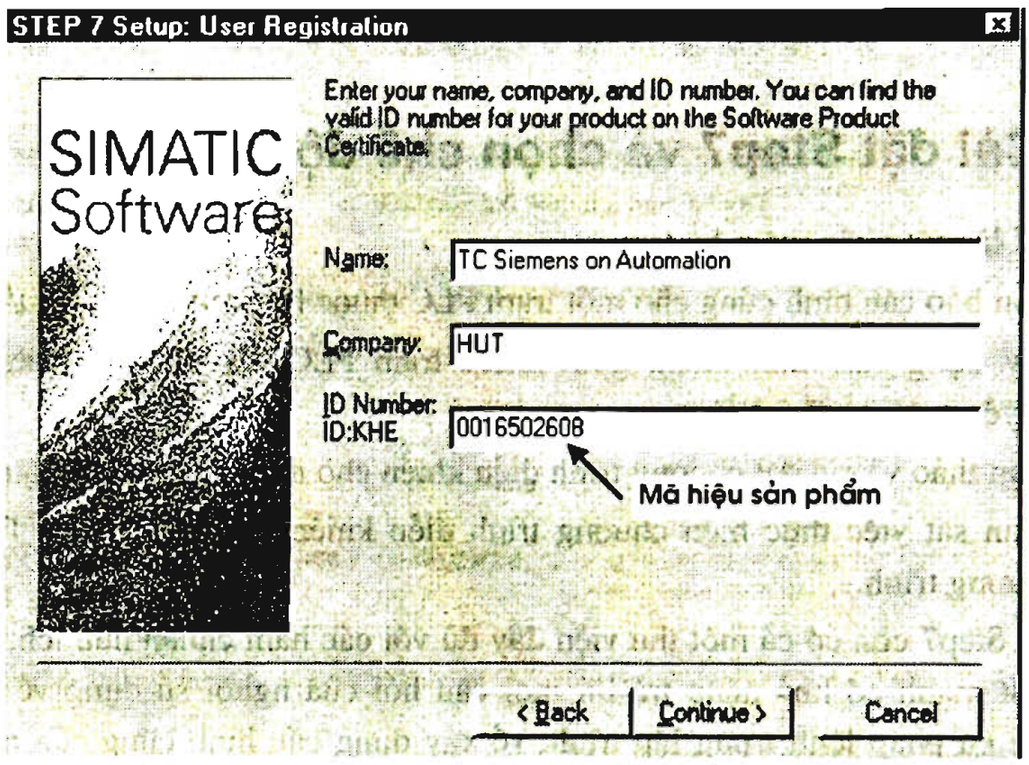
4.1.1 Cài đặt Step7

Có nhiều phiên bản của bộ phần mềm gốc của Step7 hiện có tại Việt nam. Đang được sử dụng nhiều nhất là phiên bản (version) 4.2 và 5.0. Trong khi phiên bản 4.2 khá phù hợp với những PC có cấu hình trung bình (CPU 80586, 8MB RAM, 90MB còn trống trong ổ cứng, màn hình VGA) nhưng lại đòi hỏi tuyệt đối phải có bản quyền thì phiên bản 5.0, mặc dù đòi hỏi cấu hình PC phải mạnh tốc độ nhanh nhưng không đòi hỏi phải có bản quyền một cách tuyệt đối, tức là phiên bản này vẫn chạy (ở mức hạn chế) khi không có bản quyền.

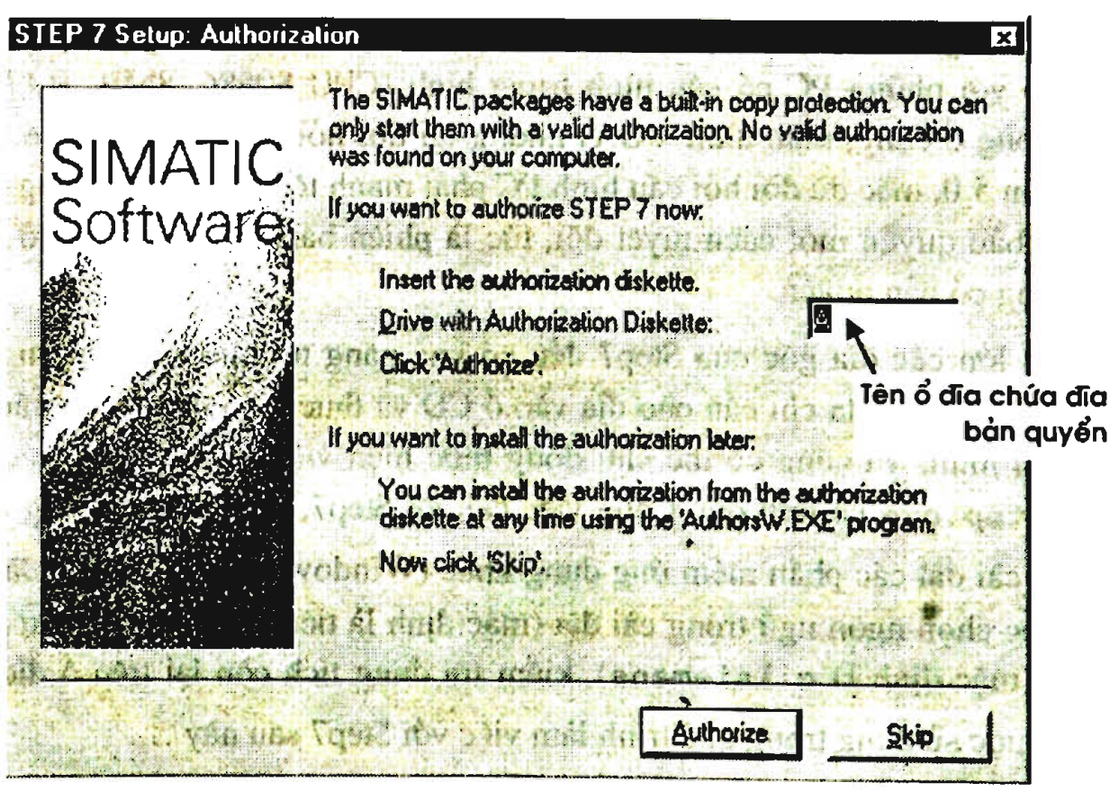
Phần lớn các đĩa gốc của Step7 đều có khả năng tự thực hiện chương trình cài đặt (autorun). Bởi vậy ta chỉ cần cho đĩa vào ổ CD và thực hiện theo đúng các chỉ dẫn hiện trên màn hình. Ta cũng có thể chủ động thực hiện việc cài đặt bằng cách gọi chương trình **`setup.exe`** có trên đĩa. Công việc cài đặt Step7, về cơ bản, không khác nhiều so với việc cài đặt các phần mềm ứng dụng khác (Window, Office ...), tức là cũng bắt đầu bằng việc chọn ngôn ngữ trong cài đặt (mặc định là tiếng Anh), chọn thư mục đích trên ổ cứng (mặc định là **`c:\siemens`**), kiểm tra dung tích còn lại trên ổ đích, chọn ngôn ngữ sẽ được sử dụng trong quá trình làm việc với Step7 sau này ....

Tuy nhiên, so với các phần mềm khác thì việc cài đặt Step7 sẽ có vài điểm khác biệt cần phải được giải thích rõ thêm:

1) **Khai báo mã hiệu sản phẩm**: Mã hiệu sản phẩm luôn đi kèm theo phần mềm Step7 và được in ngay trên đĩa chứa bộ cài Step7. Khi trên màn hình hiện ra cửa sổ yêu cầu cho biết mã hiệu sản phẩm, ta phải điền đầy đủ vào tất cả các mục của ô cửa sổ đó, kể cả tên và địa chỉ người sử dụng. Sau đó ấn phím **continue** để tiếp tục.



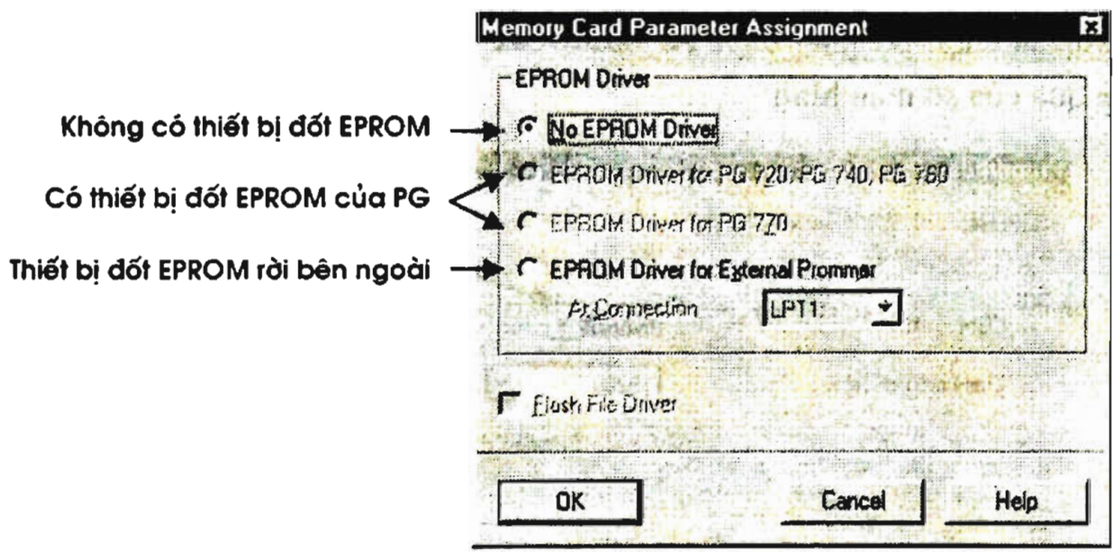
2) **Chuyển bản quyền**: Bản quyền của Step7 nằm trên một đĩa mềm riêng (thường có mầu vàng hoặc đỏ). Trong quá trình cài đặt, trên màn hình sẽ xuất hiện cửa sổ yêu cầu chuyển bản quyền sang ổ đích (mặc định là **c:**) có dạng như sau:



Ta có thể chuyển bản quyền từ đĩa mềm sang ổ **C:** ngay trong khi cài đặt Step7 bằng cách cho đĩa bản quyền vào ổ **A:** rồi ấn phím **Authorize**. Ta cũng có thể bỏ qua và sẽ chuyển bản quyền sau vào lúc khác bằng cách ấn phím **Skip**. Trong trường hợp bỏ qua thì sau này, lúc chuyển bản quyền, ta phải sử dụng chương trình chuyển bản quyền có tên **AuthorsW.EXE** cũng có trên đĩa bản quyền (ver. 4.2) hoặc có cùng trong đĩa CD với phần mềm Step7 gốc (ver. 5.0).

Chú ý rằng đĩa mềm chứa bản quyền (Author disk) đã được bảo vệ cấm sao chép. Cho dù bản quyền đã được chuyển từ đĩa mềm sang ổ cứng và trên đĩa mềm không còn bản quyền, nhưng nó vẫn là một đĩa đặc biệt có chỗ chứa bản quyền. Bản quyền được sao chép sang ổ cứng sẽ nằm trong thư mục **Ax nf zz**. Nếu thư mục này bị hỏng, ta sẽ mất bản quyền. Bởi vậy, mỗi khi muốn cài đặt lại hệ thống hay dọn dẹp lại đĩa cứng thì trước đó phải rút bản quyền ra khỏi ổ **C:** và chuyển nó ngược trở về đĩa mềm Author cũng bằng chương trình **AuthorsW.EXE**.

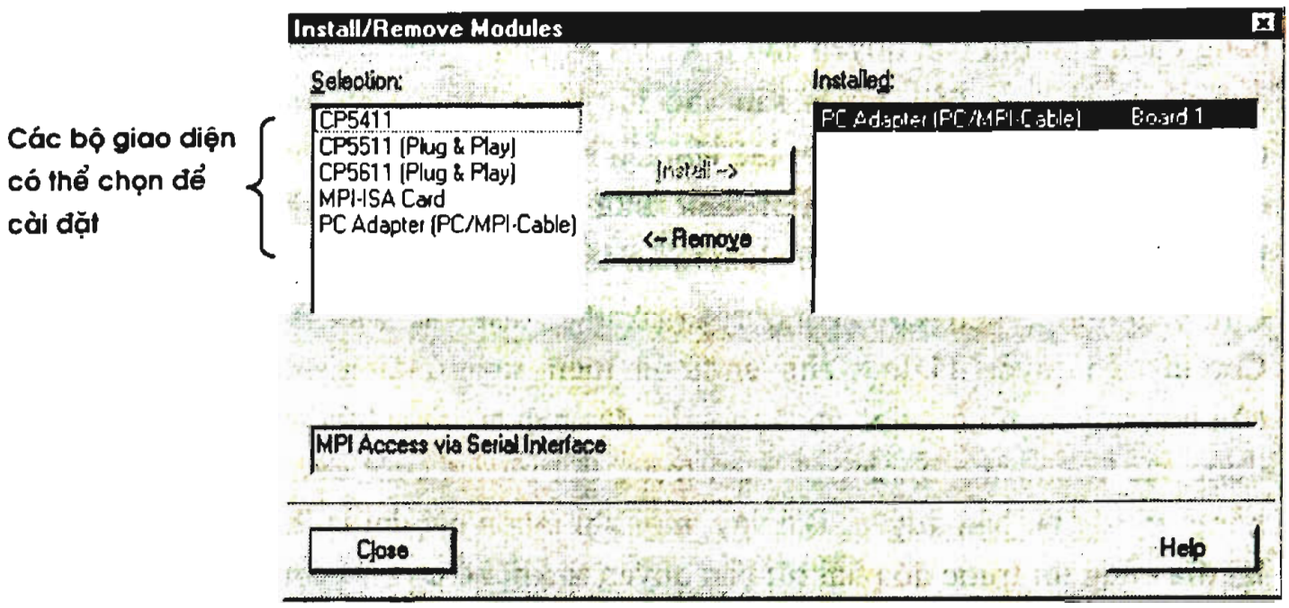
3) **Khai báo thiết bị đốt EPROM**: Chương trình Step7 có khả năng đốt chương trình ứng dụng lên thẻ EPROM cho PLC. Nếu máy tính PC của ta có thiết bị đốt EPROM thì cần phải thông báo cho Step7 biết khi trên màn hình xuất hiện cửa sổ:



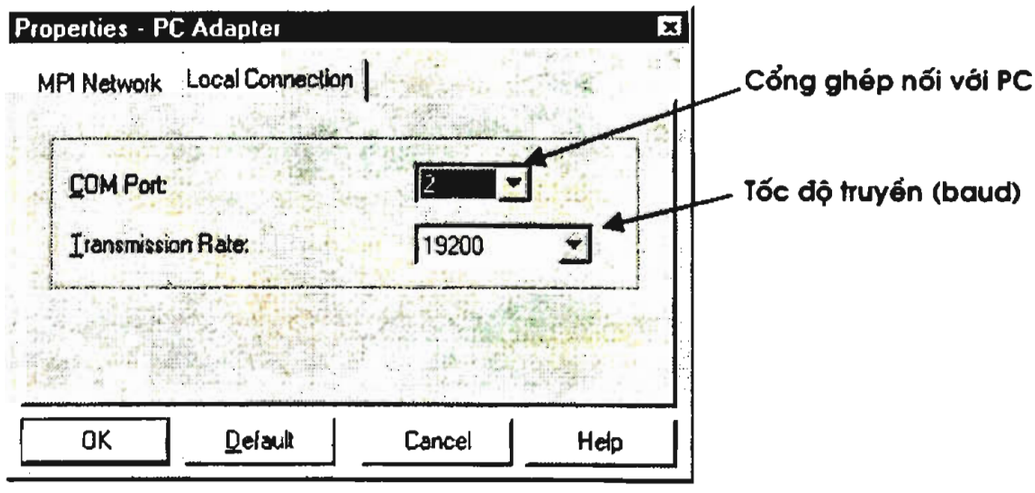
4) **Chọn giao diện PC/PLC**: Chương trình Step7 được cài đặt trên PC hoặc PG để hỗ trợ việc soạn thảo cấu hình cứng cũng như chương trình cho PLC, tức là sau đó toàn bộ những gì đã soạn thảo sẽ được dịch và chuyển sang PLC. Không những thế, Step7 còn tạo khả năng quan sát việc thực hiện chương trình của PLC. Muốn như vậy ta cần phải có bộ giao diện ghép nối giữa PC với PLC để truyền thông tin, dữ liệu.

Step7 có thể được ghép nối với PLC qua nhiều bộ giao diện khác nhau như qua thẻ MPI, qua bộ chuyển đổi PC/PPI, qua thẻ PROFIBUS (CP) ... nhưng chúng phải được khai báo sử dụng.

Ngay sau khi Step7 vừa được cài đặt xong, trên màn hình xuất hiện cửa sổ thông báo cho ta chọn các bộ giao diện sẽ được sử dụng. Cửa sổ này có dạng như sau:



Muốn chọn bộ giao diện nào, ta đánh dấu bộ giao diện đó ở ô phía bên trái rồi ấn phím **Install**. Những bộ giao diện đã được chọn sẽ được ghi lại vào ô phía phải. Sau khi chọn xong các bộ giao diện sử dụng, ta còn phải đặt tham số làm việc cho những bộ giao diện đó bao gồm tốc độ truyền, cổng ghép nối với máy tính ... Chẳng hạn khi đã chọn bộ giao diện **PC Adapter** ta phải đặt tham số làm việc cho nó thông qua cửa sổ màn hình:



4.1.2 Đặt tham số làm việc

Sau khi cài đặt xong Step7, trên màn hình (desktop) sẽ xuất hiện biểu tượng (icon) của nó như ở hình bên. Đồng thời trong Menu của Window cũng có thư mục **Simatic** với tất cả các tên của những thành phần liên quan, từ các phần mềm trợ giúp đến các phần mềm đặt cấu hình, chế độ làm việc của Step7 ....

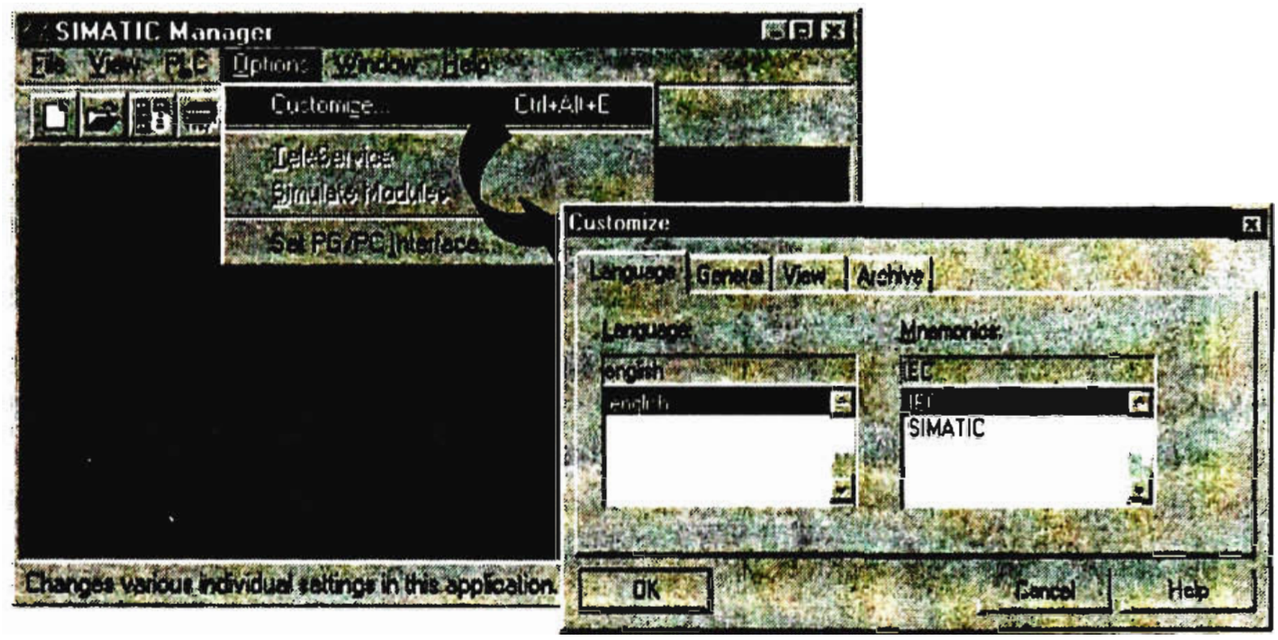

SIMATIC Manager

Biểu tượng của Step7

Khi vừa được cài đặt, Step7 có cấu hình mặc định về chế độ làm việc của **Simatic**, chẳng hạn như cú pháp các lệnh lại được

viết theo tiếng Đức như **JU** được viết thành **SPA**, **JC** thành **SPB**, **CAD** thành **TAD** .... Muốn chuyển về dạng thông dụng quốc tế ta phải đặt lại cấu hình cho Step7.

Để làm việc này, trước hết ta phải vào Step7 bằng cách nháy kép phím chuột trái tại biểu tượng của nó. Trên màn hình sẽ xuất hiện cửa sổ chính của Step7. Chọn tiếp **Option→Customize→Language→IEC** rồi ấn phím OK (xem hình dưới).



Tất nhiên, bên cạnh việc chọn ngôn ngữ cho cú pháp lệnh ta còn có thể sửa đổi nhiều chức năng khác của Step7 như nơi sẽ chứa chương trình trên đĩa cứng, những thanh ghi sẽ được hiển thị nội dung khi gỡ rối chương trình ..., song các việc đó không ảnh hưởng quyết định tới việc sử dụng Step7 theo thói quen của ta như ngôn ngữ cú pháp lệnh.

4.2 Soạn thảo một Project

Khái niệm Project trong Simatic không đơn thuần chỉ là chương trình ứng dụng mà rộng hơn bao gồm tất cả những gì liên quan đến việc thiết kế phần mềm ứng dụng để điều khiển, giám sát một hay nhiều trạm PLC. Theo khái niệm như vậy, trong một Project sẽ có:

a) bảng cấu hình cứng về tất cả các module của từng trạm PLC,

b) bảng tham số xác định chế độ làm việc cho từng module của mỗi trạm PLC,

c) các logic block chứa chương trình ứng dụng của từng trạm PLC,

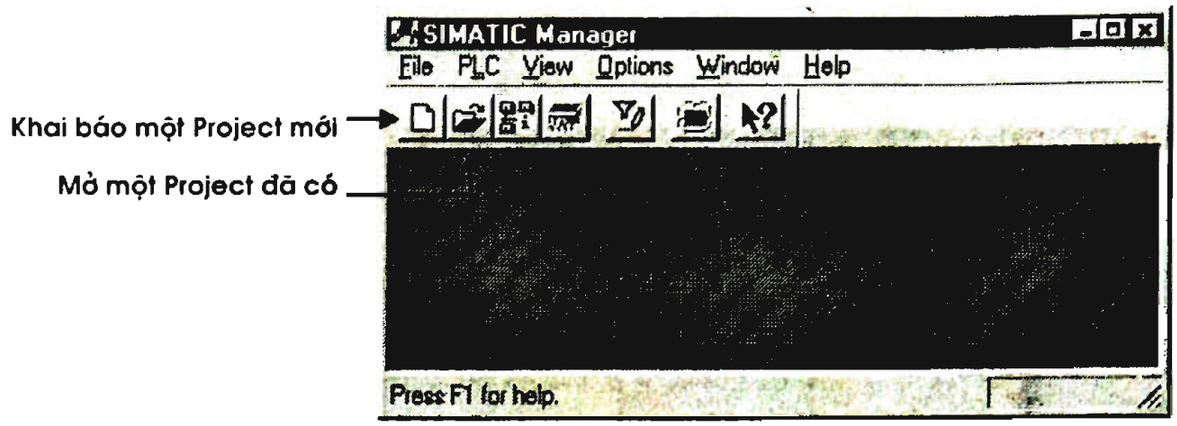
d) cấu hình ghép nối và truyền thông giữa các trạm PLC,

e) các màn hình giao diện phục vụ việc giám sát toàn bộ mạng hoặc giám sát từng trạm PLC của mạng.

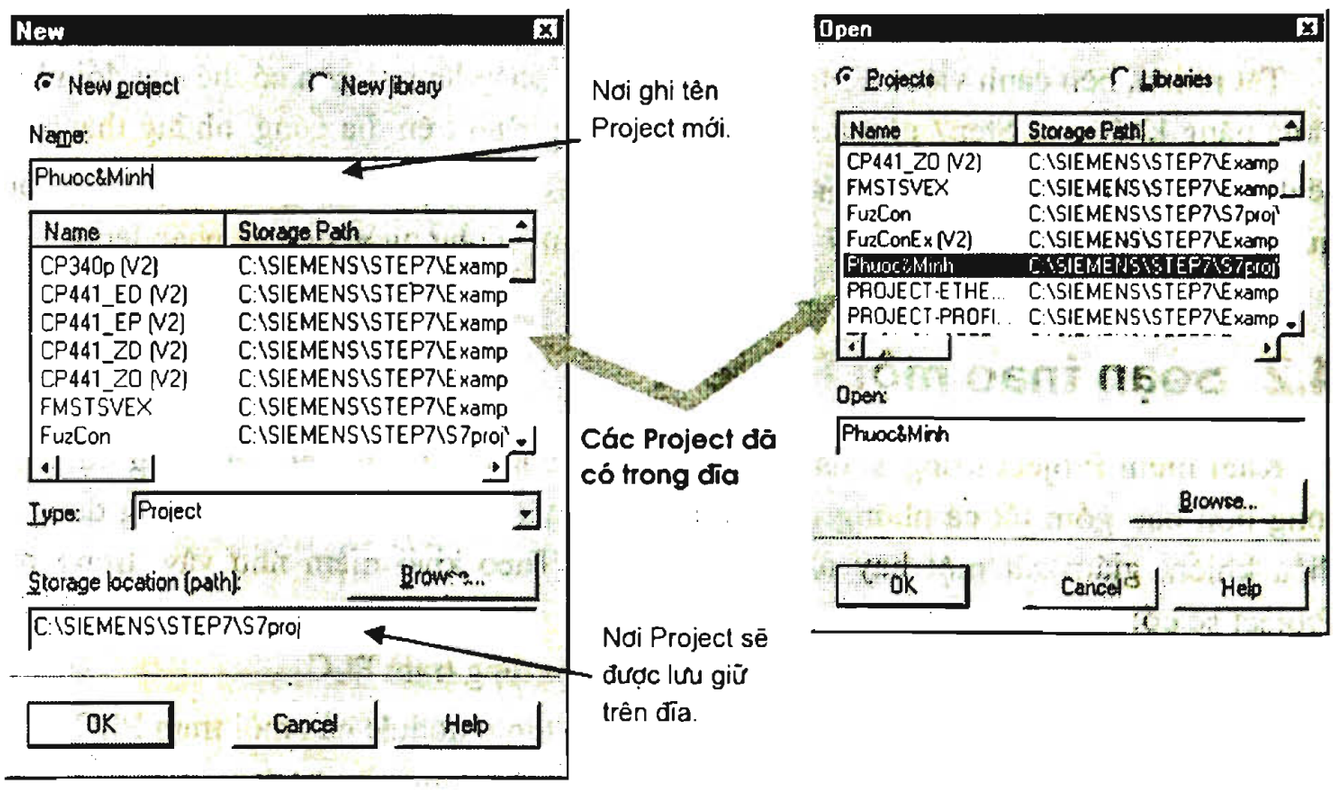
Ở đây, trong khuôn khổ phần mềm Step7, chúng tôi chỉ giới thiệu việc soạn thảo một Project gồm các phần a), b) và c). Những phần còn lại bạn đọc có thể tham khảo thêm trong tài liệu [1].

4.2.1 Khai báo và mở một Project

Để khai báo một Project, từ màn hình chính của Step7 ta chọn **File→New** hoặc kích chuột tại biểu tượng "**New Project/Library**".



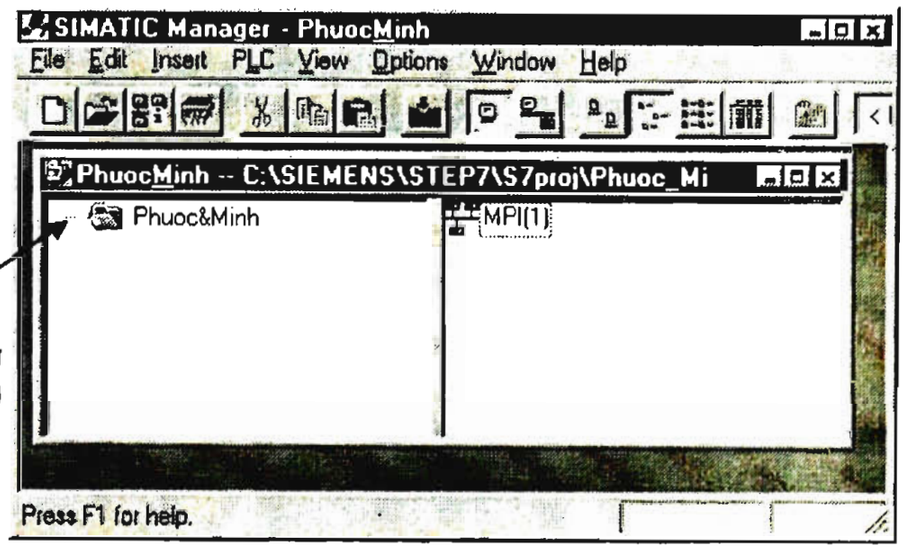
Khi đó trên màn hình sẽ xuất hiện hộp hội thoại như hình dưới, bên trái. Gõ tên Project rồi ấn phím OK và như vậy ta đã khai báo xong một Project mới. Ngoài ra ta còn có thể chọn nơi Project sẽ được cất lên đĩa. Mặc định, nơi cất sẽ là thư mục đã được quy định khi cài đặt Step7, ở đây là thư mục **c:\siemens\step7\s7proj**.



Trong trường hợp muốn mở một Project đã có, ta chọn **File→Open** hoặc kích chuột tại biểu tượng "**Open Project/Library**" từ cửa sổ chính của Step7 rồi chọn tên Project muốn mở từ hộp hội thoại có dạng như ở hình trên, bên phải. Cuối cùng ấn phím OK để kết thúc.

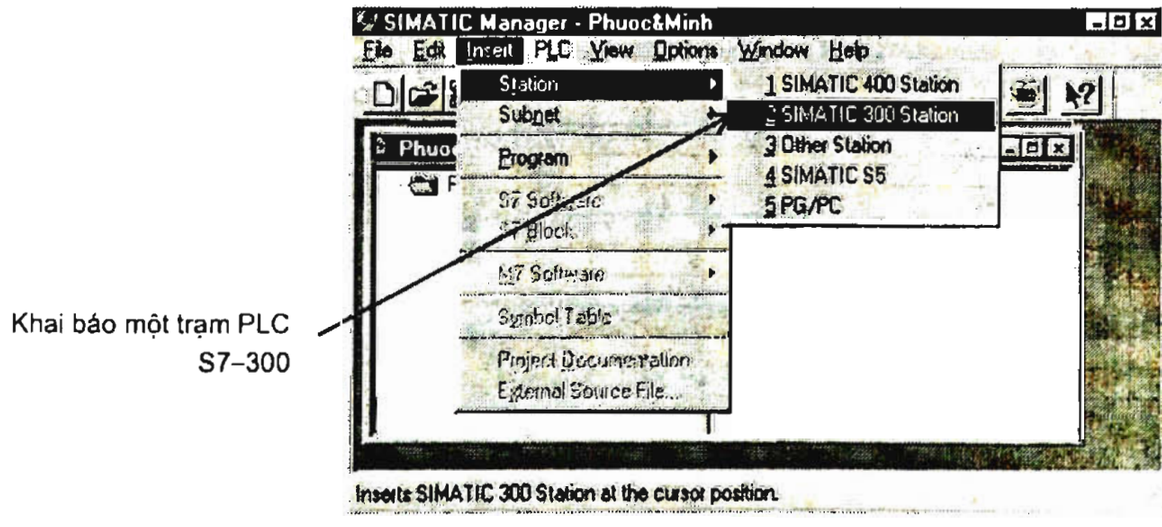
4.2.2 Xây dựng cấu hình cứng cho trạm PLC

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



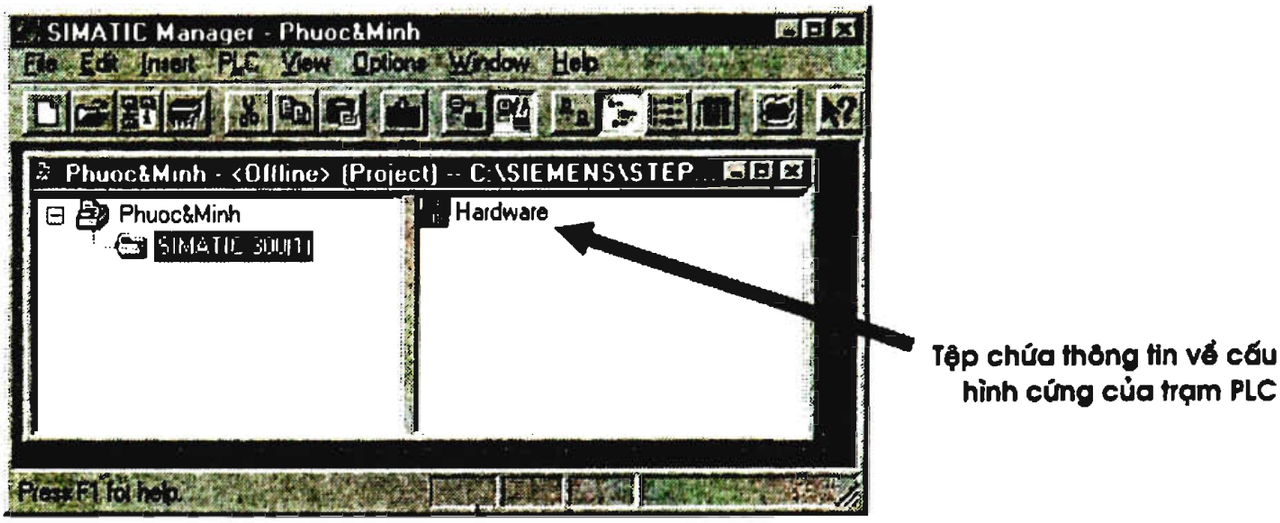
Công việc tiếp theo ta có thể làm là xây dựng cấu hình cứng cho một trạm PLC. Điều này là không bắt buộc, ta có thể không cần khai báo cấu hình cứng cho trạm mà đi ngay vào phần chương trình ứng dụng. Song kinh nghiệm cho thấy công việc này nên làm vì khi có cấu hình trong Project, lúc bật nguồn PLC, hệ điều hành của S7–300 bao giờ cũng đi kiểm tra các module hiện có trong trạm, so sánh với cấu hình mà ta xây dựng và nếu phát hiện thấy sự không đồng nhất sẽ phát ngay tín hiệu báo ngắt lỗi hoặc thiếu module chứ không cần phải đợi tới khi thực hiện chương trình ứng dụng.

