1. GIỚI THIỆU PLC S7-200

2. SỬ DỤNG PLC S7-200

3. BẮT ĐẦU LẬP TRÌNH CHO S7-200

4. CƠ BẢN TRONG LẬP TRÌNH VỚI S7-200

5. BỘ NHỚ DỮ LIỆU VÀ CÁCH ĐỊNH ĐỊA CHỈ

6. CÁC ĐẦU VÀO, RA

7. CÁC PHƯƠNG THỨC TRUYỀN THÔNG

8. TẬP LỆNH SIMATIC

Trước hết, ta điểm qua những giới hạn thông số hợp lệ trong S7-200:

Description	CPU 221	CPU 222	CPU 224
User program size	2 Kwords	2 Kwords	4 Kwords
User data size	1 Kwords	1 Kwords	2.5 Kwords
Process-image input register	I0.0 to I15.7	I0.0 to I15.7	I0.0 to I15.7
Process-image output register	Q0.0 to Q15.7	Q0.0 to Q15.7	Q0.0 to Q15.7
Analog inputs (read only)	--	AIW0 to AIW30	AIW0 to AIW30
Analog outputs (write only)	--	AQW0 to AQW30	AQW0 to AQW30
Variable memory (V)[1]	VB0.0 to VB2047.7	VB0.0 to VB2047.7	VB0.0 to VB5119.7
Local memory (L)[2]	LB0.0 to LB63.7	LB0.0 to LB63.7	LB0.0 to LB63.7
Bit memory (M)	M0.0 to M31.7	M0.0 to M31.7	M0.0 to M31.7
Special Memory (SM) Read only	SM0.0 to SM179.7 SM0.0 to SM29.7	SM0.0 to SM179.7 SM0.0 to SM29.7	SM0.0 to SM179.7 SM0.0 to SM29.7
Timers Retentive on-delay 1 ms Retentive on-delay 10 ms Retentive on-delay 100 ms On/Off delay 1 ms On/Off delay 10 ms On/Off delay 100 ms	256 (T0 to T255) T0, T64 T1 to T4, T65 to T68 T5 to T31, T69 to T95 T32, T96 T33 to T36, T97 to T100 T37 to T63, T101 to T255	256 (T0 to T255) T0, T64 T1 to T4, T65 to T68 T5 to T31, T69 to T95 T32, T96 T33 to T36, T97 to T100 T37 to T63, T101 to T255	256 (T0 to T255) T0, T64 T1 to T4, T65 to T68 T5 to T31, T69 to T95 T32, T96 T33 to T36, T97 to T100 T37 to T63, T101 to T255
Counters	C0 to C255	C0 to C255	C0 to C255
High-speed counter	HC0, HC3, HC4, HC5	HC0, HC3, HC4, HC5	HC0 to HC5
Sequential control relays (S)	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7
Accumulator registers	AC0 to AC3	AC0 to AC3	AC0 to AC3
Jumps/Labels	0 to 255	0 to 255	0 to 255
Call/Subroutine	0 to 63	0 to 63	0 to 63
Interrupt routines	0 to 127	0 to 127	0 to 127
PID loops	0 to 7	0 to 7	0 to 7
Port	Port 0	Port 0	Port 0

[1] All V memory can be saved to permanent memory.
[2] LB60 to LB63 are reserved by STEP 7-Micro/WIN 32, version 3.0 or later.

Access Method	CPU 221		CPU 222		CPU 224	
Bit access (byte.bit)	V	0.0 to 2047.7	V	0.0 to 2047.7	V	0.0 to 5119.7
	I	0.0 to 15.7	I	0.0 to 15.7	I	0.0 to 15.7
	Q	0.0 to 15.7	Q	0.0 to 15.7	Q	0.0 to 15.7
	M	0.0 to 31.7	M	0.0 to 31.7	M	0.0 to 31.7
	SM	0.0 to 179.7	SM	0.0 to 179.7	SM	0.0 to 179.7
	S	0.0 to 31.7	S	0.0 to 31.7	S	0.0 to 31.7
	T	0 to 255	T	0 to 255	T	0 to 255
	C	0 to 255	C	0 to 255	C	0 to 255
	L	0.0 to 63.7	L	0.0 to 63.7	L	0.0 to 63.7
Byte access	VB	0 to 2047	VB	0 to 2047	VB	0 to 5119
	IB	0 to 15	IB	0 to 15	IB	0 to 15
	QB	0 to 15	QB	0 to 15	QB	0 to 15
	MB	0 to 31	MB	0 to 31	MB	0 to 31
	SMB	0 to 179	SMB	0 to 179	SMB	0 to 179
	SB	0 to 31	SB	0 to 31	SB	0 to 31
	LB	0 to 63	LB	0 to 63	LB	0 to 63
	AC	0 to 3	AC	0 to 3	AC	0 to 3
	Constant		Constant		Constant	
Word access	VW	0 to 2046	VW	0 to 2046	VW	0 to 5118
	IW	0 to 14	IW	0 to 14	IW	0 to14
	QW	0 to 14	QW	0 to 14	QW	0 to 14
	MW	0 to 30	MW	0 to 30	MW	0 to 30
	SMW	0 to 178	SMW	0 to 178	SMW	0 to 178
	SW	0 to 30	SW	0 to 30	SW	0 to 30
	T	0 to 255	T	0 to 255	T	0 to 255
	C	0 to 255	C	0 to 255	C	0 to 255
	LW	0 to 62	LW	0 to 62	LW	0 to 62
	AC	0 to 3	AC	0 to 3	AC	0 to 3
	AIW	0 to 30	AIW	0 to 30	AIW	0 to 30
	AQW	0 to 30	AQW	0 to 30	AQW	0 to 30
	Constant		Constant		Constant	
Double word access	VD	0 to 2044	VD	0 to 2044	VD	0 to 5116
	ID	0 to 12	ID	0 to 12	ID	0 to 12
	QD	0 to 12	QD	0 to 12	QD	0 to 12
	MD	0 to 28	MD	0 to 28	MD	0 to 28
	SMD	0 to 176	SMD	0 to 176	SMD	0 to 176
	SD	0 to 28	SD	0 to 28	SD	0 to 28
	LD	0 to 60	LD	0 to 60	LD	0 to 60
	AC	0 to 3	AC	0 to 3	AC	0 to 3
	HC	0, 3, 4, 5	HC	0, 3, 4, 5	HC	0 to 5
	Constant		Constant		Constant	

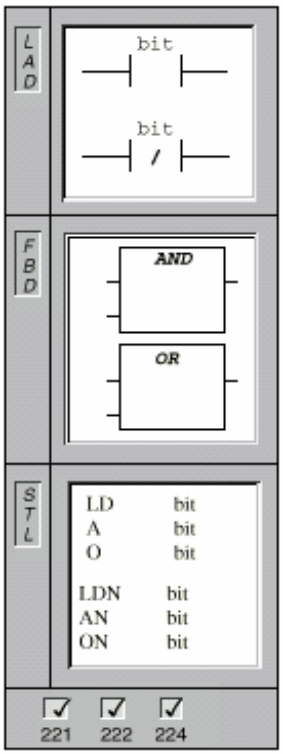
Một số qui định khi tra cứu lệnh và sử dụng lệnh:

- Trên cùng là phần tên lệnh hoặc nhóm lệnh.
- Tiếp theo là cú pháp lệnh, lần lượt trong LAD, FBD và STL.
- Dưới cùng là những loại CPU S7-200 cho phép sử dụng lệnh, lưu ý ở đây chỉ bao gồm 03 loại CPU mới: 221, 222 và 224.
- Bên cạnh là phần mô tả hoạt động của lệnh.

- Các trường hợp lỗi là các trường hợp gây lỗi khiến đầu ra ENO = 0, bình thường khi lệnh được thực hiện thì ENO = 1.
- Các bit đặc biệt bị ảnh hưởng là các bit trong vùng SM có giá trị thay đổi tùy theo kết quả thực hiện lệnh.
- Bảng các toán hạng chỉ ra các thông số hợp lệ của lệnh.
- Sau đây là những ký hiệu khi gõ lệnh trong STEP 7:
  - Trong LAD: ---> nghĩa là có thể nối tiếp lệnh khác (nhưng không bắt buộc).
  - Trong LAD: --->> nghĩa là bắt buộc phải nối tiếp lệnh khác.
  - Tên biến nằm trong ngoặc kép (ví dụ "var") là biến toàn cục.
  - Tên biến có ký tự # đằng trước là biến cục bộ.
  - Ký hiệu ? hay ???? nghĩa là yêu cầu toán hạng.
  - Ký hiệu << hoặc >> yêu cầu hoặc toán hạng hoặc nối lệnh khác.
  - Ký hiệu >I cho biết đó là đầu ra ENO.
  - Ký tự % trước tên biến nghĩa là biến trực tiếp trong IEC.
  - Trong FBD, dấu tròn nhỏ ở đầu vào đánh dấu đảo (như trong điện tử); một gạch dọc ngắn (|) ở đầu vào đánh dấu giá trị tức khắc (đầu vào trực tiếp).

8.1 Các lệnh lô gic với bit

Các công tắc:



Có 02 loại công tắc: công tắc thường mở (Normally Open, viết tắt là NO) và công tắc thường đóng (Normally Closed, viết tắt là NC).

Đối với PLC, mỗi công tắc đại diện cho trạng thái một bit trong bộ nhớ dữ liệu hay vùng ảnh của các đầu vào, ra. Công tắc thường mở sẽ đóng (ON - nghĩa là cho dòng điện đi qua) khi bit bằng 1 còn công tắc thường đóng sẽ đóng (ON) khi bit bằng 0.

Trong LAD, các lệnh này được biểu diễn bằng chính các công tắc thường mở và thường đóng.

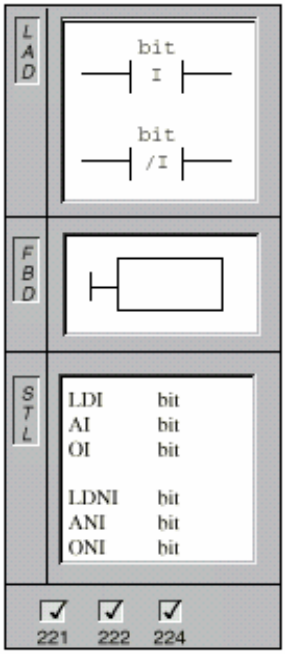
Trong FBD, các công tắc thường mở được biểu diễn như các đầu vào hoặc ra của các khối chức năng AND hoặc OR. Công tắc thường đóng được thêm dấu đảo (vòng tròn nhỏ) ở đầu vào tương ứng.

Trong STL, các công tắc thường mở được sử dụng trong các lệnh LOAD, AND hoặc OR. Lệnh LOAD ghi giá trị bit được đánh địa chỉ bởi toán hạng của lệnh vào đỉnh ngăn xếp, những giá trị cũ trong ngăn xếp bị đẩy xuống một bậc (giá trị dưới cùng sẽ mất). Các lệnh AND và OR thực hiện phép toán lô

gic And hay Or giữa giá trị được trỏ đến bởi toán hạng với đỉnh ngăn xếp, kết quả được ghi vào đỉnh ngăn xếp, những giá trị cũ trong ngăn xếp bị đẩy xuống một bậc. Hoàn toàn tương tự đối với các công tắc thường đóng, được sử dụng trong các lệnh LOAD NOT, AND NOT và OR NOT (giá trị được trỏ đến bởi toán hạng sẽ bị đảo).

Inputs/Outputs	Operands	Data Types
bit (LAD, STL)	I, Q, M, SM, T, C, V, S, L	BOOL
Input (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL
Output (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL

Các công tắc trực tiếp (tức khắc, tức thời):



Mỗi công tắc đại diện cho trạng thái một đầu vào (digital) vật lý (vùng ảnh không cập nhật khi lệnh này được thực hiện).

Công tắc thường mở trực tiếp (Normally Open Immediate) đóng khi đầu vào vật lý bằng 1 và công tắc thường đóng trực tiếp (Normally Closed Immediate) đóng khi đầu vào vật lý bằng 0.

Trong LAD, các lệnh này được biểu diễn bằng các công tắc thường mở và thường đóng trực tiếp.

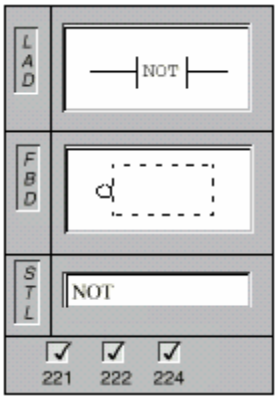
Trong FBD, các công tắc này được biểu diễn như các đầu vào của các khối với ký hiệu tức thời (một gạch dọc ngắn). Các công tắc thường đóng tức thời cũng được ký hiệu thêm bởi dấu đảo (một vòng tròn nhỏ).

Trong STL, các công tắc thường mở tức khắc được sử dụng trong các lệnh LOAD IMMEDIATE, AND IMMEDIATE hoặc OR IMMEDIATE. Lệnh LOAD IMMEDIATE ghi giá trị đầu vào vật lý vào đỉnh ngăn xếp, những giá trị cũ trong ngăn xếp bị đẩy xuống một bậc (giá trị dưới cùng sẽ mất). Các lệnh AND IMMEDIATE và OR IMMEDIATE thực hiện phép toán lô gic And hay Or giữa giá trị đầu vào vật lý với đỉnh ngăn xếp, kết quả được ghi vào đỉnh ngăn xếp, những giá trị cũ trong ngăn xếp bị đẩy xuống một bậc. Hoàn toàn tương tự đối với các công tắc thường đóng tức khắc, được sử dụng trong các lệnh LOAD NOT IMMEDIATE, AND NOT IMMEDIATE và OR NOT IMMEDIATE (giá trị đầu vào vật lý sẽ bị đảo).

Inputs/Outputs	Operands	Data Types
bit (LAD, STL)	I	BOOL
Input (FBD)	I	BOOL

Lệnh đảo (Not):

Lệnh đảo thay đổi dòng năng lượng (Power Flow). Nếu dòng năng lượng gặp lệnh này, nó sẽ bị chặn lại. Ngược lại nếu phía trước lệnh này không có dòng năng lượng, nó sẽ trở thành nguồn cung cấp dòng năng lượng.



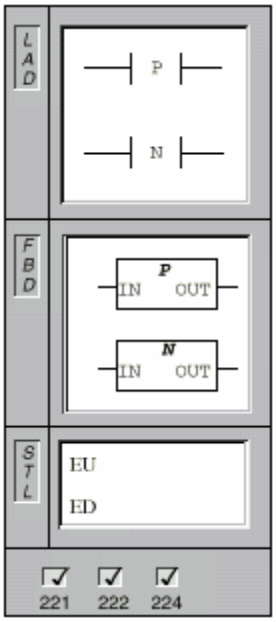
Trong LAD, lệnh này được biểu diễn như một công tắc.

Trong FBD, lệnh đảo không có biểu tượng riêng. Nó được tích hợp như là đầu vào đảo của những khối chức năng khác (với vòng tròn nhỏ ở đầu vào của các khối chức năng đó).

Trong STL, lệnh đảo đảo giá trị của đỉnh ngăn xếp: 0 thành 1 và 1 thành 0.

Lệnh này không có toán hạng.

Sườn dương và sườn âm:



Các lệnh trên đây đều thuộc nhóm lệnh các công tắc, ghi nhận trạng thái các bit dữ liệu (0 hay 1), quen thuộc với khái niệm "mức". Các lệnh về sườn ghi nhận không phải mức đơn thuần mà là sự biến đổi mức.