Trước hết ta khai báo cấu hình cứng cho một trạm PLC với Simatic S7–300 bằng cách vào **Insert→Station→Simatic 300 Station**:

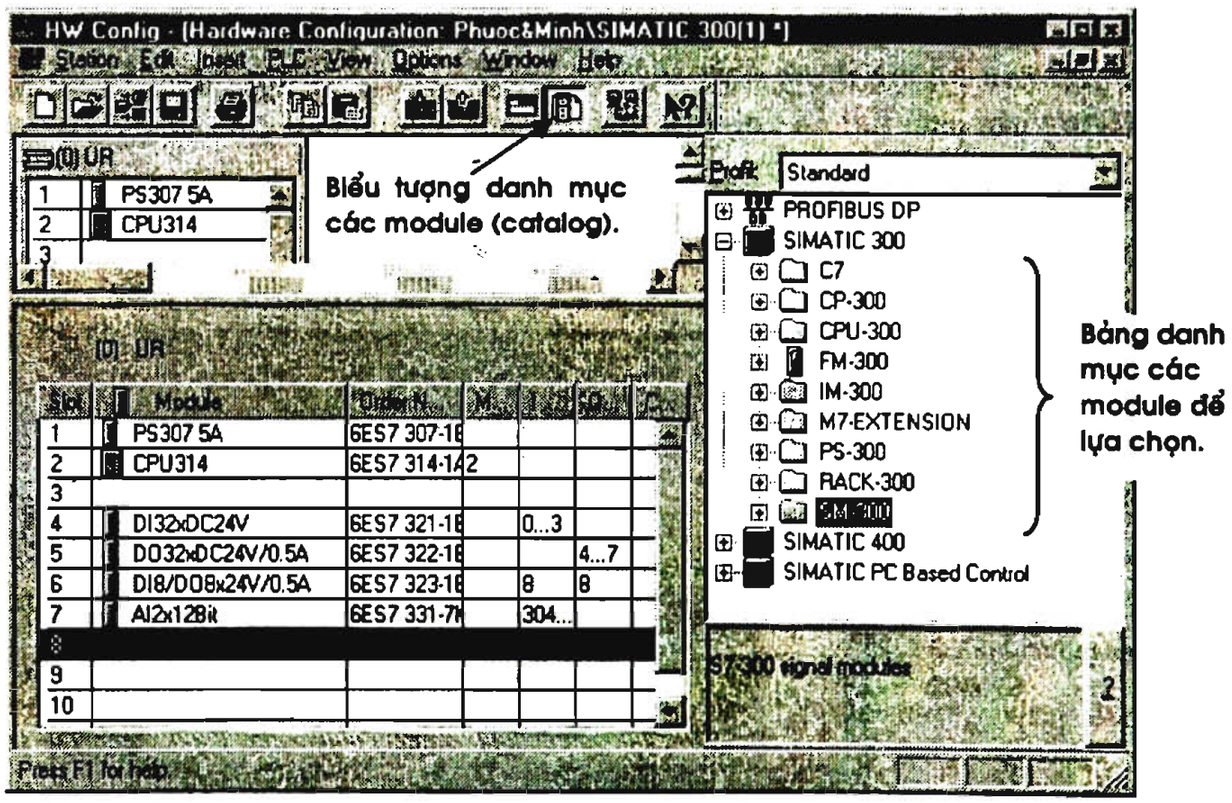


Trường hợp không muốn khai báo cấu hình cứng mà đi ngay vào chương trình ứng dụng ta có thể chọn thẳng **Insert→Program→S7 Program**. Động tác này sẽ hữu ích cho những trường hợp một trạm PLC có nhiều phiên bản chương trình ứng dụng khác nhau.

Sau khi đã khai báo một trạm (chèn một station), thư mục Project chuyển sang dạng không rỗng với thư mục con trong nó có tên mặc định là **Simatic 300(1)**. Tất nhiên ta có thể đổi lại tên mặc định này. Thư mục **Simatic 300(1)** chứa tệp thông tin về cấu hình cứng của trạm.



Để vào màn hình khai báo cấu hình cứng, ta nháy chuột tại biểu tượng **Hardware**. Trong hộp hội thoại hiện ra ta khai báo thanh ray (**rack**) và các module có trên thanh rack đó. Hình dưới là bảng khai báo cấu hình cứng cho trạm PLC theo ví dụ ở mục 3.1.3 (trang 110).



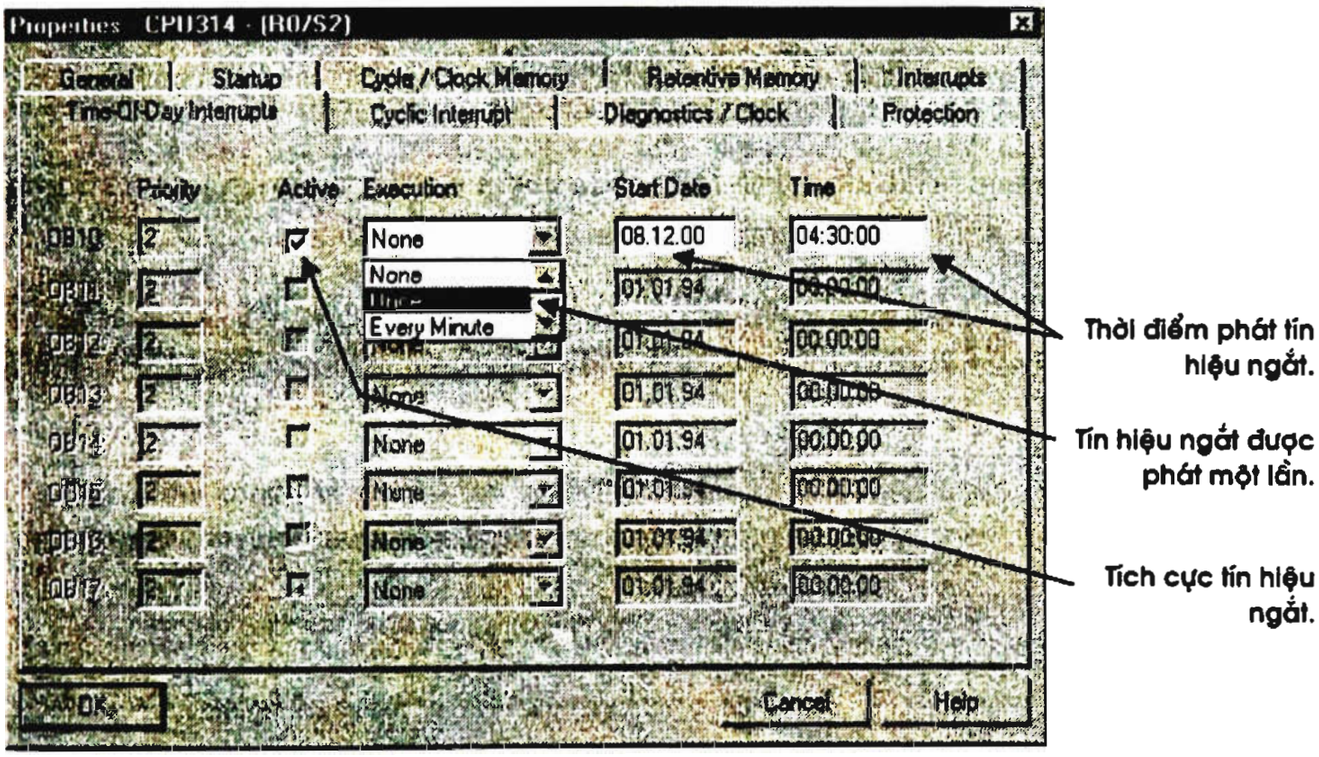
Step7 giúp việc khai báo cấu hình cứng được đơn giản nhờ bảng danh mục các module của nó. Muốn đưa module nào vào bảng cấu hình ta chỉ cần đánh dấu slot nơi module sẽ được đưa vào rồi nháy kép chuột tại tên của module đó trong bảng danh mục các module kèm theo.

4.2.3 Đặt tham số quy định chế độ làm việc cho module

Với bảng cấu hình cứng phần mềm Step7 cũng xác định luôn cho ta địa chỉ từng module theo quy tắc như đã trình bày tại mục 3.1.3.

Ngoài ra, như đã đề cập ở chương 3 (các mục 3.4.2, 3.5.3, 3.5.4), Step7 còn hỗ trợ việc đặt tham số quy định chế độ làm việc cho từng module.

Chẳng hạn Step7 có thể hỗ trợ việc tích cực ngắt theo thời điểm cho module CPU để module này phát một tín hiệu ngắt gọi khối OB10 một lần vào đúng ngày 8/12/2000 lúc 4 giờ 30. Để làm được điều này ta nháy kép chuột tại tên của module CPU ở slot 2 rồi chọn ô **`Time-Of-Day Interrupt`**, trên màn hình sẽ xuất hiện hộp hội thoại như ở hình dưới. Điền thời điểm, tần suất phát tín hiệu ngắt rồi đánh dấu tích cực chế độ ngắt vào các ô tương ứng trong hộp hội thoại. Cuối cùng ấn phím OK.



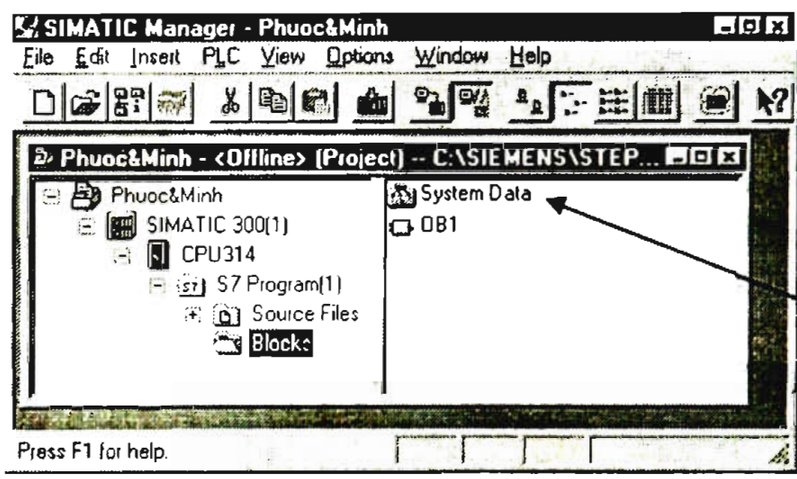
Cũng trong hộp hội thoại ta thấy module CPU314 chỉ cho phép sử dụng OB10 trong số các module OB10÷OB17 với mức ưu tiên là 2 để chứa chương trình xử lý tín hiệu ngắt theo thời điểm.

Các chế độ làm việc khác của module CPU cũng được quy định nhờ Step7. Ví dụ để sửa đổi thời gian vòng quét cực đại cho phép từ giá trị mặc định 150ms thành 100ms, ta chọn **`Cycle/Clock memory`** trong hộp hội thoại rồi sửa nội dung ô **`Scan time`** thành 100.

Hoàn toàn tương tự ta cũng có thể sử dụng Step7 để quy định chế độ làm việc cho các module mở rộng khác, như xác định chế độ làm việc với dạng tín hiệu điện áp với dải ±10V cho module AI, tích cực tín hiệu ngắt tự chẩn đoán cho module DI/DO, tích cực ngắt cứng theo sườn lên tại cổng vào I0.0 cho module DI ....

4.2.4 Soạn thảo chương trình cho các khối logic

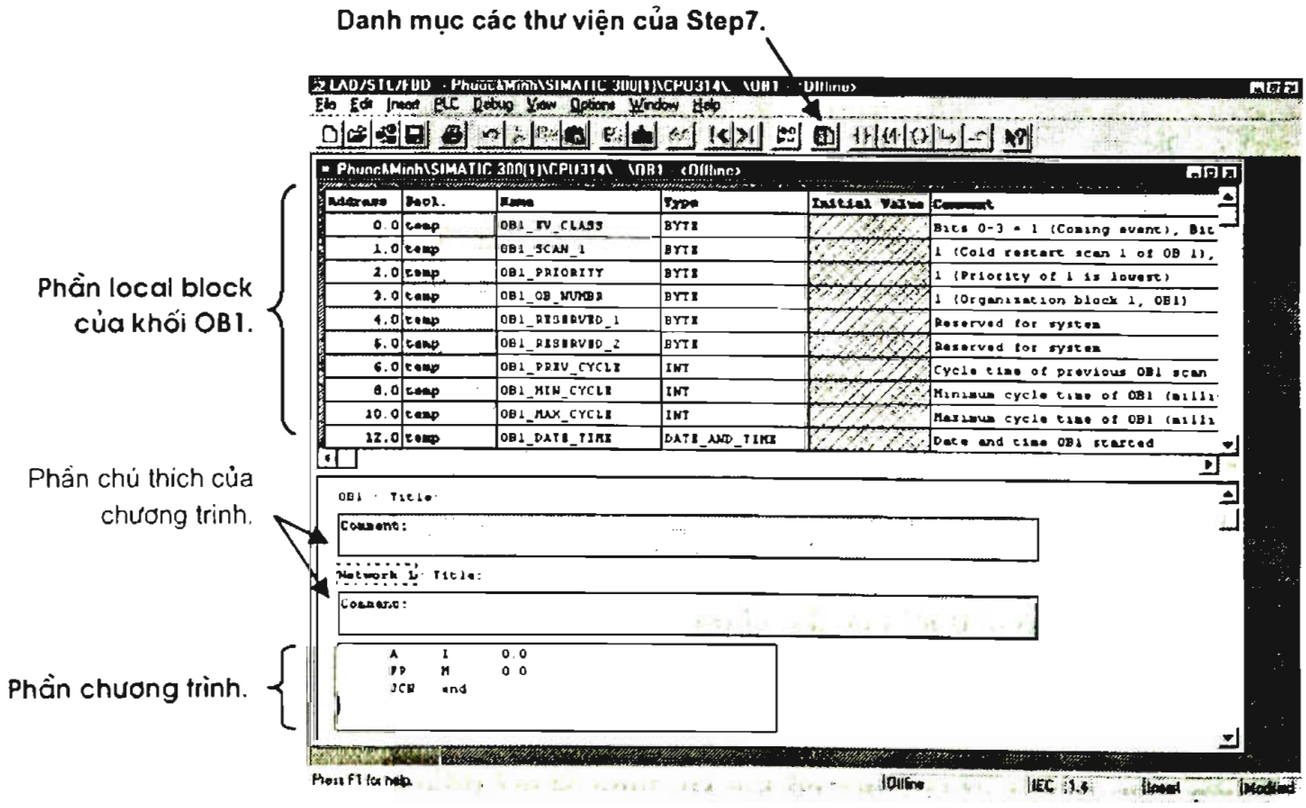
Sau khi khai báo xong cấu hình cứng cho một trạm PLC và quay trở về cửa sổ chính của Step7 ta sẽ thấy trong thư mục **Simatic 300(1)** bây giờ có thêm các thư mục con **CPU314**, **S7 Program(1)**, **Source files**, **Blocks** và tất nhiên ta có thể đổi tên các thư mục đó.



Tham số xác định chế độ làm việc của các module trong trạm vừa soạn thảo nhờ Step7 sẽ nằm trong thư mục System data.

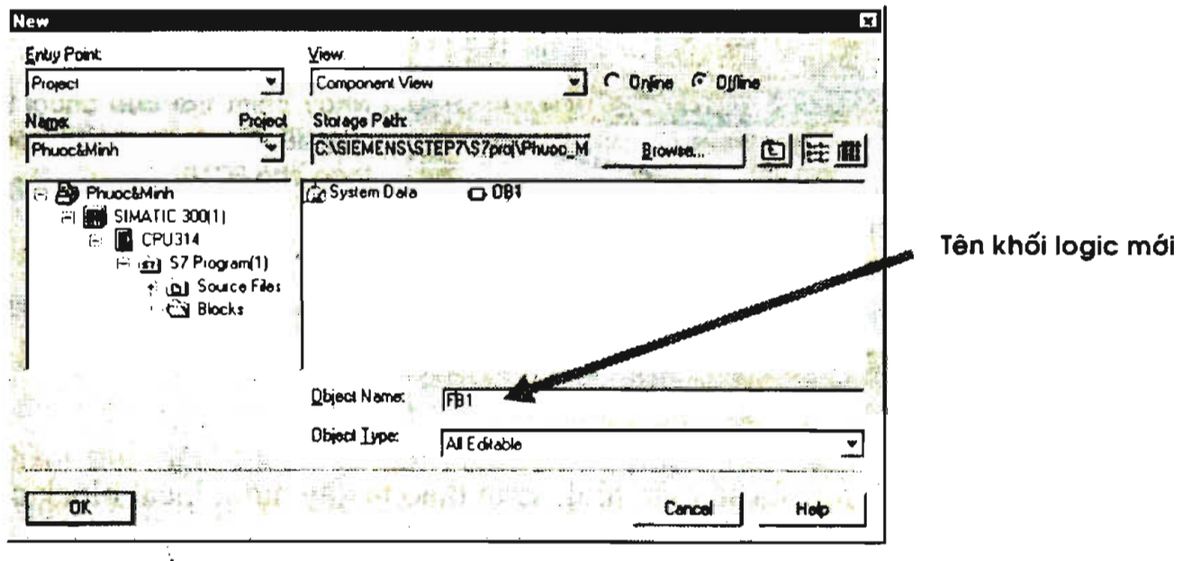
Tất cả các khối logic (OB, FC, FB, DB) chứa chương trình ứng dụng sẽ nằm trong thư mục **Block**. Mặc định trong thư mục này đã có sẵn khối OB1.

Muốn soạn thảo chương trình cho khối OB1 ta nháy chuột tại biểu tượng OB1 bên nửa cửa sổ bên phải. Trên màn hình sẽ xuất hiện của sổ của chế độ soạn thảo chương trình như sau:



Chức năng chương trình soạn thảo của Step7 về cơ bản cũng giống như các chương trình soạn thảo khác, tức là cũng có các phím nóng để gõ nhanh, có chế độ cắt và dán, có chế độ kiểm tra lỗi cú pháp lệnh ....

Để khai báo và soạn thảo chương trình cho các khối OB khác hoặc cho các khối FC, FB hay DB, ta có thể tạo một khối mới ngay trực tiếp từ chương trình soạn thảo bằng cách kích chuột tại biểu tượng **New** rồi ghi tên khối vào ô tương ứng của cửa sổ hiện ra:



hoặc cũng có thể chèn thêm khối mới đó trước từ cửa sổ chính của Step7 bằng phím **Insert→S7 Block** rồi sau đó mới vào soạn thảo chương trình cho khối vừa được chèn thêm như đã làm với OB1 ở ví dụ vừa rồi.

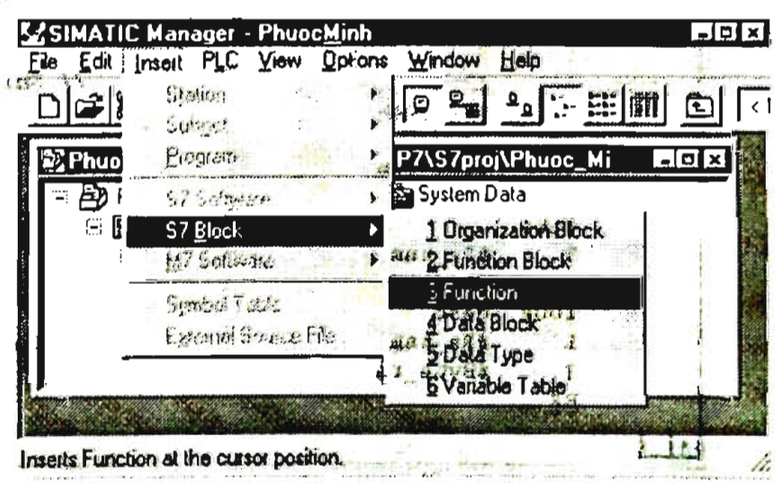
Trong màn hình soạn thảo chương trình cho các khối logic, ta có thể thay đổi không riêng phần chương trình mà cả phần local block của khối đó bao gồm tên hình thức, kiểu dữ liệu, giá trị ban đầu. Chú ý rằng không được thay đổi 20 bytes đầu trong local block của các khối OB.

Các bước soạn thảo một khối logic cho chương trình ứng dụng được tóm tắt như sau:

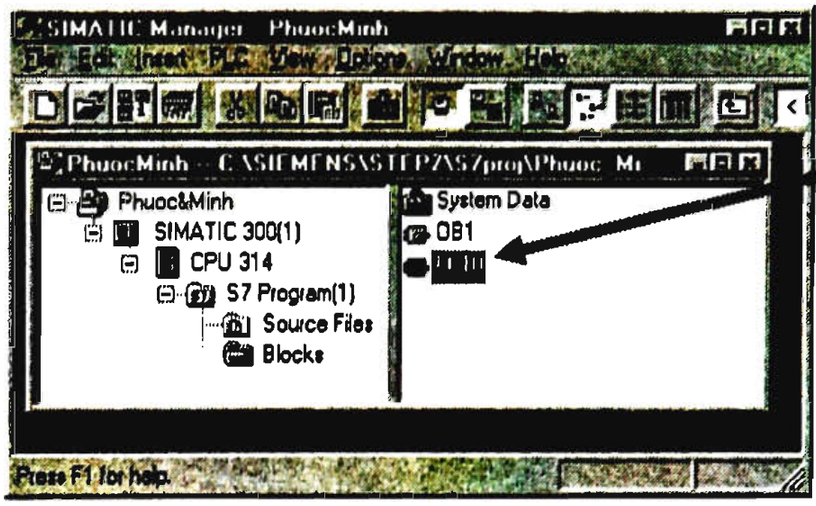
- tạo khối logic hoặc từ cửa sổ màn hình chính của Step7 bằng cách chọn **Insert** trên thanh công cụ rồi vào **S7 Block** để chọn loại khối logic mong muốn (OB, FB, FC) hoặc vào chương trình soạn thảo rồi từ đó kích biểu tượng **New**,
- thiết kế local block cho khối logic vừa tạo.
- viết chương trình

và để minh họa ta sẽ tiến hành việc soạn thảo khối FC10 của ví dụ 2 mục 3.3.1 (trang 121) theo 3 bước vừa nêu:

1) **Tạo khối**: Từ thư mục **Block** của Step7 chọn **Insert→S7 Block→Function** trên thanh công cụ (xem hình bên). Trên màn hình sẽ hiện ra hộp hội thoại hỏi tên khối FC ta muốn khởi tạo. Viết FC10 rồi ấn phím OK. Trong thư mục **Block** lúc này sẽ có thêm khối FC10.



Nháy kép chuột tại biểu tượng của FC10 để vào chế độ soạn thảo chương trình cho FC10.

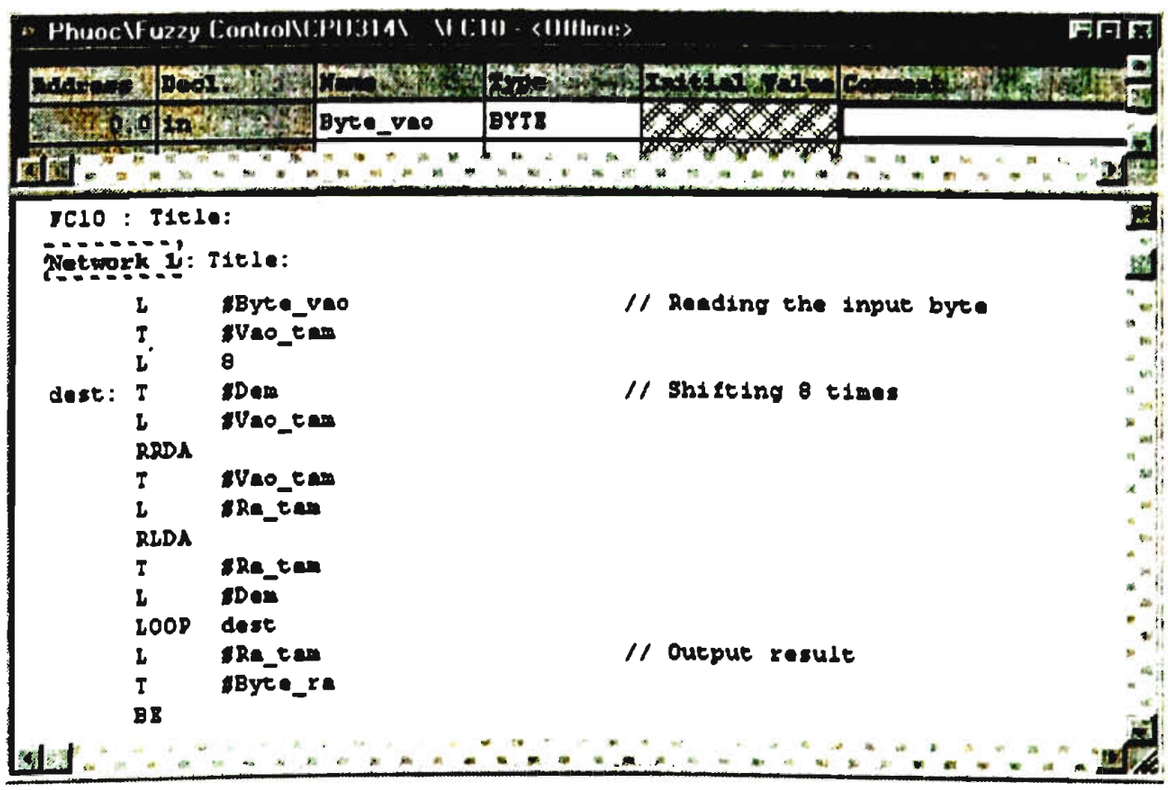


Nháy phím trái của chuột tại đây để vào chương trình soạn thảo cho FC10.

2) **Xây dựng local block**: Trong cửa sổ màn hình soạn thảo ta xây dựng local block cho FC10 với cấu trúc như sau:

Address	Decl.	Name	Type	Initial Value	Comment
0.0	in	Byte_vao	BYTE		
1.0	out	Byte_ra	BYTE		
	in_out				
0.0	temp	Dem	INT		
2.0	temp	Vao_tam	BYTE		
3.0	temp	Ra_tam	BYTE		

3) **Soạn thảo chương trình**: Tiếp theo ta viết phần chương trình. Toàn bộ chương trình của FC10 như ở ví dụ 2 trang 121 có thể viết chung trong một Network.

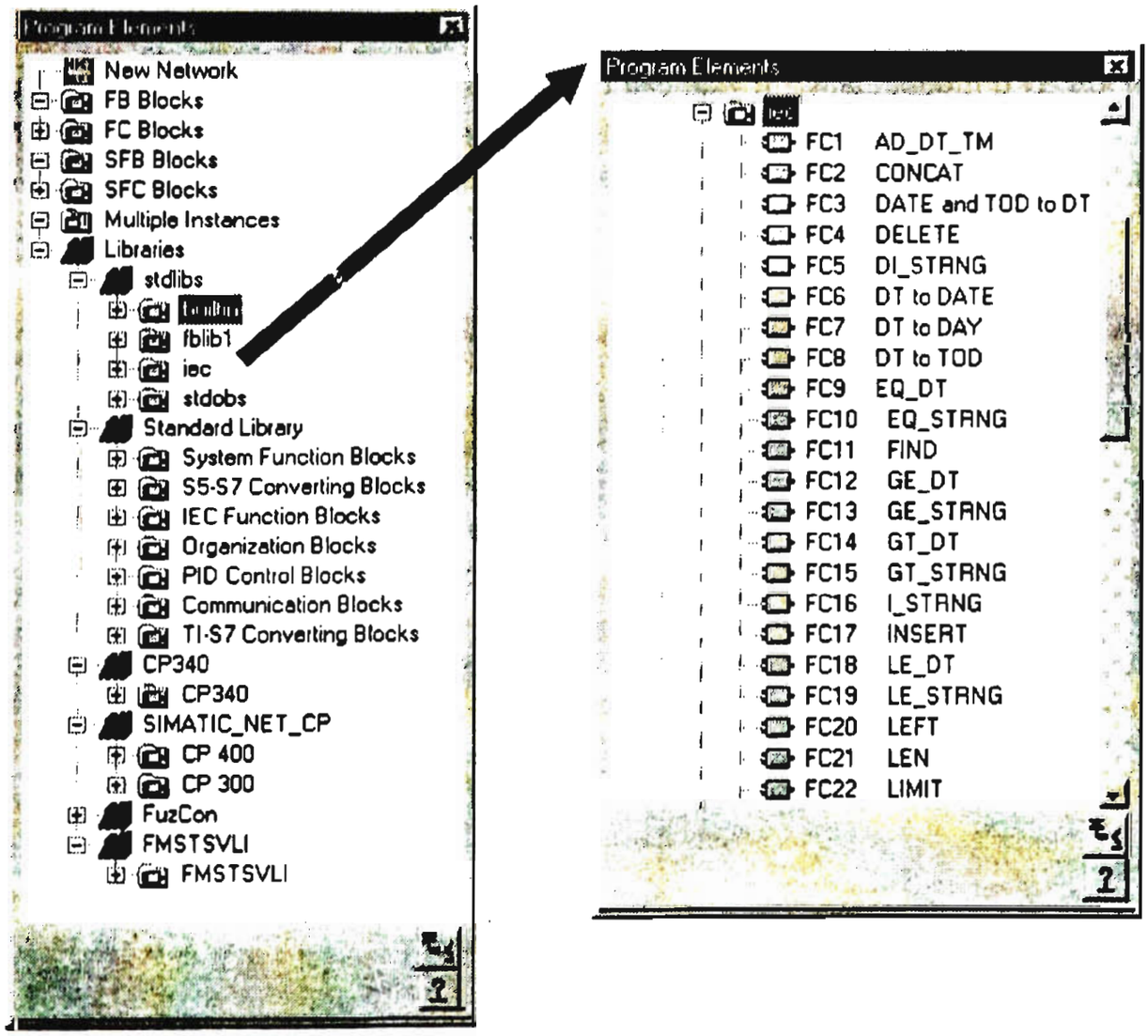


4.2.5 Sử dụng thư viện của Step7

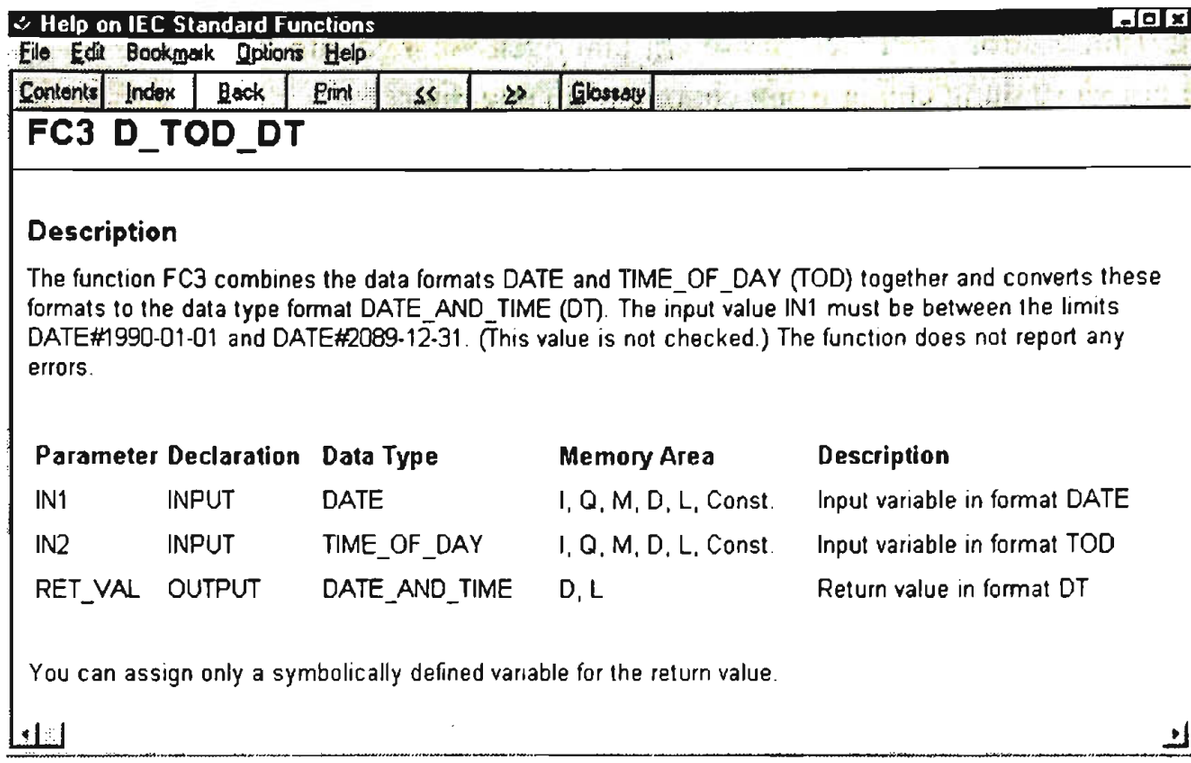
Phần mềm soạn thảo chương trình của Step7 có một bộ thư viện khá phong phú gồm những khối chương trình FC, FB, SFC và SFB đã được chuẩn hóa mà ta có thể sử dụng. Tài liệu hướng dẫn chi tiết từng công dụng các khối chương trình này có ngay trong phần mềm trợ giúp của Step7. Ở đây ta sẽ không đi vào chi tiết chức năng của từng khối chương trình có trong thư viện mà sẽ chỉ làm quen với việc làm thế nào để gọi và sử dụng chúng trong chương trình ứng dụng.

Biểu tượng của thư viện trong Step7.

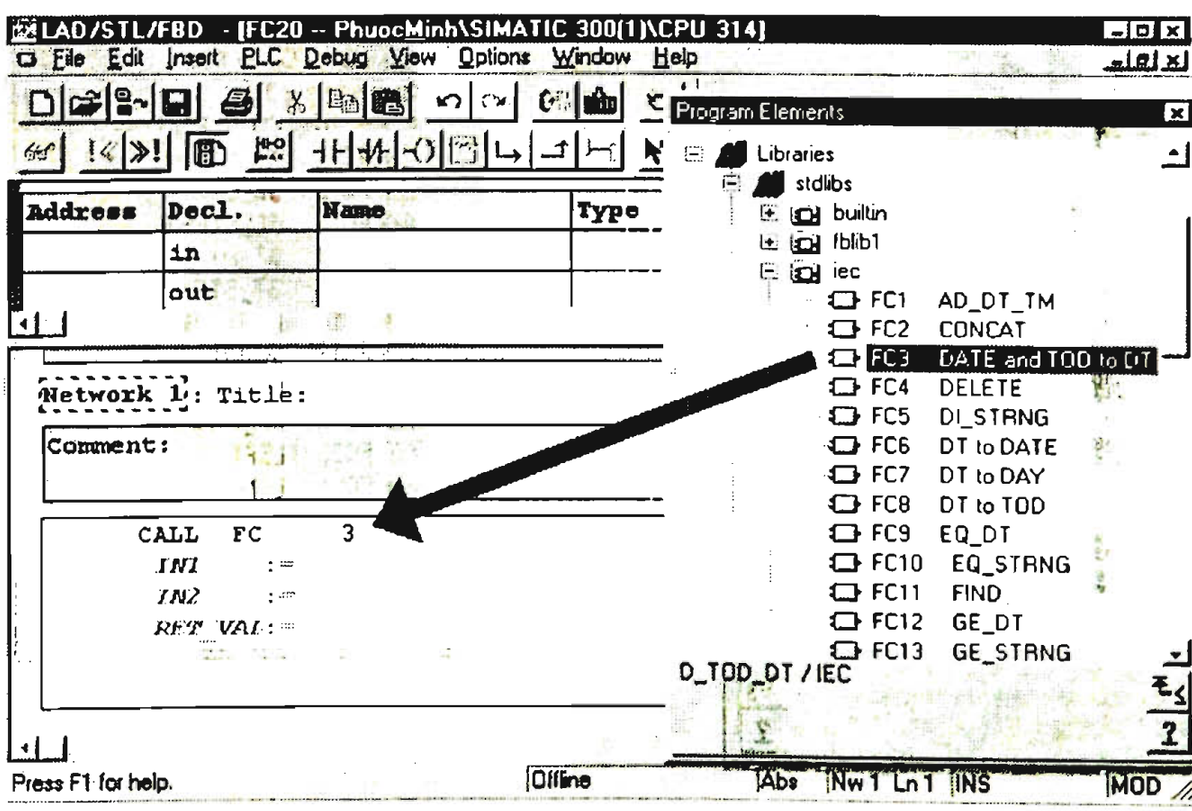
Thanh công cụ trong cửa sổ màn hình soạn thảo có biểu tượng của các hàm thư viện có trong Step7 (**Program Elements**) như hình bên. Kích vào biểu tượng này ta sẽ có được bảng danh mục các khối hàm thư viện ngay trong cửa sổ màn hình soạn thảo. Step7 có đủ các loại hàm thư viện được sắp xếp theo từng nhóm chức năng trong bảng danh mục (xem hình dưới, bên trái), mỗi nhóm là một thư mục. Muốn sử dụng một hàm cụ thể nào đó, trước hết ta phải xác định hàm đó thuộc nhóm chức năng nào, sau đó đi tìm trong bảng danh mục bằng cách mở thư mục nhóm chức năng đó. Ví dụ để sử dụng hàm tạo dữ liệu kiểu **Date_And_Time**, trước hết ta mở thư mục **Libraries**, trong đó lại mở tiếp **stdlibs→iec** sẽ thấy hàm ta cần là FC3 với tên hình thức **DATE and TOD to DT** (hình dưới, bên phải).



Trong trường hợp muốn tự tìm hiểu chức năng, công dụng của một hàm thư viện nào đó, ta có thể nhờ phần trợ giúp online của Step7 bằng cách đánh dấu tên hàm cần tìm hiểu rồi ấn phím F1. Chẳng hạn để tìm hiểu xem hàm FC3 có tính năng và công dụng gì ta kích chuột tại tên hàm FC3 để đánh dấu rồi ấn phím F1, trên màn hình sẽ hiện ra lời giải thích:



Sau khi đã tìm được hàm thư viện cần dùng, để sử dụng hàm đó trong chương trình ứng dụng ta chỉ cần nháy phím trái chuột tại tên hàm. Ví dụ, để sử dụng hàm FC3 khi soạn thảo chương trình cho khối FC20 (ví dụ của mục 3.5.3, trang 157) ta nháy chuột tại tên FC3 trong bảng danh mục các hàm thư viện, hàm FC3 sẽ được lấy vào chương trình như sau:



4.2.6 Sử dụng tên hình thức

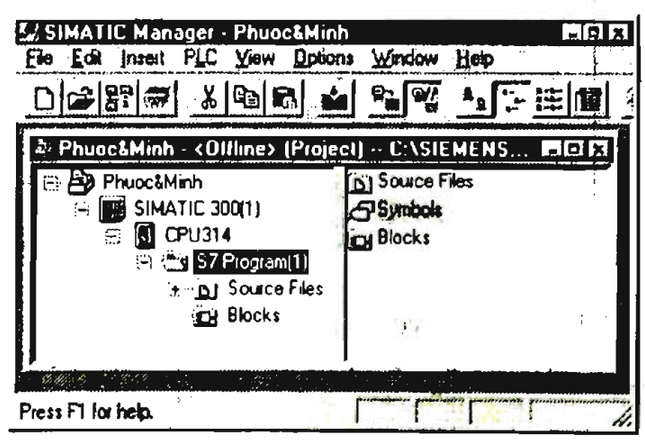
Nếu như ngôn ngữ lập trình không bị khô khan bằng những ký hiệu "vô hồn" thì có lẽ chương trình sẽ dễ hiểu và dễ theo dõi hơn. Chẳng hạn như chương trình điều khiển đèn giao thông đã giới thiệu ở mục 2.7.4, trang 87 chắc chắn sẽ trở nên sinh động đối với người đọc hơn nếu như thay vì viết Q4.0 ta có thể viết "Đèn_đỏ", thay Q4.1 bằng tên "Đèn_vàng" và thay Q4.2 bằng tên "Đèn_xanh" như sau:

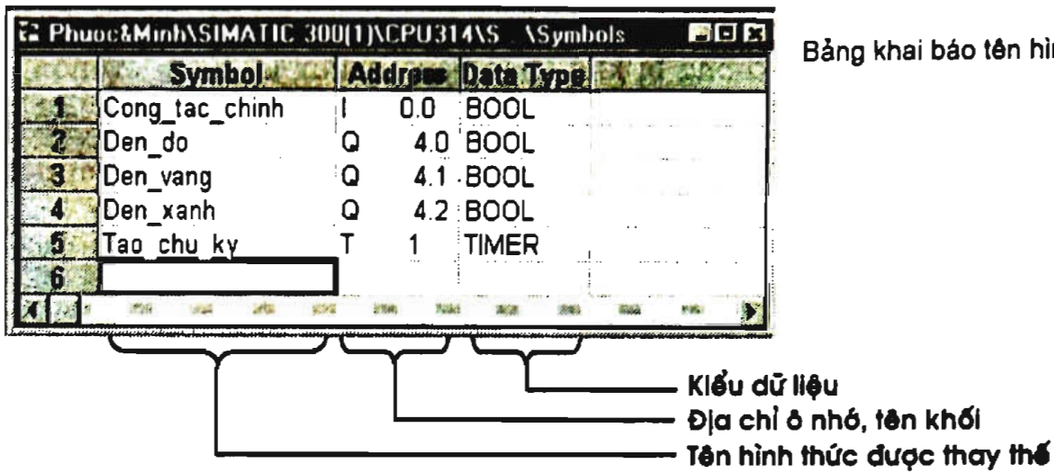
```
      A     I 0.0              // Công tắc chuyển sang điều khiển tự động (I0.0=1)
      AN    T1                 //Tạo chu kỳ 95s.
      L     W#16#1950          //Thời gian trễ là 100ms×950.
      SD    T1
      L     W#16#1370          // Đèn xanh ?.
      LC    T1                 //Đọc giá trị đếm tức thời CV.
      <=I
      JC    xng
      L     W#16#1350          // Đèn vàng ?.
      >I
      JC    vng
      S     "Đèn_đỏ"           // Bật đèn đỏ và tắt đèn xanh, vàng.
      R     "Đèn_vàng"
      R     "Đèn_xanh"
      BEU
xng:  S     "Đèn_xanh"         //Bật đèn xanh và tắt đèn đỏ, vàng.
      R     "Đèn_đỏ"
      R     "Đèn_vàng"
      BEU
vng:  S     "Đèn_vàng"         // Bật đèn vàng và tắt đèn xanh, đỏ.
      R     "Đèn_đỏ"
      R     "Đèn_xanh"
      BEU
```

Nắm bắt được điều này, Step7 đã cung cấp thêm khả năng sử dụng tên hình thức trong lập trình thay vì các ký hiệu địa chỉ, chữ số khối FC, FB ... khó nhớ. Các tên hình thức của một địa chỉ hay một tên khối ... phải được khai báo trước trong bảng có tên là **Symbols**.

Do tên hình thức sẽ có giá trị thay thế trong toàn bộ chương trình ứng dụng nên bảng khai báo tên hình thức phải có vị trí ngang bằng với chương trình ứng dụng, nói cách khác nó phải nằm trong cùng thư mục với thư mục **Block** (chứ không thể nằm trong thư mục **Block**).

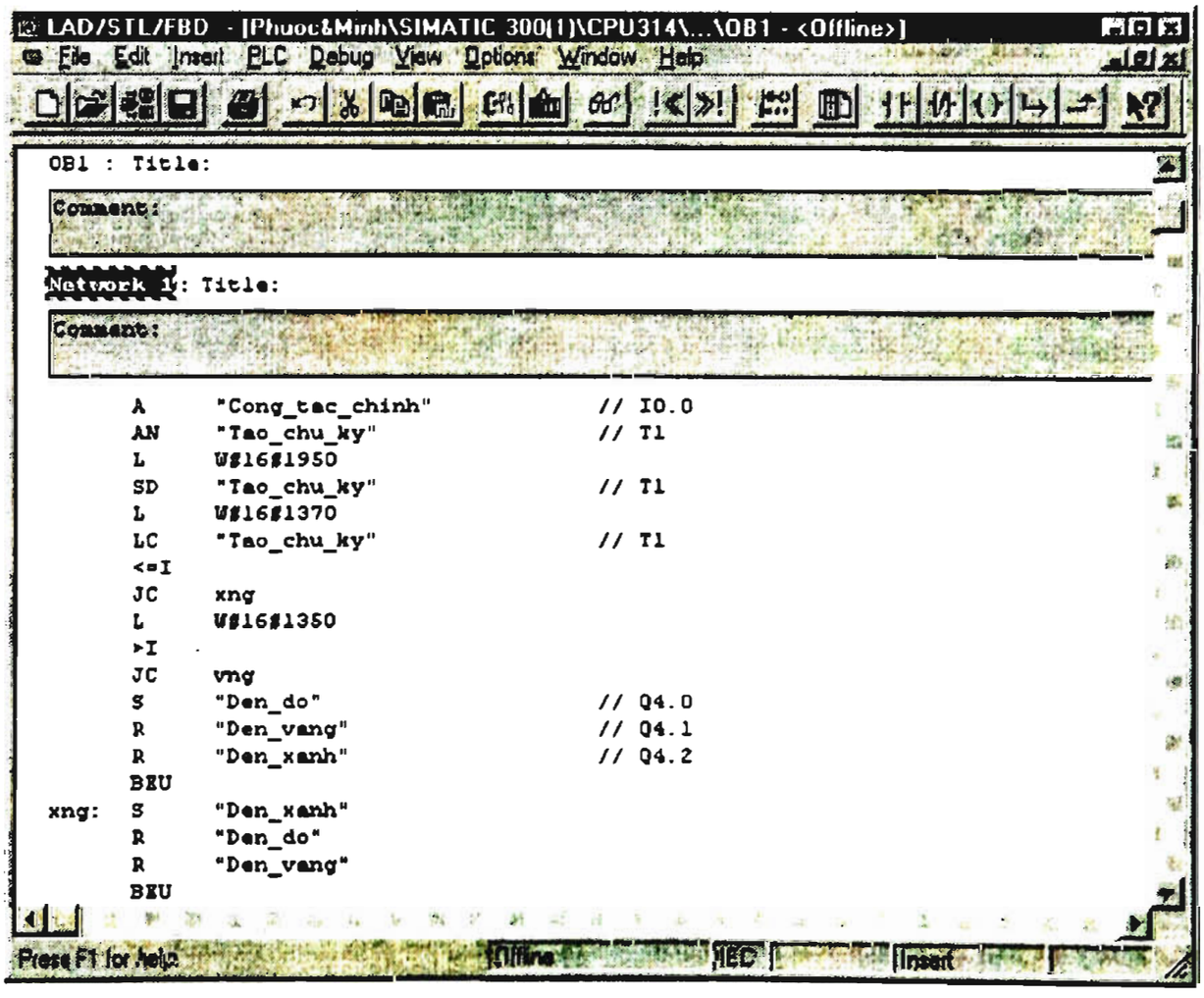
Kích chuột vào thư mục mẹ của **Block**, ở đây là thư mục với tên mặc định **S7 Program(1)**, sau đó nháy phím chuột trái tại biểu tượng **Symbols** (xem hình bên) ta sẽ có màn hình soạn thảo bảng các tên hình thức như sau:





Bảng khai báo tên hình thức.

Sau khi điền đầy đủ tên hình thức, địa chỉ ô nhớ mà nó thay thế (phần lớn kiểu dữ liệu sẽ được Step7 tự xác định căn cứ vào địa chỉ ô nhớ) và cất vào Project, ta sẽ quay trở lại màn hình chính của Step7. Mở một khối chương trình, ví dụ OB1, và chọn **View→Symbolic Representation** trên thanh công cụ của màn hình soạn thảo hoặc dùng phím nóng **CTRL+Q**, chương trình trong OB1 sẽ được chuyển sang dạng biểu diễn với những tên hình thức có trong bảng **Symbol** (hình dưới).

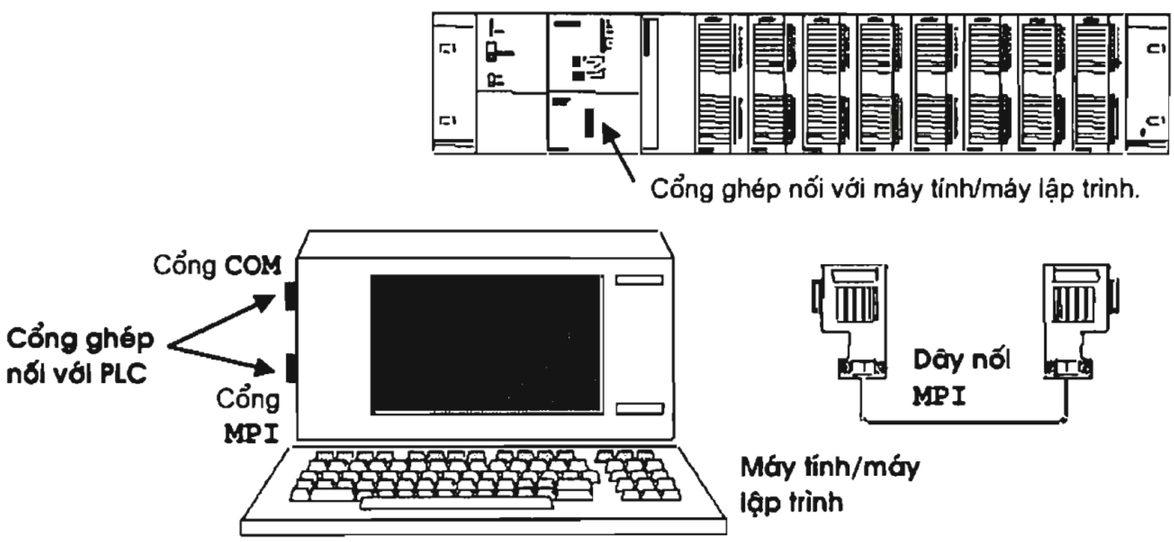


Muốn quay về sử dụng lại ký hiệu địa chỉ tuyệt đối ta lại ấn phím nóng **CTRL+Q** hoặc chọn **View→Symbolic Representation** trên thanh công cụ.

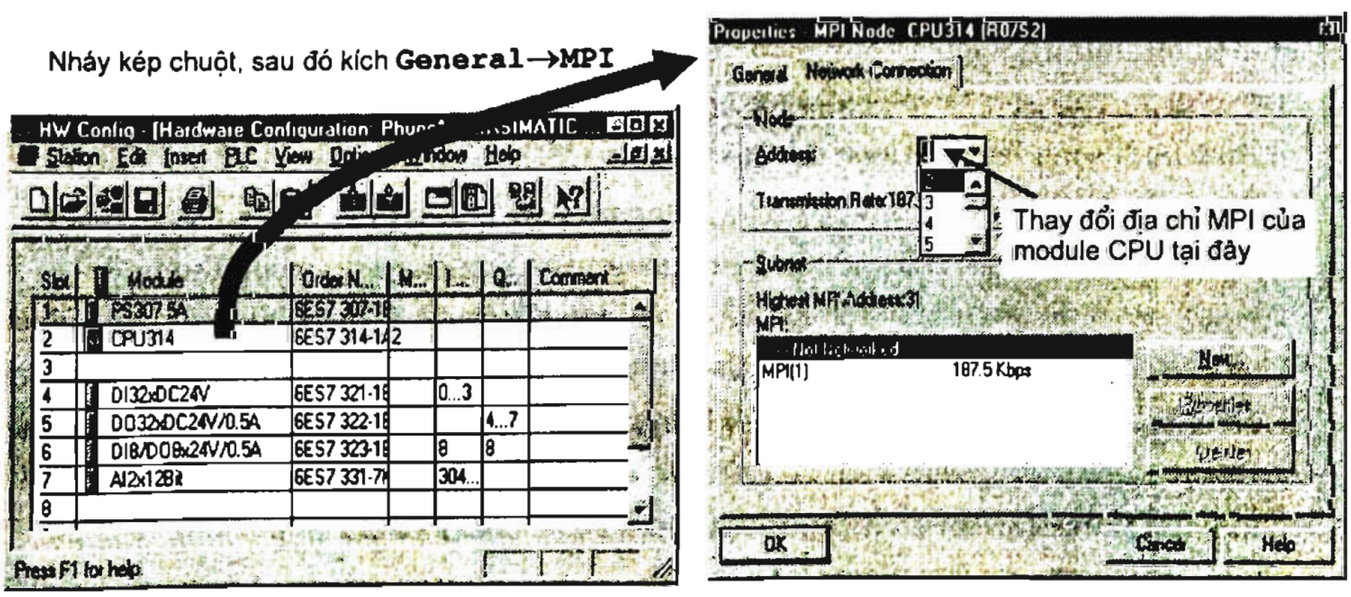
4.3 Làm việc với PLC

4.3.1 Quy định địa chỉ MPI cho module CPU

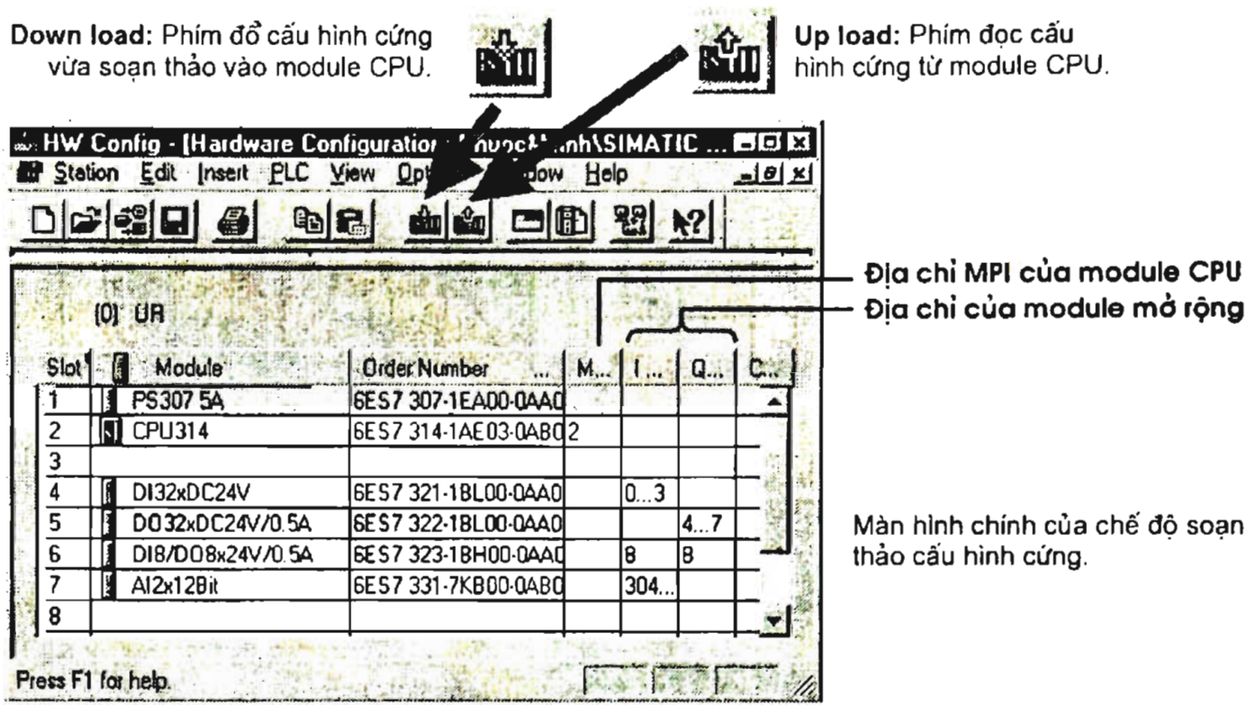
Ở trang 167 thuộc mục 4.1.1, khi nói về việc cài đặt Step7 ta đã có đề cập đến vấn đề ghép nối máy tính hay máy lập trình (PG) với trạm PLC. Máy tính/máy lập trình được ghép nối với module CPU qua cổng truyền thông nối tiếp RS232 (COM) của máy tính hay qua cổng MPI (MPI Card) hay cổng CP (CP Card) là còn tùy thuộc vào bộ giao diện được sử dụng. Tương tự cũng có nhiều khả năng nối PLC với máy tính, song để truyền thông nhờ Step7 thì PLC luôn phải được nối với máy tính qua cổng lập trình (RS485).



Sau khi ghép nối module CPU với máy tính về phần cứng ta còn phải định nghĩa thêm địa chỉ truyền thông cho trạm PLC. Điều này là cần thiết vì một máy tính/máy lập trình có thể cùng một lúc làm việc được với nhiều trạm PLC. Mặc định, các module CPU đều có địa chỉ là 2 (địa chỉ MPI). Muốn thay đổi địa chỉ module CPU ta nháy kép phím chuột trái tại tên của module trong bảng khai báo cấu hình cứng để vào chế độ đặt lại tham số làm việc (trang 173), trong đó ta lại chọn tiếp **General→MPI** và sửa lại địa chỉ MPI như hình dưới mô tả.



Sau khi đã định nghĩa lại địa chỉ MPI cho trạm PLC, ta phải ghi lại địa chỉ đó lên module CPU và chỉ khi đó module CPU mới thực sự làm việc theo địa chỉ mới này. Công việc ghi địa chỉ MPI mới này lên module CPU được thực hiện cùng với việc ghi tất cả tham số quy định chế độ làm việc của module bằng cách kích vào biểu tượng **Down load** trên thanh công cụ hoặc chọn **PLC→Down load**.



Bên cạnh việc ghi cấu hình cứng vừa soạn thảo vào module CPU ta cũng có thể đọc ngược bảng cấu hình cứng hiện có từ module CPU vào Project bằng cách kích chuột vào biểu tượng **Up load** trên thanh công cụ của màn hình (hoặc chọn **PLC→Up load**). Với việc đọc ngược cấu hình cứng này ta cũng đọc được luôn cả toàn bộ chương trình hiện có trong Load memory của module CPU vào Project.

4.3.2 Ghi chương trình lên module CPU

Có hai cách đổ chương trình ứng dụng, sau khi đã soạn thảo xong, vào module CPU (cụ thể là vào vùng Load memory) như sau:

1) Đổ từ màn hình soạn thảo chương trình bằng cách kích vào biểu tượng **Down load** trên thanh công cụ của màn hình. Với cách đổ này, chỉ riêng khối chương trình đang ở màn hình soạn thảo sẽ được đổ vào module CPU.
2) Đổ từ màn hình chính của Step7 cũng bằng cách kích vào biểu tượng **Down load**. Với cách đổ này ta có thể đổ toàn bộ chương trình ứng dụng có trong thư mục **Block** hoặc đổ những khối mà ta đã đánh dấu. Muốn đổ toàn bộ thư mục **Block** ta phải kích chuột vào tên thư mục trước sau đó mới được kích vào **Down load**. Trong trường hợp chỉ đổ một số khối, ta đánh dấu những khối sẽ được đổ trước bằng cách giữ phím **CTRL** đồng thời kích chuột tại tên của từng khối. Cuối cùng, sau khi đã chọn xong các khối thì kích chuột vào biểu tượng **Down load**.

4.3.3 Giám sát việc thực hiện chương trình

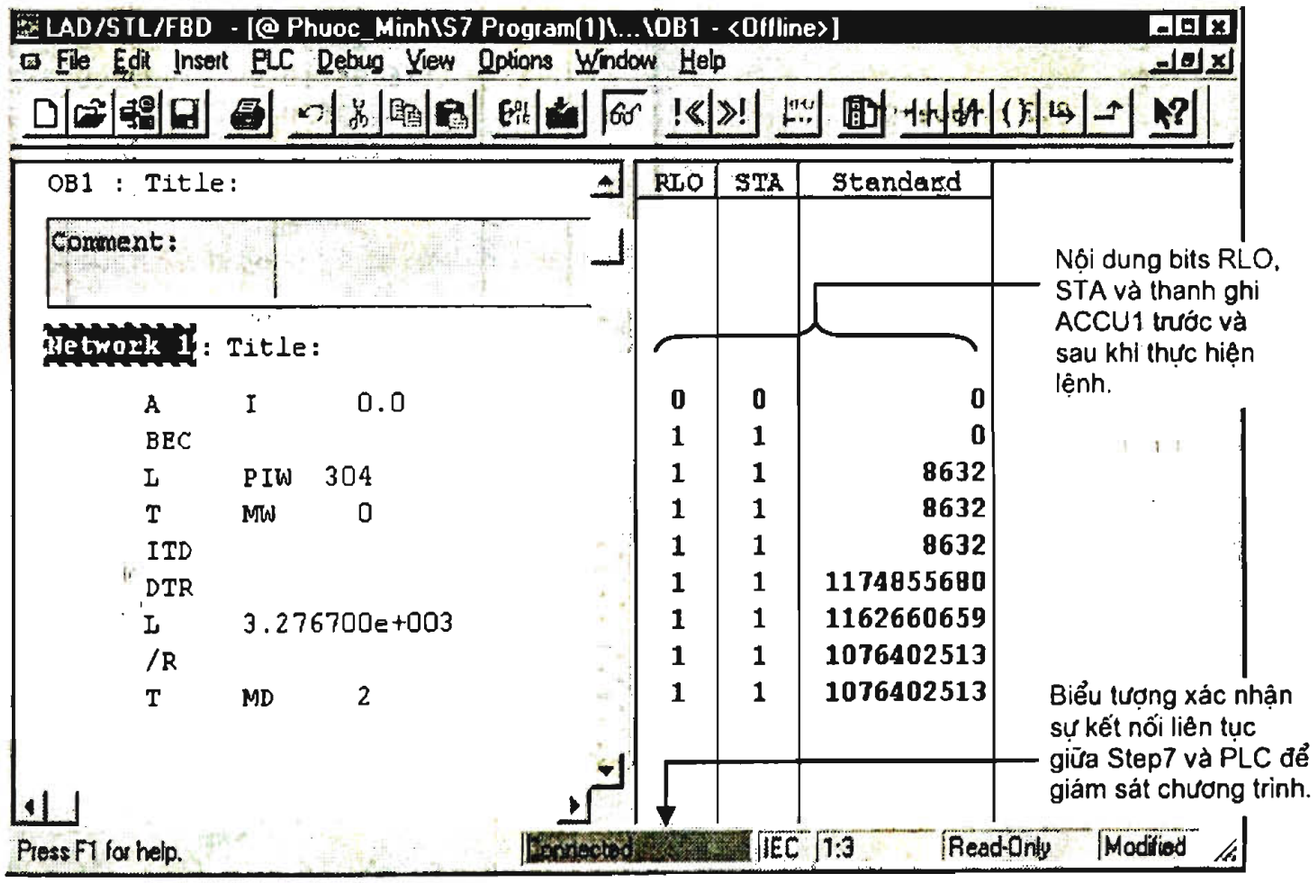
Sau khi ghi chương trình lên module CPU thì nội dung Load memory của module CPU và thư mục **Block** của Project trong máy tính sẽ đồng nhất. Nếu bật công tắc module CPU từ **STOP** sang **RUN**, CPU sẽ thực hiện chương trình trong Load memory của nó theo vòng quét và quá trình thực hiện lệnh này được Step7 giám sát thông qua chương trình tương ứng trong Project.

Việc giám sát chương trình được Step7 tiến hành bằng cách cho hiển thị nội dung các thanh ghi của CPU trước và sau khi thực hiện từng lệnh một của chương trình.

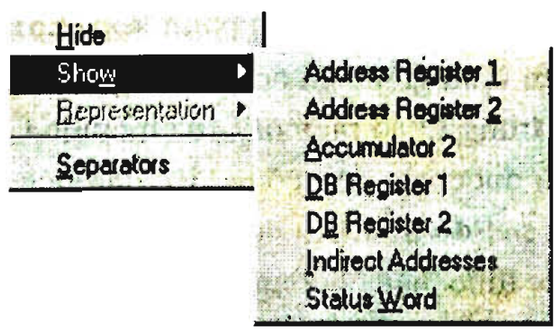
Monitor: Phím giám sát việc thực hiện chương trình.

Để vào màn hình giám sát, ta chỉ cần kích chuột tại phím **Monitor** trên thanh công cụ của màn hình soạn thảo. Phím **Monitor** có biểu tượng cho ở hình bên.