Lệnh sườn dương (Positive Transition) cho dòng năng lượng đi qua trong khoảng thời gian bằng thời gian một vòng quét khi ở đầu vào của nó có sự thay đổi mức từ 0 lên 1.

Lệnh sườn âm (Negative Transition) cho dòng năng lượng đi qua trong khoảng thời gian bằng thời gian một vòng quét khi ở đầu vào của nó có sự thay đổi mức từ 1 xuống 0.

Trong LAD, các lệnh này được biểu diễn cũng như các công tắc.

Trong FBD, các lệnh này được biểu diễn bằng các khối chức năng P và N.

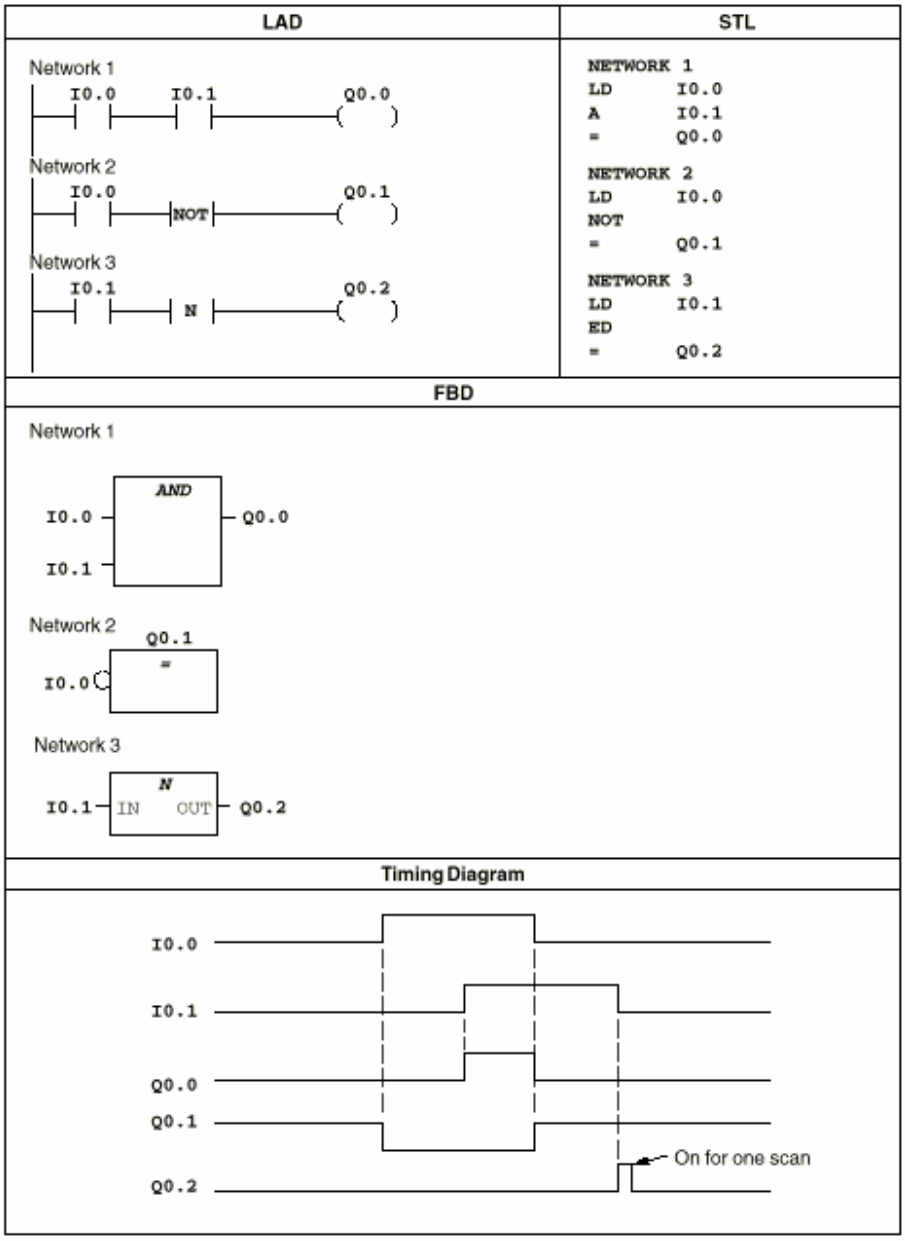
Trong STL, lệnh Edge Up, nếu phát hiện có sự thay đổi mức của đỉnh ngăn xếp từ 0 lên 1, sẽ đặt vào đỉnh ngăn xếp giá

trị 1. Trong trường hợp ngược lại, nó đặt vào đó giá trị 0. Tương tự, lệnh Edge Down, nếu phát hiện có sự thay đổi mức của đỉnh ngăn xếp từ 1 xuống 0, sẽ đặt vào đỉnh ngăn xếp giá trị 1. Trong trường hợp ngược lại, nó cũng đặt vào đó giá trị 0.

Chú ý: Theo cấu trúc hoạt động của PLC, sự thay đổi mức tất nhiên chỉ được phát hiện giữa các vòng quét liên tiếp. Do đó mỗi lệnh sườn này cần một bit nhớ để nhớ trạng thái đầu vào của nó ở vòng quét kế trước. Vì đặc tính này mà tổng số lệnh sườn được sử dụng trong một chương trình bị hạn chế (do dung lượng bộ nhớ dành cho chúng có hạn). Ví dụ trong một chương trình với CPU 212 chỉ có thể được sử dụng tối đa 128 lệnh sườn. Con số giới hạn này đối với CPU 214 là 256.

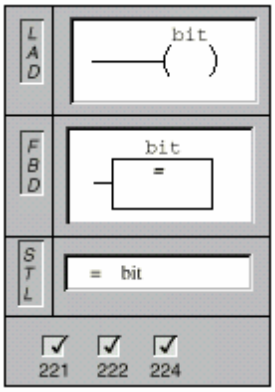
Inputs/Outputs	Operands	Data Types
IN (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL
OUT (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL

Sau đây là một vài ví dụ đơn giản về cách sử dụng các lệnh công tắc trên:



<u>Lệnh ra</u>:

Giá trị bit được định địa chỉ bởi toán hạng của lệnh ra phản ảnh trạng thái của dòng năng lượng (Power Flow) ở đầu vào lệnh này.

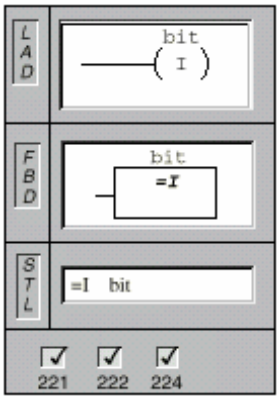


Trong LAD và FBD, lệnh ra đặt giá trị bit được trỏ đến bởi toán hạng của nó bằng giá trị dòng năng lượng ở đầu vào của lệnh.

Trong STL, lệnh ra sao chép giá trị đỉnh ngăn xếp ra giá trị bit được trỏ đến bởi toán hạng của lệnh.

Inputs/Outputs	Operands	Data Types
bit	I, Q, M, SM, T, C, V, S, L	BOOL
Input (LAD)	Power Flow	BOOL
Input (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL

<u>Lệnh ra trực tiếp</u>:

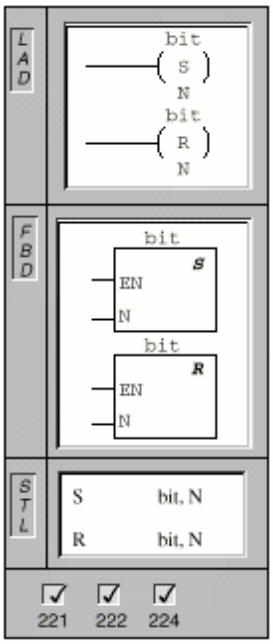


Giá trị đầu ra rời rạc (digital) vật lý được định địa chỉ bởi toán hạng của lệnh ra trực tiếp phản ảnh trạng thái của dòng năng lượng (Power Flow) ở đầu vào lệnh này.

Trong LAD và FBD, lệnh ra trực tiếp đặt đồng thời giá trị đầu ra vật lý được trỏ đến bởi toán hạng của nó và bit ảnh của đầu ra này bằng giá trị dòng năng lượng ở đầu vào của lệnh. Điều đó khác với lệnh ra thông thường ở chỗ lệnh ra thông thường chỉ ghi giá trị vào bit ảnh của đầu ra.

Trong STL, lệnh ra trực tiếp sao chép giá trị đỉnh ngăn xếp ra đồng thời giá trị đầu ra vật lý được trỏ đến bởi toán hạng của lệnh và bit ảnh của đầu ra này.

Inputs/Outputs	Operands	Data Types
bit	Q	BOOL
Input (LAD)	Power Flow	BOOL
Input (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL



Các lệnh ghi xóa giá trị tiếp điểm:

Các lệnh SET và RESET đặt một số các bit liên tiếp trong bộ nhớ dữ liệu thành 1 (Set) hay 0 (Reset). Số lượng các bit được định bởi toán hạng [N] và bắt đầu từ bit được định địa chỉ bởi toán hạng [bit].

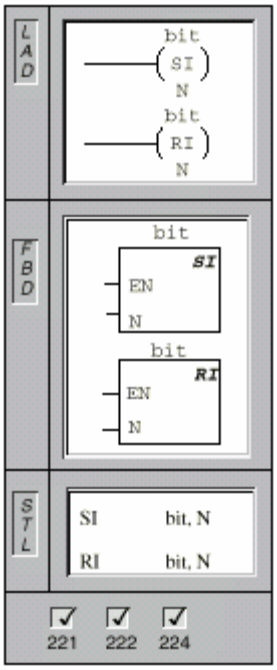
Số lượng các bit có thể Set hoặc Reset nằm trong khoảng từ 1 đến 255.

Trong trường hợp sử dụng lệnh Reset với các bit nằm trong những vùng T hay C, các bộ định thời hay bộ đếm tương ứng sẽ bị reset. Nghĩa là bit trạng thái của chúng được đưa về 0 và số đang đếm cũng bị xóa (sẽ có giá trị 0).

Những lỗi có thể được gây nên bởi các lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Lỗi 0091: toán hạng vượt quá giới hạn cho phép.

Inputs/Outputs	Operands	Data Types
bit	I, Q, M, SM, T, C, V, S, L	BOOL
N	VB, IB, QB, MB, SMB, SB, LB, AC, Constant,*VD, *AC, *LD	BYTE



Các lệnh ghi xóa giá trị tiếp điểm trực tiếp:

Các lệnh SET IMMEDIATE và RESET IMMEDIATE đặt một số các đầu ra rời rạc (digital) vật lý liên tiếp thành 1 (Set) hay 0 (Reset). Số lượng các đầu ra được định bởi toán hạng [N] và bắt đầu từ đầu ra được định địa chỉ bởi toán hạng [bit].

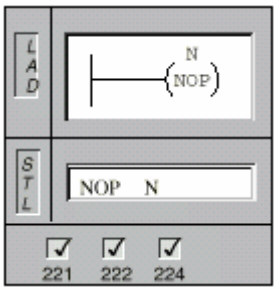
Số lượng các đầu ra vật lý có thể Set hoặc Reset nằm trong khoảng từ 1 đến 128.

Ký tự "I" trong những lệnh này (Immediate) nói lên tính tức thời. Các lệnh này ghi giá trị mới ra các đầu ra vật lý đồng thời ghi cả vào các giá trị ảnh của chúng. Điều đó khác với những lệnh Set và Reset thông thường chỉ ghi giá trị mới vào vùng ảnh của các đầu ra.

Những lỗi có thể được gây nên bởi các lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Lỗi 0091: toán hạng vượt quá giới hạn cho phép.

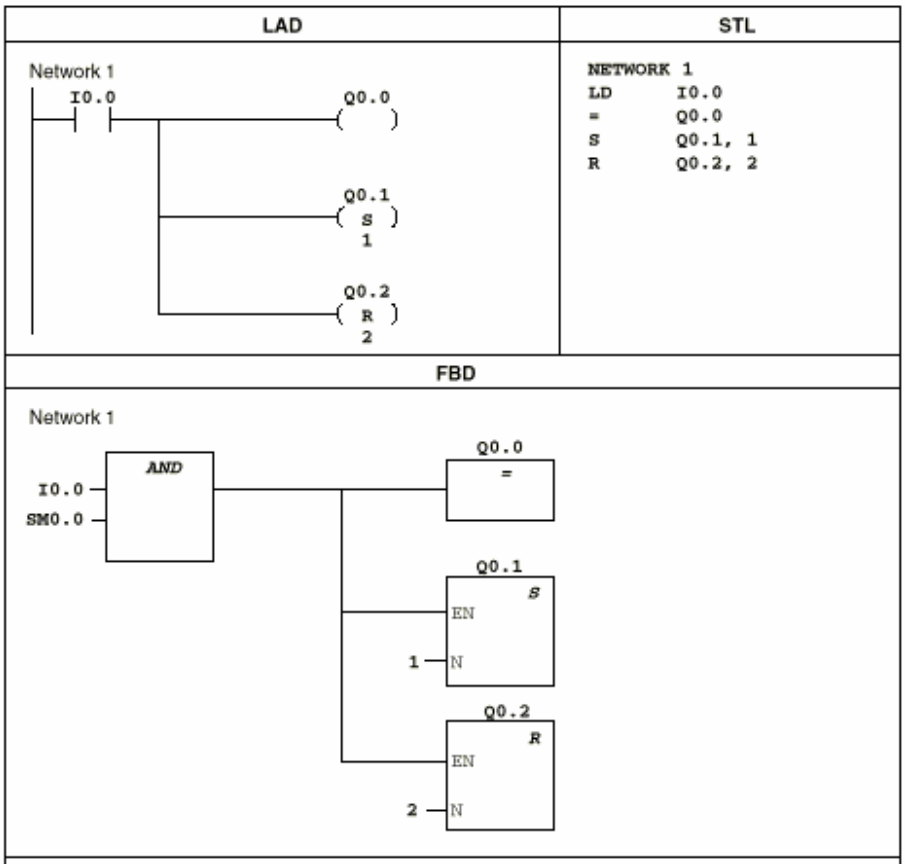
Inputs/Outputs	Operands	Data Types
bit	Q	BOOL
N	VB, IB, QB, MB, SMB, SB, LB, AC, Constant,*VD, *AC, *LD	BYTE

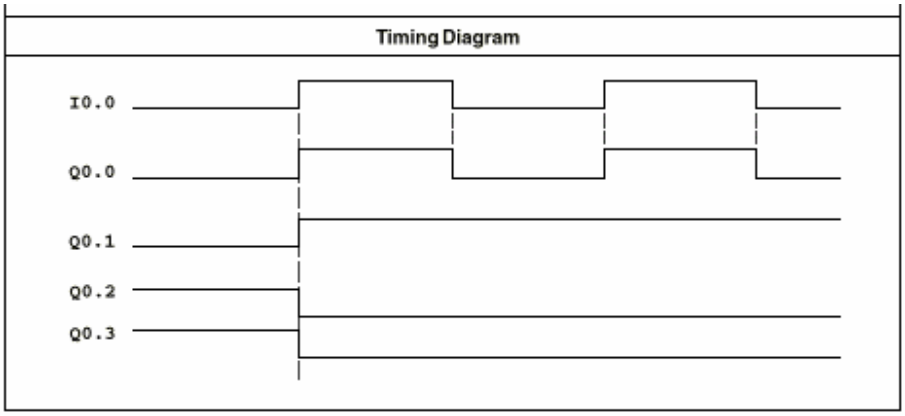


<u>Lệnh không làm gì cả</u>:

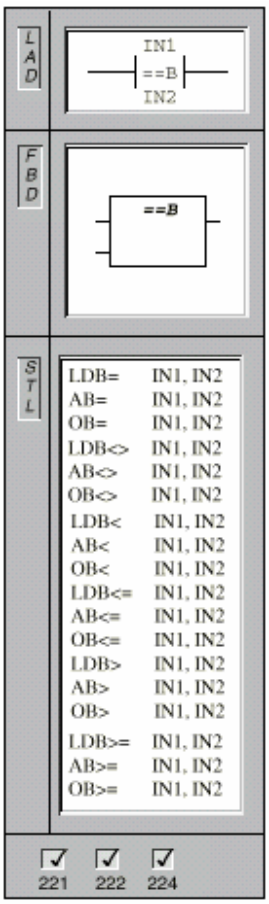
Lệnh không làm gì (No Operation) không tác động đến chương trình. Mặc dù nó cũng có một toán hạng [N] dạng Byte, là một hằng số trong khoảng từ 1 đến 255.

Một số ví dụ về các lệnh ra:





8.2 Các lệnh so sánh



<u>So sánh Byte</u>:

Lệnh so sánh Byte dùng để so sánh 02 giá trị dạng byte được định địa chỉ bởi hai toán hạng ở đầu vào của lệnh: [IN1] và [IN2]. Có tất cả 06 phép so sánh có thể được thực hiện: [IN1] = [IN2], [IN1] >= [IN2], [IN1] <= [IN2], [IN1] > [IN2], [IN1] < [IN2], [IN1] <> [IN2].

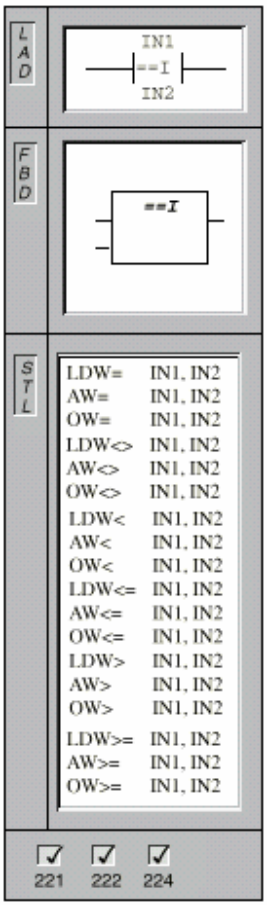
Các byte được đem so sánh là những giá trị không dấu.

Trong LAD, lệnh này có dạng một công tắc và công tắc đó đóng (ON) khi điều kiện đem so sánh có giá trị đúng.

Trong FBD, đầu ra sẽ có giá trị 1 nếu điều kiện đem so sánh là đúng.

Trong STL, lệnh được thực hiện sẽ ghi giá trị 1 vào đỉnh ngăn xếp (với những lệnh Load) hoặc thực hiện phép toán lô gic AND hay OR (tùy theo lệnh cụ thể) giá trị 1 với đỉnh ngăn xếp nếu điều kiện so sánh đúng.

Inputs/Outputs	Operands	Data Types
Inputs	IB, QB, MB, SMB, VB, SB, LB, AC, Constant, *VD, *AC,*LD	BYTE
Outputs (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL



<u>So sánh số nguyên (Integer)</u>:

Lệnh so sánh số nguyên dùng để so sánh 02 giá trị dạng Integer được định địa chỉ bởi hai toán hạng ở đầu vào của lệnh: [IN1] và [IN2]. Có tất cả 06 phép so sánh có thể được thực hiện: [IN1] = [IN2], [IN1] >= [IN2], [IN1] <= [IN2], [IN1] > [IN2], [IN1] < [IN2], [IN1] <> [IN2].

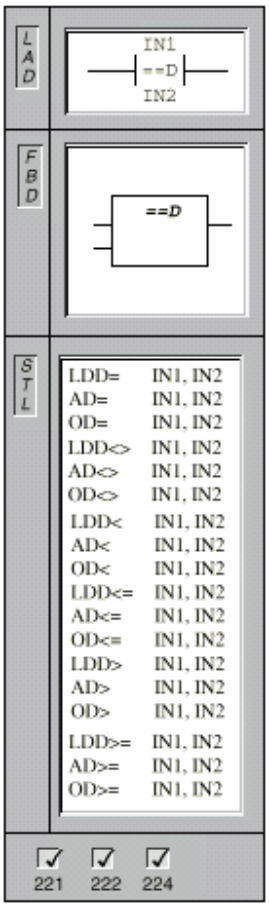
Các số nguyên được đem so sánh là những giá trị có dấu: 16#7FFF > 16#8000.

Trong LAD, lệnh này có dạng một công tắc và công tắc đó đóng (ON) khi điều kiện đem so sánh có giá trị đúng.

Trong FBD, đầu ra sẽ có giá trị 1 nếu điều kiện đem so sánh là đúng.

Trong STL, lệnh được thực hiện sẽ ghi giá trị 1 vào đỉnh ngăn xếp (với những lệnh Load) hoặc thực hiện phép toán lô gic AND hay OR (tùy theo lệnh cụ thể) giá trị 1 với đỉnh ngăn xếp nếu điều kiện so sánh đúng.

Inputs/Outputs	Operands	Data Types
Inputs	IW, QW, MW, SW, SMW, T, C, VW, LW, AIW, AC, Constant, *VD, *AC,*LD	INT
Outputs (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL



So sánh từ kép (Double Word):

Lệnh so sánh từ kép dùng để so sánh 02 giá trị dạng Double Word được định địa chỉ bởi hai toán hạng ở đầu vào của lệnh: [IN1] và [IN2]. Có tất cả 06 phép so sánh có thể được thực hiện: [IN1] = [IN2], [IN1] >= [IN2], [IN1] <= [IN2], [IN1] > [IN2], [IN1] < [IN2], [IN1] <> [IN2].

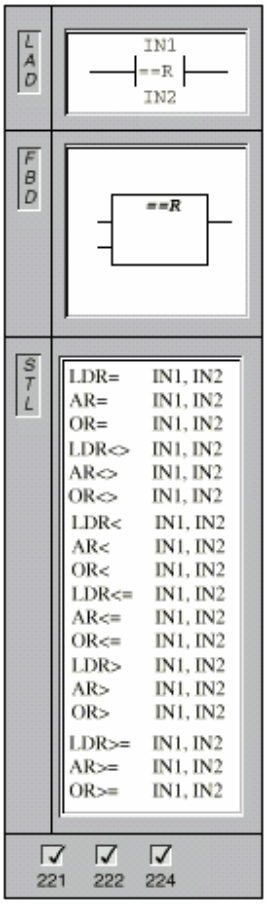
Các giá trị từ kép được đem so sánh là những giá trị có dấu: 16#7FFFFFFF > 16#80000000.

Trong LAD, lệnh này có dạng một công tắc và công tắc đó đóng (ON) khi điều kiện đem so sánh có giá trị đúng.

Trong FBD, đầu ra sẽ có giá trị 1 nếu điều kiện đem so sánh là đúng.

Trong STL, lệnh được thực hiện sẽ ghi giá trị 1 vào đỉnh ngăn xếp (với những lệnh Load) hoặc thực hiện phép toán lô gic AND hay OR (tùy theo lệnh cụ thể) giá trị 1 với đỉnh ngăn xếp nếu điều kiện so sánh đúng.

Inputs/Outputs	Operands	Data Types
Inputs	ID, QD, MD, SD, SMD, VD, LD, HC, AC, Constant, *VD, *AC, *LD	DINT
Outputs (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL



So sánh số thực (Real):

Lệnh so sánh số thực dùng để so sánh 02 giá trị dạng Real được định địa chỉ bởi hai toán hạng ở đầu vào của lệnh: [IN1] và [IN2]. Có tất cả 06 phép so sánh có thể được thực hiện: [IN1] = [IN2], [IN1] >= [IN2], [IN1] <= [IN2], [IN1] > [IN2], [IN1] < [IN2], [IN1] <> [IN2].

Các số thực được đem so sánh là những giá trị có dấu theo kiểu dấu phẩy động.

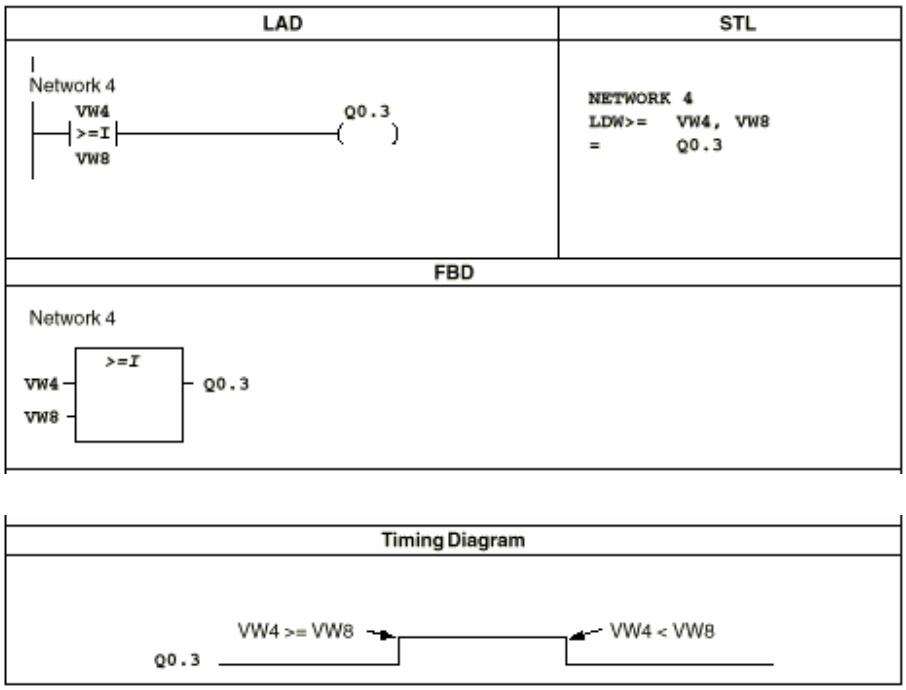
Trong LAD, lệnh này có dạng một công tắc và công tắc đó đóng (ON) khi điều kiện đem so sánh có giá trị đúng.

Trong FBD, đầu ra sẽ có giá trị 1 nếu điều kiện đem so sánh là đúng.

Trong STL, lệnh được thực hiện sẽ ghi giá trị 1 vào đỉnh ngăn xếp (với những lệnh Load) hoặc thực hiện phép toán lô gic AND hay OR (tùy theo lệnh cụ thể) giá trị 1 với đỉnh ngăn xếp nếu điều kiện so sánh đúng.

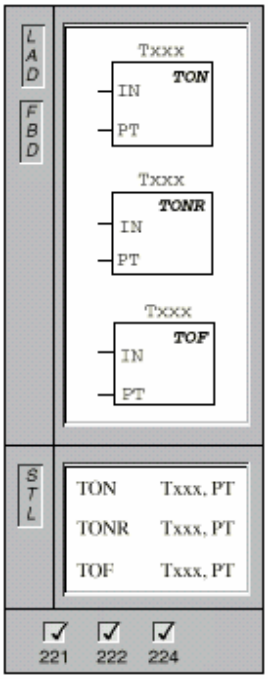
Inputs/Outputs	Operands	Data Types
Inputs	ID, QD, MD,SD, SMD, VD, LD, AC, Constant, *VD, *AC, *LD	REAL
Outputs (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL

Ví dụ sử dụng lệnh so sánh:



8.3 Các lệnh làm việc với các bộ định thời

SIMATIC S7-200 có 03 loại bộ định thời:



- Bộ đóng trễ (On - Delay Timer).
- Bộ đóng trễ có nhớ (Retentive On - Delay Timer).
- Bộ ngắt trễ (Off - Delay Timer).

loại thứ ba không có trong các CPU 212 và 214.

Các bộ đóng trễ và đóng trễ có nhớ bắt đầu đếm thời gian khi có đầu vào EN (Enable) ở mức 1 (ON). Lúc giá trị đếm được lớn hơn hoặc bằng giá trị đặt trước tại đầu vào PT (Preset Time) thì bit trạng thái sẽ được đặt bằng 1 (ON). Điều khác nhau giữa hai loại bộ đóng trễ này là: bộ đóng trễ bình thường sẽ bị reset (cả giá trị đang đếm lẫn bit trạng thái đều bị xóa về 0) khi đầu vào EN bằng 0; trong khi đó bộ định thời có nhớ lưu lại giá trị của nó khi đầu vào EN bằng 0 và tiếp tục đếm nếu đầu vào EN lại bằng 1. Như vậy ta có thể dùng loại có nhớ để cộng thời gian những lúc đầu vào EN bằng 1. Loại bộ định thời này có thể reset (xóa giá trị đang đếm về 0) bằng lệnh R (Reset).

Cả hai loại bộ đóng trễ vẫn tiếp tục đếm thời gian ngay cả sau khi đã đạt đến giá trị đặt trước PT và chỉ dừng đếm khi

đạt giá trị tối đa 32767 (16#7FFF).

Bộ ngắt trễ dùng để đưa giá trị đầu ra (bit trạng thái) về 0 (OFF) trễ một khoảng thời gian sau khi đầu vào (EN) đổi về 0. Khi đầu vào EN được đặt bằng 1 (ON) thì bit trạng thái của bộ ngắt trễ cũng bằng 1 ngay lúc đó đồng thời giá trị đếm của nó bị xóa về 0. Khi đầu vào EN về 0, bộ định thời bắt đầu đếm và đếm cho đến khi đạt giá trị đặt trước PT. Lúc đó bit trạng thái của bộ ngắt trễ sẽ về 0 đồng thời nó cũng ngừng đếm.

Nếu đầu vào EN chỉ bằng 0 trong khoảng thời gian ngắn hơn thời gian được đặt rồi quay lại bằng 1 thì bit trạng thái của bộ định thời vẫn giữ nguyên bằng 1. Bộ ngắt trễ chỉ bắt đầu đếm khi có sườn thay đổi từ 1 thành 0 ở đầu vào EN.

Nếu bộ ngắt trễ ở trong vùng một SCR (Sequence Control Relay) và vùng SCR đó không được kích hoạt thì giá trị đếm của nó được xóa về 0, bit trạng thái cũng bằng 0 (OFF) và bộ định thời không đếm. Khái niệm vùng SCR sẽ được định nghĩa ở phần sau của tài liệu này (xem phần 8.10).

Inputs/Outputs	Operands	Data Types
IN (LAD)	Power Flow	BOOL
IN (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL
PT	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, Constant, *VD, *AC, *LD	INT

Trên đây chúng ta thường nói đến giá trị đang đếm của các bộ định thời, mà các bộ định thời thì lại đếm thời gian. Thực tế, thời gian trễ được tính như là tích của số đang đếm với một hằng số thời gian (base time), hằng số thời gian này còn được gọi là độ phân giải của của bộ định thời.

Các bộ định thời trong S7-200 (đóng trễ, đóng trễ có nhớ và ngắt trễ) bao gồm 03 nhóm với 03 độ phân giải khác nhau: 1ms, 10 ms và 100 ms. Mỗi bộ định thời (được định địa chỉ trong vùng T) có một độ phân giải xác định theo bảng sau:

Timer Type	Resolution in milliseconds (ms)	Maximum Value in seconds (s)	Timer Number
TONR	1 ms	32.767 s	T0, T64
	10 ms	327.67 s	T1 to T4, T65 to T68
	100 ms	3276.7 s	T5 to T31, T69 to T95
TON, TOF	1 ms	32.767 s	T32, T96
	10 ms	327.67 s	T33 to T36, T97 to T100
	100 ms	3276.7 s	T37 to T63, T101 to T255

Chú ý những bộ định thời có nhớ có địa chỉ được qui định riêng. Những bộ định thời còn lại (không nhớ) có thể được khai báo như là bộ đóng trễ hoặc ngắt trễ, nhưng không thể là cả hai. Nghĩa là không thể có, ví dụ TON 33 và TOF 33 đồng thời.