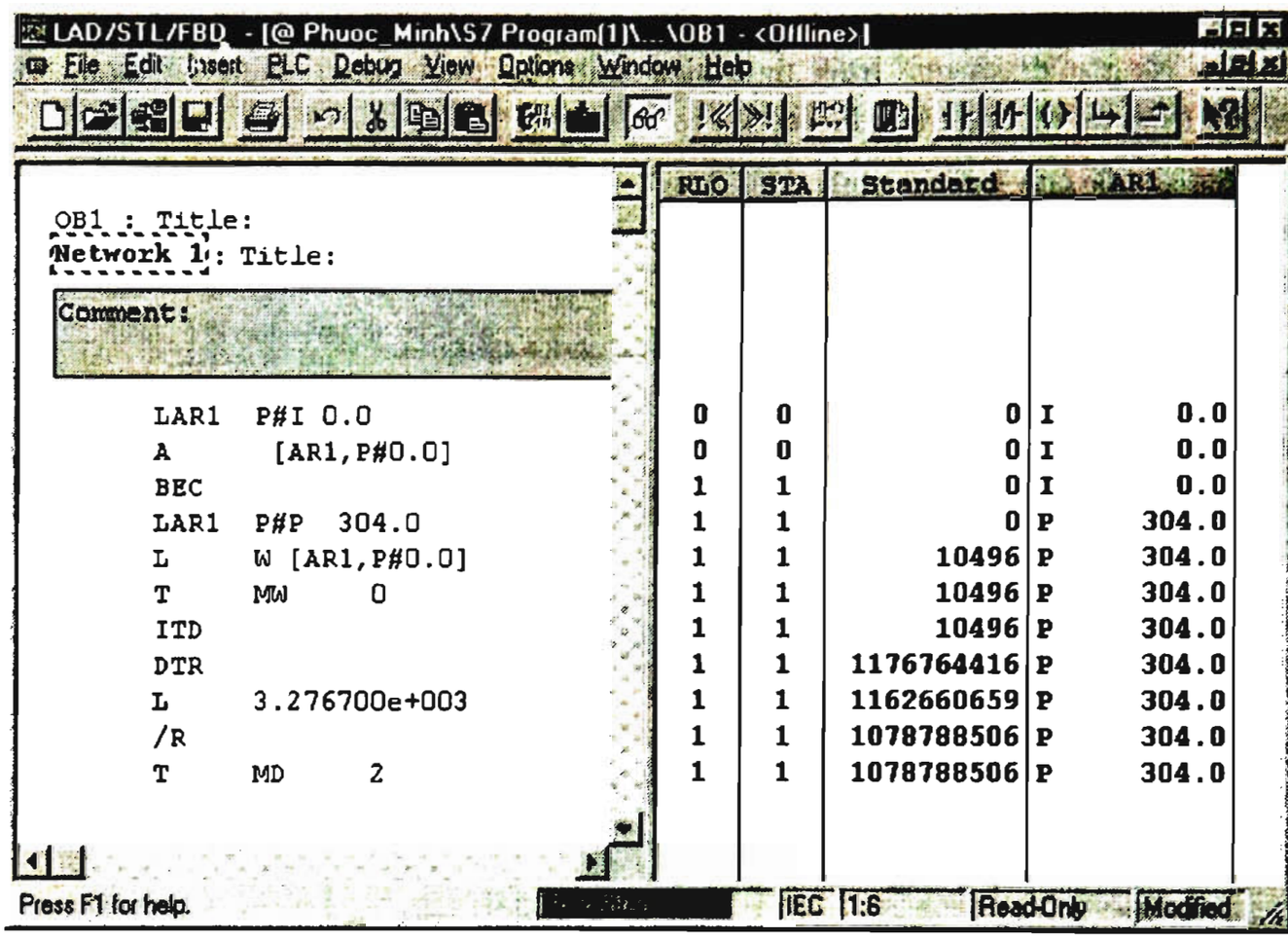
Sau khi kích phím **Monitor**, trên màn hình xuất hiện cửa sổ giám sát như sau:



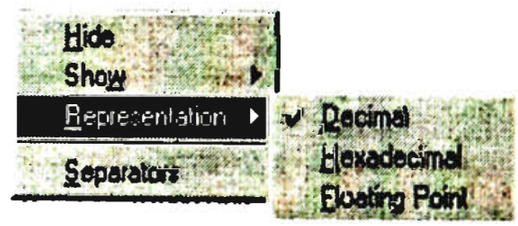
Mặc định, Step7 chỉ cho hiển thị nội dung các bits RLO, STA (từ thanh ghi trạng thái) và của ACCU1. Tuy nhiên ta có thể cho hiển thị thêm nội dung toàn bộ thanh ghi trạng thái hoặc của các thanh ghi khác bằng cách ấn phím chuột bên phải rồi chọn **Show→Tên thanh ghi** từ hộp hội thoại hiện ra (xem hình bên).



Chẳng hạn để quan sát thêm nội dung thanh ghi AR1 chỉ địa chỉ (con trỏ) ta kích phím chuột bên phải, sau đó chọn **Show** và tiếp theo là **Address Register 1**. Trên cửa sổ giám sát sẽ hiện ra thêm cột hiển thị nội dung của thanh ghi con trỏ AR1 như sau:



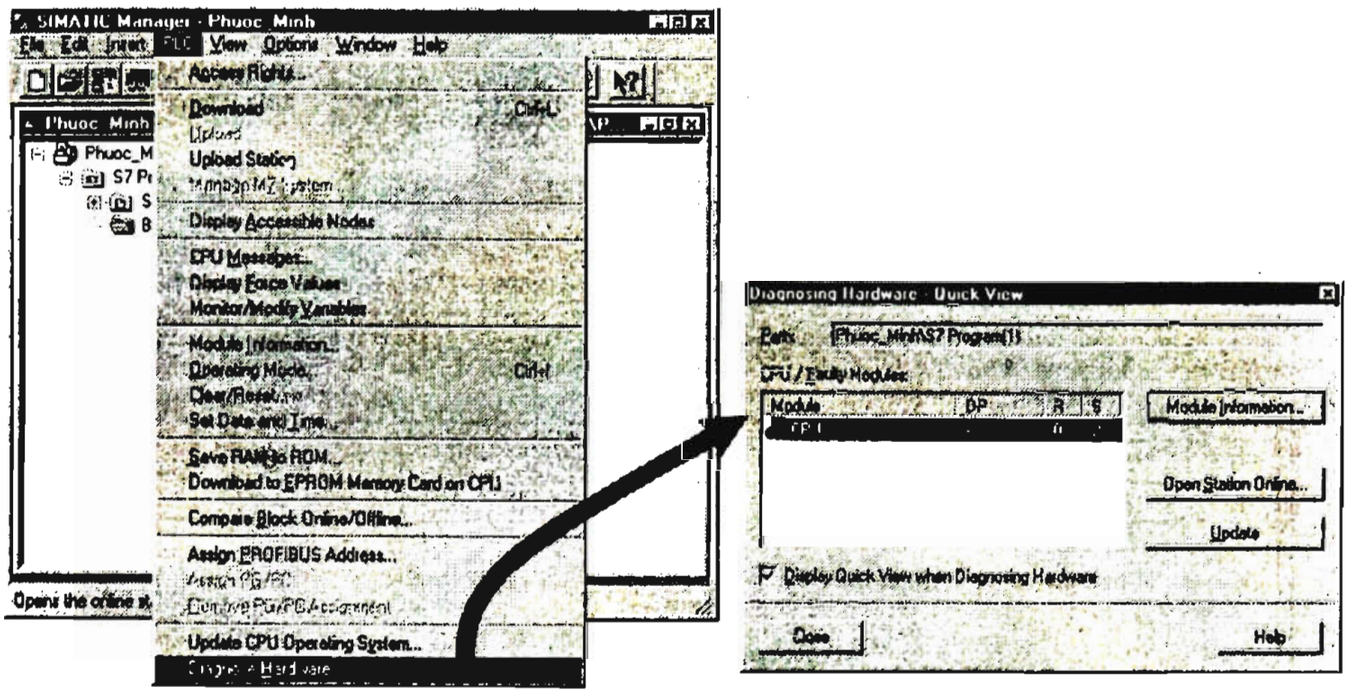
Ngoài ra, ta cũng có thể thay đổi kiểu dữ liệu được hiển thị. Mặc định Step7 sẽ cho hiển thị nội dung các thanh ghi dưới dạng mã hexadecimal, song ta có thể thay đổi sang các dạng khác như decimal hay số thực bằng cách đưa chuột vào vùng dữ liệu được hiển thị, ấn phím chuột bên phải rồi chọn **Representation→Kiểu dữ liệu** trong hộp hội thoại hiện ra có dạng như ở hình bên.



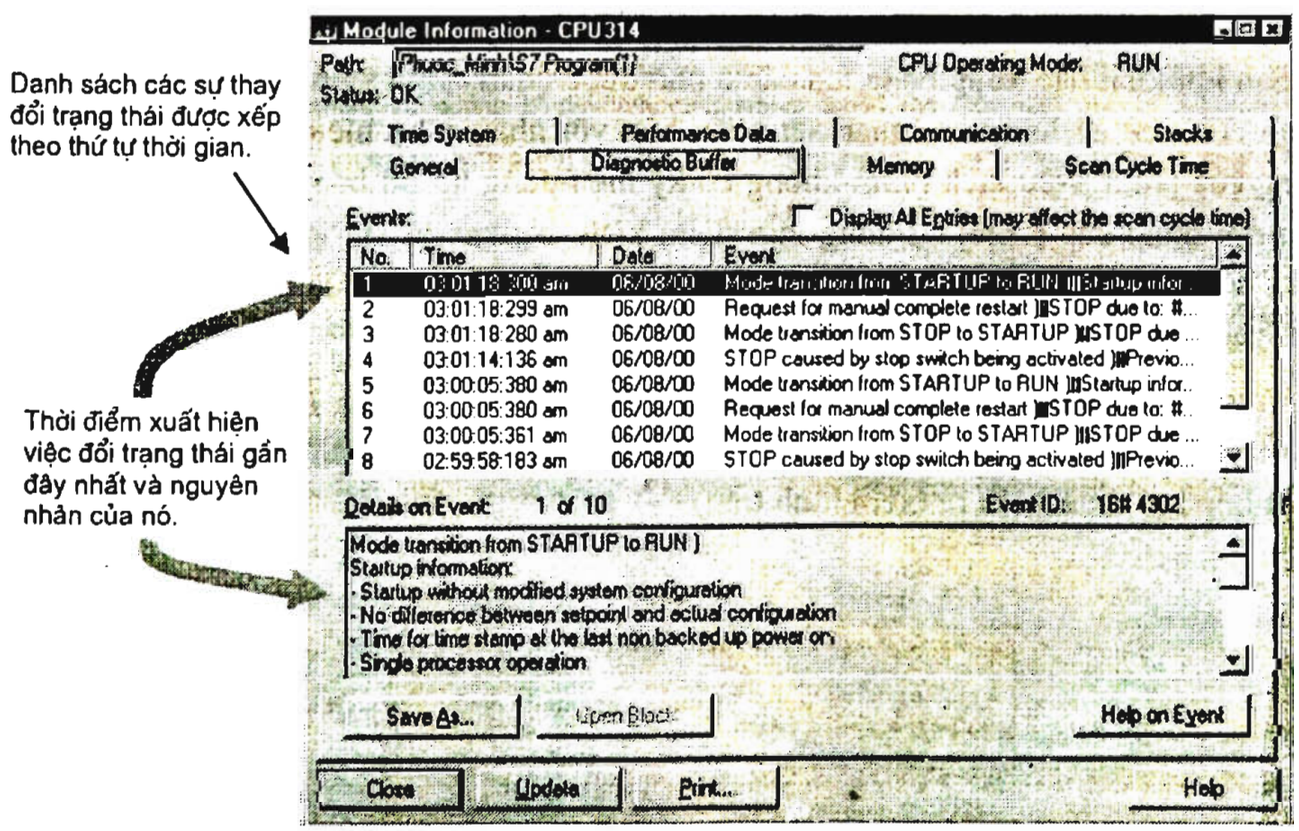
Chú ý rằng ta không thể sửa đổi được chương trình nếu cửa sổ màn hình giám sát đang ở trạng thái tích cực. Muốn quay trở về chế độ soạn thảo, ta phải rời khỏi màn hình giám sát bằng cách ấn phím **Monitor**. Tương tự, ta cũng không thể tích cực được cửa sổ màn hình giám sát nếu chương trình có trong Project không đồng nhất với chương trình có trong Load memory của module CPU. Bởi vậy để có thể giám sát được chương trình vừa được sửa đổi, công việc đầu tiên phải làm là ghi chương trình đó vào module CPU rồi sau đó mới tích cực cửa sổ màn hình giám sát. Hơn nữa ta cũng chỉ có thể giám sát việc thực hiện chương trình trong một khối và đó là khối đang được mở ở màn hình soạn thảo.

4.3.4 Giám sát module CPU

Bên cạnh việc giám sát chương trình, ta có thể giám sát cả công việc của module CPU bằng cách vào cửa sổ **PLC** trên thanh công cụ, sau đó chọn **Diagnose Hardware** sẽ có được hộp hội thoại:

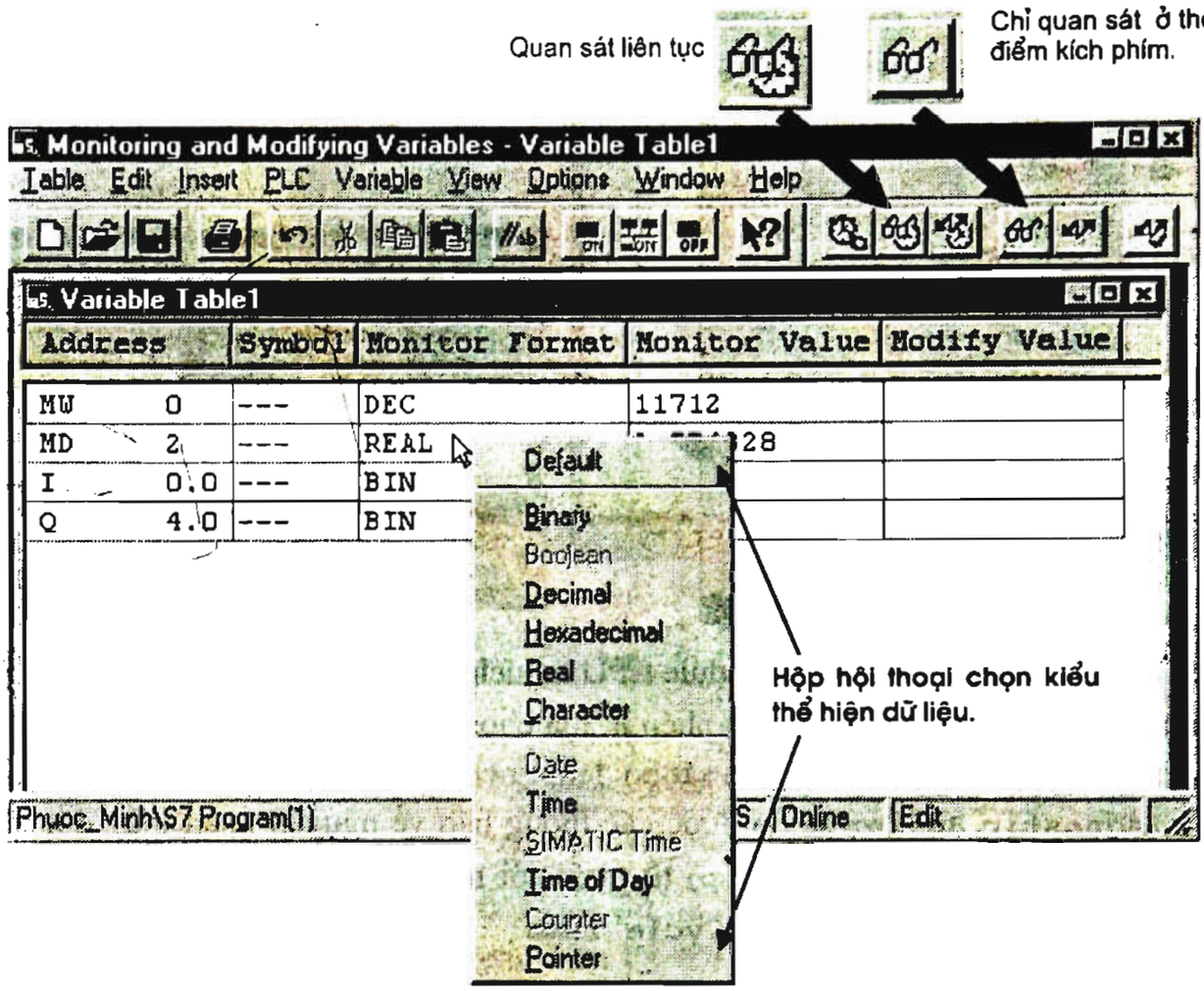


Nếu chỉ muốn giám sát riêng module CPU ta kích vào **Module Information**. Trên màn hình sẽ hiện tiếp ra cửa sổ cho phép ta lựa chọn cụ thể hình thức công việc được giám sát. Chẳng hạn nếu muốn quan sát bộ đệm tự chẩn đoán của module ta kích chuột vào ô **Diagnostic Buffer** sẽ có được các thông báo về nguyên nhân thay đổi trạng thái của module CPU (**Start↔Stop**) từ trước tới nay hoặc muốn quan sát thời gian thực hiện vòng quét ta chọn ô **Scan Cycle Time**.



4.3.5 Quan sát nội dung ô nhớ

Step7 cho phép quan sát nội dung mọi ô nhớ thuộc System memory và các ô nhớ có địa chỉ định nghĩa như PI, PQ. Những ô nhớ được quan sát phải được khai báo trước trong bảng có tên là **Variable Table** và để làm được điều này ta kích chuột tại **PLC** từ thanh công cụ màn hình chính của Step7 sau đó chọn **Monitor/Modify Variable**.



Sau khi khai báo xong bảng tên các ô nhớ được quan sát ta kích phím quan sát. Trên thanh công cụ có hai phím quan sát phân biệt với nhau ở ký hiệu đồng hồ trong biểu tượng của phím. Nếu kích phím không có ký hiệu đồng hồ thì bảng quan sát sẽ chỉ thể hiện nội dung của ô nhớ tại đúng thời điểm kích. Ngược lại khi kích phím có ký hiệu đồng hồ, Step7 sẽ liên tục truy nhập và đọc nội dung các ô nhớ của module CPU để thể hiện vào bảng.

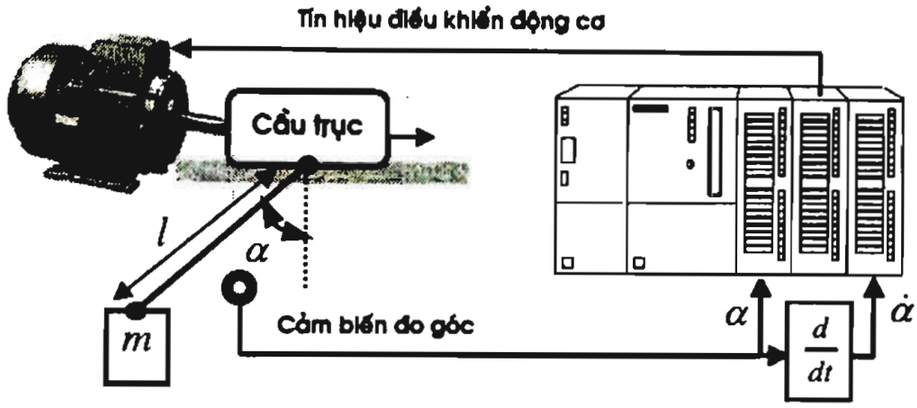
Ta cũng có thể thay đổi cách trình bày kiểu dữ liệu cho từng ô nhớ bằng cách đưa chuột vào ô nhớ cần thay đổi và kích phím phải của chuột. Sau đó chọn kiểu thích hợp trong hộp hội thoại hiện ra (xem hình trên).

5 ĐIỀU KHIỂN MỜ VỚI S7-300

5.1 Điều khiển mờ là gì?

5.1.1 Điều khiển không cần mô hình đối tượng

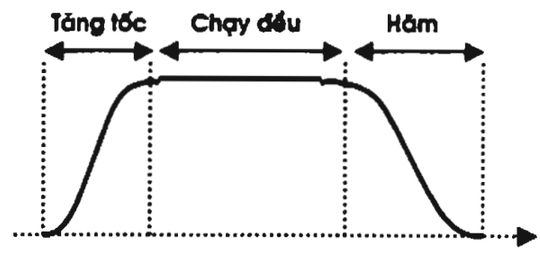
Xét một chiếc cần cẩu treo được mô tả về mặt nguyên lý trong hình 5.1. Cầu trục chuyển động theo phương nằm ngang để chuyển hàng dưới tác động của động cơ.



Hình 5.1. Điều khiển mờ cầu trục.

Hàng có khối lượng m được buộc với cẩu bằng một sợi dây độ dài l. Khi cầu trục chuyển động để vận chuyển hàng thì do tác động của những lực khác nhau, hàng sẽ dao động dưới cẩu treo như một con lắc. Dao động này được mô tả bằng góc α và tốc độ thay đổi góc $\dot{\alpha}$. Dao động sẽ càng lớn nếu gia tốc của cầu trục càng cao hoặc dây buộc hàng càng dài. Nếu xem toàn bộ cẩu treo và hàng như một đối tượng điều khiển thì tín hiệu vào của đối tượng sẽ là giá trị v đặt trước cho tốc độ động cơ.

Quá trình cẩu hàng được chia thành 3 giai đoạn: giai đoạn tăng tốc, giai đoạn chạy đều và giai đoạn hãm để dừng lại (hình 5.2). Mục đích của bài toán điều khiển là phải xác định tín hiệu điều khiển động cơ sao cho dao động của hàng bị triệt tiêu trong suốt quá trình cẩu hàng hoặc ít nhất thì cũng phải là trong giai đoạn hãm. Đây là bài toán khá toàn diện vì đặc thù



Hình 5.2. Ba giai đoạn của quá trình cẩu hàng

tham số thay đổi nhiều của đối tượng như khối lượng của các vật được cẩu không đồng đều, độ dài của dây buộc hàng cho các lần cẩu khác nhau ....

Về nguyên tắc, trước khi thiết kế bộ điều khiển người ta phải xây dựng một mô hình toán học cho đối tượng điều khiển. Mô hình xấp xỉ của cầu trục+hàng có dạng:

$$\underline{\dot{x}} = \begin{pmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & b & 0 \end{pmatrix} \underline{x} + \begin{pmatrix} 0 \\ c \\ 0 \\ d \end{pmatrix} \cdot v$$

trong đó x_1 là quãng đường đi của cầu trục, x_2 là tốc độ cầu trục, x_3 là quãng đương đi được của hàng và x_4 là tốc độ của hàng. Các tham số a, b, c, d phụ thuộc vào khối lượng hàng được vận chuyển và độ dài của dây treo hàng.

Theo những phương pháp điều khiển kinh điển thì để thiết kế bộ điều khiển cho cầu trục người ta có hai cách:

- Cách thứ nhất là thiết kế bộ điều khiển thỏa mãn yêu cầu đặt ra cho dù tham số mô hình đối tượng có thể bị thay đổi (bộ điều khiển bền vững).
- Cách thứ hai là là thiết kế bộ điều khiển có chế độ làm việc thay đổi phù hợp với sự thay đổi của các tham số mô hình đối tượng (điều khiển thích nghi).

Các lời giải thích nghi hoặc bền vững cho bài toán điều khiển này đã được tìm ra và cũng đã được ứng dụng nhiều. Ví dụ ở lời giải thích nghi, trong giai đoạn tăng tốc người ta cho gia tốc của động cơ không đổi và thông qua độ lớn của góc α họ xác định được khối lượng hàng được cẩu ..., từ đó tìm được giá trị điều khiển động cơ thích hợp để làm triệt tiêu góc α ở các giai đoạn sau. Tuy nhiên thiết bị thực hiện các lời giải đó là khá phức tạp kéo theo độ tin cậy của bộ điều khiển không cao.

Theo kinh nghiệm của các công nhân điều khiển cầu trục, ta chia các miền giá trị của góc α, tốc độ góc $\dot{\alpha}$, tốc độ động cơ v thành

- Biến α có các giá trị: dương (P), không (Z), âm ít (NS), âm nhiều (NB).
- Biến tốc độ góc $\dot{\alpha}$ có các giá trị: dương it (PS), không (Z), âm it (NS).
- Biến tốc độ động cơ v có các giá trị: dương ít (PS), không (Z), âm it (NS).

và đủ để hãm cầu trục mà hàng không dao động nếu sử dụng luật điều khiển:

R_1: Nếu α=P và $\dot{\alpha}$=PS thì v=PS

R_2: Nếu α=NB và $\dot{\alpha}$=Z thì v=NS

R_3: Nếu α=NS và $\dot{\alpha}$=PS thì v=Z

R_4: Nếu α=Z và $\dot{\alpha}$=PS thì v=Z

R_5: Nếu α=Z và $\dot{\alpha}$=NS thì v=Z

Một bộ điều khiển làm việc theo luật như trên sẽ được gọi là bộ điều khiển mờ. Khác hẳn với những phương pháp kinh điển, điều khiển mờ không cần đến mô hình toán học của đối tượng.

5.1.2 Bộ điều khiển mờ

Lý do chính dẫn tới suy nghĩ áp dụng logic mờ để điều khiển nằm ở chỗ trong rất nhiều trường hợp, con người chỉ cần dựa vào kinh nghiệm vẫn có thể điều khiển được đối tượng cho dù đối tượng có *thông số kỹ thuật không đúng hoặc thường xuyên bị thay đổi ngẫu nhiên và do đó mô hình toán học của đối tượng điều khiển không chính xác*, đó là chưa nói tới chúng có thể hoàn toàn sai. Việc điều khiển theo kinh nghiệm như vậy, mặc dù không thể đánh giá là chính xác như các thông số kỹ thuật đề ra (ví dụ như Điều khiển tối ưu), song đã giải quyết được vấn đề trước mắt là vẫn đảm bảo được về mặt định tính các chỉ tiêu chất lượng định trước.

Như vậy, kỹ thuật điều khiển mờ có thể được hiểu là việc *tự động hóa quá trình điều khiển theo kinh nghiệm*. Mở rộng ra, nhiều bộ điều khiển kinh điển cũng đang được chuyển dần sang nguyên lý mờ (xem thêm [3]). Về cơ bản, bộ điều khiển mờ cũng không có gì khác với các bộ điều khiển tự động thông thường khác. Sự khác biệt duy nhất là bộ điều khiển mờ làm việc theo luật được tổ chức thành tập các mệnh đề dạng:

Nếu có A_1 và $\cdots$ và có A_m thì phải có B_1 và $\cdots$ và phải có B_s

Nếu nói rằng làm việc với bộ điều khiển mờ có thể giải quyết được mọi vấn đề từ trước đến nay chưa giải quyết được theo phương pháp kinh điển thì sẽ hơi quá, song với điều khiển mờ ta có một khả năng tích hợp bộ điều khiển đơn giản làm việc theo kinh nghiệm của con người mà nếu theo phương pháp kinh điển việc thiết kế một bộ điều khiển như vậy thường bị bế tắc do hệ thống có độ phức tạp cao, độ phi tuyến lớn, có sự thường xuyên thay đổi trạng thái và cấu trúc của đối tượng... hoặc giả nếu có tổng hợp được trong phạm vi lý thuyết thì khi thực hiện cũng gặp không ít khó khăn về giá thành và độ tin cậy của sản phẩm.

Tóm lại, bộ điều khiển mờ có những ưu điểm cơ bản:

1) Việc tổng hợp bộ điều khiển mờ đơn giản. Những bộ điều khiển phi tuyến phức tạp cũng có thể được tổng hợp không khó khăn gì theo nguyên tắc mờ.
2) Việc tổng hợp bộ điều khiển mờ không cần đến mô hình toán học của đối tượng.
3) Khối lượng công việc thiết kế và khối lượng tính toán giảm dẫn đến giá thành thành sản phẩm hạ.
4) Bộ điều khiển mờ dễ hiểu hơn so với các bộ điều khiển khác.
5) Trong nhiều trường hợp bộ điều khiển mờ làm việc ổn định hơn, bền vững hơn và có chất lượng cao hơn.

Chính những ưu điểm và khả năng này mà trong những năm gần đây kỹ thuật điều khiển mờ đã nhanh chóng tìm được miền ứng dụng rộng lớn.

5.2 Những khái niệm cơ bản

5.2.1 Tập mờ

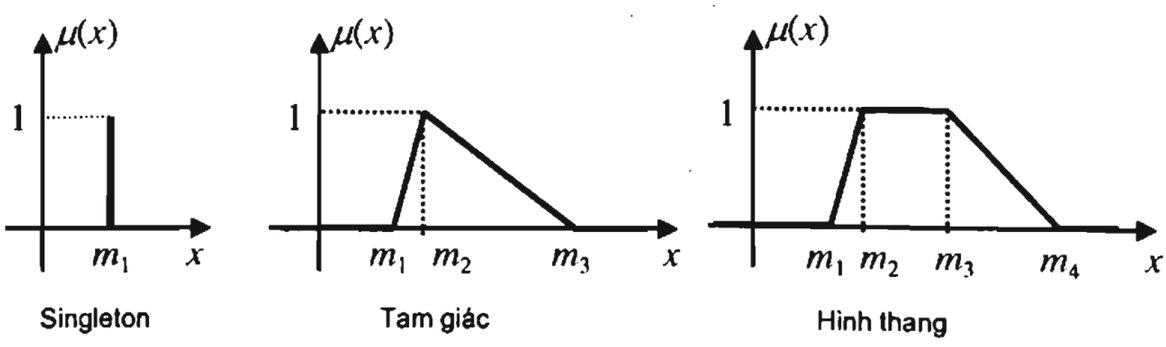
Khái niệm cơ bản được sử dụng trong hệ thống điều khiển mờ là tập mờ, dùng để mô tả các giá trị ngôn ngữ. Hãy lấy ví dụ về việc mô tả độ cao của cột nước trong bình, nếu cột nước hiện thời là 2m thì cột nước này vẫn có thể được đánh giá là thuộc cả 3 loại giá trị ngôn ngữ là "trung bình", "hơi cao" và "rất cao ", tuy nhiên với những mức độ đánh giá (độ phụ thuộc) khác nhau.

Tổng quát thì tập mờ $\mathcal{F}$ là một tập hợp mà mỗi phần tử được gán thêm một số thực $\mu(x)$ trong khoảng [0,1] để chỉ thị *độ phụ thuộc* của phần tử đó vào tập đã cho. Khi $\mu(x)=0$, phần tử x sẽ hoàn toàn không thuộc $\mathcal{F}$ (xác suất phụ thuộc bằng 0), ngược lại với $\mu(x)=1$, phần tử cơ bản sẽ thuộc $\mathcal{F}$ với xác suất 100%.

Phần tử x được gọi là phần tử cơ bản và tập kinh điển chỉ chứa riêng x (không có độ phụ tuộc $\mu(x)$) có tên là *tập nền* của tập mờ $\mathcal{F}$. Như vậy, tập mờ là tập hợp các cặp $(x,\mu(x))$. Khi x chạy khắp trên tập nền thì $\mu(x)$ sẽ là một hàm thực và được gọi là *hàm thuộc*. Nói chung một hàm bất kỳ không âm và không lớn hơn 1 đều có thể là hàm thuộc $\mu(x)$. Nhưng trong điều khiển, với mục đích sử dụng các dạng hàm thuộc sao cho khả năng tích hợp chúng là đơn giản, người ta thường chỉ quan tâm đến ba dạng hàm thuộc cơ bản. Đó là (hình 5.3)

- Hàm Singleton (còn gọi là Kronecker).
- Hàm hình tam giác.
- Hàm hình thang.

Mỗi một *giá trị ngôn ngữ* được biểu diễn thành một tập mờ. Ví dụ các giá trị "cột nước cao trung bình", "cột nước hơi cao" hoặc "cột nước rất cao" là những tập mờ.



Hình 5.3: Những dạng hàm thuộc thường dùng.

5.2.2 Phép tính trên tập mờ

Những phép tính trên tập mờ là những quy định về các phép toán được thực hiện như phép hợp hai tập mờ, giao hai tập mờ .... Các quy định này không thể tùy ý mà phải tuân thủ nguyên tắc là không được mâu thuẫn với phép tính logic kinh điển.

- Hợp $A \cup B$ của hai tập mờ A và B được hiểu là một tập mờ gồm các phần tử của hai tập A, B đã cho, trong đó hàm thuộc $\mu_{A\cup B}$ của phần tử của $A \cup B$ không được mâu thuẫn với phép hợp của hai tập kinh điển. Ví dụ

$$\mu_{A\cup B}(x) = \max\{\mu_A(x) , \mu_B(x)\} \quad \text{Luật max} \quad (5.1a)$$

$$\mu_{A\cup B}(x) = \min\{1, \mu_A(x)+\mu_B(x)\} \quad \text{Luật tổng} \quad (5.1b)$$

- Giao $A \cap B$ của hai tập mờ A và B được hiểu là một tập mờ gồm các phần tử của hai tập A, B đã cho, trong đó hàm thuộc $\mu_{A\cap B}$ của phần tử của $A \cap B$ không được mâu thuẫn với phép giao của hai tập kinh điển. Ví dụ

$$\mu_{A\cap B}(x) = \min\{\mu_A(x) , \mu_B(x)\} \quad \text{Luật min} \quad (5.2a)$$

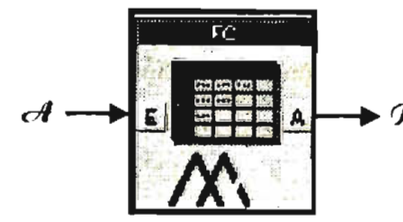
$$\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x) \quad \text{Luật tích} \quad (5.2b)$$

5.2.3 Mệnh đề hợp thành

Mệnh đề hợp thành

R: Nếu $\mathcal{A}=A$ thì $\mathcal{B}=B$

là phép suy diễn mờ "Từ A suy ra B", trong đó tập mờ A với hàm thuộc $\mu_A(x)$ là một giá trị của biến ngôn ngữ đầu vào $\mathcal{A}$ và tập mờ B với hàm thuộc $\mu_B(y)$ cũng là một giá trị của biến ngôn ngữ đầu ra $\mathcal{B}$. Do mệnh đề hợp thành này có 1 biến đầu vào $\mathcal{A}$ và một biến đầu ra $\mathcal{B}$ nên nó được gọi là mệnh đề SISO hay phép suy diễn SISO (single input–single output).



Hình 5.4: Động cơ suy diễn SISO

Khi cho trước một giá trị rõ cụ thể x_0 , tính đúng đắn của phép suy diễn trên sẽ được đánh giá bởi một tập mờ B' cùng nền với B. Nói cách khác kết quả của phép suy diễn ứng với x_0 tại đầu vào là một tập mờ B'. Hàm thuộc của B' được ký hiệu bằng $\mu_{B'}(y)$. Hai công thức xác định $\mu_{B'}(y)$ thường được dùng trong điều khiển là

$$\mu_{B'}(y)= \min\{\mu_A(x_0) , \mu_B(y)\} \quad \text{Luật min} \quad (5.3a)$$

$$\mu_{B'}(y)= \mu_A(x_0)\mu_B(y) \quad \text{Luật tích} \quad (5.3b)$$

trong đó giá trị $H =\mu_A(x_0)$ được gọi là *độ thỏa mãn đầu vào*. Để ngắn gọn ta sẽ viết phép suy diễn "Nếu $\mathcal{A}=A$ thì $\mathcal{B}=B$" với giá trị rõ x_0 là $H \mapsto \mu_{B'}(y)$.