Bảng sau tóm tắt những đặc điểm hoạt động của ba loại bộ định thời nêu trên:

Timer Type	Current >= Preset	Enabling Input ON	Enabling Input OFF	Power Cycle/ First Scan
TON	Timer bit ON, Current continues counting to 32,767	Current value counts time	Timer bit OFF, Current value = 0	Timer bit OFF, Current value = 0
TONR	Timer bit ON, Current continues counting to 32,767	Current value counts time	Timer bit and current value maintain last state	Timer bit OFF, Current value may be maintained[1]
TOF	Timer bit OFF, Current = Preset, stops counting	Timer bit ON, Current value = 0	Timer counts after ON to OFF transition	Timer bit OFF, Current value = 0

[1] The retentive timer current value can be selected for retention through a power cycle. See Section 5.3 for information about memory retention for the S7-200 CPU.

Lệnh Reset (R) có thể được sử dụng để reset bất kỳ bộ định thời nào. Các bộ định thời có nhớ (loại TONR) chỉ có thể reset bằng lệnh này. Các bộ định thời sau khi reset có bit trạng thái cũng như giá trị đếm đều được xóa về 0. Các bộ ngắt trễ (TOF) chỉ bắt đầu đếm khi có sự thay đổi từ 1 xuống 0 ở đầu vào IN.

Các bộ định thời có độ phân giải khác nhau có cách hoạt động cũng khác nhau. Chúng ta xem xét kỹ hơn về vấn đề này:

Bộ định thời với độ phân giải 1 ms

Bộ định thời loại này đếm số khoảng thời gian 1 ms trôi qua kể từ khi nó được kích hoạt. Bộ định thời với độ phân giải 1 ms được kích hoạt bằng lệnh khai báo của nó nhưng sau đó nó được cập nhật (bit trạng thái cũng như giá trị đếm) mỗi giây một lần một cách độc lập không phụ thuộc vào vòng quét chương trình. Nói một cách khác, một bộ định thời loại này có thể được cập nhật nhiều lần trong một vòng quét nếu như thời gian vòng quét lớn hơn 1 ms.

Bởi vì một bộ định thời với độ phân giải 1 ms có thể được kích hoạt ở bất kỳ một thời điểm nào trong vòng 1 ms nên ta nên đặt giá trị đặt trước lớn hơn 1 đơn vị so với giá trị yêu cầu cần đếm. Ví dụ để đếm khoảng thời gian 56 ms, ta thường đặt giá trị đặt trước bằng 57.

Bộ định thời với độ phân giải 10 ms

Bộ định thời loại này đếm số khoảng thời gian 10 ms trôi qua kể từ khi nó được kích hoạt. Bộ định thời với độ phân giải 10 ms được kích hoạt bằng lệnh khai báo của nó và sau đó nó được cập nhật (bit trạng thái cũng như giá trị đếm) mỗi vòng quét một lần ở ngay đầu mỗi vòng quét bằng cách cộng vào giá trị đang đếm của nó số khoảng thời gian 10 ms trôi qua kể từ đầu vòng quét trước. Nói một cách khác, giá trị đang đếm của bộ định thời loại này giữ nguyên không đổi trong suốt thời gian một vòng quét.

Bởi vì một bộ định thời với độ phân giải 10 ms có thể được kích hoạt ở bất kỳ một thời điểm nào trong vòng 10 ms nên ta nên đặt giá trị đặt trước lớn hơn 1 đơn vị so với giá trị yêu cầu cần đếm. Ví dụ để đếm khoảng thời gian 140 ms, ta thường đặt giá trị đặt trước bằng 15.

Bộ định thời với độ phân giải 100 ms

Bộ định thời loại này tính số khoảng thời gian 100 ms trôi qua kể từ khi nó được cập nhật lần cuối. Lệnh khai báo bộ định thời với độ phân giải 100 ms cập nhật bit trạng thái cũng như giá trị đếm của nó bằng cách cộng vào giá trị đang đếm của nó số khoảng thời gian 100 ms trôi qua kể từ vòng quét trước.

Như vậy, giá trị đang đếm của bộ định thời loại này chỉ được cập nhật khi có lệnh khai báo nó thực hiện. Vì thế nếu bộ định thời với độ phân giải 100 ms đã được kích hoạt nhưng lệnh khai báo nó không được thực hiện trong mỗi vòng quét thì nó có thể không được cập nhật kịp thời và đếm thiếu thời gian. Ngược lại nếu lệnh khai báo bộ định thời được thực hiện nhiều lần trong một vòng quét thì nó có thể đếm dư thời gian do một số khoảng thời gian 100 ms được cộng nhiều lần. Tóm lại nên sử dụng bộ định thời loại này với lệnh khai báo thực hiện chính xác mỗi vòng quét một lần.

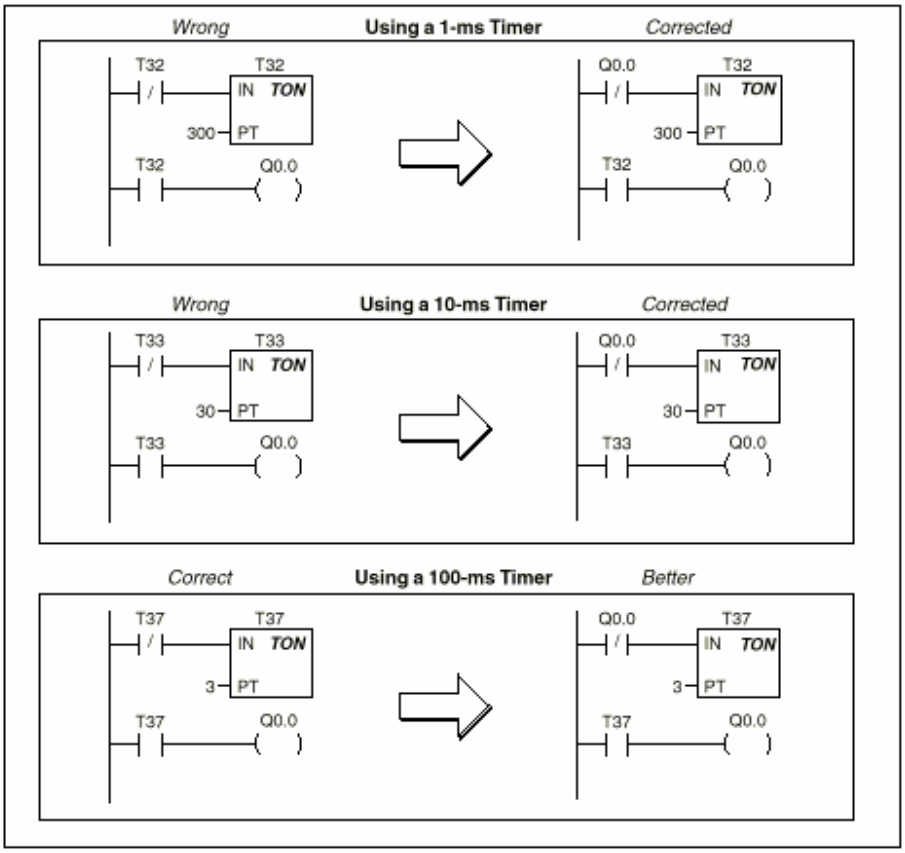
Bởi vì một bộ định thời với độ phân giải 100 ms có thể được khởi động ở bất kỳ một thời điểm nào trong vòng 100 ms nên ta nên đặt giá trị đặt trước lớn hơn 1 đơn vị so với giá trị yêu cầu cần đếm. Ví dụ để đếm khoảng thời gian 2100 ms, ta thường đặt giá trị đặt trước bằng 22.

Để hiểu thêm về cơ chế cập nhật của các bộ định thời với những độ phân giải khác nhau, chúng ta xem xét ví dụ sau:

Tạo bộ định thời 3 giây với lần lượt ba bộ định thời khác nhau (xem chương trình kèm theo):

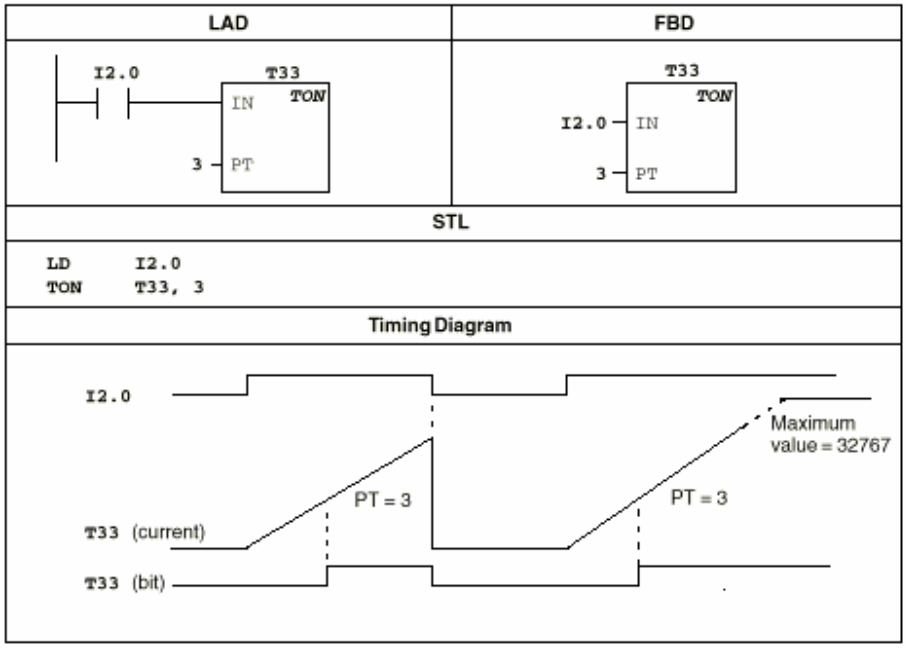
- Đầu tiên bộ định thời với độ phân giải 1 ms được sử dụng (T32, giá trị đặt trước 300). Q0.0 sẽ có giá trị bằng 1 (ON) trong thời gian một vòng quét khi và chỉ khi nào thời điểm cập nhật của bộ định thời mà giá trị đếm vượt qua giá trị đặt trước rơi vào đúng giữa lúc thực hiện hai lệnh này. Nghĩa là sau khi lệnh trước được thực hiện nhưng phải trước khi thực hiện lệnh sau.
- Nếu sử dụng bộ định thời với độ phân giải 10 ms (T33, giá trị đặt trước 30), Q0.0 không bao giờ có giá trị 1 (luôn luôn OFF).
- Trường hợp cuối cùng sử dụng bộ định thời với độ phân giải 100 ms (T37, giá trị đặt trước bằng 3). Q0.0 luôn luôn có giá trị bằng 1 (ON) trong đúng thời gian một vòng quét.

Để đảm bảo chắc chắn Q0.0 sẽ có giá trị 1 (ON) trong thời gian một vòng quét, ta phải dùng công tắc thường đóng Q0.0 để kích hoạt các bộ định thời thay vì dùng công tắc thường đóng với bit trạng thái của nó.

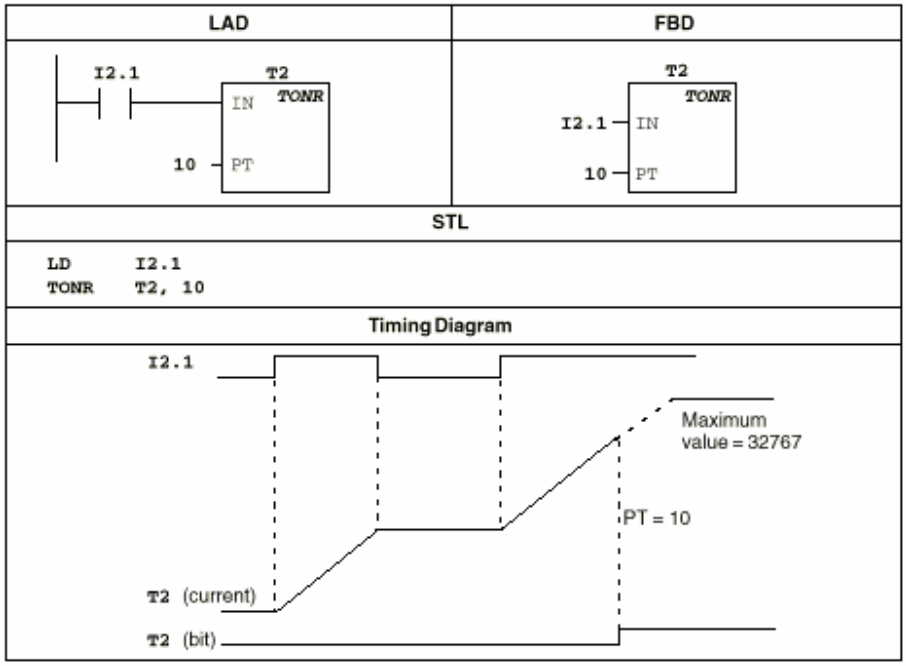


Sau đây là những ví dụ về các loại bộ định thời:

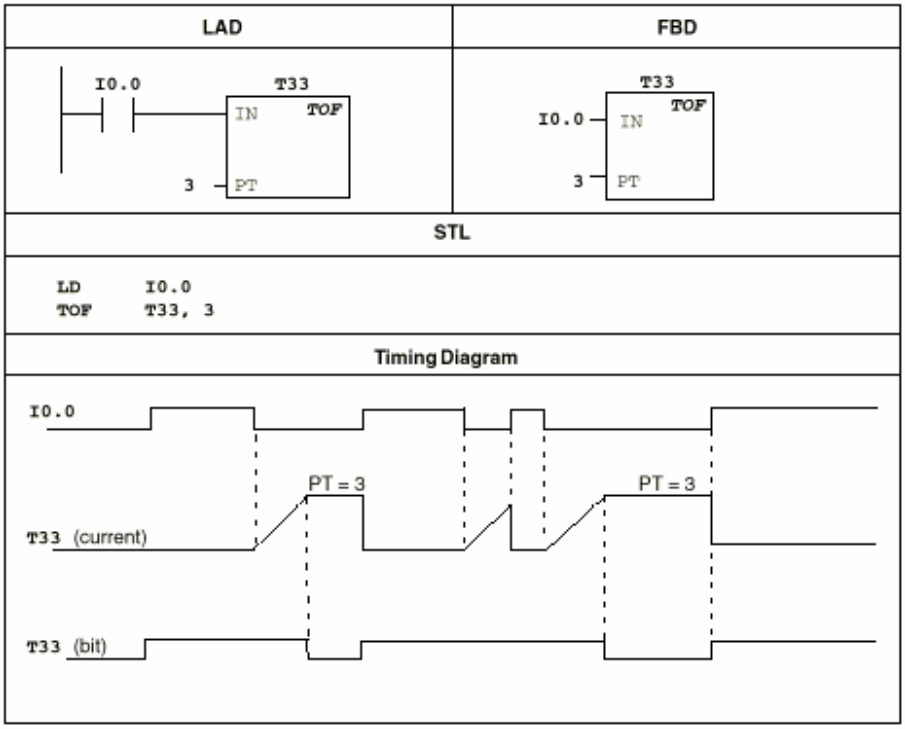
On-Delay Timer Example



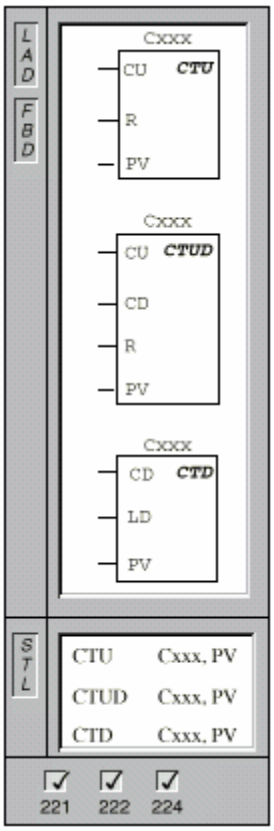
Retentive On-Delay Timer Example



Off-Delay Timer Example



8.4 Các lệnh làm việc với các bộ đếm



S7-200 có ba loại bộ đếm: bộ đếm lên (Count Up), bộ đếm xuống (Count Down) và loại bộ đếm có thể vừa đếm lên vừa đếm xuống (Count Up / Down).

Bộ đếm lên đếm cho đến giá trị tối đa của nó (32767) mỗi khi có sườn lên ở đầu vào đếm lên (CU). Khi giá trị đếm (Cxxx) lớn hơn hoặc bằng giá trị đặt trước (PV) thì bit trạng thái (Cxxx) sẽ có giá trị 1 (ON). Bộ đếm có thể bị xóa (reset) bởi mức 1 ở đầu vào reset (R), lúc đó cả giá trị đếm lẫn bit trạng thái sẽ bị xóa về 0.

Bộ đếm xuống đếm từ giá trị đặt trước (PV) mỗi khi có sườn lên ở đầu vào đếm xuống (CD). Khi giá trị đếm (Cxxx) bằng 0, bit trạng thái (Cxxx) sẽ bằng 1 đồng thời bộ đếm ngừng đếm. Mức cao ở đầu vào LD xóa bit trạng thái về 0 và tải giá trị đặt trước PV vào giá trị đếm.

Bộ đếm vừa đếm lên vừa đếm xuống đếm lên khi có sườn lên ở đầu vào đếm lên (CU) và đếm xuống khi có sườn lên ở đầu vào đếm xuống (CD). Khi giá trị đếm (Cxxx) lớn hơn hoặc bằng giá trị đặt trước (PV) thì bit trạng thái (Cxxx) sẽ có giá trị 1 (ON). Bộ đếm có thể bị xóa (reset) bởi mức 1 ở đầu vào reset (R), lúc đó cả giá trị đếm lẫn bit trạng thái sẽ bị xóa về 0.

Số các bộ đếm có trong S7-200: C0 đến C255. Chú ý CPU 212 chỉ có 64 (C0 - C63), CPU 214 có 128 (C0 - C127) và mỗi bộ đếm đã được xác định cố định là bộ đếm tiến hay bộ đếm có thể vừa đếm tiến vừa đếm lùi (không có bộ đếm lùi). Trong CPU 221, 222 và 224 mỗi bộ đếm được xác định loại tùy theo lệnh khai báo nhưng không thể khai báo các bộ đếm loại khác nhau với cùng một địa chỉ (trong vùng C).

Trong STL, đầu vào reset (R) của bộ đếm tiến là bit đỉnh của ngăn xếp và đầu vào đếm của nó (CU) là bit thứ hai của ngăn xếp.

Trong STL, đầu vào tải (LD) của bộ đếm lùi là bit đỉnh của ngăn xếp và đầu vào đếm của nó (CD) là bit thứ hai của ngăn xếp.

Trong STL, đầu vào reset (R) của bộ đếm vừa đếm tiến vừa đếm lùi là bit đỉnh của ngăn xếp, đầu vào đếm lùi của nó (CD) là bit thứ hai của ngăn xếp và đầu vào đếm tiến của nó (CU) là bit thứ ba của ngăn xếp.

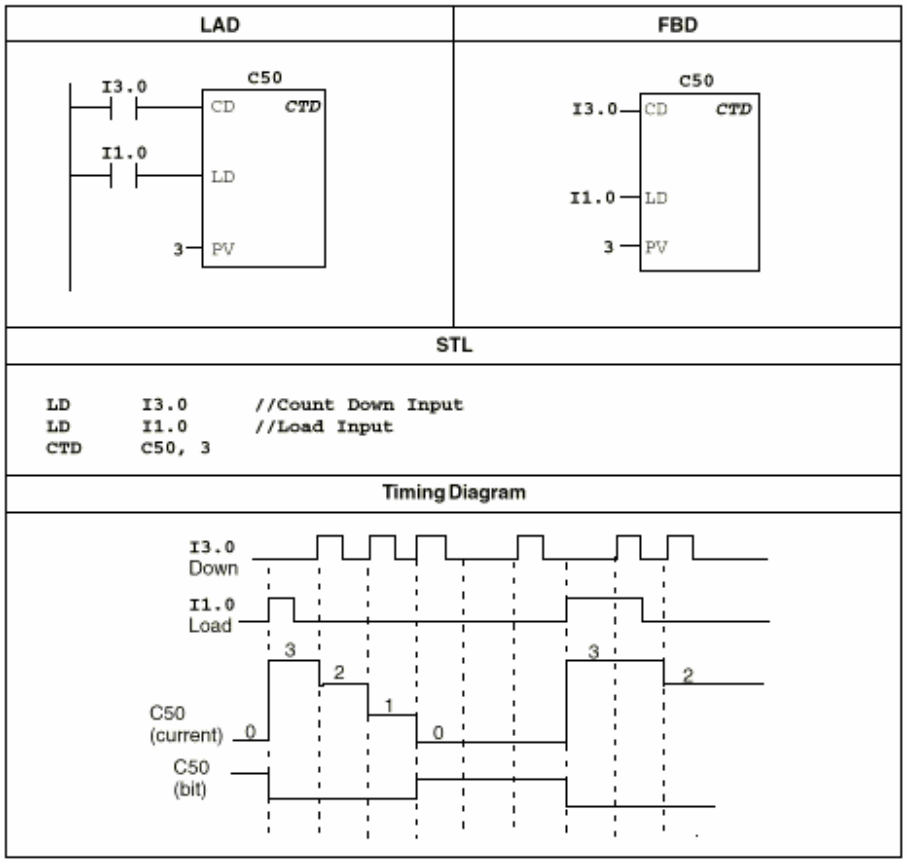
Inputs/Outputs	Operands	Data Types
CU, CD (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL
R, LD (FBD)	I, Q, M, SM, T, C, V, S, L, Power Flow	BOOL
PV	VW, IW, QW, MW, SMW, LW, AIW, AC, T, C, Constant, *VD, *AC, *LD, SW	INT

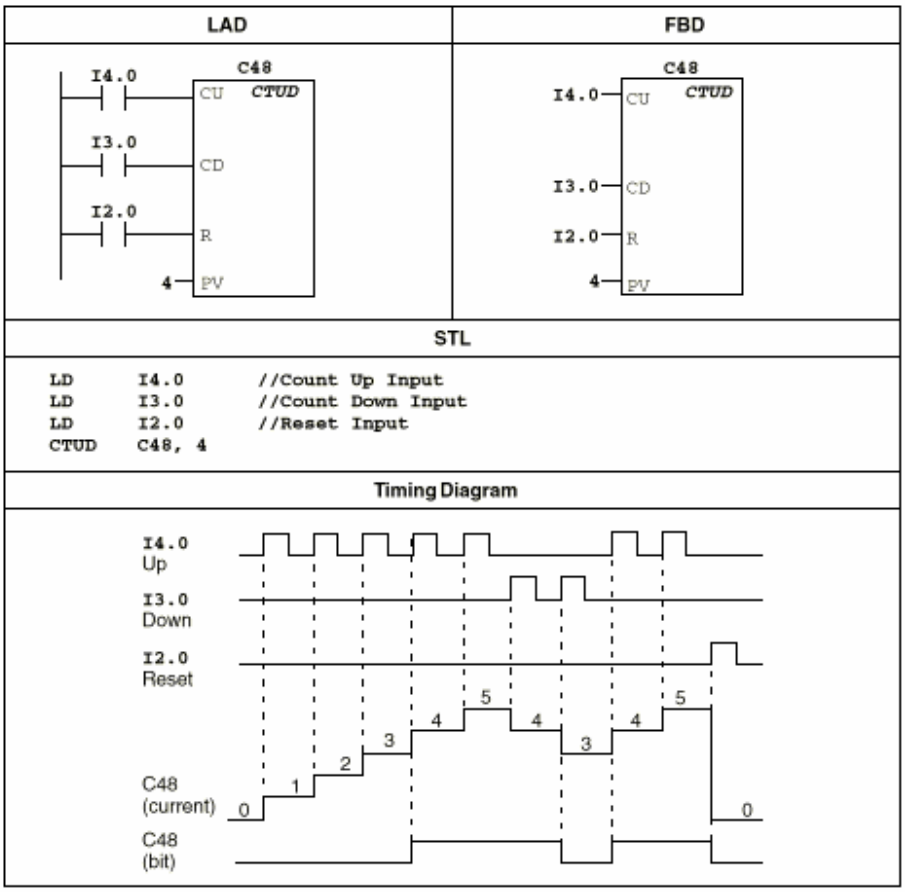
Các bộ đếm còn có thể bị reset bởi lệnh Reset.

Bộ đếm vừa tiến vừa lùi khi đếm đến giá trị tối đa (32767) mà tiếp tục đếm lên thi số đếm sẽ nhảy sang giá trị tối thiểu (-32768) và tiếp tục đếm bình thường. Tương tự, nếu nó đếm lùi khi đã ở giá trị nhỏ nhất (-32768) thì số đếm sẽ nhảy thành giá trị lớn nhất (32767).

Sau đây là những ví dụ sử dụng bộ đếm:

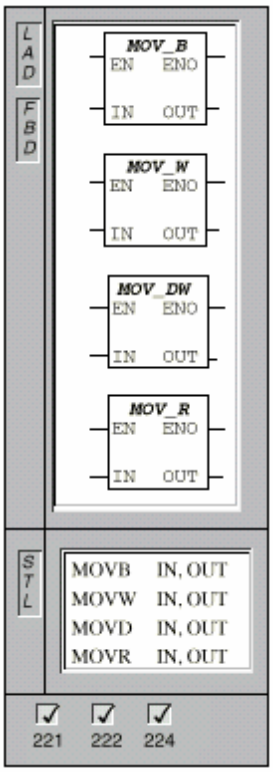
Counter Examples





8.5 Các lệnh dịch chuyển ô nhớ

Các lệnh dịch chuyển một Byte, một từ đơn (Word), một từ kép (Double Word) hay một số thực (Real):



Lệnh dịch chuyển một Byte, Move Byte, sao chép nội dung ô nhớ kích thước một byte được định địa chỉ ở đầu vào IN lên ô nhớ kích thước một byte được định địa chỉ ở đầu ra OUT. Nội dung byte nhớ ở địa chỉ [IN] không thay đổi.

Lệnh dịch chuyển một Từ đơn, Move Word, sao chép nội dung ô nhớ kích thước một word được định địa chỉ ở đầu vào IN lên ô nhớ kích thước một word được định địa chỉ ở đầu ra OUT. Nội dung từ đơn ở địa chỉ [IN] không thay đổi.

Lệnh dịch chuyển một Từ kép, Move Double Word, sao chép nội dung ô nhớ kích thước một từ kép được định địa chỉ ở đầu vào IN lên ô nhớ kích thước một từ kép được định địa chỉ ở đầu ra OUT. Nội dung từ kép ở địa chỉ [IN] không thay đổi.

Lệnh dịch chuyển một Số thực, Move Real, sao chép số thực kích thước 32 bit được định địa chỉ ở đầu vào IN lên số thực kích thước 32 bit được định địa chỉ ở đầu ra OUT. Số thực ở địa chỉ [IN] không thay đổi.

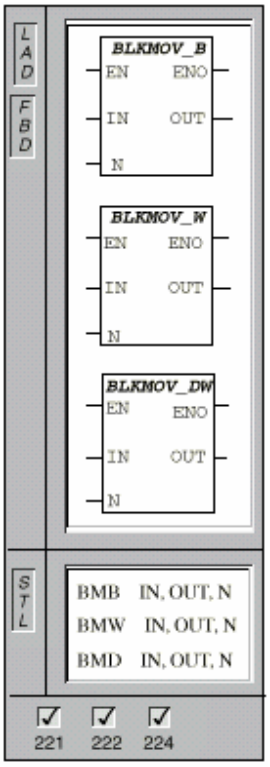
Những lỗi có thể được gây nên bởi các lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

Move...	Inputs/Outputs	Operands	Data Types
Byte	IN	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
	OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE
Word	IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, Constant, AC *VD, *AC, *LD	WORD, INT
	OUT	VW, T, C, IW, QW, SW, MW, SMW, LW, AC, AQW, *VD, *AC, *LD	WORD, INT
Double Word	IN	VD, ID, QD, MD, SD, SMD, LD, HC, &VB, &IB, &QB, &MB, &SB, &T, &C, AC, Constant, *VD, *AC, *LD	DWORD, DINT
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD, DINT
Real	IN	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD	REAL
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	REAL

Các lệnh dịch chuyển một khối các byte, một khối các từ đơn (Word) và một khối các từ kép (Double Word):



Lệnh dịch chuyển một khối các byte, Block Move Byte, sao chép nội dung một số các ô nhớ liên tiếp (xác định bởi toán hạng ở đầu vào N), mỗi ô kích thước một byte với byte đầu tiên được định địa chỉ ở đầu vào IN lên khối các ô nhớ liên tiếp kích thước mỗi ô nhớ một byte và byte đầu tiên được định địa chỉ ở đầu ra OUT. Số lượng các byte có thể sao chép nằm trong khoảng từ 1 đến 255.

Lệnh dịch chuyển một khối các từ đơn, Block Move Word, sao chép nội dung một số các ô nhớ liên tiếp (xác định bởi toán hạng ở đầu vào N), mỗi ô kích thước một word với word đầu tiên được định địa chỉ ở đầu vào IN lên khối các ô nhớ liên tiếp kích thước mỗi ô nhớ một word và word đầu tiên được định địa chỉ ở đầu ra OUT. Số lượng các word có thể sao chép nằm trong khoảng từ 1 đến 255.

Lệnh dịch chuyển một khối các từ kép, Block Move Double Word, sao chép nội dung một số các ô nhớ liên tiếp (xác định bởi toán hạng ở đầu vào N), mỗi ô kích thước một từ kép với từ kép đầu tiên được định địa chỉ ở đầu vào IN lên khối các ô nhớ liên tiếp kích thước mỗi ô nhớ một từ kép và từ kép đầu tiên được định địa chỉ ở đầu ra OUT. Số lượng các từ kép có thể sao chép nằm trong khoảng từ 1 đến 255.