5.2.4 Luật hợp thành

Luật hợp thành là tập hợp của nhiều mệnh đề hợp thành cùng cấu trúc

R_1: Nếu $\mathcal{A}_1=A_{11}$ và $\mathcal{A}_2=A_{12}\cdots$ và $\mathcal{A}_m=A_{1m}$ thì $\mathcal{B}=B_1$ hoặc (5.4a)

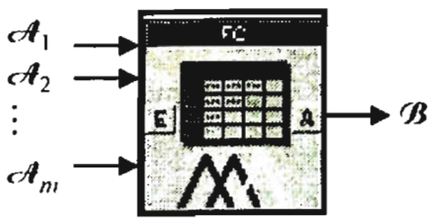
R_2: Nếu $\mathcal{A}_1=A_{21}$ và $\mathcal{A}_2=A_{22}\cdots$ và $\mathcal{A}_m=A_{2m}$ thì $\mathcal{B}=B_2$ hoặc (5.4b)

$\vdots$

R_n: Nếu $\mathcal{A}_1=A_{n1}$ và $\mathcal{A}_2=A_{n2}\cdots$ và $\mathcal{A}_m=A_{nm}$ thì $\mathcal{B}=B_n$ (5.4c)

trong đó A_{ij} là các giá trị ngôn ngữ (tập mờ) của biến ngôn ngữ $\mathcal{A}_j$ và B_i là những giá trị ngôn ngữ của $\mathcal{B}$, $i=1,2,\ldots,m$; $j=1,2,\ldots,n$.

Ở đây có điểm khác biệt so với mệnh đề hợp thành vừa trình bày ở trên là mỗi một mệnh đề hợp thành dạng (5.4) R_k , $k=1,2,\ldots,n$ có nhiều biến ngôn ngữ đầu vào $\mathcal{A}_j$, $j=1,2,\ldots,n$ do đó nó còn được gọi là *mệnh đề hợp thành* MISO (multi inputs, single output). Tuy nhiên giá trị của nó cũng là một tập mờ B_k' và cũng được xác định theo (5.3), trong đó tại mệnh đề R_k giá trị H_k được xác định bởi



Hình 5.5. Động cơ suy diễn MISO.

$$H_k = \min_{1\le j\le m} \mu_{A_{kj}}(x_j) \tag{5.5}$$

nói cách khác

$$B_k' = H_k \mapsto \mu_{B_k'}(y) \tag{5.6}$$

trong đó vector $\underline{x}$ với các phần tử x_j, $j=1,\ldots,m$ là những giá trị rõ của m tín hiệu đầu vào. Quá trình biến đổi x_j, $j=1,\ldots,m$ thành H_k theo (5.5) được gọi là quá trình mờ hóa.

Do kết quả của mỗi mệnh đề hợp thành R_k là một tập mờ B_k' nên kết quả B' của luật hợp thành (5.4) sẽ phải là hợp của các tập mờ đó:

$$B' = \bigcup_{k=1}^{n} B_k' \tag{5.7}$$

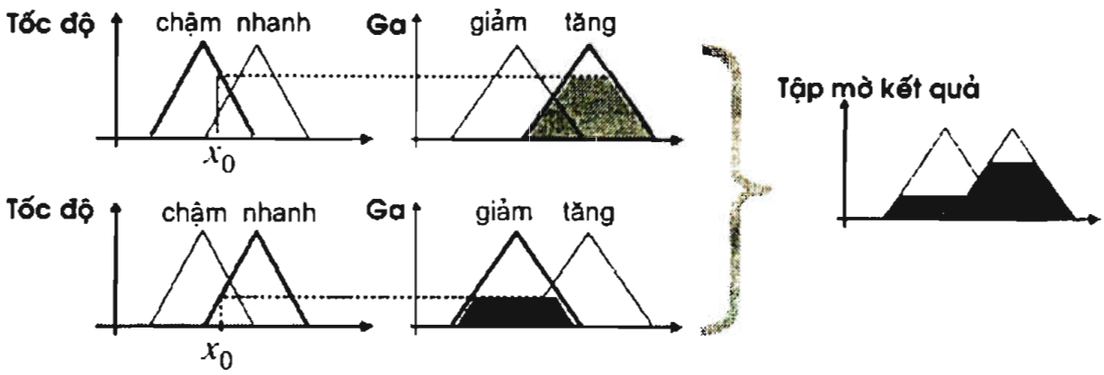
Như vậy, để xác định B' của luật hợp thành (5.4) một cách cụ thể ta cần phải có:

- Một công thức trong số hai công thức (5.3a),(5.3b) để thực hiện phép suy diễn mờ (5.6) để có B_k' .
- Một công thức trong số hai công thức (5.1a),(5.1b) để thực hiện phép hợp các tập mờ theo (5.7).

Với những cách quy định khác nhau ta có các thuật toán xác định B' khác nhau. Mỗi một thuật toán như vậy có tên là động cơ suy diễn. Trong điều khiển người ta hay dùng bốn động cơ suy diễn chính. Đó là:

1) Động cơ suy diễn max–MIN, nếu (5.1a) được sử dụng cho phép hợp mờ, (5.3a) cho phép suy diễn mờ.
2) Động cơ suy diễn max–PROD, nếu (5.1a) được sử dụng cho phép hợp mờ, (5.3b) cho phép suy diễn mờ.
3) Động cơ suy diễn sum–MIN, nếu (5.1b) được sử dụng cho phép hợp mờ, (5.3a) cho phép suy diễn mờ.
4) Động cơ suy diễn sum–PROD, nếu (5.1b) được sử dụng cho phép hợp mờ, (5.3b) cho phép suy diễn mờ và (5.2a) cho phép giao mờ.

Hình 5.6. Ví dụ về cách xác định giá trị (mờ) của luật hợp thành SISO.



Hình 5.6 là một ví dụ minh họa cho việc xác định giá trị (mờ) của một luật hợp thành mô phỏng kinh nghiệm giữ ổn định tốc độ xe ô tô, gồm hai mệnh đề hợp thành SISO (một biến vào và một biến ra) ứng với giá trị rõ x_0 tại đầu vào khi động cơ suy diễn được áp dụng là max–MIN.

Do luật hợp thành (5.4) có n biến ngôn ngữ đầu vào $\mathcal{A}_j$, j=1,2,...n và một biến ngôn ngữ đầu ra $\mathcal{B}$ nên động cơ suy diễn thực hiện nó còn được gọi là động cơ suy diễn thực hiện luật hợp thành MISO (multi inputs–single output).

Đôi khi, trong ứng dụng người ta lại cần phải thực hiện luật hợp thành có nhiều đầu vào và nhiều đầu ra (MIMO)

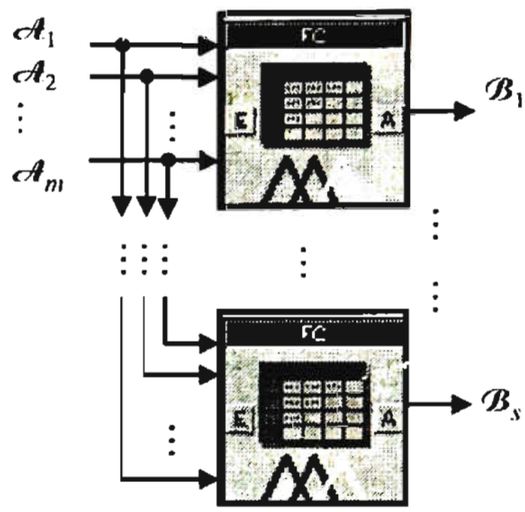
$$R_1: \quad \text{Nếu } \mathcal{A}_1=A_{11} \text{ và } \cdots \text{ và } \mathcal{A}_m=A_{1m} \text{ thì } \mathcal{B}_1=B_{11} \text{ và } \cdots \text{ và } \mathcal{B}_s=B_{1s} \qquad (5.8a)$$

$$R_2: \quad \text{Nếu } \mathcal{A}_1=A_{21} \text{ và } \cdots \text{ và } \mathcal{A}_m=A_{2m} \text{ thì } \mathcal{B}_1=B_{21} \text{ và } \cdots \text{ và } \mathcal{B}_s=B_{2s} \qquad (5.8b)$$

$$\vdots$$

$$R_n: \quad \text{Nếu } \mathcal{A}_1=A_{n1} \text{ và } \cdots \text{ và } \mathcal{A}_m=A_{nm} \text{ thì } \mathcal{B}_1=B_{n1} \text{ và } \cdots \text{ và } \mathcal{B}_s=B_{ns} \qquad (5.8c)$$

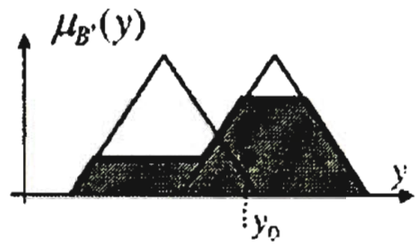
Để có được động cơ suy diễn cho (5.8), cách đơn giản nhất là chuyển mệnh đề hợp thành R_k có nhiều đầu vào/ra (MIMO) về dạng hợp của s mệnh đề nhiều vào một ra (MISO) *rời nhau* và như vậy động cơ suy diễn thực hiện (5.8) sẽ chính là sự ghép nối song song của s động cơ suy diễn kiểu MISO (hình 5.7).



Hình 5.7. Động cơ suy diễn MIMO.

5.2.5 Giải mờ

Giải mờ có nhiệm vụ chuyển đổi một tập mờ B' với hàm thuộc $\mu_{B'}(y)$ thành một giá trị rõ y_0 bằng cách lấy một phần tử cơ bản của tập nền làm đại diện. Về nguyên tắc có nhiều phương pháp lấy một phần tử đại diện như



Hình 5.8. Giải mờ theo phương pháp điểm trọng tâm.

- phương pháp giá trị cực đại bên phải,
- phương pháp các giá trị cực đại bên trái,
- phương pháp giá trị trung bình,

song trong điều khiển, phương pháp được ưa chuộng nhất là phương pháp điểm trọng tâm (hình 5.8):

$$y_0 = \frac{\int_{-\infty}^{\infty} y\mu_{B'}(y)dy}{\int_{-\infty}^{\infty} \mu_{B'}(y)dy}. \qquad (5.6)$$

Như vậy giá trị y_0 rõ chính là hoành độ của điểm trọng tâm của tập mờ B'.

5.2.6 Các bước tổng hợp bộ điều khiển mờ

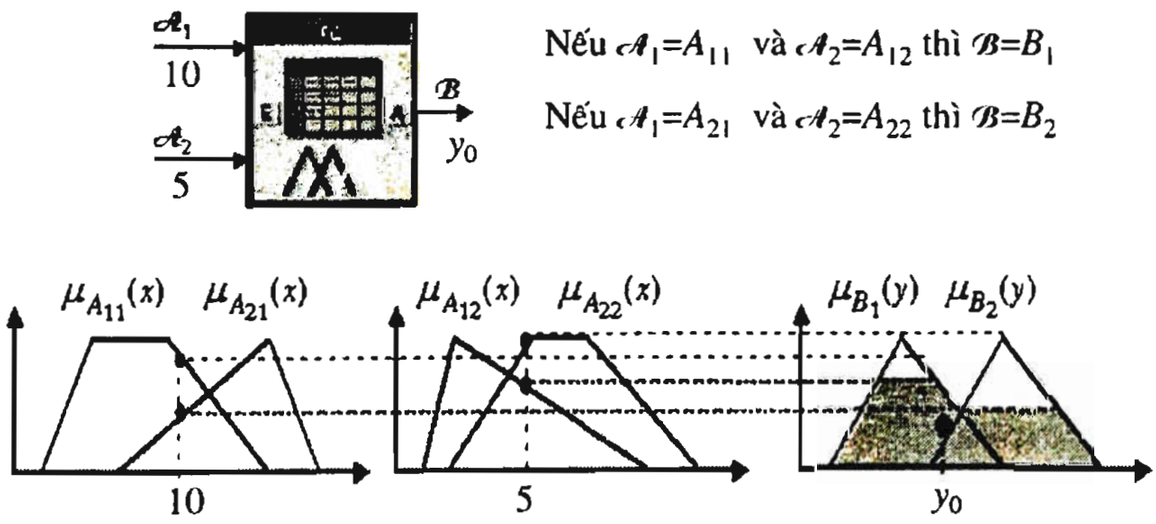
Giả thiết rằng, người thiết kế đã có đủ các kinh nghiệm và muốn chuyển nó thành bộ điều khiển thì phải tiến hành các bước sau đây:

- định nghĩa tất cả các biến ngôn ngữ vào và ra,
- định nghĩa tập mờ (giá trị ngôn ngữ) cho các biến vào/ ra,
- xây dựng các luật điều khiển (luật hợp thành),
- chọn động cơ suy diễn,
- chọn phương pháp giải mờ.

Các bước tiến hành việc thiết kế một bộ điều khiển mờ trên đây sẽ được trình bày rõ hơn trong với phần hướng dẫn sử dụng *Fuzzy Control Parameter Assignment* cho PLC Simatic S7–300.

5.2.7 Ví dụ minh họa

Hình 5.8 là một ví dụ minh họa cho việc xác định giá trị (mờ) của một luật hợp thành gồm hai mệnh đề hợp thành MISO ứng vớigiá trị rõ 10 và 5 tại đầu vào. Động cơ suy diễn được áp dụng là max–MIN và phương pháp giải mờ là phương pháp điểm trọng tâm.



Hình 5.8. Xác định giá trị rõ tại đầu ra của một luật hợp thành MISO với hai đầu vào, một đầu ra khi tại đầu vào có các giá trị rõ 10 và 5. Động cơ suy diễn là max–MIN và phương pháp giải mờ là phương pháp điểm trọng tâm.

5.3 Chương trình FCPA

5.3.1 Chuẩn bị một Project cho việc khai báo bộ điều khiển mờ bằng FCPA

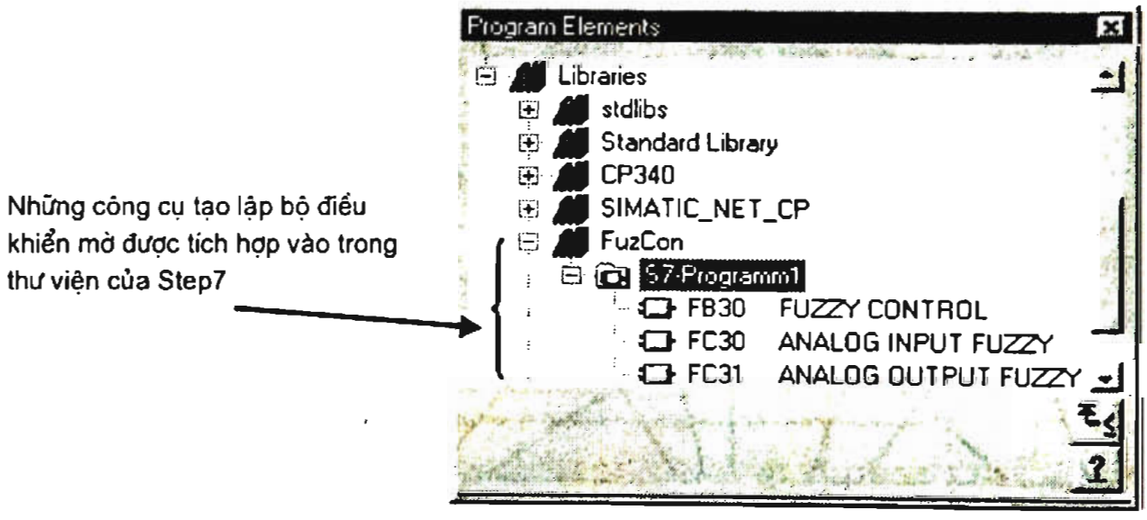
Chương trình FCPA (chữ viết tắt của *Fuzzy Control Parameter Assignment*) là phần mềm hỗ trợ việc tạo lập bộ điều khiển mờ cho PLC Simatic S7–300 theo từng bước như đã trình bày tại mục 5.2.6.

Trước hết ta phải cài đặt FCPA trên máy tính cá nhân. Việc cài đặt thành công FCPA đòi hỏi:

- Có ít nhất 1MBytes còn trống trong ổ cứng.
- Chạy dưới hệ điều hành Window 95/98 hoặc NT.

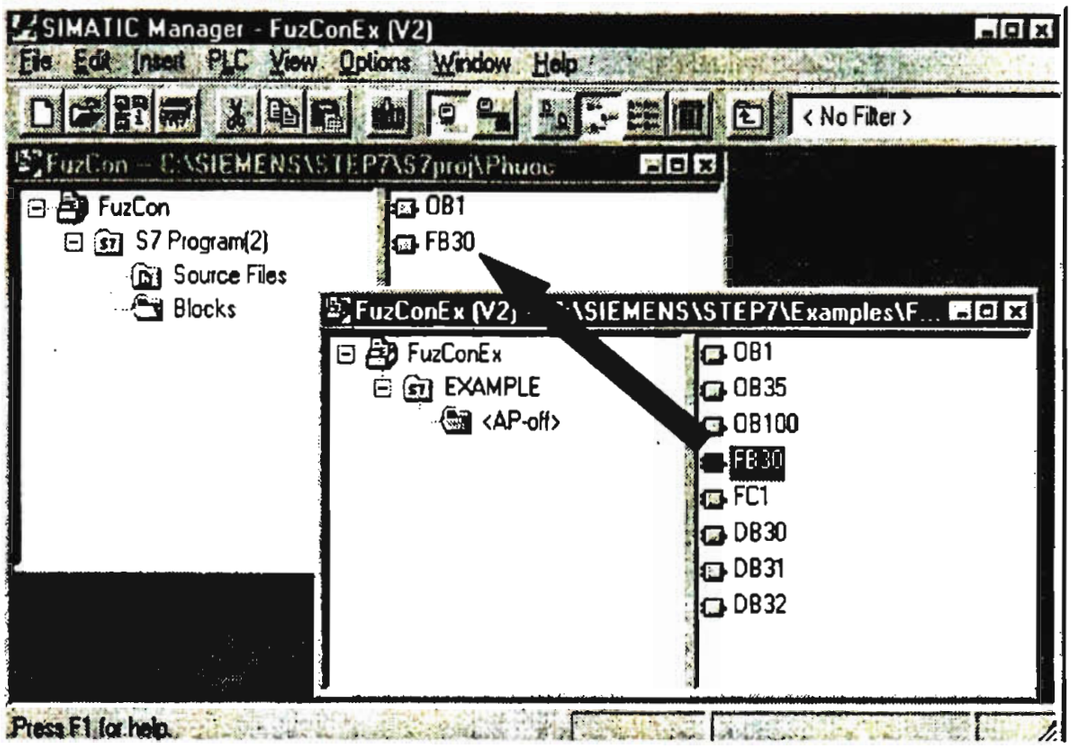
Toàn bộ chương trình gốc của FCPA gồm 2 phần **Fuzzy/Tool** và **Fuzzy/FB** với dung lượng tổng cộng 2,27MB. Để cài đặt, ta gọi tệp **Setup.exe** của **Fuzzy/Tool** và của **Fuzzy/FB** từ Window và thực hiện những chỉ dẫn hiện trên màn hình. Chú ý là chỉ cài đặt FCPA sau khi đã cài Step7.

Sau khi đã được cài đặt, phần chính của FCPA sẽ được tích hợp trong Step7 dưới thư mục **S7WRFUZ**, các công cụ hỗ trợ khác được đưa vào thư viện của phần mềm Step7, cũng như Project **FuzConEx**.



Bộ điều khiển mờ được tổng hợp với FCPA có dạng một khối dữ liệu (DB) cho Project ứng dụng. Khối DB tạo bởi FCPA sẽ được gọi là *khối DB mờ* và được sử dụng cùng với FB **Fuzzy Control** có trong Project **FuzConEx** khi cài đặt chương trình **Fuzzy/FB** với tên mặc định là FB30. Bởi vậy trước khi sử dụng FCPA để tạo lập DB mờ cho Project ứng dụng, bắt buộc Project ứng dụng đã phải có FB **Fuzzy Control**.

Ví dụ, Project ứng dụng của ta có tên là **FuzCon**. Trước khi sử dụng FCPA để tạo lập khối DB mờ cho Project ứng dụng **FuzCon**, ta phải sao chép FB **Fuzzy Control** có tên mặc định FB30 từ Project **FuzConEx** sang Project **FuzCon**. Có thể thay đổi tên FB30 nếu như trong Project ứng dụng của ta đã có một FB trùng tên.

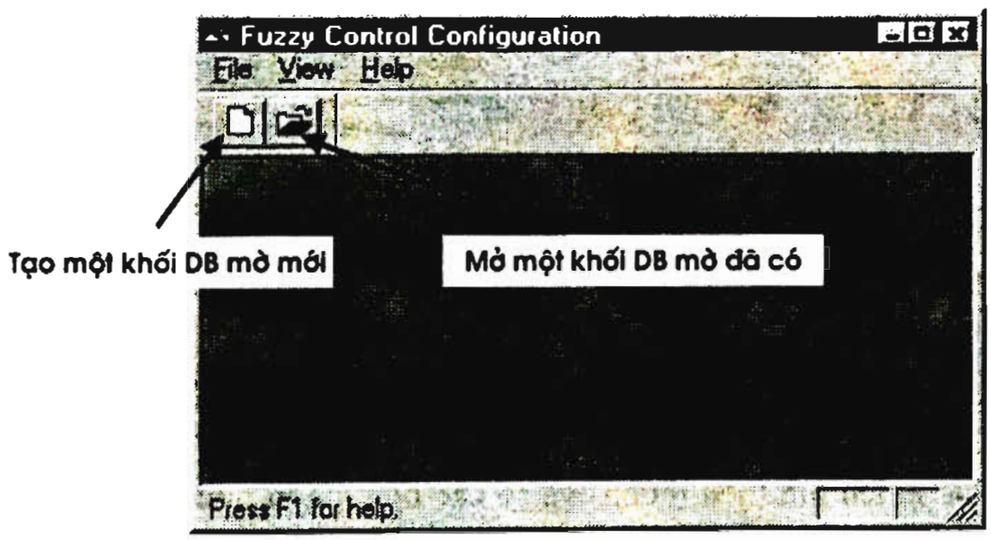


5.3.2 Tạo DB mờ

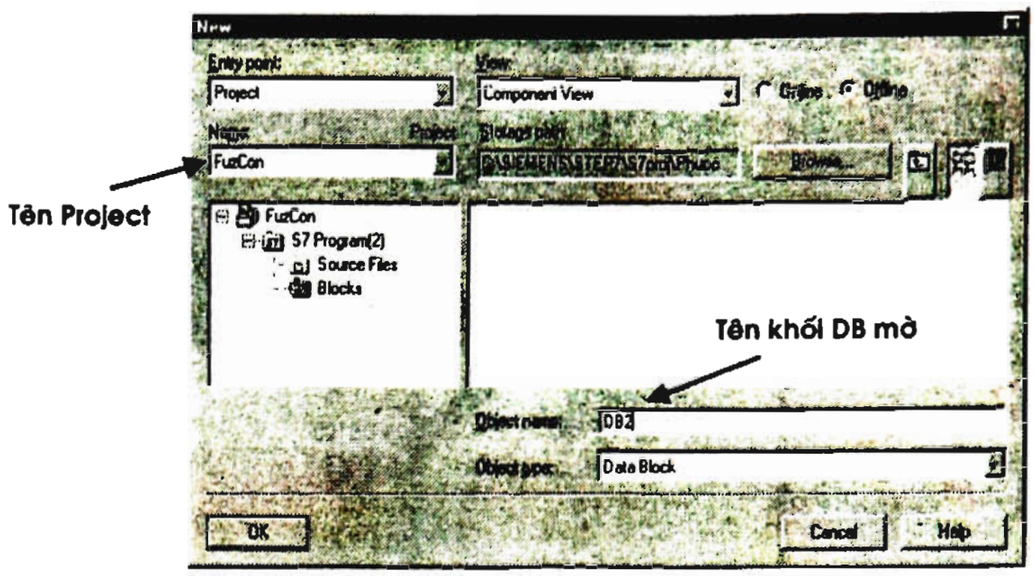
Sau khi đã chuẩn bị một Project ứng dụng cho bộ điều khiển mờ (Project có chứa FB Fuzzy Control), ta có thể bắt đầu sử dụng FCPA để tạo lập DB mờ cho bộ điều khiển mờ và khối DB mờ này phải nằm trong cùng một thư mục với FB Fuzzy Control của Project ứng dụng. Để vào FCPA ta thực hiện lệnh gọi từ Window theo thứ tự:

`Start → Simatic → Step7 → Fuzzy Control Parameter Assignment`

khi đó trên màn hình sẽ xuất hiện của sổ



Do khối DB mờ phải nằm trong một Project nào đó nên khi kích vào một trong hai biểu tượng trên, FCPA sẽ yêu cầu ta cho biết tên Project chứa khối DB mờ đó. Chẳng hạn khi kích vào biểu tượng tạo DB mờ mới và khối DB mờ được tạo ra này sẽ phải nằm trong Project có tên FuzCon thì ta phải cho FCPA biết tên sẽ được đặt cho khối DB mờ, (ví dụ DB2) và tên của Project là FuzCon. Cửa sổ màn hình khai báo các dữ liệu đó có dạng như sau:

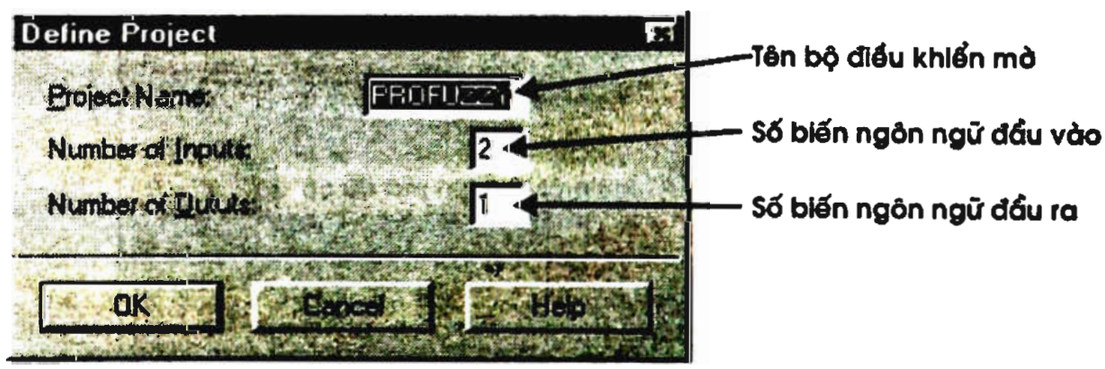


Sau khi đã cho đầy đủ tên Project, tên khối DB mở, ta ấn phím OK. Chương trình FCPA sẽ kiểm tra lại trong Project FuzCon thực sự đã có khối hàm Fuzzy Control hay chưa bằng thông báo liệt kê tất cả các khối hàm đã có trong Project ứng dụng. Ta phải chọn trong bảng danh mục được liệt kê ra đó khối hàm Fuzzy Control đã được lấy từ Project **FuzConEx** sang.

Ấn phím OK để xác nhận và ta bắt đầu vào công việc tổng hợp bộ điều khiển mờ với phần mềm FCPA.

5.3.3 Khai báo số các biến ngôn ngữ vào ra

Nếu tạo một DB mờ mới thì sau khi ấn phím OK xác nhận khối FB Fuzzy Control, chương trình FCPA sẽ hỏi số các biến ngôn ngữ vào/ra của bộ điều khiển mờ bằng hộp hội thoại

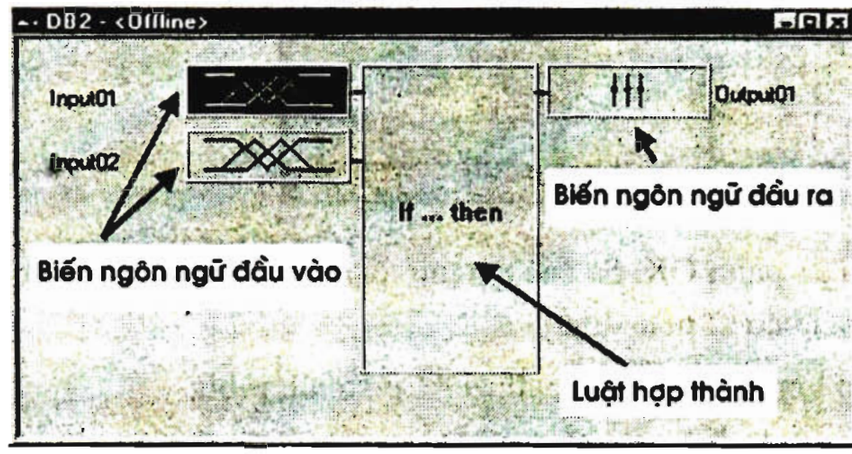


Viết tên bộ điều khiển mờ (nếu muốn) và số các biến ngôn ngữ vào ra vào những ô tương ứng. Hạn chế của FCPA là:

- chỉ tạo lập được những bộ điều khiển mờ với tối đa 8 biến vào.
- chỉ tạo lập được bộ điều khiển với tối đa 4 biến ra.

Ấn phím OK để xác nhận các giá trị vừa cho. Những biến ngôn ngữ đầu vào sẽ có tên mặc định Input01, Input02, ... và Output01, Output02, ... lần lượt là tên mặc định của các biến ngôn ngữ đầu ra.

Sau khi ấn OK, trên màn hình sẽ xuất hiện cửa sổ soạn thảo tiếp giá trị ngôn ngữ của từng biến vào/ra cũng như luật hợp thành của bộ điều khiển mờ, như sau:

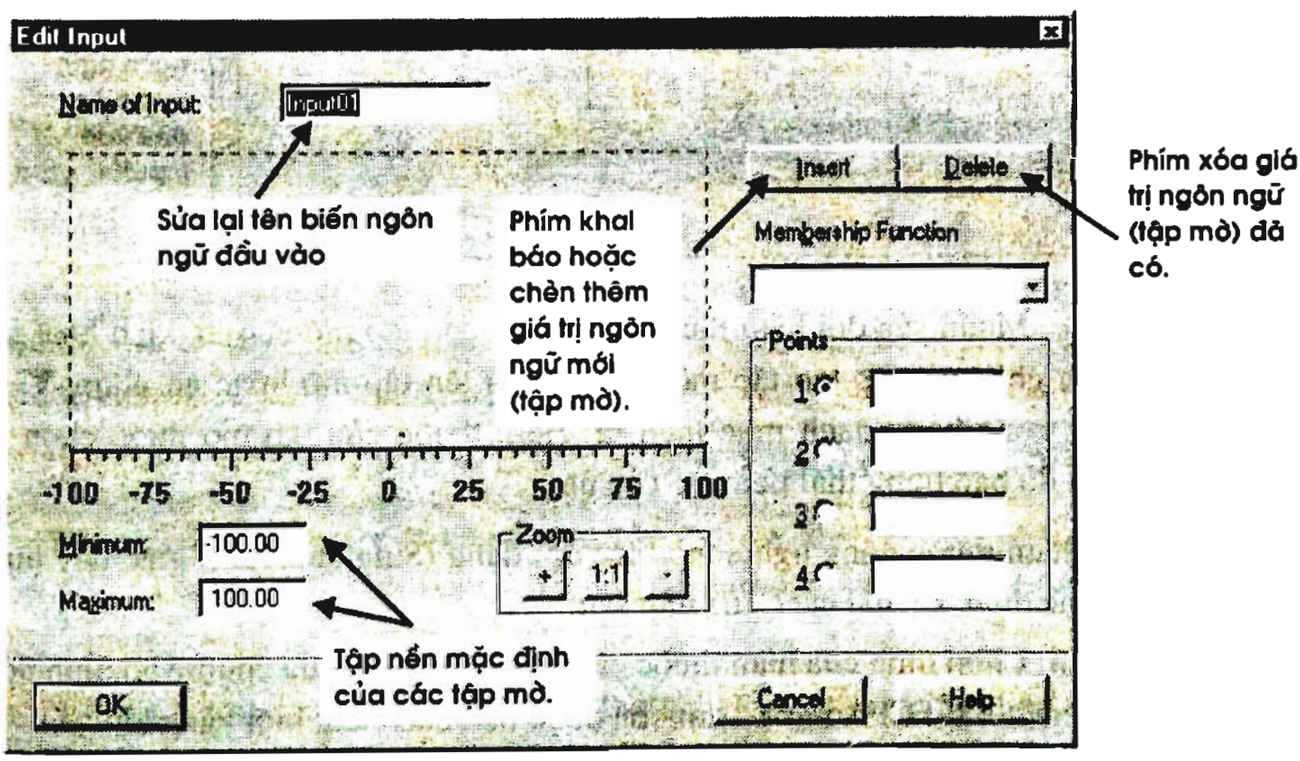


5.3.4 Soạn thảo giá trị cho từng biến (ngôn ngữ) đầu vào

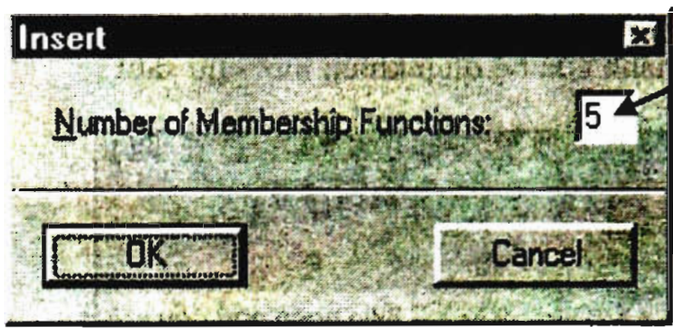
Các giá trị của mỗi một biến ngôn ngữ đầu vào được gọi là biến ngôn ngữ. Vì bản chất của giá trị ngôn ngữ là tập mờ, nên để soạn thảo giá trị ngôn ngữ cho một biến ngôn ngữ ta cần phải:

1) Khai báo số các giá trị ngôn ngữ (tập mờ) của biến.
2) Soạn thảo tập nền, cũng như hàm thuộc cho từng giá trị ngôn ngữ.