Những lỗi có thể được gây nên bởi các lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

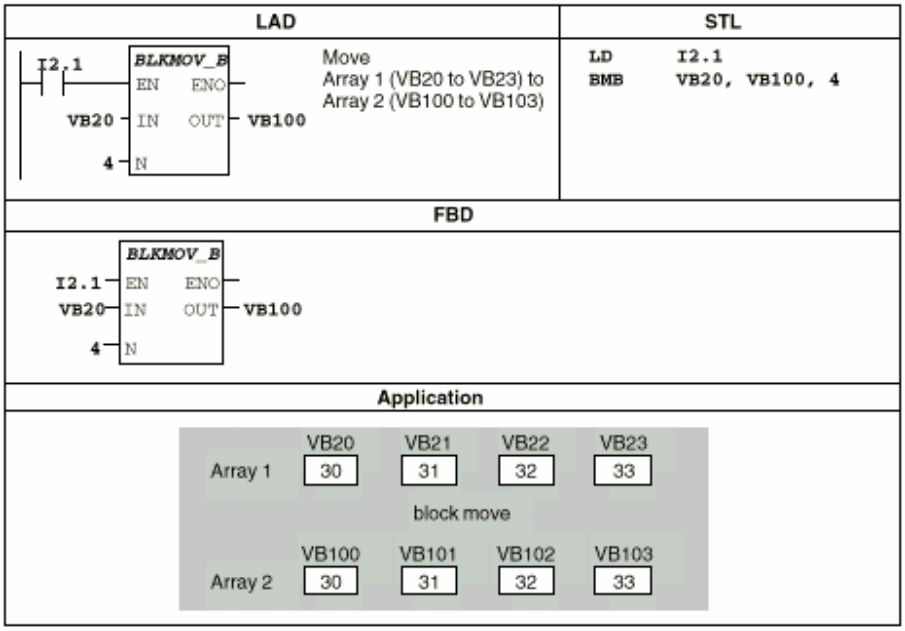
+ Lỗi 0006: địa chỉ gián tiếp.

+ Lỗi 0091: toán hạng vượt quá giới hạn cho phép.

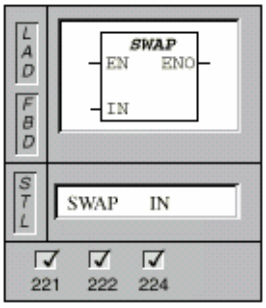
Block Move...	Inputs/Outputs	Operands	Data Types
Byte	IN, OUT	VB, IB, QB, MB,SB, SMB, LB, *VD, *AC, *LD	BYTE
	N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
Word	IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, *VD, *AC, *LD	WORD
	N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
	OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AQW, *VD, *LD, *AC	WORD
Double Word	IN, OUT	VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD	DWORD
	N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE

Ví dụ cách sử dụng lệnh dịch chuyển một khối dữ liệu:

Block Move Example



Lệnh SWAP:



Lệnh này (Swap Bytes) có toán hạng là một từ đơn (Word) được định địa chỉ bởi đầu vào IN. Lệnh Swap tráo đổi nội dung hai byte nhớ của một từ đơn: byte cao thành byte thấp và byte thấp thành byte cao. Kết quả được ghi vào chính từ đơn là toán hạng của lệnh.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

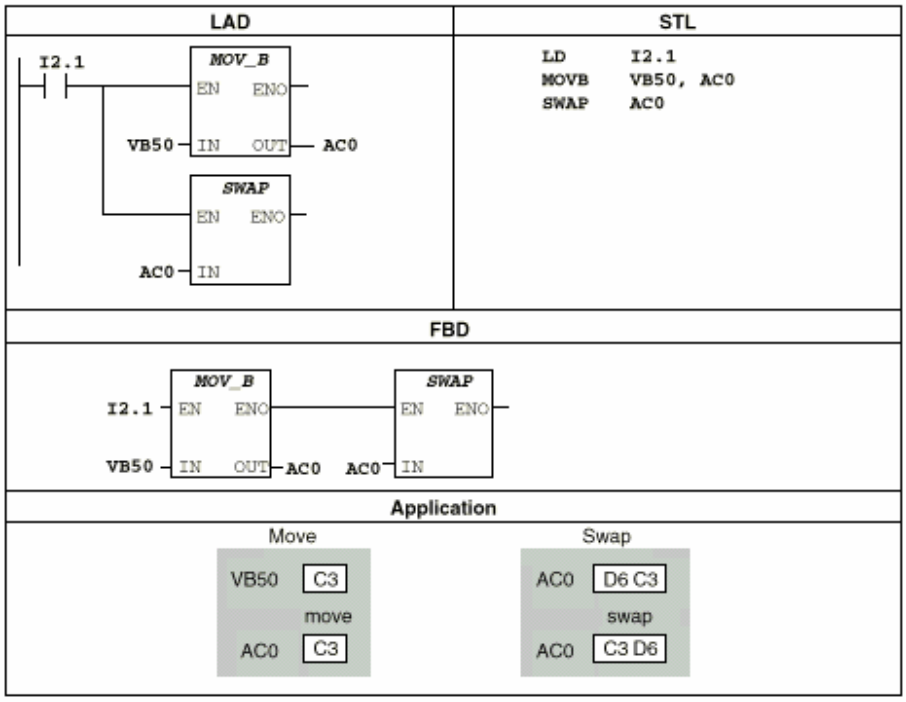
+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

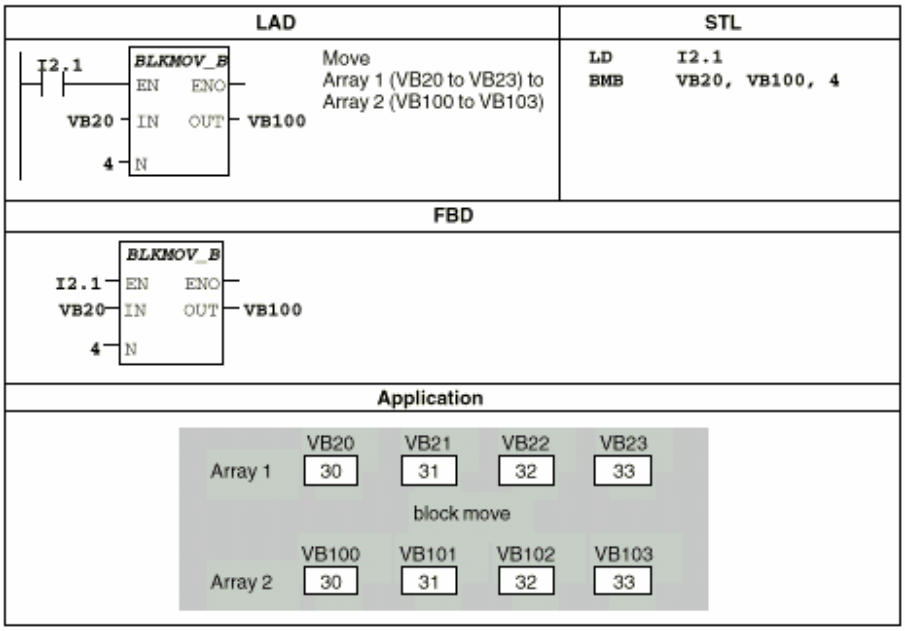
Inputs/Outputs	Operands	Data Types
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

Ví dụ về lệnh dịch chuyển và lệnh Swap:

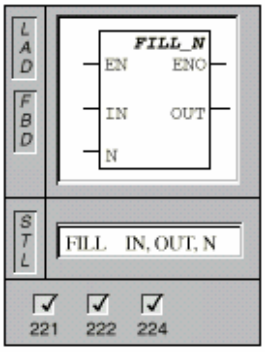
Move and Swap Examples



Block Move Example



Lệnh MEMORY FILL:



Lệnh này điền đầy một khoảng nhớ bao gồm một số các từ đơn liên tiếp (được xác định bởi đầu vào N) với từ đơn (Word) đầu tiên được định địa chỉ bởi đầu ra OUT bằng từ đơn được định địa chỉ ở đầu vào IN. Kích thước khoảng nhớ có thể nằm trong khoảng từ 1 đến 255 từ đơn.

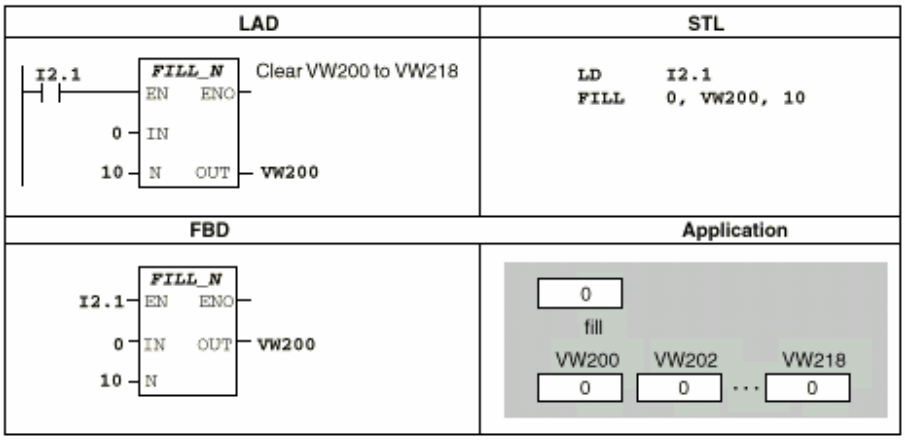
Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Lỗi 0091: toán hạng vượt quá giới hạn cho phép.

Inputs/Outputs	Operands	Data Types
IN	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, Constant, *VD, *AC, *LD	WORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AQW, *VD, *AC, *LD	WORD

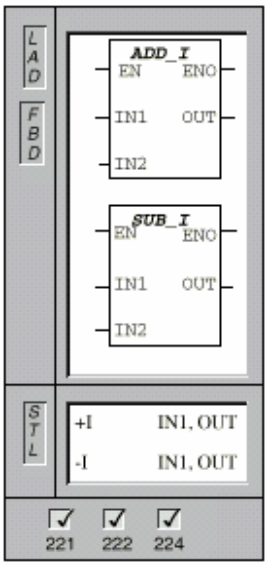
Ví dụ:

Fill Example



8.6 Các lệnh toán số học

Các lệnh Cộng, Trừ hai số nguyên (Integer):



Các lệnh này cộng (Add) hay trừ (Subtract) hai số nguyên được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số nguyên được định địa chỉ bởi đầu ra OUT.

Trong LAD và FBD: [IN1] + [IN2] = [OUT]
[IN1] - [IN2] = [OUT]

Trong STL: [IN1] + [OUT] = [OUT]
[OUT] - [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

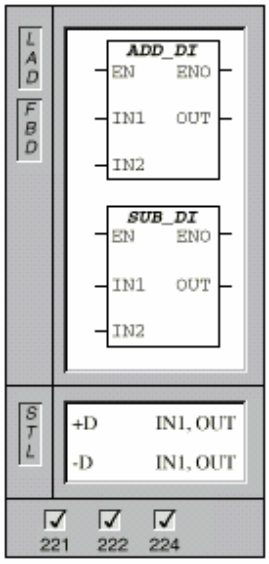
+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, Constant, *VD, *AC, *LD	INT
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	INT

Các lệnh Cộng, Trừ hai số nguyên dài (Double Integer):



Các lệnh này cộng (Add) hay trừ (Subtract) hai số nguyên 32 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số nguyên 32 bit được định địa chỉ bởi đầu ra OUT.

Trong LAD và FBD: [IN1] + [IN2] = [OUT]
[IN1] - [IN2] = [OUT]

Trong STL: [IN1] + [OUT] = [OUT]
[OUT] - [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

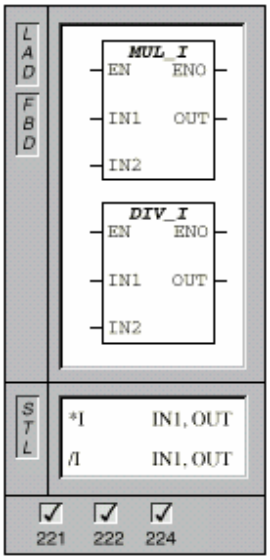
+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VD, ID, QD, MD, SMD, SD, LD, AC, HC, Constant, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SM, SD, LD, AC, *VD, *AC, *LD	DINT

Các lệnh Nhân, Chia hai số nguyên (Integer):



Các lệnh này nhân (Multiply) hay chia (Divide) hai số nguyên 16 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số nguyên được định địa chỉ bởi đầu ra OUT. Trong phép chia, số dư bị bỏ qua. Bit báo tràn sẽ thành 1 nếu kết quả lớn hơn một số nguyên 16 bit. Những lệnh này không có trong các CPU 212, 214.

Trong LAD và FBD: [IN1] * [IN2] = [OUT]
[IN1] / [IN2] = [OUT]

Trong STL: [IN1] * [OUT] = [OUT]
[OUT] / [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).
+ Bit đặc biệt SM1.3 = 1: lỗi chia cho 0 (Divide-by-zero).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.
+ SM1.3 (Divide-by-zero): bằng 1 nếu số chia bằng 0.

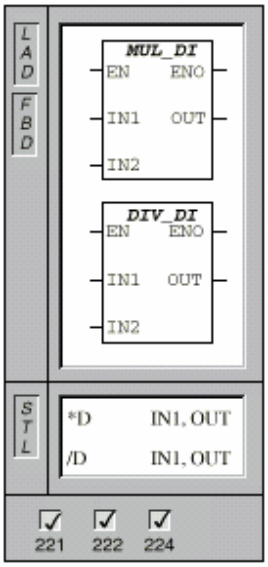
Inputs/Outputs	Operands	Data Types
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, Constant, *VD, *AC, *LD	INT
OUT	VW, QW, IW, MW, SW, SMW, LW, T, C, AC, *VD, *LD, *AC	INT

Trong trường hợp bit SM1.1 (Overflow) bằng 1, kết quả sẽ không được ghi và các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được xóa về 0.

Trong trường hợp bit SM1.3 (Divide-by-zero) bằng 1, các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được giữ nguyên không thay đổi và các toán hạng ở đầu vào cũng không đổi.

Trong các trường hợp còn lại, các bit đặc biệt nói trên sẽ có giá trị phản ảnh trạng thái của kết quả theo tính năng của chúng.

Các lệnh Nhân, Chia hai số nguyên dài (Double Integer):



Các lệnh này nhân (Multiply) hay chia (Divide) hai số nguyên 32 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số nguyên 32 bit được định địa chỉ bởi đầu ra OUT. Trong phép chia, số dư bị bỏ qua. Bit báo tràn sẽ thành 1 nếu kết quả lớn hơn một số nguyên 32 bit. Những lệnh này không có trong các CPU 212, 214.

Trong LAD và FBD: [IN1] * [IN2] = [OUT]
[IN1] / [IN2] = [OUT]

Trong STL: [IN1] * [OUT] = [OUT]
[OUT] / [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).
+ Bit đặc biệt SM1.3 = 1: lỗi chia cho 0 (Divide-by-zero).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

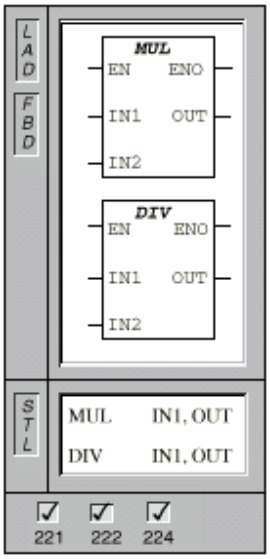
+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.
+ SM1.3 (Divide-by-zero): bằng 1 nếu số chia bằng 0.

Trong trường hợp bit SM1.1 (Overflow) bằng 1, kết quả sẽ không được ghi và các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được xóa về 0.

Trong trường hợp bit SM1.3 (Divide-by-zero) bằng 1, các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được giữ nguyên không thay đổi và các toán hạng ở đầu vào cũng không đổi.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VD, ID, QD, MD, SMD, SD, LD, HC, AC, Constant, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC	DINT

Các lệnh Nhân, Chia hai số nguyên (Integer) và ghi kết quả vào số nguyên dài (Double Integer):



Các lệnh này nhân (Multiply) hay chia (Divide) hai số nguyên 16 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số nguyên 32 bit được định địa chỉ bởi đầu ra OUT. Trong phép chia, kết quả bao gồm số dư ở 16 bit cao và thương số ở 16 bit thấp.

Trong LAD và FBD: [IN1] * [IN2] = [OUT]
[IN1] / [IN2] = [OUT]
Trong STL: [IN1] * [OUT] = [OUT]
[OUT] / [IN1] = [OUT]

Trong STL, lệnh MUL chỉ sử dụng 16 bit thấp của từ kép [OUT] làm số nhân. Tương tự lệnh DIV cũng chỉ sử dụng 16 bit thấp của từ kép [OUT] làm số bị chia.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).
+ Bit đặc biệt SM1.3 = 1: lỗi chia cho 0 (Divide-by-zero).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

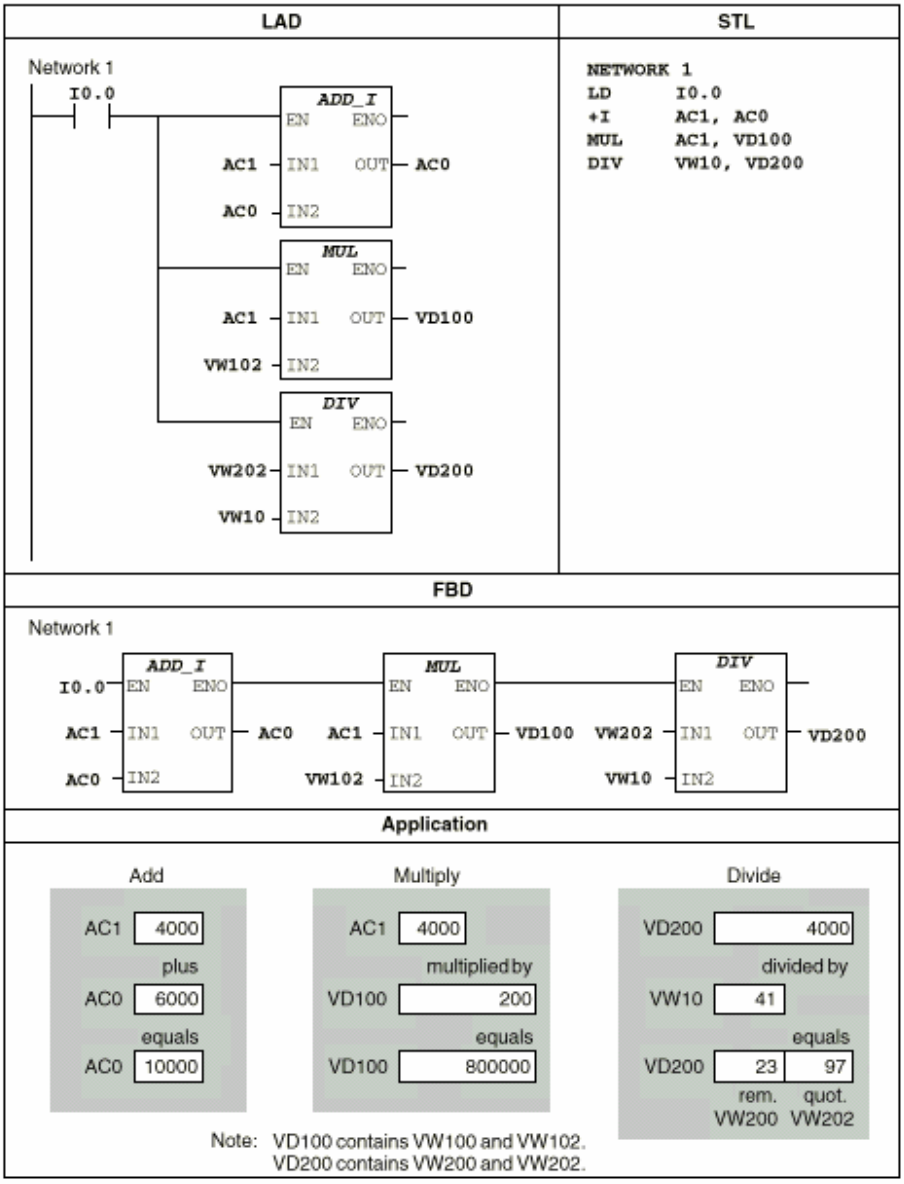
+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.
+ SM1.3 (Divide-by-zero): bằng 1 nếu số chia bằng 0.

Trong trường hợp bit SM1.3 (Divide-by-zero) bằng 1, các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được giữ nguyên không thay đổi và các toán hạng ở đầu vào cũng không đổi.

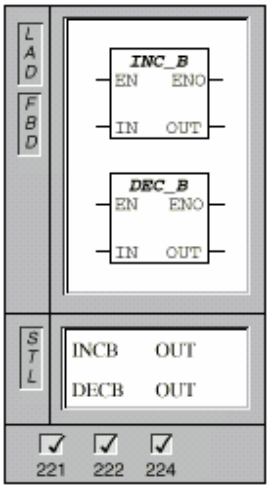
Inputs/Outputs	Operands	Data Types
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AC, AIW, T, C, Constant, *VD, *AC, *LD	INT
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC	DINT

Ví dụ về các lệnh số học:

Math Examples



Các lệnh tăng giảm một Byte một đơn vị:



Các lệnh này thêm vào hay bớt đi một đơn vị từ một Byte được định địa chỉ ở đầu vào IN, kết quả lưu vào Byte được định địa chỉ bởi đầu ra OUT.

Các số trong Byte toán hạng được xem là các số không dấu.

Trong LAD và FBD: [IN] + 1 = [OUT]
[IN] - 1 = [OUT]
Trong STL: [OUT] + 1 = [OUT]
[OUT] - 1 = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

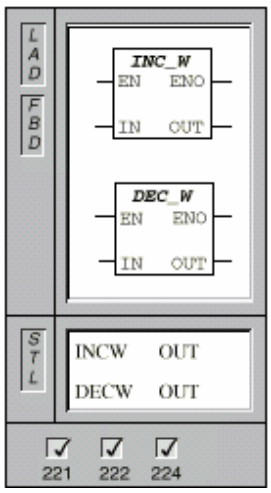
+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

Inputs/Outputs	Operands	Data Types
IN	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SB, SMB, LB, AC,*VD, *AC, *LD	BYTE

<u>Các lệnh tăng giảm một từ đơn một đơn vị</u>:



Các lệnh này thêm vào hay bớt đi một đơn vị từ một Word được định địa chỉ ở đầu vào IN, kết quả lưu vào Word được định địa chỉ bởi đầu ra OUT.

Các số trong từ đơn toán hạng được xem là các số có dấu (16#7FFF > 16#8000).

Trong LAD và FBD: [IN] + 1 = [OUT]
[IN] - 1 = [OUT]
Trong STL: [OUT] + 1 = [OUT]
[OUT] - 1 = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

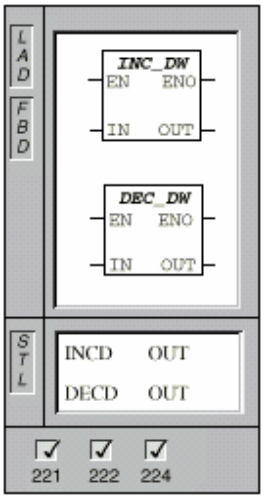
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Inputs/Outputs	Operands	Data Types
IN	VW, IW, QW, MW, SW, SMW, AC, AIW, LW, T, C, Constant, *VD, *AC, *LD	INT
OUT	VW, IW, QW, MW, SW, SMW, LW, AC, T, C, *VD, *AC, *LD	INT

Các lệnh tăng giảm một từ kép một đơn vị:



Các lệnh này thêm vào hay bớt đi một đơn vị từ một từ kép được định địa chỉ ở đầu vào IN, kết quả lưu vào từ kép được định địa chỉ bởi đầu ra OUT.

Các số trong từ kép toán hạng được xem là các số có dấu (16#7FFFFFFF > 16#80000000).

Trong LAD và FBD: [IN] + 1 = [OUT]
[IN] - 1 = [OUT]

Trong STL: [OUT] + 1 = [OUT]
[OUT] - 1 = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

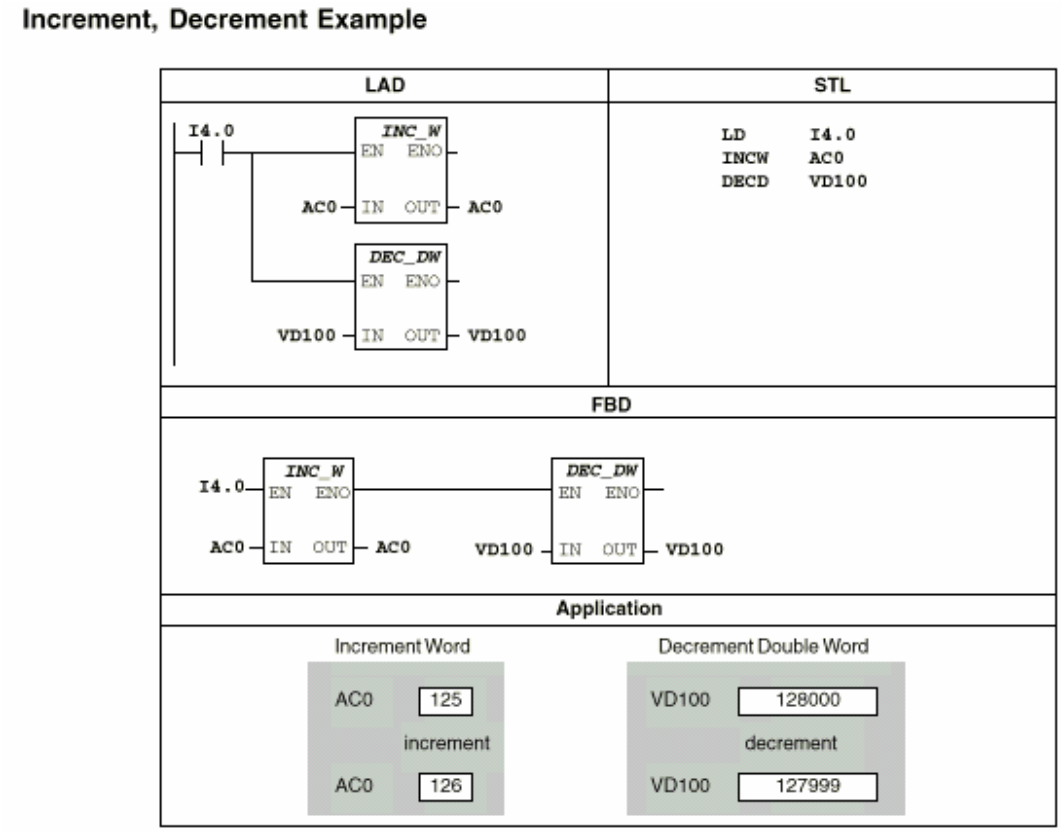
+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.
+ Lỗi 0006: địa chỉ gián tiếp.
+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

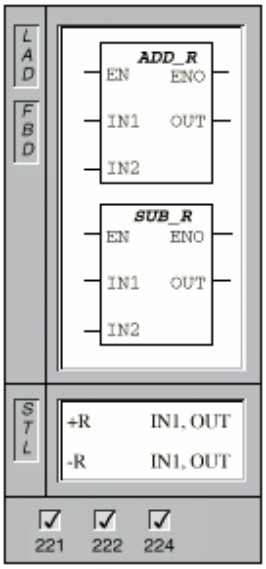
+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Inputs/Outputs	Operands	Data Types
IN	VD, ID, QD, MD, SD, SMD, LD, AC, HC, Constant, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DINT

Ví dụ:



8.7 Các lệnh toán số thực



Các lệnh cộng trừ hai số thực:

Các lệnh này cộng (Add) hay trừ (Subtract) hai số thực 32 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số thực 32 bit được định địa chỉ bởi đầu ra OUT. Những lệnh này không có trong CPU 212.

Trong LAD và FBD: [IN1] + [IN2] = [OUT]
[IN1] - [IN2] = [OUT]

Trong STL: [OUT] + [IN1] = [OUT]
[OUT] - [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi

lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

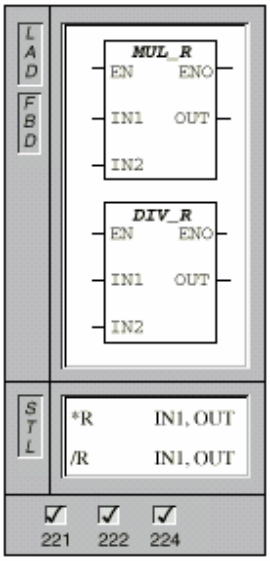
+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Bit đặc biệt SM1.1 dùng để xác định lỗi tràn hoặc giá trị không hợp lệ. Nếu SM1.1 = 1, giá trị các bit SM1.0, SM1.2 không có ý nghĩa và các giá trị đầu vào không thay đổi. Nếu SM1.1 = 0, phép toán hoàn thành với kết quả hợp lệ và các bit SM1.0, SM1.2 phản ảnh trạng thái kết quả theo đúng chức năng của chúng.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VD, ID, QD, MD, SD, SMD, AC, LD, Constant, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SD, SMD, AC, LD, *VD, *AC, *LD	REAL

Các số thực được biểu diễn bằng 32 bit dưới dạng dấu phẩy động theo chuẩn ANSI / IEEE 754 - 1985.



<u>Các lệnh nhân chia hai số thực</u>:

Các lệnh này nhân (Multiply) hay chia (Divide) hai số thực 32 bit được định địa chỉ ở các đầu vào IN1 và IN2, kết quả lưu vào số thực 32 bit được định địa chỉ bởi đầu ra OUT. Những lệnh này không có trong CPU 212.

Trong LAD và FBD: [IN1] * [IN2] = [OUT]
[IN1] / [IN2] = [OUT]

Trong STL: [OUT] * [IN1] = [OUT]
[OUT] / [IN1] = [OUT]

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow) hoặc giá trị đầu vào không hợp lệ (Invalid value).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

+ SM1.3 (Divide-by-zero): bằng 1 nếu số chia bằng 0.

Trong trường hợp bit SM1.3 (Divide-by-zero) bằng 1, các bit đặc biệt khác liên quan đến các phép toán (Zero, Negative, ...) đều được giữ nguyên không thay đổi và các toán hạng ở đầu vào cũng không đổi. Bit đặc biệt SM1.1 dùng để xác định lỗi tràn hoặc

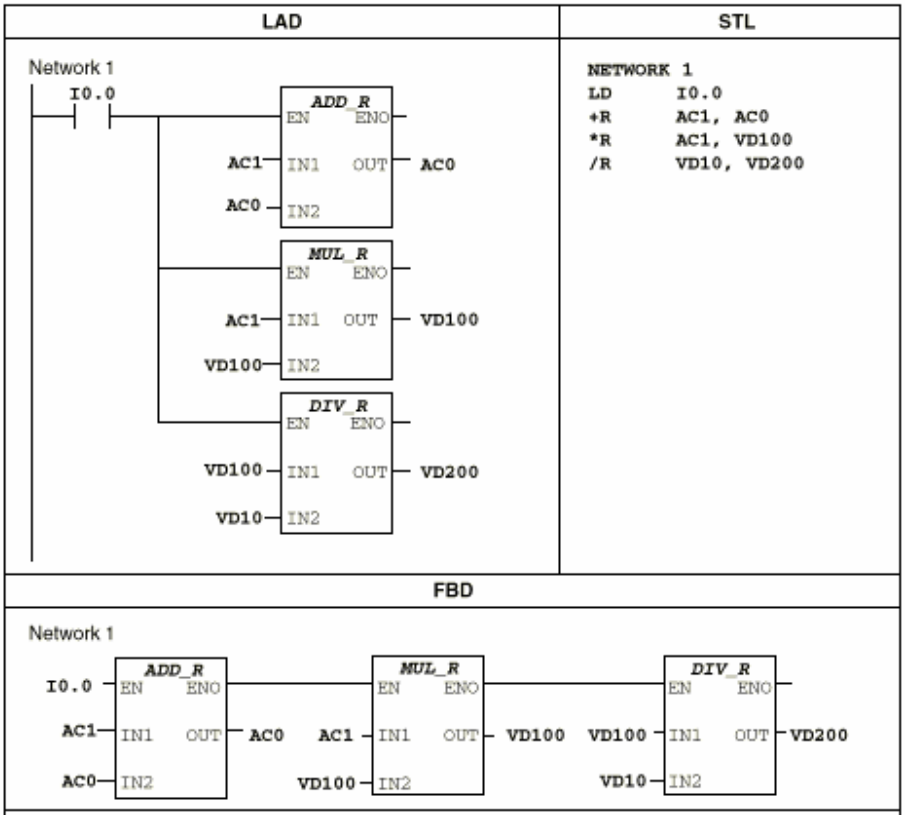
giá trị không hợp lệ. Nếu SM1.1 = 1, giá trị các bit SM1.0, SM1.2 không có ý nghĩa và các giá trị đầu vào không thay đổi. Nếu SM1.1 = 0 đồng thời SM1.3 = 0 , phép toán hoàn thành với kết quả hợp lệ và các bit SM1.0, SM1.2 phản ảnh trạng thái kết quả theo đúng chức năng của chúng.