Để vào chế độ soạn thảo giá trị ngôn ngữ (tập mờ) cho một biến đầu vào nào đó, ta nháy kép phím chuột trái tại biểu tượng của biến đó. Ví dụ để soạn thảo giá trị cho biến vào Input01, ta nháy kép vào biểu tượng của nó (đã được đánh dấu trên màn hình). Khi đó cửa sổ soạn thảo hiện ra:



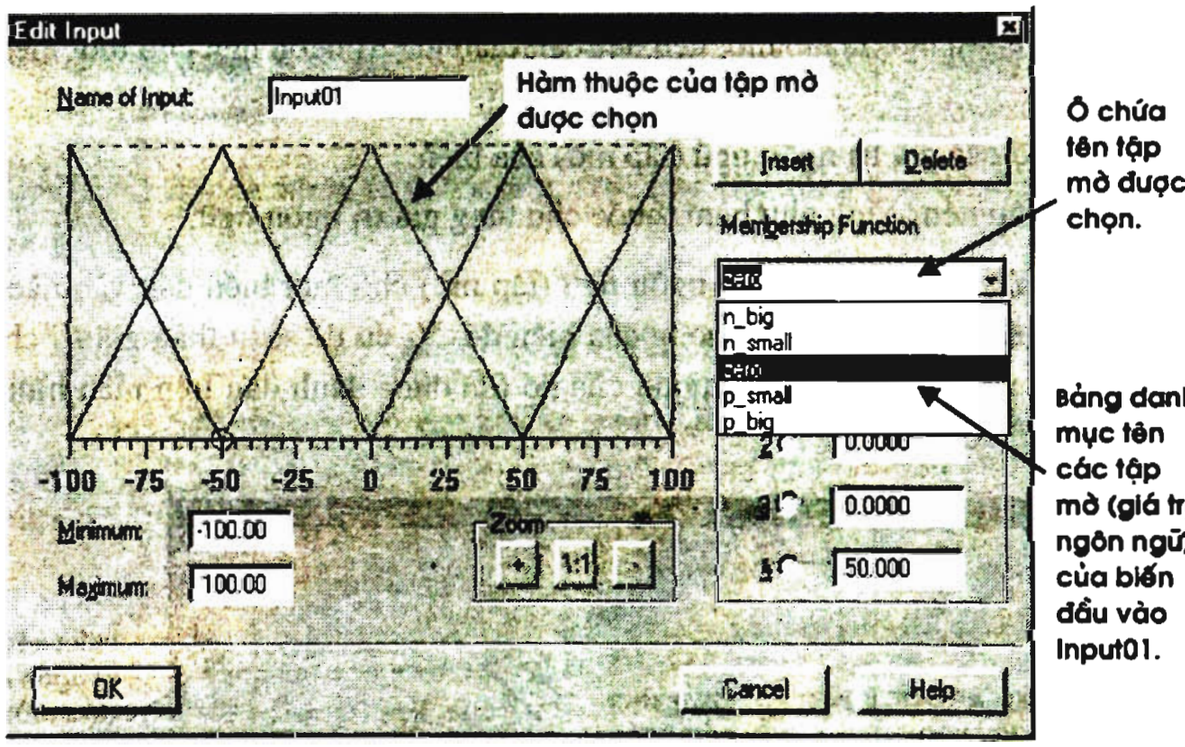
Khai báo số các giá trị ngôn ngữ (tập mờ): Để khai báo số các tập mờ cho biến Input01, ta chỉ cần kích chuột vào phím `Insert` rồi viết số các tập mờ cần có vào ô tương ứng trong cửa số hiện ra dạng (tối đa là 7):



Tiếp theo ta ấn phím OK. Số các tập mờ tối đa mà FCPA cho phép khai báo là 7. Các tập mờ được khai báo sẽ mặc định:

- có tên lần lượt là n-big, n-small, zero, p-small, p-big.
- có hàm thuộc hình tam giác được chia đều trên tập nền.

Sau khi ấn phím OK, FCPA sẽ in ra màn hình cửa sổ soạn thảo hàm thuộc cho mỗi tập mờ như sau:

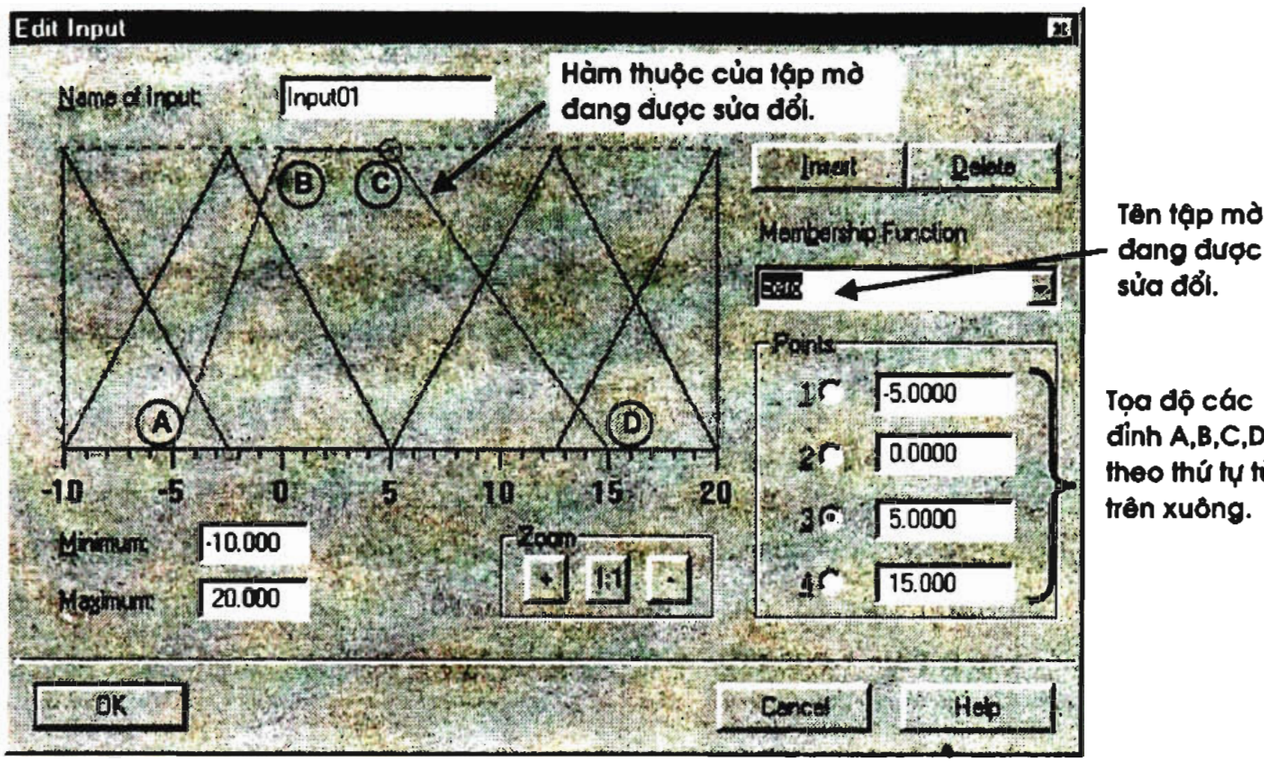


Sửa đổi hàm thuộc: Muốn sửa đổi hàm thuộc mặc định cho tập mờ nào, ta kích hoạt tập mờ đó bằng cách ghi trực tiếp tên tập mờ vào ô chứa tên tập mờ hoặc ấn phím ▼ và chọn tên tập mờ trong bảng danh mục hiện ra. Hàm thuộc của tập mờ được chọn sẽ chuyển sang mầu đỏ báo trạng thái tích cực của nó.

Việc sửa đổi hàm thuộc đồng nghĩa với việc đổi dạng (Singleton, tam giác hay hình thang) và miền xác định. Có hai cách sửa như sau:

1) *Cách thứ nhất*: Chọn đỉnh của hàm thuộc cần sửa bằng cách đưa chuột vào đỉnh đó và ấn phím chuột trái FCPA sẽ báo đỉnh đã được tích cực bằng một vành khuyên nhỏ quanh đỉnh đó, ví dụ như ở hình dưới, thì đỉnh C là đỉnh đã được tích cực. Giữ nguyên phím chuột rồi kéo đỉnh đó sang phải hoặc sang trái để thay đổi tọa độ của đỉnh.

2) *Cách thứ hai*: Sửa trực tiếp bằng cách ghi tọa độ mới vào các ô trong cửa sổ `Point`.



Như vậy muốn có hàm thuộc hình tam giác, ta cho đỉnh B trùng với đỉnh C (hai đỉnh có cùng tọa độ). Để có dạng singleton ta cho A trùng với D, B trùng với C.

Sau khi đã soạn thảo hay sửa đổi xong tất cả các giá trị của một biến vào, ta ấn phím OK để kết thúc. FCPA sẽ quay trở lại màn hình ban đầu.

5.3.5 Soạn thảo giá trị cho từng biến (ngôn ngữ) đầu ra

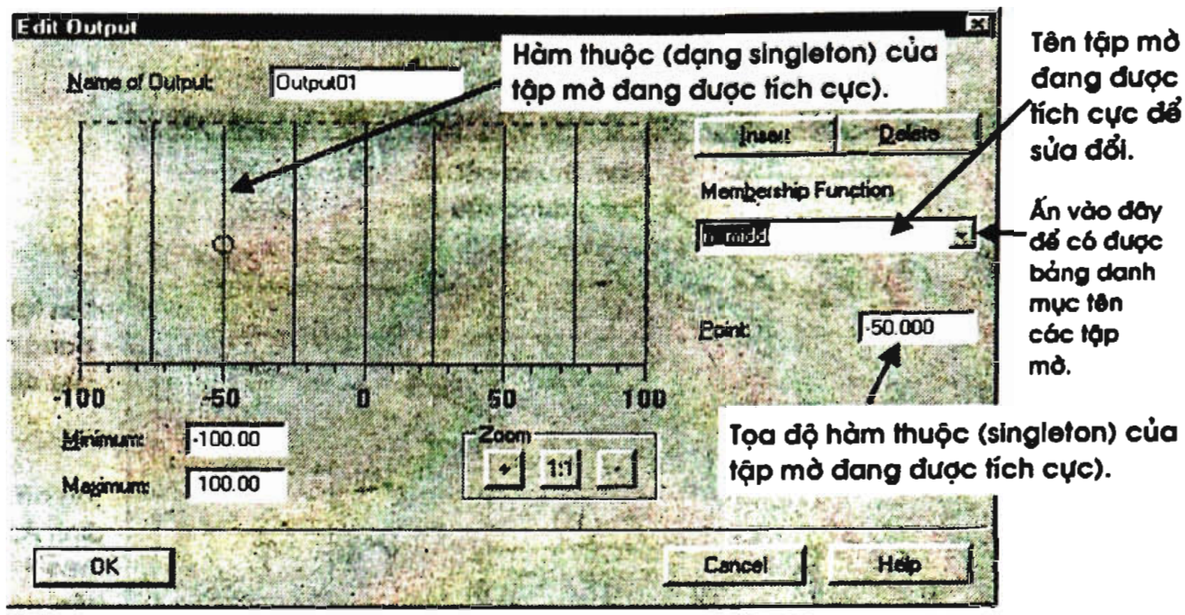
Tương tự như đã khai báo hay sửa đổi giá trị cho biến vào, việc khai báo các giá trị (tập mờ) cho biến ra cũng được bắt đầu bằng cách nháy kép phím chuột trái tại biểu tượng của biến đầu ra. Muốn soạn thảo hay sửa đổi giá trị ngôn ngữ (tập mờ) cho một biến đầu ra nào đó, ta nháy kép phím chuột trái tại biểu tượng của biến đó.

Ví dụ để soạn thảo giá trị cho biến ra Output01, ta nháy kép vào biểu tượng của nó. Khi đó cửa sổ soạn thảo sẽ hiện ra. Tiếp tục ta kích chuột vào phím `Insert` để khai báo số các tập mờ cho biến Output01.

Chú ý là FCPA chỉ cho phép khai báo tối đa 9 giá trị cho mỗi biến ra.

Sau khi khai báo xong số các giá trị (tập mờ) cho biến ra Output01 ta ấn phím OK để vào màn hình soạn thảo. Khác với biến ngôn ngữ đầu vào, giá trị (tập mờ) của các biến ra chỉ có duy nhất một dạng singleton.

Muốn sửa đổi giá trị ngôn ngữ nào, ta tích cực nó bằng cách chọn tên tập mờ của giá trị đó trong bảng danh mục hiện ra khi ấn phím ▼. FCPA sẽ báo trạng thái tích cực của hàm thuộc của tập mờ được chọn bằng cách chuyển sang nó mầu đỏ và thêm một hình khuyên ở chính giữa.



Để sửa đổi hàm thuộc dạng singleton, đơn giản ta chỉ cần sửa đổi tọa độ của nó bằng cách đưa con trỏ vào hình khuyên, giữ phím chuột trái rồi kéo sang phải/trái, hoặc trực tiếp ghi tọa độ mới vào ô **Point** của cửa sổ màn hình soạn thảo.

5.3.6 Soạn thảo luật hợp thành

Sau khi khai báo xong biến ngôn ngữ vào/ra và các giá trị (tập mờ) cho chúng, chẳng hạn như ta đã khai báo m biến vào Input1, ... , Input$m$ với các giá trị A_{i1} , ⋯ , A_{im} và s biến ra Output1=B_{n1} , ⋯ , Output$s$ với các giá trị B_{i1} , ⋯ , B_{is} , bước tiếp theo là ta xây dựng luật hợp thành. Để vào chế độ xây dựng luật hợp thành có cấu trúc

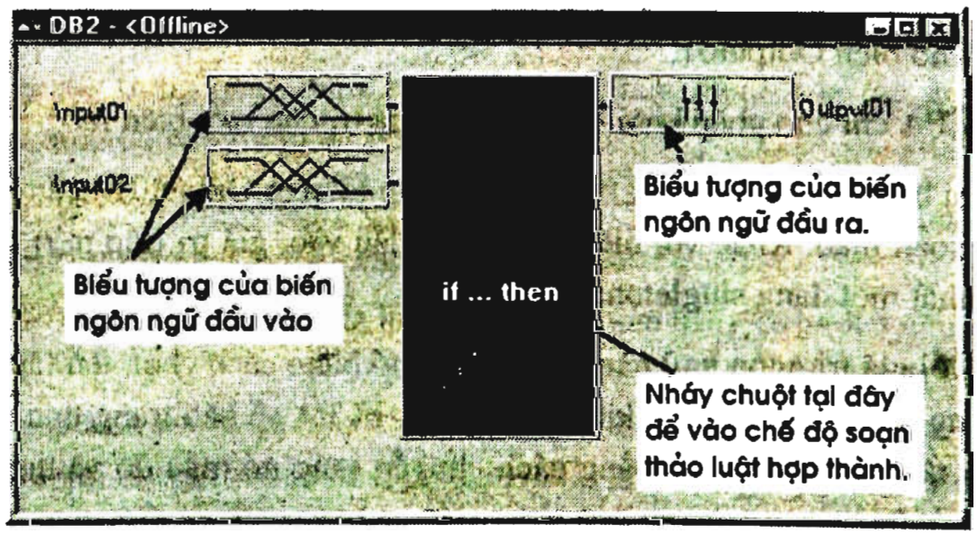
R_1: Nếu Input1=A_{11} và ⋯ và Input$m$=A_{1m} thì Output1=B_{11} và ⋯ và Output$s$=B_{1s}

R_2: Nếu Input1=A_{21} và ⋯ và Input$m$=A_{2m} thì Output1=B_{21} và ⋯ và Output$s$=B_{2s}

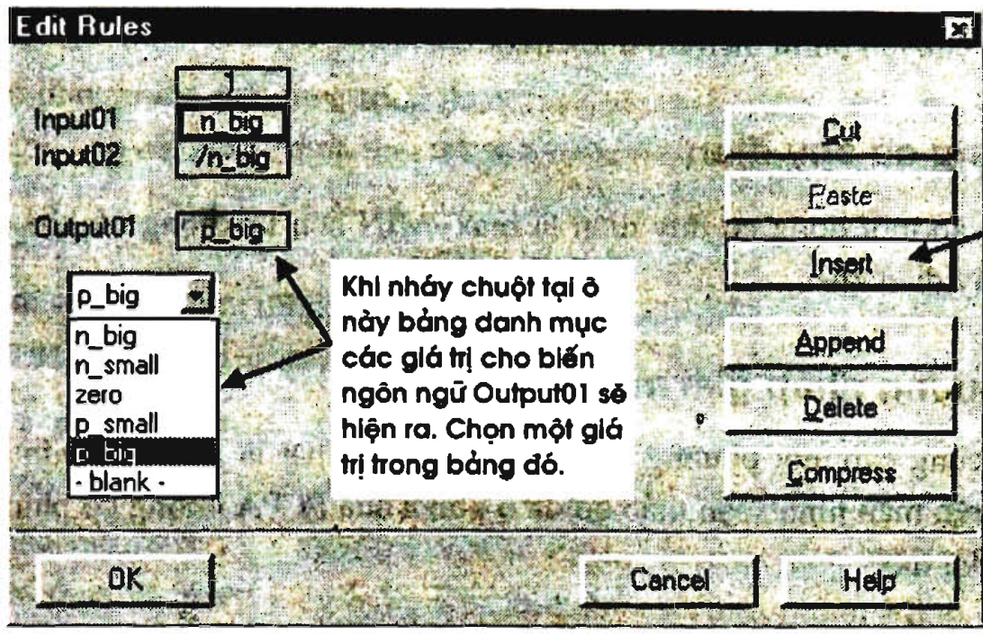
⋮

R_n: Nếu Input1=A_{n1} và ⋯ và Input$m$=A_{nm} thì Output1=B_{n1} và ⋯ và Output$s$=B_{ns}

ta nháy kép phím trái của chuột tại ô **if⋯then**:



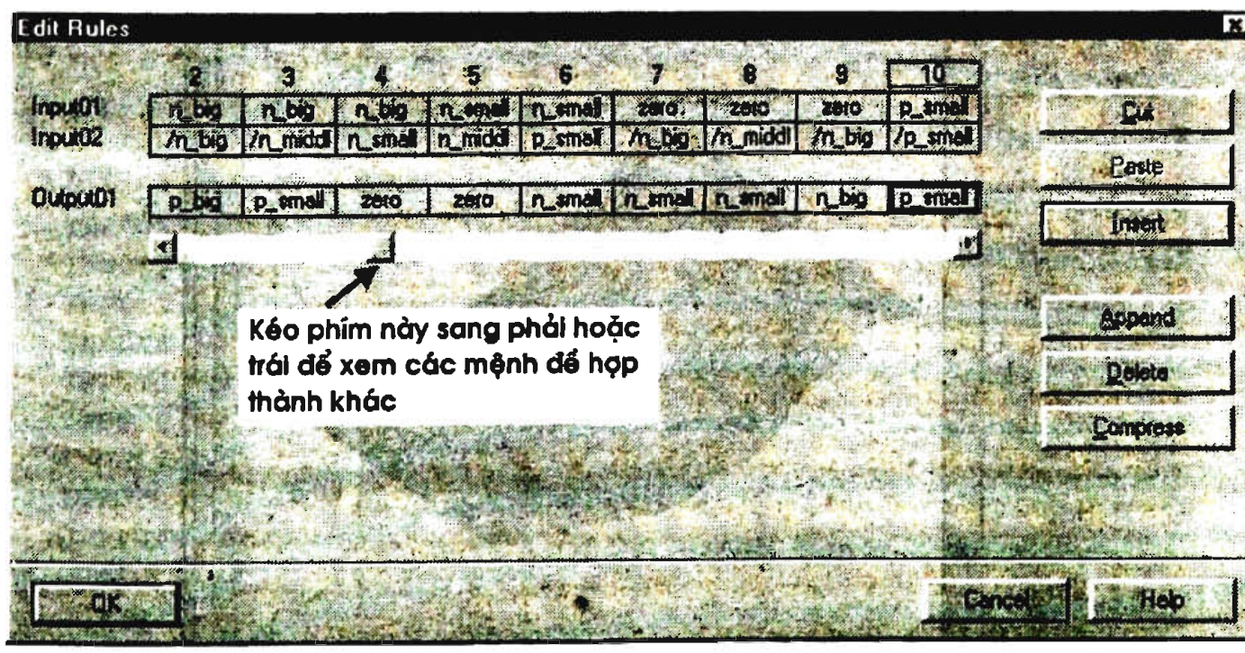
khi đó sẽ xuất hiện cửa sổ màn hình soạn thảo dạng



Để soạn thảo luật hợp thành ta soạn thảo từng mệnh đề hợp thành. Ấn phím **`Insert`** để chèn thêm một mệnh đề hợp thành vào trong luật. Mệnh đề hợp thành được chèn thêm sẽ là một cột gồm các ô trống. Số các ô trống này được quy định bởi số các biến ngôn ngữ vào/ra mà ta đã khai báo từ trước. Mỗi ô trống ứng với một biến ngôn ngữ. Tiếp theo, nếu ta nháy chuột tại ô trống của biến ngôn ngữ nào, trên màn hình sẽ hiện ra bảng các giá trị (tập mờ) của biến ngôn ngữ đó để ta chọn. Ví dụ ở màn hình soạn thảo phía trên, mệnh đề hợp thành thứ nhất mà ta vừa soạn thảo bằng cách chọn giá trị cho nó từ bảng các giá trị chính là:

Nếu Input01=n_big và Input02=n_big thì Output01=p_big

Sau khi khai báo xong đầy đủ các mệnh đề hợp thành, màn hình soạn thảo luật hợp thành sẽ có dạng như sau:



trong đó, do bị khống chế về kích thước cửa sổ màn hình, số các mệnh đề hợp thành nhiều nhất có thể được hiển thị là 9. Tuy nhiên ta có thể xem các mệnh đề hợp thành khác bằng cách dịch chuyển phím hiển thị nhờ chuột.

5.3.7 Chọn động cơ suy diễn

FCPA chỉ cung cấp một động cơ duy diễn là max–MIN nên ta không có khả năng chọn một động cơ suy diễn khác.

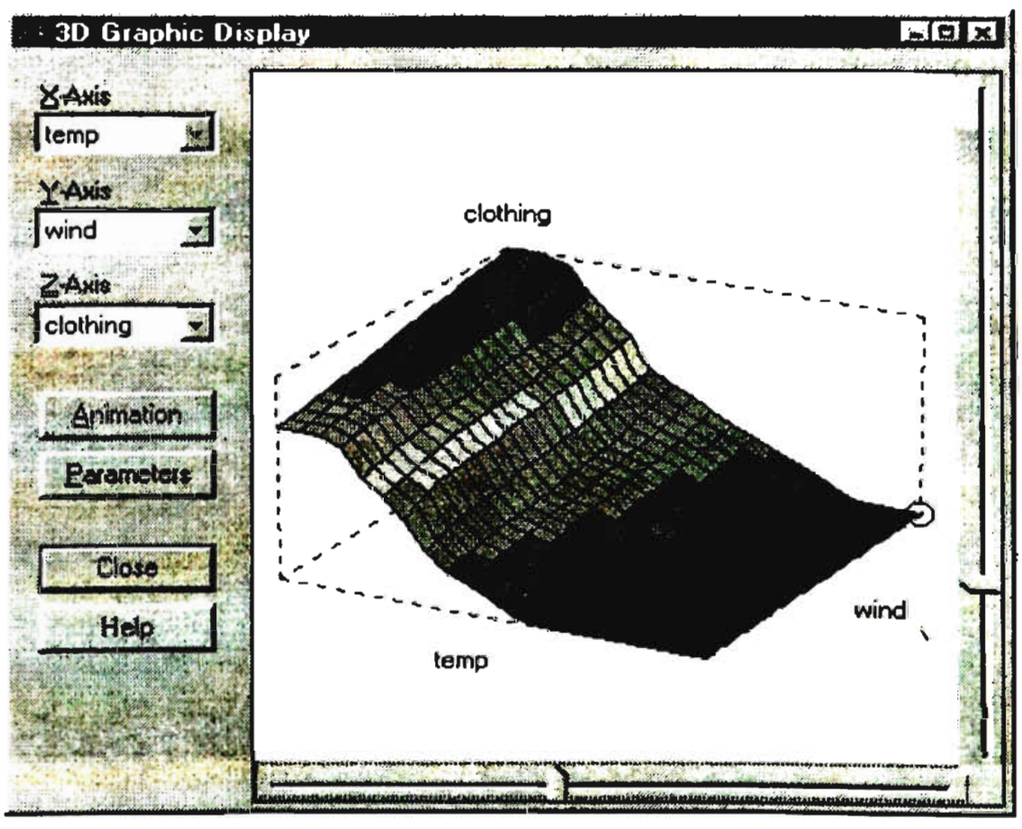
5.3.8 Chọn phương pháp giải mờ

FCPA cũng chỉ cung cấp một phương pháp giải mờ duy nhất là *phương pháp điểm trọng tâm*. Bởi vậy trên màn hình soạn thảo bộ điều khiển của FCPA không có phím lựa chọn phương pháp giải mò.

5.3.9 Quan sát quan hệ vào ra của bộ điều khiển mờ

Cùng với việc khai báo xong luật hợp thành ta đã kết thúc quá trình soạn thảo một bộ điều khiển mờ. Ấn phím OK để kết thúc quá trình soạn thảo và quay trở về cửa sổ màn hình chính của FCPA. Ví dụ sau khi khai báo một bộ điều khiển mờ có 3 đầu vào, 2 đầu ra với luật hợp thành gồm 13 mệnh đề hợp thành và ấn phím OK, FCPA sẽ quay về màn hình chính.

Để quan sát một cách trực quan quan hệ vào ra của bộ điều khiển mờ vừa soạn thảo ta chọn **Debug** → **3D Graphic Display**, khi đó trên màn hình xuất hiện đồ thị mô tả quan hệ vào/ra của bộ điều khiển mờ như sau:



5.4 Sử dụng DB mờ với FB30 (Fuzzy control)

5.4.1 Các tham biến hình thức của FB30

Bộ điều khiển mờ được soạn thảo xong cần phải được cất giữ vào Project bằng lệnh **File → Save**. Nó sẽ được lưu trữ vào Project dưới dạng một khối DB mà ta đã đặt tên. Khối dữ liệu mờ này được sử dụng cùng với khối hàm FB30 đã được lấy từ Project **FuzConEx** trong thư viện của Simatic Manager khi cài đặt chương trình **Fuzzy/FB**. Bởi vậy khi sử dụng khối dữ liệu mờ ta phải kết thúc FCPA bằng lệnh **File → Exit** và quay trở lại **Simatic Manager** để viết lệnh sử dụng theo cấu trúc:

Cú pháp CALL FB30,DBx

trong đó **DBx** là tên khối dữ liệu mờ. Khối FB30 (tên hình thức **Fuzzy Control**) có 8 biến đầu vào **INPUT1 ÷ INPUT8** kiểu số thực, 5 biến ra gồm **OUTPUT1 ÷ OUTPUT4** cũng kiểu số thực và **INFO** kiểu byte. Khi thực hiện lệnh gọi khối FB30 như trên, toàn bộ 8 biến hình thức đầu vào và 5 biến đầu ra sẽ hiện trên màn hình chờ ta truyền tham trị:

```
CALL  FB    30 , DBx
 INPUT1 :=
 INPUT2 :=
 INPUT3 :=
 INPUT4 :=
 INPUT5 :=
 INPUT6 :=
 INPUT7 :=
 INPUT8 :=
```

Chỉ gán tham trị cho những biến ngôn ngữ đầu vào nào đã được khai báo trong DBx nhờ phần mềm FCPA.

```
 OUTPUT1:=
 OUTPUT2:=
 OUTPUT3:=
 OUTPUT4:=
```

Chỉ gán tham trị cho những biến ngôn ngữ đầu ra nào đã được khai báo trong DBx nhờ phầm mềm FCPA.

```
 INFO   :=        // Thanh ghi báo trạng thái của FB30
```

Ví dụ, xét lại bài toán điều khiển cầu trục đã được đề cập tới ở mục 5.1.1. Gọi tên khối dữ liệu mờ với hai biến vào α , $\dot{\alpha}$, một biến ra v và luật hợp thành như đã mô tả được soạn thảo bằng FCPA là **DB2** thì khi sử dụng ta dùng lệnh

```
CALL  FB    30 , DB2
 INPUT1 :=MD0          // Giá trị tín hiệu đo góc
 INPUT2 :=MD4          // Giá trị tín hiệu đo tốc độ góc
 INPUT3 :=             // Không sử dụng
 INPUT4 :=             //      -
 INPUT5 :=             //      -
 INPUT6 :=             //      -
 INPUT7 :=             //      -
 INPUT8 :=             //      -
 OUTPUT1:=MD8          // Giá trị tín hiệu điều khiển động cơ
 OUTPUT2:=             // Không sử dụng
 OUTPUT3:=             //      -
 OUTPUT4:=             //      -
 INFO   :=             // Thanh ghi báo trạng thái
```

nếu như trước đó giá trị tín hiệu đo góc α đã được ghi vào ô nhớ MD0, giá trị đo tốc độ thay đổi góc $\dot{\alpha}$ được ghi vào MD4. Tín hiệu điều khiển động cơ sẽ được FB30 chuyển vào ô nhớ MD8.

5.4.2 Thanh ghi báo trạng thái làm việc của FB30

Giá trị trả về có tên **INFO** với kích thước một byte là mã báo trạng thái thực hiện công việc của khối hàm FB30. Nó được quy định như sau:

B#16#00	Khối hàm FB30 đã được thực hiện bình thường.
B#16#01	Khối hàm FB30 không được thực hiện. Giá trị trả về ở đầu ra vẫn là những giá trị cũ.
B#16#11	Không tìm thấy khối DB mờ đã chỉ thị. Có thể khối DB mờ này đã không được đổ vào CPU.
B#16#21	Khối dữ liệu DB mờ được gọi theo hàm FB30 không cùng kích thước về biến vào ra. Chẳng hạn như khối DB mờ đã được soạn thảo cho 4 biến vào và 2 biến ra, nhưng khi gọi cùng với FB30 lại khai báo 5 biến vào và 2 biến ra.

Liên quan tới mã B#16#01 báo FB30 không làm việc là nội dung từ kép có tên **START_STOP** trong DB mờ đã được soạn thảo bằng FCPA. Từ kép này có tác dụng như một biến điều kiện để thực hiện lệnh **CALL FB30,DBx**:

- Nếu **START_STOP = W#16#0000** lệnh sẽ được thực hiện.
- Ngược lại khi **START_STOP ≠ W#16#0000** thì lệnh không được thực hiện.

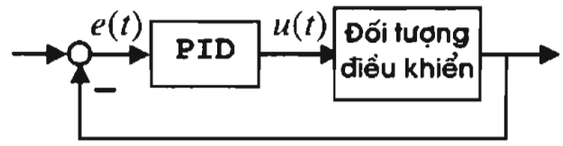
6 MODULE MỀM PID

Nhiều năm trước đây, bộ điều khiển PID được coi là bộ điều khiển lý tưởng đối với các đối tượng có mô hình liên tục. Bộ PID thực sự là bộ điều khiển động mà việc thay đổi các tham số của bộ điều khiển có khả năng làm thay đổi đặc tính động và tĩnh của hệ thống điều khiển tự động.

Bộ điều khiển PID thực chất là thiết bị điều khiển thực hiện luật điều khiển được mô tả bằng phương trình sau:

$$u(t)=k_p e(t)+\frac{1}{T_I}\int_0^t e(\tau)d\tau+T_D\dot{e}(t) \qquad (6.1)$$

trong đó $e(t)$ là tín hiệu vào, $u(t)$ là tín hiệu ra của bộ điều khiển, k_p là hệ số khuếch đại của luật điều khiển tỷ lệ, T_I hằng số thời gian tích phân và T_D là hằng số thời gian vi phân.



Hình 6.1. Điều khiển với bộ điều khiển **PID**.

Đối với hệ thống có độ dự trữ ổn định lớn, nếu muốn tăng độ chính xác điều khiển ta chỉ cần tăng hệ số khuếch đại của luật điều khiển tỷ lệ [7].

Hệ thống sẽ không có sai lệch tĩnh khi tín hiệu vào là hàm bậc thang đơn vị và hằng số thời gian tích phân T_I được chọn khác không. Luật điều khiển tích phân còn gọi là điều khiển chậm sau vì sai số điều khiển được tích lũy cho đến khi đủ lớn thì quyết định điều khiển mới được đưa ra.

Tăng khả năng tác động nhanh của hệ, giảm bớt thời gian quá điều chỉnh bằng cách thay đổi hằng số thời gian của luật điều khiển vi phân. Luật điều khiển vi phân còn được gọi là điều khiển vượt trước.

Luật điều khiển trong phương trình (6.1) thường còn được biểu diễn dưới dạng hàm truyền đạt như sau

$$W_{DK}(p)=k_p\left(1+\frac{1}{T_I p}+T_D p\right)=\frac{k_i}{p}(1+T_{D1}p)(1+T_{D2}p) \qquad (6.2)$$

Từ năm 1975 trở lại đây, do sự phát triển không ngừng của kỹ thuật điện tử và kỹ thuật vi xử lý, các PID số ngày càng được sử dụng rộng rãi trong công nghiệp. PID số được mô tả qua phương trình vi sai phân sau:

$$u_k = u_{k-1} + r_0 e_k + r_1 e_{k-1} + r_2 e_{k-2} \tag{6.3}$$

hoặc bằng hàm truyền đạt gián đoạn:

$$W^*(z) = \frac{r_0 + r_1 z^{-1} + r_2 z^{-2}}{1 + z^{-1}} \tag{6.4}$$

Với bộ điều khiển PID, người sử dụng dễ dàng tích hợp các luật điều khiển khác như luật điều khiển tỷ lệ (luật P), điều khiển tỷ lệ–tích phân (luật PI), luật điều khiển tỷ lệ–vi phân (luật PD). Bộ điều khiển PID luôn là một phần tử không thể thay thế được trong các quá trình tự động khống chế nhiệt độ, mức, tốc độ .... Ngay cả khi lý thuyết điều khiển tự động hiện đại được ứng dụng vào việc thiết kế, các bộ điều khiển như bộ điều khiển mờ, bộ điều khiển nơ ron, bộ điều khiển bền vững thì việc kết hợp giữa các phương pháp điều khiển hiện đại và bộ điều khiển PID kinh điển vẫn đem lại những hiệu quả bất ngờ mà không bộ điều khiển nào có khả năng đem lại.

Một trong những ứng dụng của bộ điều khiển PID trong điều khiển thích nghi và điều khiển mờ là thường xuyên phải chỉnh định lại các tham số của nó cho phù hợp với sự thay đổi không biết trước của đối tượng cũng như của môi trường nhằm đảm bảo được các chỉ tiêu chất lượng đã đề ra cho hệ thống. Nếu như ta đã tự động hóa được công việc thay đổi tham số này thì bộ điều khiển PID đó sẽ là một bộ điều khiển bền vững với mọi tác động của nhiễu nội cũng như nhiễu ngoại lên hệ thống [3].

Cũng chính vì vậy mà các thiết bị điều khiển quá trình như DCS *Disbuted Control system*, PLC *Programmable Logic Control*, PCS *Process Control System* của các hãng sản xuất thiết bị tự động trên thế giới không thể thiếu được module điều khiển PID hoặc cứng hoặc mềm.

Để sử dụng tốt các module này, người thiết kế phải nắm được các phương pháp chọn luật điều khiển và các tham số cho bộ điều khiển.

6.1 Xác định tham số cho bộ điều khiển PID

Luật điều khiển thường được chọn trên cơ sở đã xác định được mô hình toán học của đối tượng và phải phù hợp với đối tượng cũng như thỏa mãn các yêu cầu của bài toán thiết kế [7], [15].

Trong trường hợp mô hình toán học của đối tượng không xác định được có thể chọn luật điều khiển và các tham số của bộ điều khiển theo phương pháp thực nghiệm. Tuy nhiên để tiến hành được phương pháp thực nghiệm, hệ thống phải đảm bảo thỏa mãn thêm một số điều kiện.

6.1.1 Phương pháp Reinisch

Phương pháp thiết kế thuật điều khiển của Reinisch dựa trên cơ sở mô hình toán học của đối tượng đã xác định một cách tường minh. Mô hình động học của đối tượng được đưa về hai dạng cơ bản sau:

1) Dạng khâu nguyên hàm với mô hình đặc trưng:

$$W(p) = k_{dt}\frac{(1+bp)e^{-T_t p}}{\prod_{i=1}^{n}(1+pT_i)} = k_{dt}\frac{(1+bp)e^{-T_t p}}{1+a_1 p+\cdots+a_n p^n} \tag{6.5}$$

với T_i là các số thực thỏa mãn $T_1 \geq T_2 \geq \cdots \geq T_n \geq 0$ và hằng số thời gian trễ T_t là một số thực hữu hạn không âm. Không mất tính tổng quát nếu ta giả thiết T_1 hằng số thời gian lớn nhất và T_2 là hằng số thời gian lớn thứ hai.

Nếu $0 \leq b \leq T_3$ thì bộ điều khiển thích hợp sẽ là P hoặc PI. Trong trường hợp $0 \leq b \leq T_4$ người ta lại thường hay chọn bộ điều khiển PD hoặc PID.

2) Dạng khâu động học có thành phần tích phân

$$W(p) = k_{idt}\frac{(1+bp)e^{-T_t p}}{p\prod_{i=1}^{n}(1+pT_i)} = k_{idt}\frac{(1+bp)e^{-T_t p}}{p(1+a_1 p+\cdots+a_n p^n)} \tag{6.6}$$

với những điều kiện hạn chế giống như của (6.5).

Để thuận lợi cho việc thiết kế hệ thống với luật điều khiển I cho đối tượng dạng 1 và không có luật điều khiển I cho đối tượng dạng 2, Reinisch đã đề nghị đưa hàm truyền phải có của hệ hở về dạng gần đúng sau:

$$W_0(p) = \frac{1}{pT(1+c_1 p+c_2 p^2)} \tag{6.7}$$

với hai trường hợp phân biệt $c_2 = 0$ hoặc $c_2 \neq 0$. Tham số T được tính bởi:

$$\frac{1}{T} = \begin{cases} k_{dt}k_i \text{ cho đối tượng dạng 1} \\ k_{idt} \text{ cho đối tượng dạng 2} \end{cases} \tag{6.8}$$

và c_1 được xác định từ các tham số của đối tượng như sau:

$$c_1 = \sum_{i=1}^{n} T_i - b + T_t = a_1 - b + T_t . \tag{6.9}$$

Tham số k_i của bộ điều khiển PID sẽ được xác định từ T theo (6.8). Các tham số T_{D1}, T_{D2} còn lại thì được tính đơn giản là $T_{D1} = T_1$ và $T_{D2} = T_2$.

Điều khiển đối tượng dạng 1

Để chọn T cho đối tượng dạng 1 ta đi từ độ quá điều chỉnh cực đại mong muốn σ_{max} thông qua hệ số chỉnh định $\alpha = f(\sigma_{max})$ theo công thức:

$$T = c_1\alpha \qquad \Rightarrow \qquad k_i = \frac{1}{k_{dt}c_1\alpha} \tag{6.10}$$

1) Cho trường hợp (6.7) có $c_2 = 0$, hệ số chỉnh định α được tính theo

$$\alpha = \frac{4\ln^2\sigma_{max}}{\pi^2 + \ln^2\sigma_{max}} \tag{6.11}$$

2) Cho trường hợp (6.7) có $c_2 \neq 0$ thì

$\alpha = a + c\gamma$

với a và c xác định từ σ_{max} theo bảng bên, hằng số γ có thể được xác định theo các cách:

$\sigma_{max}(\%)$	0	5	10	15	20	30	40	50	60
a	0	1,9	1,4	1,1	0,83	0,51	0,31	0,18	0,11
c	0	0	1	1	1,4	1,4	1,4	1,4	1,4

a) $\gamma = \frac{c_2}{c_1^2}$ nếu bộ điều khiển được sử dụng là I. (6.12)

b) $\gamma = \frac{c_2'}{c_1'^2}$ nếu bộ điều khiển được sử dụng là P hoặc PI. (6.13)

c) $\gamma = \frac{c_2''}{c_1''^2}$ nếu bộ điều khiển được sử dụng là PD hoặc PID. (6.14)

trong đó

$$c_1 = a_1 - b + T_t \quad , \quad c_1' = c_1 - T_1 \quad , \quad c_1'' = c_1 - T_1 - T_2 \tag{6.15}$$

$$c_2 = a_2 + (T_1 - b)(a_1 - b) + \frac{T_t^2}{2}, \quad c_2' = c_2 - T_1c_1' \quad , \quad c_2'' = c_2 - T_1c_1' - T_2c_1'' \tag{6.16}$$

Ví dụ 1: Cho một đối tượng thuộc dạng 1 (theo phương trình (6.5)) với mô hình

$$W(p) = \frac{1}{1 + 14p + 40p^2}e^{-6p} = \frac{1}{(1+10p)(1+4p)}e^{-6p}.$$

Hãy thiết kế luật điều khiển và chọn tham số sao cho độ quá điều chỉnh σ_{max} không vượt quá 10%.

Để điều khiển đối tượng trên ta có thể sử dụng các bộ điều khiển I, P hoặc PI.

Theo như bảng trên thì yêu cầu $\sigma_{max} \leq 10\%$ dẫn đến a=1,4 và c=1. Hơn nữa đối tượng có các tham số k_{dt}=1, b=0, a_1=14s, a_2=40s^2, T_1=10s, T_2=4s và T=6. Bởi vậy theo (6.15) và (6.16) thì c_1=20s, c_1'=10s, c_1''=6s, c_2=198s^2, c_2'=98s^2, c_2''=74s^2. Suy ra γ có thể có những giá trị sau:

- $\gamma = \frac{c_2}{c_1^2}$ =0,495 nếu bộ điều khiển được sử dụng là I.
- $\gamma = \frac{c_2'}{c_1'^2}$ =0,98 nếu bộ điều khiển được sử dụng là P hoặc PI.

Giả sử rằng ta sử dụng bộ điều khiển I. Vậy thì do $\alpha = a + c\gamma$=1,9 nên từ (6.10) có

k_i=0,03 và $T_{D1} = T_{D2}$= 0.

Nếu bộ điều khiển mà ta sử dụng lại là PI thì các tham số cần xác định của bộ điều khiển là k_i và T_{D1}. Từ $\alpha = a + c\gamma$=2,38 ta suy ra được

k_i=0,05 và T_{D1}=10s , T_{D2}=0.

Điều khiển đối tượng dạng 2

Ưu điểm của phương pháp Reinisch là ngay cả trong trường hợp đối tượng có thành phần tích phân (dạng 2), các giá trị cần thiết cho công việc tính toán tham số bộ điều khiển như c_1, c_1', c_1'', c_2, c_2', c_2'' cũng được tính giống như cho đối tượng dạng 1.

Đối với vấn đề điều khiển đối tượng dạng 2, Reinisch đề xuất sử dụng bộ điều khiển P hoặc PD (không có I) và do đó theo công thức hàm truyền đạt (6.2) của bộ điều khiển thì chỉ còn hai tham số k_p và T_D là phải xác định.

Với những giá trị trung gian c_1, c_1', c_1'', c_2, c_2', c_2'', tính theo (6.12)÷(6.16), ta có γ :

a) $\gamma = \frac{c_2}{c_1^2}$ nếu bộ điều khiển được sử dụng là P (6.17)

b) $\gamma = \frac{c_2'}{c_1'^2}$ nếu bộ điều khiển được sử dụng là PD (6.18)

Từ đó suy ra:

1) $k_p = \frac{1}{k_{idt}\, c_1 \alpha}$ cho bộ điều khiển P.

2) $k_p = \frac{1}{k_{idt}\, c_1'' \alpha}$ và $T_D = T_1$ cho bộ điều khiển PD.

trong đó $\alpha = a + c\gamma$ và a, c được tính từ độ quá điều chỉnh cực đại mong muốn σ_{max} theo bảng đã cho ở trang 210.

Ví dụ 2: Tìm bộ điều khiển cho đối tượng thuộc dạng 2 với mô hình

$$W(p)=\frac{1}{p(1+14p+40p^2)}e^{-6p}$$

để $\sigma_{max} \le 10\%$. Giống như ở ví dụ 1, các giá trị trung gian là $c_1=20s$, $c_1'=10s$, $c_1''=6s$, $c_2=198s^2$, $c_2'=98s^2$, $c_2''=74s^2$. Bởi vậy nếu chọn bộ điều khiển PD thì

$$\gamma=\frac{c_2''}{c_1''^2}=2{,}05 \quad \text{hay} \quad \alpha=3{,}45$$

và do đó

$$k_p=0{,}05\ ,\ T_D=10.$$

Cuối cùng, ta tóm tắt lại phương pháp Reinisch dưới dạng bảng như sau để tiện cho việc tra cứu và sử dụng sau này.

<table>
<tr><td>Mô hình đối tượng</td><td colspan="5">$W(p)=k_{dt}\dfrac{(1+bp)e^{-T_t p}}{1+a_1p+\cdots+a_np^n}$</td><td colspan="2">$W(p)=k_{idt}\dfrac{(1+bp)e^{-T_t p}}{p(1+a_1p+\cdots+a_np^n)}$</td></tr>
<tr><td>Các giá trị trung gian c_i</td><td colspan="7">$c_1=a_1-b+T_t$, $c_1'=c_1-T_1$, $c_1''=c_1-T_1-T_2$
$c_2=a_2+(T_1-b)(a_1-b)+\dfrac{T_t^2}{2}$, $c_2'=c_2-T_1c_1'$, $c_2''=c_2-T_1c_1'-T_2c_1''$</td></tr>
<tr><td>Bộ điều khiển</td><td>Luật P</td><td>Luật I</td><td>Luật PI</td><td>Luật PD</td><td>Luật PID</td><td>Luật P</td><td>Luật PD</td></tr>
<tr><td></td><td>k_p</td><td>$\dfrac{k_I}{p}$</td><td>$\dfrac{k_I(1+T_Dp)}{p}$</td><td>$k_p(1+T_Dp)$</td><td>xem (6.2)</td><td>k_p</td><td>$k_p(1+T_Dp)$</td></tr>
<tr><td>k_p</td><td>$\dfrac{T_1}{k_{dt}c_1''\alpha}$</td><td></td><td></td><td>$\dfrac{T_1}{k_{dt}c_1''\alpha}$</td><td></td><td>$\dfrac{T_1}{k_{idt}c_1\alpha}$</td><td>$\dfrac{T_1}{k_{idt}c_1''\alpha}$</td></tr>
<tr><td>k_I</td><td></td><td>$\dfrac{T_1}{k_{dt}c_1\alpha}$</td><td>$\dfrac{T_1}{k_{dt}c_1'\alpha}$</td><td></td><td>$\dfrac{T_1}{k_{dt}c_1''\alpha}$</td><td></td><td></td></tr>
<tr><td>T_D</td><td></td><td></td><td>T_1</td><td>T_2</td><td>$T_{D1}=T_1$
$T_{D2}=T_2$</td><td></td><td>T_1</td></tr>
<tr><td>α</td><td colspan="7">$a+c\gamma$
$\sigma_{max}(\%)$ | 0 | 5 | 10 | 15 | 20 | 30 | 40 | 50 | 60
a | 0 | 1,9 | 1,4 | 1,1 | 0,83 | 0,51 | 0,31 | 0,18 | 0,11
c | 0 | 0 | 1 | 1 | 1,4 | 1,4 | 1,4 | 1,4 | 1,4</td></tr>
<tr><td>γ</td><td>$\dfrac{c_2'}{c_1'^2}$</td><td>$\dfrac{c_2}{c_1^2}$</td><td>$\dfrac{c_2'}{c_1'^2}$</td><td>$\dfrac{c_2''}{c_1''^2}$</td><td>$\dfrac{c_2''}{c_1''^2}$</td><td>$\dfrac{c_2}{c_1^2}$</td><td>$\dfrac{c_2'}{c_1'^2}$</td></tr>
<tr><td>Những hạn chế</td><td>$\dfrac{T}{c_1'\alpha}>10$
$b<T_3$</td><td>$b<T_2$</td><td>$b<T_3$</td><td>$\dfrac{T}{c_1'\alpha}>10$
$b<T_4$</td><td>$b<T_4$</td><td>$b<T_2$</td><td>$b<T_3$</td></tr>
</table>

6.1.2 Phương pháp thực nghiệm

Trong trường hợp không thể xây dựng mô hình cho đối tượng thì phương pháp thiết kế thích hợp là phương pháp thực nghiệm. Thực nghiệm chỉ có thể tiến hành nếu hệ thống đảm bảo điều kiện: *khi đưa trạng thái làm việc của hệ đến biên giới ổn định thì mọi giá trị của các tín hiệu trong hệ thống đều phải nằm trong giới hạn cho phép.*

Phương pháp Ziegler và Nichols

Trước khi tiến hành thực nghiệm hệ thống phải được lắp đặt theo sơ đồ ở hình 6.1, bao gồm đối tượng và bộ điều khiển theo luật PID. Sau khi lắp đặt xong, thực nghiệm được tiến hành theo các bước sau:

1) Cho hệ thống làm việc ở biên giới ổn định
   - Điều khiển đối tượng theo luật P, tức là cho $T_D \to 0$ và $T_I \to \infty$.
   - Tăng hệ số khuếch đại k_p của luật điều khiển P cho đến khi hệ thống ở biên giới ổn định. Xác định hệ số k_{pth} và chu kỳ giao động tới hạn T_{th}.
2) Chọn luật điều khiển và tính toán tham số từ k_{pth}, T_{th} theo bảng sau

Luật điều khiển	$\frac{k_p}{k_{pth}}$	$\frac{T_p}{T_{th}}$	$\frac{T_D}{T_{th}}$
Luật P	0.5		
Luật PI	0.45	0.8	
Luật PID	0.6	0.5	0.12

Trong nhiều trường hợp, việc xác định chu kỳ giao động riêng gặp khó khăn và không đảm bảo độ chính xác thì phương pháp giới thiệu sau đây sẽ khắc phục nhược điểm đó.

Phương pháp Jassen và Offerein

Thực nghiệm theo phương pháp này được tiến hành theo các bước sau đây:

1) Cho hệ thống làm việc ở biên giới ổn định
   - Điều khiển đối tượng theo luật P ($T_D \to 0$ và $T_I \to \infty$)
   - Xác định hệ số k_{pth}.
2) Chọn tham số cho luật PI
   - Cho hệ làm việc với luật PI và với hệ số $k_p = 0{,}45k_{pth}$, T_I tuỳ chọn
   - Giảm hằng số thời gian tích phân T_I cho đến khi hệ thống làm việc ở biên giới biên giới ổn định. Xác định hằng số thời gian tích phân T_{Ith} ở chế độ này.
   - Chọn $T_I = 3\,T_{Ith}$.

3) Chọn luật điều khiển PID

- Cho hệ thống làm việc theo luật PID với $k_p = k_{pth} - \xi$ (ξ đủ nhỏ), T_D và T_I tuỳ chọn.
- Tăng hằng số thời gian vi phân cho đến khi hệ thống đạt được độ quá điều chỉnh cực đại lớn nhất $\sigma_{max} \rightarrow$ max. Xác định T_{DMAX}.
- Chọn $T_D = \frac{1}{3} T_{DMAX}$ và $T_I = 4{,}5\, T_D$.
- Giảm k_p cho đến khi hệ thống đạt được đặc tính động học mong muốn.

6.2 Module mềm PID

6.2.1 Những module PID mềm có trong Step7

Phần mềm Step7 cung cấp các module mềm PID để điều khiển các đối tượng có mô hình liên tục như lò, động cơ, mức .... Đầu ra của đối tượng được đưa vào đầu vào của bộ điều khiển qua các cổng vào tương tự của các module vào tương tự của Simatic S7–300/400. Tín hiệu ra của bộ điều khiển có nhiều dạng và được đưa đến các cơ cấu chấp hành qua những module vào ra khác nhau như:

- qua cổng ra tương tự của module ra tương tự (AO), hoặc
- qua các cổng ra số của module ra số (DO), hoặc
- qua các cổng phát xung ra tốc độ cao.

Phụ thuộc vào cơ cấu chấp hành, người sử dụng có thể chọn được module mềm PID tương thích. Ba module PID được tích hợp trong phần mềm Step7 phù hợp với ba kiểu cơ cấu chấp hành nêu trên, đó là:

1) Điều khiển liên tục với module mềm FB41 (tên hình thức **CONT_C**)
2) Điều khiển bước với module mềm FB42 (tên hình thức **CONT_S**)
3) Điều khiển kiểu phát xung với khối hàm hỗ trợ FB43 (tên hình thức **PULSEGEN**)

Mỗi module mềm PID đều có một khối dữ liệu riêng (DB) để lưu giữ các dữ liệu phục vụ cho chu trình tính toán thực hiện luật điều khiển. Các khối hàm FB của module mềm PID đều cập nhật được những khối dữ liệu này ở mọi thời điểm.

Module mềm FB **PULSEGEN** được sử dụng kết hợp với module mềm FB **CONT_C** nhằm tạo ra bộ điều khiển có tín hiệu ra dạng xung tốc độ cao thích ứng với những cơ cấu chấp hành kiểu tỷ lệ.

Một bộ điều khiển PID mềm được hoàn thiện thông qua các khối hàm FB nhiều chức năng tạo ra tính linh hoạt cao trong thiết kế . Người sử dụng có thể chọn chức năng này hoặc loại bỏ các chức năng không cần cho một hệ thống. Các chức năng cơ bản

khác như xử lý tín hiệu chủ đạo, tín hiệu quá trình và tính toán các biến khác cùng với bộ điều khiển theo thuật điều khiển PID cũng được tích hợp sẵn trong một module điều khiển mềm.

Một điều cần chú ý là những module PID mềm không toàn năng tới mức có thể ứng dụng được vào mọi bài toán điều khiển. Đặc tính điều khiển và tốc độ xử lý của module PID mềm phụ thuộc vào loại CPU được chọn để giải quyết bài toán điều khiển. Do khi xử lý một mạch vòng điều khiển người ta phải thực hiện công việc trích mẫu tín hiệu đầu vào cho mạch vòng điều khiển đó (liên quan đến tín hiệu báo ngắt theo chu kỳ thời gian OB30÷OB38), nên cần phải có sự tương thích giữa số mạch vòng điều khiển PID và khả năng cũng như tốc độ tính toán của CPU. Nếu bài toán điều khiển yêu cầu tần suất cập nhật càng cao thì số vòng điều khiển phải càng giảm. Chỉ ở những bài toán có số vòng điều khiển ít người ta mới có thể sử dụng các bộ module PID mềm có tần suất truy nhập cao.

Tất cả các modul;e PID mềm đều cung cấp nhiều giải pháp lựa chọn luật điều khiển trong khi thiết kế để bộ điều khiển phù hợp được với đối tượng như: luật điều khiển tỷ lệ (luật P), luật điều khiển tỷ lệ–vi phân (luật PD), luật điều khiển tỷ lệ–tích phân (luật PI) .... Chất lượng của hệ thống hoàn toàn phụ thuộc vào các tham số của bộ điều khiển. Do đó, điều kiện bắt buộc để đảm bảo thành công trong thiết kế là người sử dụng phải có mô hình đối tượng chính xác. Đó cũng chính là nhược điểm cơ ban của các phương pháp điều khiển kinh điển.

Các đại lượng vật lý của đối tượng và đặc tính của bộ điều khiển quyết định đặc tính động của hệ thống trong quá trình điều khiển và chỉ bị thay đổi rất ít so với thiết kế. Chỉ có thể đạt được chất lượng điều khiển tốt nếu như người thiết kế chọn thuật điều khiển và thời gian trích mẫu phù hợp với đối tượng.

Hoàn toàn có thể thiết kế bộ điều khiển (cấu trúc, tham số, gọi module mềm PID trong chương trình hệ thống) mà không cần lập trình. Tuy nhiên muốn làm được như vậy phải nắm vững phần mềm Step7.

6.2.2 Khai báo tham số và các biến của module mềm PID

Người thiết kế có thể khai báo tham số và các biến cho bộ điều khiển trong một khối dữ liệu địa phương (instance data block) bằng cách sử dụng giao diện của module mềm PID. Để vào chương trình khai báo tham số ta thực hiện:

Start → SIMATIC → STEP7 → PID Control Parameter Assignment

Trong hộp hội thoại đầu tiên, người thiết kế có thể mở khối dữ liệu (DB) đã tích hợp sẵn cho FB41 "**CON_C**", FB42 "**CON_S**" hoặc mở một khối dữ liệu mới hoàn toàn. Riêng FB43 "**PULSEGEN**" không thể thực hiện chọn tham số và biến qua giao diện, trong trường hợp này người thiết kế phải sử dụng công cụ của Step7 để thiết lập tham số và khai báo biến cho bộ điều khiển.

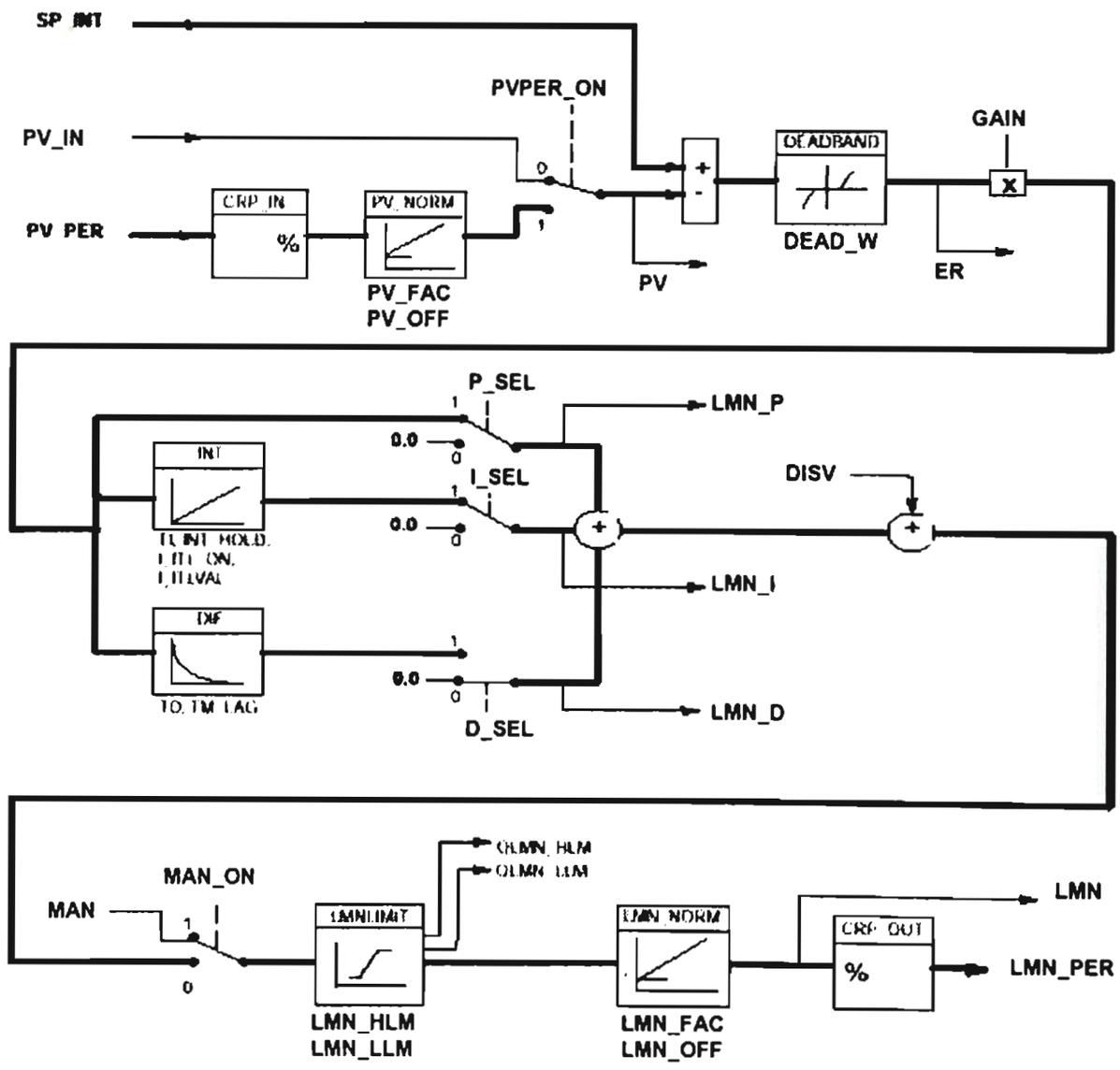
Đối với CPU 314 IFM có thể thiết lập tham số và biến cho module mềm SFB41 hoặc SFB42 bằng cách nhập trực tiếp một khối dữ liệu bất kỳ và chọn nó làm khối dữ liệu cục bộ cho những module này.

6.3 Điều khiển liên tục với FB41 "CONT_C"

6.3.1 Giới thiệu chung về FB41

Sơ đồ cấu trúc của module mềm FB41 "**CONT_C**" được minh họa trong hình 6.2.

FB41 "**CONT_C**" được sử dụng để điều khiển các quá trình kỹ thuật với các biến đầu vào và đầu ra tương tự trên cơ sở thiết bị khả trình Simatic. Trong khi thiết lập tham số, có thể tích cực hoặc không tích cực một số thành phần chức năng của bộ điều khiển PID cho phù hợp với đối tượng.



Hình 6.2. Cấu trúc của module mềm **FB41 "CONT_C"**

Có thể sử dụng module mềm PID như một bộ điều khiển với tính hiệu chủ đạo đặt cứng (fixed setpoint) hoặc thiết kế một hệ thống điều khiển nhiều mạch vòng theo kiểu điều khiển cascade. Những chức năng điều khiển được thiết kế trên cơ sở của thuật điều khiển PID của bộ điều khiển mẫu với tín hiệu tương tự.

Module mềm PID bao gồm tín hiệu chủ đạo **SP_INT**, tín hiệu ra của đối tượng **PV_PER**, tín hiệu giả để mô phỏng tín hiệu ra của đối tượng **PV_IN**, các biến trung gian trong quá trình thực hiện luật và thuật điều khiển PID như **PVPER_ON**, **P_SEL**, **I_SEL**, **D_SEL**, **MAN_ON** ....

Tín hiệu chủ đạo SP_INT: được nhập dưới dạng số thực dấu phảy động.

Tín hiệu ra của đối tượng PV_PER: Thông qua hàm nội của FB41 có tên **CRP_IN**, tín hiệu ra của đối tượng có thể được nhập dưới dạng số nguyên có dấu hoặc số thực dấu phảy động. Chức năng của **CRP_IN** là chuyển đổi kiểu biểu diễn của **PV_PER** từ dạng số nguyên sang số thực dấu phảy động có giá trị nằm trong khoảng −100 đến 100% theo công thức:

$$\text{Tín hiệu ra của } \mathbf{CRP\_IN} = \mathbf{PV\_PER} \times \frac{100}{27648}$$

Chuẩn hóa: Chức năng của hàm chuẩn hóa **PV_NORM** tín hiệu ra của đối tượng là chuẩn hóa tín hiệu ra của hàm **CRP_IN** theo công thức:

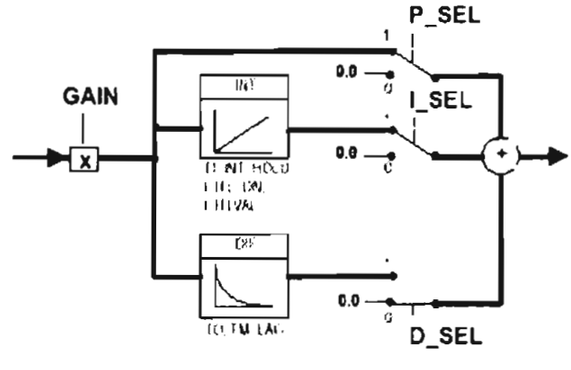
Tín hiệu ra của **PV_NORM** = (Tín hiệu ra của **CRP_IN**)×**PV_FAC_OFF**

Hai tham trị khống chế dải giá trị cho phép của **PV_NORM** là **PV_FAC** và **PV_OFF**. Mặc định **PV_FAC** của hàm **PV_NORM** có giá trị bằng 1 và **PV_OFF** có giá trị bằng 0.

Lọc nhiễu tác động trong lân cận điểm làm việc: Tín hiệu sai lệch là hiệu giữa tín hiệu chủ đạo và tín hiệu ra của đối tượng. Nó được tạo ra ngay trong FB41 và là đầu vào của khối **DEADBAND** có tác dụng lọc những dao động nhỏ xung quanh giá trị xác lập. Nếu không muốn sử dụng **DEADBAND** hoặc với đối tượng mà có thể bỏ qua sự ảnh hưởng của nhiễu trong lân cận điểm làm việc ta chọn **DEAD_W** = 0.

6.3.2 Chọn luật điều khiển trên module FB41 "CONT_C"

Hình 6.3 mô tả thuật PID được thiết kế theo kiểu song song của ba thuật điều khiển đơn lẻ: tỷ lệ (P), tích phân (I) và vi phân (D) theo sơ đồ cấu trúc trong hình 6.2 (sau khối **DEADBAND**). Chính vì cấu trúc song song như vậy nên ta có thể thông qua các tham trị **P_SEL**, **I_SEL** hay **D_SEL** mà tích hợp được các thuật điều khiển khác nhau từ bộ điều khiển mẫu này như thuật điều khiển P, PI, PD và PID.



Hình 6.3: Thuật điều khiển PID.

6.3.3 Đặt giá trị

Phần mềm cho phép chọn chế độ tự động (automatic mode) hoặc chế độ bằng tay. Ở chế độ bằng tay các giá trị của các biến được chọn bằng tay. Bộ tích phân (**INT**) tự thiết lập chế độ **LNM_LNM_P-DISV** và bộ vi phân (**DIF**) tự động về 0. Điều đó đảm bảo cho việc chuyển chế độ từ thiết lập giá trị bằng tay về chế độ tự động không gây một biến đổi đột ngột nào đối với các biến đã được thiết lập giá trị bằng tay.

Cũng có thể đặt giới hạn cho các giá trị được thiết lập bằng tay nhờ hàm **LMNLIMIT**. Một bit cờ sẽ có giá trị logic bằng 1 khi biến vào có giá trị vượt quá giới hạn đã chọn. Hàm **LMN_NORM** sẽ chuẩn hoá tín hiệu ra của hàm **LMNLIMIT** theo công thức:

LMN = (Tín hiệu ra của **LMNLIMIT**)***LMN_FAC** + **LMN_OFF**

Mặc định **LMN_FAC** có giá trị bằng 1, còn **LMN_OFF** có giá trị bằng 0. Các giá trị đặt bằng tay có thể theo một cách biểu diễn riêng. Hàm **CRP_OUT** có chức năng biến đổi từ kiểu biểu diễn số thực dấu phảy động sang kiểu biểu diễn riêng theo công thức:

$$\mathbf{LMN\_PER} = \mathbf{LMN}^* \frac{27648}{100}$$

Ngoài ra nhiễu có thể được lọc trước bằng cách đưa qua đầu vào **DISV**.

6.3.4 Khởi động và thông báo lỗi

FB41 "**CON_C**" có một chương trình con phục vụ cho việc khởi tạo lại hoàn toàn hệ thống. Chương trình này được gọi khi tín hiệu vào **COM_RST** có giá trị logic bằng 1.

Trong khi khởi tạo, luật điều khiển tích phân được tự động thiết lập với giá trị khởi tạo **I_ITVAL**. Nếu luật điều khiển này được gọi theo ngắt thời gian, nó sẽ luôn luôn làm việc với giá trị này. Tất cả các đầu ra khác được đặt giá trị mặc định.

Khối FB41 "**CON_C**" không có khả năng tự kiểm tra lỗi bên trong của module mềm PID. Mã báo lỗi **RET_VAL** không được sử dụng.

6.3.5 Tham biến hình thức đầu vào

Khối FB41 "**CON_C**" có 26 tham biến hình thức đầu vào như sau:

Tên biến	Kiểu dữ liệu	Phạm vi giới hạn	Giá trị mặc định	Mô tả chức năng
COM_RST	**BOOL**		**FALSE**	COMPLETE RESTART Khối có chức năng khởi tạo lại hệ thống hoàn toàn khi đầu vào "complete restart" được thiết lập giá trị logic TRUE.
MAN_ON	**BOOL**		**TRUE**	MANUAL VALUE ON Khi đầu vào "manual value on" có giá trị logic **TRUE** mạch vòng điều khiển sẽ bị ngắt, các giá trị sẽ được thết lập bằng tay

PVPER_ON	**BOOL**		**FALSE**	PROCESS VARIABLE PERIPHERAL ON Khi đọc biến quá trình từ các cổng vào/ra, đầu vào **PV_PER** phải được nối với các cổng vào/ra và đầu vào "process variable peripheral" có giá trị logic **TRUE**.
P_SEL	**BOOL**		**TRUE**	PROPORTIONAL ACTION ON Hoạt động của bộ điều khiển **PID** có thể tích cực hoặc không tích cực từng phần riêng trong thuật điều khiển **PID**. Thuật điều khiển tỷ lệ được kích hoạt khi giá trị logic **TRUE** được thiết lập tại cổng vào "proportional action on"
I_SEL	**BOOL**		**TRUE**	INTERGRAL ACTION ON Hoạt động của bộ điều khiển PID có thể tích cực hoặc không tích cực từng phần riêng trong thuật điều khiển **PID**. Thuật điều khiển tỷ lệ được kích hoạt khi giá trị logic **TRUE** được thiết lập tại cổng vào "proportional action on"
INT_ HOLD	**BOOL**		**FALSE**	INTERGRAL ACTION HOLD Đầu ra của bộ điều khiển tích phân có thể bị "đông lạnh"(không được sử dụng) khi thiết lập giá trị logic **TRUE** cho đầu vào "intergral action hold "
I_ITL_ON	**BOOL**		**FALSE**	INITIALIZATION OF THE INTERGRAL ACTION Đầu ra của bộ điều khiển tích phân có thể được nối vào cổng vào **I_ITL_VAL** nếu như cổng vào "initialization of the intergral action on" có giá trị logic **TRUE**.
D_SEL	**BOOL**		**FALSE**	DERIVATE ACTION ON Hoạt động của bộ điều khiển **PID** có thể tích cực hoặc không tích cực từng phần riêng trong thuật điều khiển **PID**. Thuật điều khiển vi phân được kích hoạt khi giá trị logic **TRUE** được thiết lập tại cổng vào "derivate action on"
CYCLE	**TIME**	≥ 1ms	**T#1s**	SAMPLING TIME Thời gian lấy mẫu là khoảng thời gian **không đổi** giữa các lần khối được cập nhật.
SP_INT	**REAl**	-100.0 ... 100.0% hoặc giá trị vật lý	**0.0**	INTERNAL SEPOINT Đầu vào "internal setpoint" được sử dụng để thiết lập tín hiệu chủ đạo (tín hiệu mẫu).
PV_IN	**REAL**	-100.0 ... 100.0% hoặc giá trị vật lý	**0.0**	PROCESS VARIABLE IN Giá trị khởi tạo có thể đặt ở đầu vào "process variable on" hoặc từ biến quá trình được biểu diễn dưới dạng số thực dấu phẩy động.
PV_PER	**WORD**		**W#16#0000**	PROCESS VARIABLE PERIPHERAL Biến quá trình được nối với CPU thông qua cổng vào tương tự
MAN	**REAL**	-100.0 ... 100.0% hoặc giá trị vật lý	**0.0**	MANUAL VALUE Cổng vào "manual value" được sử dụng để đặt giá trị bằng các hàm giao diện
GAIN	**REAL**		**2.0**	PROPORTIONAL GAIN Đầu vào "proportional gain" được sử dụng để thiết lập hệ số tỷ lệ cho bộ điều khiển theo luật tỷ lệ

TI	**TIME**	≥ **CYCLE**	**T#20s**	RESET TIME Cổng vào "reset time" được sử dụng để thiết lập hằng số thời gian tích phân cho bộ điều khiển tích phân
TD	**TIME**	≥ **CYCLE**	**T#10s**	DERIVATE TIME Cổng vào "derivate time" sử dụng để thiết lập hằng số thời gian vi phân cho bộ điều khiển vi phân
TM_LAG	**TIME**	≥ **CYCLE**	**T#2s**	TIME LAG OF DERIVATE ACTION Thời gian tích cực của luật điều khiển vi phân được chọn thông qua cổng vào "time lag of the derivate action"
DEADB_W	**REAL**	>=0.0(%) hoặc giá trị vật lý	0.0	DEAD BAND WIDTH Một vùng kém nhạy được sử dụng để xử lý tín hiệu sai lệch. Độ rộng của vùng kém nhạy được đặt thông qua cổng vào "dead band width".
LMN_HLM	**REAL**	**LMN_LLM** ... 100(%) hoặc giá trị vật lý	100.0	MANIPULATED VALUE HIGH LIMIT Giá trị hạn chế trên được thiết lập bằng tay qua cổng vào "manipulated value high limit
LMN_LLM	**REAL**	-100 ... **LMN_LLM** (%) hoặc giá trị vật lý	0.0	MANIPULATED VALUE LOW LIMIT Giá trị hạn ché dưới được thiết lập bằng tay qua cổng vào "manipulated value low limit"
PV_FAC	**REAL**		1.0	PROCESS VARIABLE FACTOR Biến quá trình được nhân với một hệ số cho phù hợp với phạm vi qui định của biến này. Hệ số được chọn thông qua cổng vào "process variable factor".
PV_OFF	**REAL**		1.0	PROCESS VARIABLE OFFSET Biến quá trình được cộng với một lượng bù cho phù hợp với phạm vi qui định của biến này. Giá trị bù được chọn thông qua cổng vào "process variable offset"
LMN_FAC	**REAL**		1.0	MANIPULATED VALUE OFFSET Giá trị giới hạn được nhân với một hệ số cho phù hợp với phạm vi qui định của biến quá trình. Hệ số này được đặt qua cổng vào "manipulated value factor"
LMN_OFF	**REAL**		0.0	MANIPULATED VALUE OFFSET Giá trị giới hạn được cộng thêm một lượng bù cho phù hợp với phạm vi qui định của biến quá trình. Giá trị bù này được đặt qua cổng vào "manipulated value offset".
I_ITLVAL	**REAL**	-100.0 ... 100.0 (%) hoặc giá trị vật lý	0.0	INITIALIZATION VALUE OF THE INTERGRAL ACTION Giá trị đầu ra của bộ điều khiển tích phân có được thiết lập thông qua cổng vào "initialization value of the intergral action".
DISV	**REAL**	-100.0 ... 100.0(%) hoặc giá trị vật lý	0.0	DISTURBANCE VARIABLE Khi điều khiển hệ thống bằng phương pháp feedforward thì một giá trị bù nhiễu được đặt thông qua cổng vào "disturbance variable".

6.3.6 Tham biến hình thức đầu ra

Khối FB41 "CON_C" có 9 tham biến hình thức đầu ra như sau:

Tên biến	Kiểu dữ liệu	Mặc định	Mô tả
LMN	REAL	0.0	MANIPULATED VALUE Giá trị ra được thiết lập bằng tay thông qua cổng ra "manipulated value".
LMN_PER	WORD	w#16#0000	MANIPULATED VALUE PERIPHERAL Giá trị đầu ra thiếtlập bằng tay theo kiểu biểu diễn phu hợp với các cổng vào/ra tương tự được chọn qua cổng ra ""manipulated value peripheral".
QLMN_HLM	BOOL	FALSE	HIGH LIMIT OF MANIPULATED VALUE REACHED Cổng ra "high limit of manipulated value reached" thông báo giá trị biến quá trình vượt quá giá rị giới hạn trên.
QLMN_LLM	BOOL	FALSE	LOW LIMIT OF MANIPULATED VALUE REACHED Cổng ra "low limit of manipulated value reached" thông báo giá trị của biến quá trinh nhỏ hơn giá trị giới hạn dưới.
LMN_P	REAL	0.0	PROPORTIONAL COMPONENT Tín hiệu ra của bộ điều khiển tỷ lệ được xuất qua cổng ra "proportional component".
LMN_I	REAL	0.0	INTEGRAL COMPONENT Tín hiệu ra của bộ điều khiển tích phân được xuất qua cổng ra "integral component".
LMN_D	REAL	0.0	DERIVATIVE COMPONENT Tín hiệu ra của bộ điều khiển vi phân được xuất qua cổng ra "derivative component"
PV	REAL	0.0	PROCESS VALUE Tín hiệu quá trình được xuất qua cổng ra "process value"
ER	REAL	0.0	ERROR SIGNAL Tín hiệu sai lệch được xuất qua cổng ra "error signal"

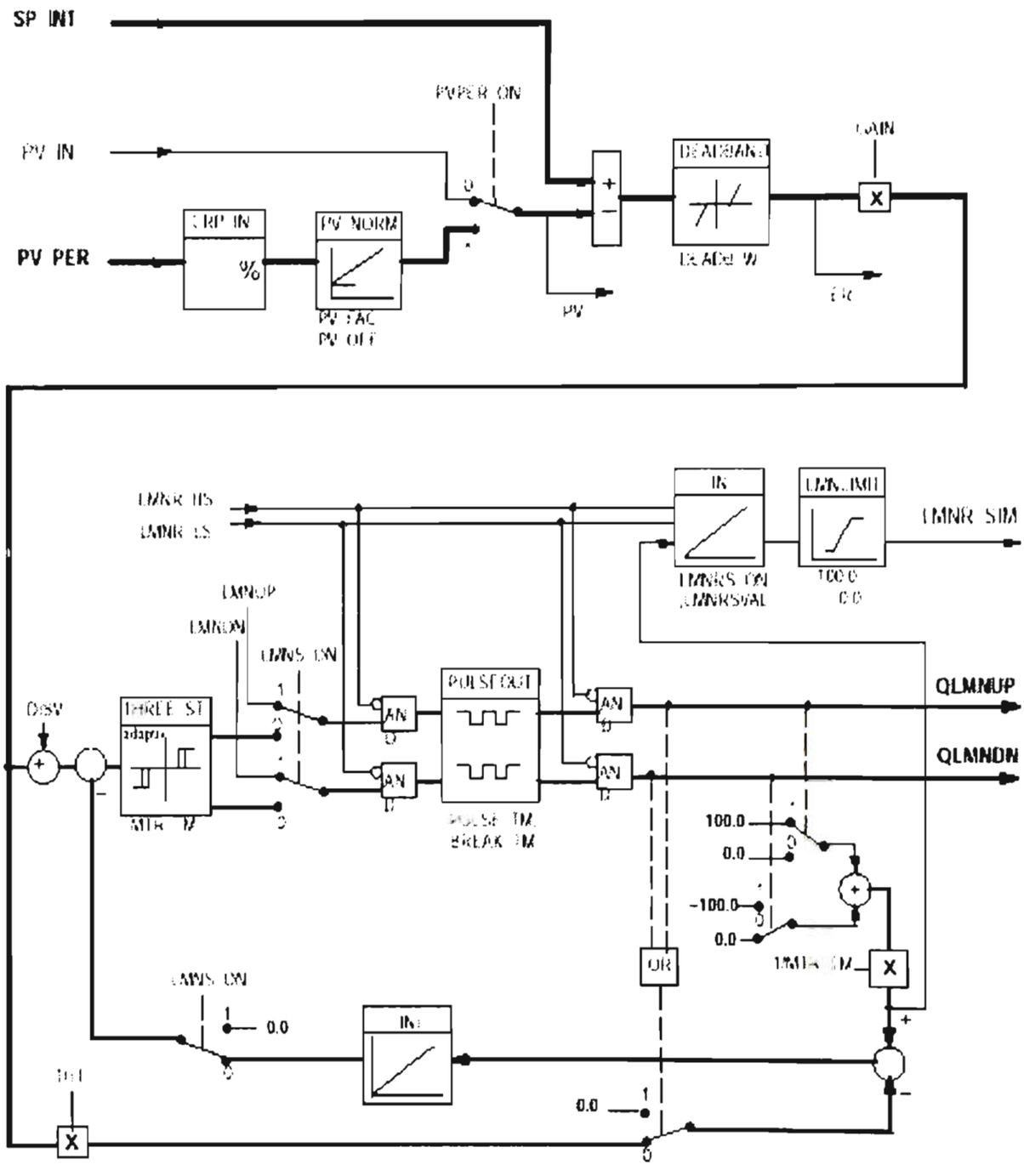
6.4 Điều khiển bước với FB42 "CONT_S"

6.4.1 Mô tả chung

FB42 "CONT_S" là module mềm được tích hợp sẵn trong phần mềm STEP 7. FB42 "CONT_S " được sử dụng trên cơ sở Simatic S7–300/400 để điều khiển các đối tượng kỹ thuật với đầu ra của bộ điều khiển là tín hiệu số. Tín hiệu ra số hoàn toàn thích hợp đối với các cơ cấu chấp hành kiểu tích phân. Trong khi thiết lập tham số, người thiết kế có thể tích cực hoặc không tích cực bộ điều khiển PI bước cho phù hợp với yêu cầu của bài toán điều khiển đặt ra. Có thể sử dụng module mềm FB42 "CONT_S" như một bộ điều khiển theo luật PI với tín hiệu chủ đạo đặt trước hoặc có thể sử dụng nó trong mạch vòng điều khiển phụ trong hệ thống thiết kế dựa trên nguyên tắc điều khiển cascade. Chức năng của bộ điều khiển này hoàn toàn tuân theo thuật điều khiển PI với tín hiệu quá trình là tín hiệu tương tự và tín hiệu ra của bộ điều khiển là tín hiệu số.

Một phần trong các chức năng của module mềm này là đóng vai trò của một bộ điều khiển PI có các gía trị và tín hiệu đầu ra số đặt bằng tay. Làm việc ở chế độ này bộ điều khiển bước không cần đến tín hiệu hồi tiếp.

Hình sau mô tả nguyên lý hoạt động của FB42 "**CONT_S**".



Hình 6.4: Sơ đồ cấu trúc nguyên lý của module mềm **FB42 "CONT_S"**

Tín hiệu chủ đạo: Tín hiệu chủ đạo được biểu diễn kiểu số thực dấu phảy động và được thiết lập từ cổng vào **SP_INT**.

Tín hiệu ra của đối tượng: Tín hiệu ra của đối tượng được đưa thẳng từ cổng vào tương tự theo kiểu số nguyên hoặc được truyền sau khi đã biến đổi sang kiểu số thực dấu phảy

động. Hàm **CRP_IN** có chức năng biến đổi giá trị truyền từ cổng vào tương tự sang kiểu số thực dấu phảy động trong khoảng từ –100.0% đến 100.0% theo công thức:

$$\text{Tín hiệu ra của } \mathbf{CRP\_IN} = \mathbf{PV\_PER} * \frac{100.0}{27648.0}$$

Chuẩn hóa: Chức năng của hàm **PV_NORM** là chuẩn hóa tín hiệu lấy từ đầu ra của hàm **CRP_IN** theo công thức:

Tín hiệu ra của **PV_NORM** = (Tín hiệu ra của **CRP_IN**)***PV_FAC** + **PV_OFF**

Mặc định **PV_FAC** có giá trị bằng 1 và **PV_OFF** có giá trị bằng 0.

Lọc nhiễu tác động trong lân cận điểm làm việc: Tín hiệu sai lệch là hiệu giữa tín hiệu chủ đạo và tín hiệu ra của đối tượng. Giống như FB41, trong FB42 cũng có khối **DEADBAND** được thiết kế ngay sau tín hiệu sai lệch và trước phần điều khiển theo luật để lọc những dao động nhỏ quanh giá trị xác lập bằng cách tạo ra ở đó một vùng kém nhạy. Nếu giá trị **DEADB_W**=0 thì vùng kém nhạy không tồn tại.

6.4.2 Thuật điều khiển PI bước

Khối hàm FB42 của module mềm PID làm việc không cần phải có tín hiệu hồi tiếp. Chức năng của luật I trong thuật điều khiển PI và tín hiệu sai lệch được tính trong một bộ tích phân **INT**, sau đó so sánh với tín hiệu ra của bộ điều khiển theo luật tỷ lệ như một giá trị hồi tiếp. Hiệu của phép so sánh này được đưa vào một rơ le ba vị trí có trễ **Three_ST** và đầu ra của rơ le này điều khiển bộ phát xung ra **PULSEOUT** để điều khiển cơ cấu chấp hành. Có thể giảm tần số đóng cắt của bộ điều khiển bằng cách tạo ra vùng trễ khi chuyển vị trí của rơ le. Sơ đồ thuật toán biểu diễn trong hình 6.4.

Ngoài ra, để giảm ảnh hưởng của nhiễu trong trường hợp điều khiển không hồi tiếp, có thể lọc nhiễu cho hệ bằng cách đưa tín hiệu vào đầu vào **DISV** của bộ lọc nhiễu.

6.4.3 Khởi động và thông báo lỗi

Hệ thống được khởi tạo lại hoàn toàn khi cổng vào **COM_RST** có giá trị logic bằng 1. Tất cả các cổng ra nhận giá trị mặc định.

Khối hàm FB42 này không có khả năng tự kiểm tra lỗi bên trong. Nó không sử dụng cổng ra báo kiểu lỗi **RET_VAL**.

6.4.4 Tham biến hình thức đầu vào

Tất cả các tham biến hình thức đầu vào của FB42 "**CON_S**" gồm:

Tham số	Kiểu dữ liệu	Phạm vi làm việc	Mặc định	Mô tả
COM_RST	**BOOL**		**FALSE**	COMPLETE RESTART Module mềm được khởi tạo lại hoàn toàn khi ở cổng vào complete restart có giá trị logic bằng 1.

LMNR_HS	**BOOL**		**FALSE**	HIGH LIMIT OF POSITION FEEDBACK SIGNAL Tín hiệu "actuator at upper limit stop" được nối đến cổng vào "high limit of position feedback" Cổng ra cơ cấu chấp hành sẽ bị cấm khi LMNR_HS=TRUE
LMNR_LS	**BOOL**		**FALSE**	LOW LIMIT OF POSITION FEEDBACK SISNAL Tín hiệu "actuator at lower limit stop" được nối đến cổng vào "low limit of position feedback". Cổng ra cơ cấu chấp hành sẽ bị cấm khi **LMNR_LS=TRUE**
LMNS_ON	**BOOL**		**FALSE**	MANUAL ACTUATING SIGNALS ON Xử lý tín hiệu chấp hành được chuyển sang chế độ bằng tay qua cổng vào "manual actuating signal on"
LMNUP	**BOOL**		**FALSE**	ACTUATING SIGNALS UP Tín hiệu ra QLMNUP được thiết lập qua cổng vào "actuating signal up" với các tín hiệu chấp hành bằng tay
LMNDN	**BOOL**		**FALSE**	ACTUATING SIGNALS DOWN Tín hiệu ra QLMNDN được thiết lập qua cổng vào "actuating signals down" với các tín hiệu chấp hành bằng tay.
PVPER_ON	**BOOL**		**FALSE**	PROCESS VARIABLE PERIPHERAL ON Muốn đọc các tín hiệu quá trình từ các cổng vào/ra thì ở cổng vào "process variable on" phải có giá trị logic 1 (cho phép đọc)
CYCLE	**TIME**	≥1ms	**T#1s**	SAMPLING TIME Khoảng thời gian giữ các lần gọi khối phải không cố định. Thời gian trích mẫu (sampling time) được thiết lập qua cổng vào "sampling time"
SP_INT	**REAL**	-100.0 ... 100.0 (%) hoặc giá trị vật lý	**0.0**	INTERNAL SETPOINT Tín hiệu chủ đạo được đặt qua cổng vào "internal set point"
PV_IN	**REAL**	-100.0 ... 100.0 (%) hoặc giá trị vật lý	**0.0**	PROCESS VARIABLE IN Giá trị đầu của biến quá trình có thể đặt trước qua cổng vào "process variable in" theo kiểu số thực dấu phảy động.
PV_PER	**WORD**		W#16#0000	PROCESS VARIABLE PERIPHERAL Biến quá trình theo kiểu biểu diễn số nguyên được truyền trực tiếp từ cổng vào/ra tương tự qua cổng vào "process variable peripheral"
GAIN	**REAL**		**2.0**	PROPORTIONAL GAIN Hệ số khuếch đại của luật **P** được đặt qua cổng vào "proportional gain"
TI	**TIME**	≥**CYCLE**	**T#20s**	RESET TIME Hằng số thời gian tích phân của luật **I** được đặt qua cổng vào "reset time"
DEADB_W	**REAL**	0.0 ... 100 (%) hoặc giá trị vật lý	**1.0**	DEAD BAND WIDTH Một vùng kém nhạy được thiết kế để xử lý tín hiệu sai lệch. Độ rộng của vùng kém nhạy được đặt qua cổng vào "dead band width"
PV_FAC	**REAL**		**1.0**	PROCESS VARIABLE FACTOR Biến quá trình được nhân với một hệ số cho phù hợp với phạm vi qui định của biến này. Hệ số được chọn thông qua cổng vào "process variable factor".

PV_OFF	**REAL**		0.0	PROCESS VARIABLE OFFSET Biến quá trình được cộng với một lượng bù cho phù hợp với phạm vi qui định của biến này. Giá trị bù được chọn thông qua cổng vào "process variable offset"
PULSE_TM	**TIME**	≥CYCLE	T#3s	MINIMUM PULSE TIME Chu kỳ phát xung nhỏ nhất được đặt qua cổng vào "minimum pulse time"
BREAK_TM	**TIME**	≥CYCLE	T#3s	MINIMUM BREAK TIME Thời gian cấm nhỏ nhất được đặt qua cổng vào "minimum break time"
MTR_TM	**TIME**	≥CYCLE	T#30s	MOTOR ACTUATING TIME Khoảng thời gian cần thiết để cơ cấu chấp hành chuyển từ giới hạn dừng này sang giới hạn dừng khác được đặt qua cổng vào "motor actuating time"
DISV	**REAL**	100.0 ... 100.0 (%) hoặc giá trị vật lý	0.0	DISTURBANCE VARIABLE Khi điều khiển hệ thống bằng phương pháp feedforward thì một giá trị bù nhiễu được đặt thông qua cổng vào "disturbance variable".

6.4.5 Tham biến hình thức đầu ra

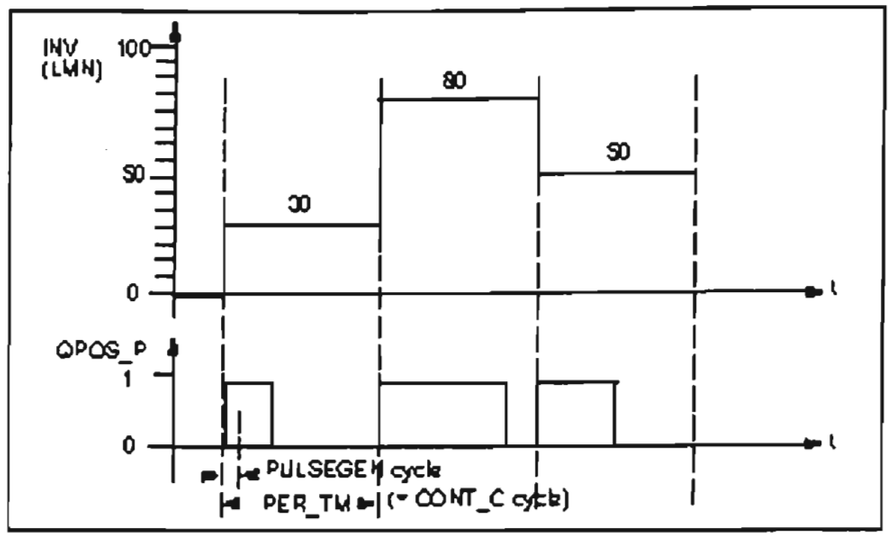
Tất cả các tham biến hình thức đầu ra của FB42 "**CON_S**" gồm:

Cổng ra	Kiểu dữ liệu	Mặc định	Mô tả
QLMNUP	**BOOL**	**FALSE**	ACTUATING SIGNAL UP Cơ cấu chấp hành van được mở khi cổng ra "actuating signal up"
QLMNDN	**BOOL**	**FALSE**	ACTUATING SIGNAL DOWN Cơ cấu chấp hành van được mở khi cổng ra "actuating signal down"
PV	**REAL**	0.0	PROCESS VARIABLE Biến quá trình có thể được xuất qua cổng ra "process variable"
ER	**REAL**	0.0	ERROR SIGNAL Tín hiệu sai lệch có thể được xuất qua cổng ra "error signal"

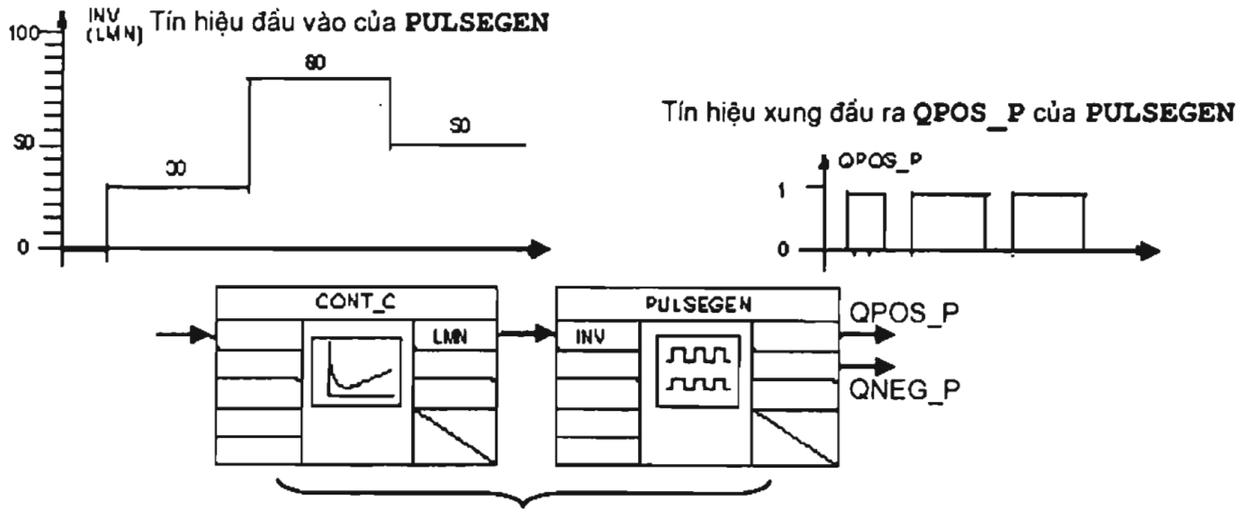
6.6 Khối hàm tạo xung FB43 "PULSEGEN"

Khối hàm FB43 "**PULSEGEN**" có tác dụng hỗ trợ việc thiết kế một bộ điều khiển PID hai hoặc ba vị trí với bộ tạo xung theo nguyên tắc điều biên (hình 6.5). Nó biến đổi tín hiệu đầu vào **INV** dạng số thực (thường là đầu ra **LMN** của module mềm PID) thành một dãy xung có chu kỳ cố định và độ rộng tương ứng với độ lớn của tín hiệu đầu vào.

Khối hàm FB43 "**PULSEGEN**" thường được sử dụng cùng với FB41 "**CONT_C**" để có được một bộ điều khiển PID với tín hiệu đầu ra dạng xung (hình 6.6). Nói cách khác một mình nó không phải là bộ điều khiển PID. Bởi vậy, ở đây chúng tôi sẽ không tiếp tục trình bày chi tiết vào nguyên lý làm việc của FB43 "**PULSEGEN**". Bạn đọc quan tâm có thể xem tài liệu hướng dẫn sử dụng PID có ngay trong các phần mềm hỗ trợ tra cứu của Step7 (**PID Control English**).



Hình 6.5: Nguyên lý điều biên của **FB43 PULSEGEN**



Hình 6.6: Khối **FB43 PULSEGEN** được sử dụng cùng với **FB41 CONT_C** để tạo thành bộ điều khiển **PID** với đầu ra dạng xung.

Tài liệu tham khảo

[1] **Berger**: Automating with Simatic, MCD Verlag, 2000.

[2] **Metz; Merbeth**: Schaltalgebra. Fachbuch Verlag, 1970.

[3] **Minh; Phước**: Lý thuyết Điều khiển mờ. Nhà xuất bản Khoa học và Kỹ thuật, 1997.

[4] **Minh; Phước**: Tự động hóa với Simatic S7–200. Nhà xuất bản Nông nghiệp, 1997.

[5] **Minh; Phước; Thanh**: S5–95U và phần mềm Step5. Giáo trình giảng dạy của Trung tâm Đào tạo Siemens Tự động hóa tại Trường ĐHBK Hà Nội, 1997.

[6] **Siemens AG**: Statement List for S7–300 and S7–400. Reference Manual, 1996.

[7] **Philippow**: Taschenbuch der Elektrrotechnik, Band 2. VEB Verlag Technik Berlin, 1987.

[8] **Siemens AG**: Simatic STEP7 Program Design. Programming Manual, 1996.

[9] **Siemens AG**: Simatic STEP7 User Manual, 1995.

[10] **Siemens AG**: S7–300 Fuzzy Control. User Manual, 1996.

[11] **Siemens AG**: S7–300 Hardware Configuration and Structure, 1995.

[12] **Siemens AG**: Step7–Standard and System Functions, 1998.

[13] **Sơn**: PLC là gì. Tự động hóa ngày nay. Số 3, 2000.

[14] **TC Mannheim**: Training of Simatic on CD, 1997.

[15] **Töpfer; Besch**: Grundlagen der Automatisierungstechnik. Hanser Verlag, 1990.

TỰ ĐỘNG HÓA VỚI SIMATIC S7–300

Nguyễn Doãn Phước, Phan Xuân Minh, Vũ Vân Hà

Chịu trách nhiệm xuất bản: PGS.TS. Tô Đăng Hải

Biên tập: Nguyễn Thị Ngọc Khuê

Trình bày và chế bản: Tác giả

Vẽ bìa: Hương Lan

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

70 Trần Hưng Đạo, Hà Nội

In 1000 cuốn khổ 16 x 24 cm tại Công ty In Hàng không. Giấy phép xuất bản số 1104 - 81 - 10/8/2004. In xong và nộp lưu chiểu tháng 9/2004.

204180

8 935048 941805

Giá: 37.500đ

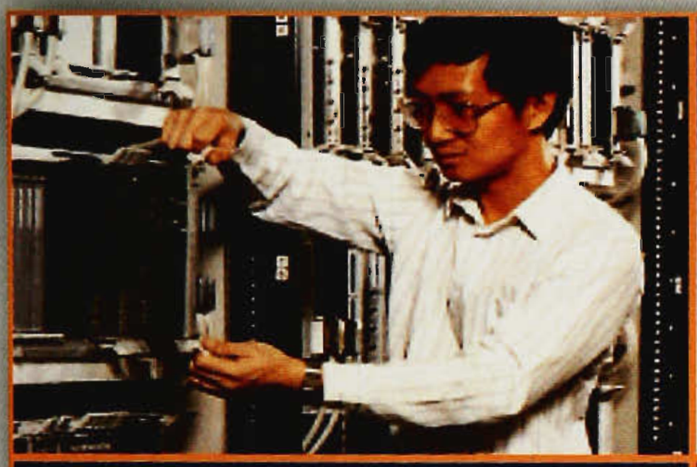
AMECO

Authorised Distributor of Siemens

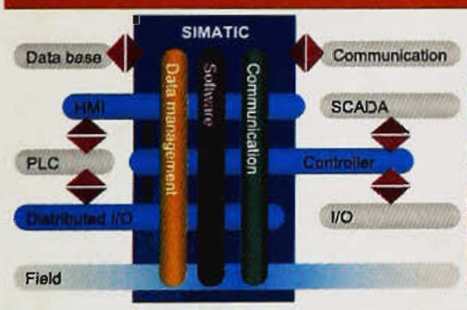
Công ty Tự động hoá - Cơ khí - Môi trường
The Automation Mechanics Environment Company Ltd.

Số 2, Lô A, Thanh Nhàn, Đường Trần Khát Chân, Hai Bà Trưng, Hà Nội

PHÂN PHỐI SẢN PHẨM VÀ HỖ TRỢ KỸ THUẬT 24 / 24



GIẢI PHÁP TỰ ĐỘNG HÓA TÍCH HỢP TOÀN DIỆN



AMECO là đại diện kinh doanh của nhiều tập đoàn nổi tiếng Tây Âu, Mỹ và Nhật Bản trên các lĩnh vực Tự động hoá - Cơ khí - Môi trường. AMECO phân phối các sản phẩm tự động hóa, máy và thiết bị công nghiệp. Cung cấp dịch vụ tư vấn thiết kế, tích hợp hệ thống, chế tạo thiết bị, lắp đặt và vận hành dây chuyền sản xuất, đào tạo và chuyển giao công nghệ.

AUTOMATION
MECHANICS
ENVIRONMENT

Automation System Instrumentation	System Integration
Motion Control	Machine Builder
Control Distribution	Panel Builder

Consultancy
Supply & Commissioning
Training
Field Service

AMECO Integrated Solution

Totally Customer Satisfaction

TEL: 4 - 9714877; FAX: 4 - 9714869; Email: ameco@hn.vnn.vn.