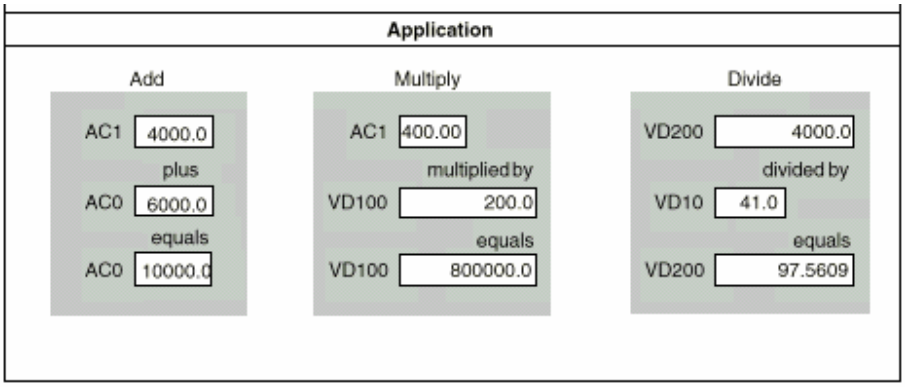
Inputs/Outputs	Operands	Data Types
IN1, IN2	VD, ID, QD, MD, SD, SMD, AC, LD, Constant, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SD, SMD, AC, LD, *VD, *AC, *LD	REAL

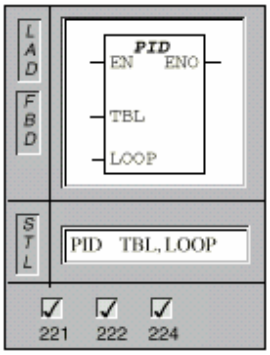
Các số thực được biểu diễn bằng 32 bit dưới dạng dấu phẩy động theo chuẩn ANSI / IEEE 754 - 1985.

Ví dụ sử dụng các phép toán số thực:

Math Examples







Vòng lặp PID:

Lệnh này tính toán vòng lặp PID (PID Loop) theo các đầu vào và những thông số từ bảng được định địa chỉ bởi TBL.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

Inputs/Outputs	Operands	Data Types
TBL	VB	BYTE
LOOP	Constant (0 to 7)	BYTE

Lệnh PID Loop (Proportional, Integral, Derivative Loop) được sử dụng để tính toán vòng lặp PID. Lệnh này chỉ được thực hiện nếu như đỉnh của ngăn xếp (Top Of Stack) bằng 1 trong STL, hay có Power Flow trong LAD. Lệnh này có hai toán hạng: [TBL] là địa chỉ byte đầu tiên của một bảng dữ liệu còn [LOOP] là một số nằm trong khoảng từ 0 đến 7. Điều này cũng có nghĩa là chỉ có tối đa 8 lệnh PID Loop có thể được sử dụng trong một chương trình. Nếu có hai lệnh PID Loop với cùng một số [LOOP] thì dù chúng có sử dụng hai bảng khác nhau đi nữa cũng vẫn ảnh hưởng đến nhau và có thể gây những hậu quả không lường trước được.

Bảng dữ liệu của lệnh PID Loop bao gồm 09 tham số dùng để điều khiển hoạt động của vòng lặp: giá trị tức thời và giá trị kế trước (current and previous value) của biến điều khiển (process variable), giá trị yêu cầu (setpoint), giá trị xử lý (output - đầu ra của PID), hệ số khuếch đại (gain), thời gian lấy mẫu (sample time), hệ số tích phân (integral time - reset), hệ số vi phân (derivative time - rate) và *integral sum (bias)*.

Để thực hiện lệnh này ở một tần suất lấy mẫu xác định, nó phải hoặc là được đặt trong một ngắt thời gian hoặc là được thực hiện trong chương trình chính qua kiểm soát bởi một bộ định thời. Đồng thời, thời gian lấy mẫu tương ứng phải được đưa vào bảng dữ liệu của lệnh.

Trong STEP 7 Micro / Win 32, chúng ta có thể sử dụng PID Wizard để tạo thuật toán với PID cho một mạch điều khiển kín bằng cách chọn **Tools Instruction Wizard -> PID** từ Menu chính.

Ở trạng thái ổn định, một bộ điều khiển PID sẽ điều chỉnh sao cho sai số giữa giá trị yêu cầu (setpoint SP) và giá trị điều khiển (process variable PV) bằng 0. Nguyên lý của một bộ điều khiển PID như vậy thể hiện trong phương trình sau:

$$M(t) = Kc * e + Ki * \int_0^t edt + Mi + Kd * \frac{de}{dt}$$

output = proportional + integral + differential

trong đó:

M(t): đầu ra của PID (đại lượng xử lý) như một hàm theo thời gian

Kc: hằng số khuếch đại

e: sai số. e = SP - PV

Mi: giá trị ban đầu của PID

Nhằm mục đích áp dụng bộ điều khiển PID trên máy vi tính hay PLC nói riêng và trong kỹ thuật số nói chung, chúng ta phải tiến hành "rời rạc hóa" phương trình nêu trên. Cụ thể là lấy mẫu và lượng tử hóa các biến. Phương trình được viết lại như sau:

$$M_n = Kc * e_n + Ki * \sum_{i=1}^{n} e_i + Mi + Kd * (e_n - e_{n-1})$$

output = proportional + integral + differential

trong đó:

M_n: đầu ra của PID (đại lượng xử lý) ở thời điểm lấy mẫu n

Kc: hằng số khuếch đại

e_n: sai số ở thời điểm lấy mẫu n. $e_n = SP_n - PV_n$

e_{n-1}: sai số ở thời điểm lấy mẫu ngay trước đó (n-1). $e_{n-1} = SP_{n-1} - PV_{n-1}$

Ki: hằng số khuếch đại của thành phần tích phân

Mi: giá trị ban đầu của PID

Kd: hằng số khuếch đại của thành phần vi phân

Từ phương trình này ta nhận thấy rằng, nếu như thành phần tỉ lệ (proportional) chỉ là hàm của sai số ở thời điểm lấy mẫu thì thành phần vi phân (differential) là hàm số của sai số ở thời điểm lấy mẫu lẫn thời điểm lấy mẫu kế trước còn thành phần tích phân (integral) lại là hàm của tất cả các sai số từ thời điểm lấy mẫu đầu tiên cho đến thời điểm

lấy mẫu hiện tại. Trong kỹ thuật số, lưu lại tất cả các sai số là điều không thể thực hiện được, cũng như thật sự không cần thiết.

Vì giá trị xử lý luôn được tính toán ở mọi thời điểm lấy mẫu, kể từ thời điểm đầu tiên, nên chỉ cần lưu lại giá trị kế trước của sai số và thành phần tích phân. Phương trình được đơn giản thành:

$$M_n = Kc * e_n + Ki * e_n + MX + Kd * (e_n - e_{n-1})$$

output = proportional + integral + differential

trong đó:

M_n: đầu ra của PID (đại lượng xử lý) ở thời điểm lấy mẫu n

Kc: hằng số khuếch đại

e_n: sai số ở thời điểm lấy mẫu n. $e_n = SP_n - PV_n$

e_{n-1}: sai số ở thời điểm lấy mẫu ngay trước đó (n-1). $e_{n-1} = SP_{n-1} - PV_{n-1}$

Ki: hằng số khuếch đại của thành phần tích phân

MX: giá trị thành phần tích phân ở thời điểm lấy mẫu kế trước (n-1)

Kd: hằng số khuếch đại của thành phần vi phân

Một cách viết khác của phương trình:

M*n* = **MP***n* + **MI***n* + **MD***n*

output = proportional + integral + differential

trong đó:

M*n*: đầu ra của PID (đại lượng xử lý) ở thời điểm lấy mẫu n

MP*n*: thành phần tỉ lệ của đầu ra PID ở thời điểm lấy mẫu n

MI*n*: thành phần tích phân của đầu ra PID ở thời điểm lấy mẫu n

MD*n*: thành phần vi phân của đầu ra PID ở thời điểm lấy mẫu n

Ta lần lượt xét đến từng thành phần một của đại lượng xử lý:

Thành phần tỉ lệ (proportional) MP là tích của hằng số khuếch đại Kc với sai số e. Trong đó Kc đặc trưng cho độ nhạy của đầu ra PID (Kc càng lớn, bộ điều khiển PID càng nhạy) còn e là sai số giữa đại lượng yêu cầu (setpoint SP) và đại lượng thực tế (process variable PV). Phương trình biểu diễn:

MP*n* = **Kc** * (**SP***n* - **PV***n*)

trong đó:

MP*n*: thành phần tỉ lệ của đầu ra PID ở thời điểm lấy mẫu n

Kc: hằng số khuếch đại

SP*n*: đại lượng yêu cầu tại thời điểm lấy mẫu n

PV*n*: đại lượng thực tế tại thời điểm lấy mẫu n

Thành phần tích phân (integral) MI tỉ lệ với tổng các sai số qua thời gian, thể hiện bằng phương trình:

MI*n* = **Kc** * **Ts** / **Ti** * (**SP***n* - **PV***n*) + **MX**

trong đó:

MI*n*: thành phần tích phân của đầu ra PID ở thời điểm lấy mẫu n

Kc: hằng số khuếch đại

Ts: thời gian lấy mẫu

Ti: hệ số tích phân

SP*n*: đại lượng yêu cầu tại thời điểm lấy mẫu n

PV*n*: đại lượng thực tế tại thời điểm lấy mẫu n

MX: giá trị của thành phần tích phân ở thời điểm lấy mẫu kế trước (n-1), còn được gọi là *integral sum* hay *bias*.

Sau khi tính toán giá trị MIn, bias MX được thay thế bởi chính giá trị MIn đó với khả năng có thể bị điều chỉnh hoặc cắt (chặn giới hạn), điều này sẽ được nói rõ ở phần sau. Giá trị ban đầu của bias MX, Mi thường được lấy là giá trị của đầu ra bộ PID ngay trước thời điểm thực hiện lệnh PID lần đầu tiên. Các hằng số khác ảnh hưởng đến thành phần này là: Kc - hằng số khuếch đại, Ts - thời gian lấy mẫu và Ti - hệ số tích phân là đặc trưng cho ảnh hưởng của thành phần này lên toàn bộ đại lượng xử lý.

Thành phần vi phân (differential) MD tỉ lệ với độ thay đổi của sai sô, thể hiện qua phương trình:

$$\mathbf{MD}n = \mathbf{Kc} * \mathbf{Td} / \mathbf{Ts} * ((\mathbf{SP}_n - \mathbf{PV}_n) - (\mathbf{SP}_{n-1} - \mathbf{PV}_{n-1}))$$

Với đặc tính có quán tính của mọi hệ vật chất, chúng ta có thể giả thiết rằng đại lượng thực tế PV không bao giờ có sự thay đổi một cách gián đoạn. Tuy nhiên đại lương yêu cầu thì có thể tăng giảm gãy khúc (do được tính trên lý thuyết). Về bản chất toán học, thành phần vi phân là phép lấy đạo hàm nên những sự thay đổi gián đoạn có thể gây nên các giá trị vô cùng lớn ở đầu ra. Để tránh hiện tượng này, trong phương trình trên ta giả thiết $\mathbf{SP}_n = \mathbf{SP}_{n-1}$ và có thể viết:

$$\mathbf{MD}n = \mathbf{Kc} * \mathbf{Td} / \mathbf{Ts} * (\mathbf{PV}_{n-1} - \mathbf{PV}_n)$$

trong đó:

MD*n*: thành phần vi phân của đầu ra PID ở thời điểm lấy mẫu n

Kc: hằng số khuếch đại

Ts: thời gian lấy mẫu

Td: hệ số vi phân

SP*n*: đại lượng yêu cầu tại thời điểm lấy mẫu n

$\mathbf{SP}_{n-1}$: đại lượng yêu cầu tại thời điểm lấy mẫu n-1

PV*n*: đại lượng thực tế tại thời điểm lấy mẫu n

$\mathbf{PV}_{n-1}$: đại lượng thực tế tại thời điểm lấy mẫu n-1

Như vậy trên thực tế không cần nhớ sai số ở thời điểm lấy mẫu kế trước mà chỉ cần nhớ đại lượng thực tế. Trong lần tính toán đầu tiên $\mathbf{PV}_{n-1}$ được lấy bằng $\mathbf{PV}_n$.

Tùy theo ứng dụng thực tế, có thể bỏ bớt thành phần trong bộ điều khiển PID chứ không nhất thiết phải bao gồm đủ cả ba thành phần, chẳng hạn có thể tạo bộ điều khiển tỉ lệ (P) hay bộ điều khiển chỉ chứa các thành phần tỉ lệ và tích phân (PI). Sự lựa chọn này dựa trên cách đặt các tham số.

Nếu muốn bỏ thành phần tích phân (bỏ I), ta chọn hệ số tích phân bằng vô cùng (Ti = :). Trong trường hợp này, thành phần tích phân vẫn không nhất thiết bằng không mà có thể bằng một giá trị không đổi thông qua giá trị bias MX ban đầu.

Nếu muốn bỏ thành phần vi phân (bỏ D), ta chọn hệ số vi phân bằng không (Td = 0.0).

Nếu muốn bỏ thành phần tỉ lệ (bỏ P), ta chọn hệ số khuếch đại bằng không (Kc = 0.0). Trong trường hợp này, vì các hằng số của các thành phần tích phân và vi phân có tính theo Kc nên đối với những thành phần ấy, Kc được hiểu là bằng 1.0.

Một bộ điều khiển PID có hai đầu vào: đại lượng yêu cầu và đại lượng thực tế. Đây là những đại lượng thật trong ứng dụng như nhiệt độ, áp suất, tốc độ, ... Để đưa vào tính toán trong một bộ điầu khiển, chúng phải được đo, chuyển đổi về giá trị thích hợp và chuẩn hóa (nếu cần). Các bước này đều cần thiết cho một bộ điều khiển PID, bộ này đòi hỏi các giá trị đầu vào là những giá trị số thực (dấu phẩy động) nằm trong khoảng từ 0.0 đến 1.0.

Thông thường, những giá trị đo được được đưa vào PLC qua các đầu vào tương tự (qui về điện áp trong khoảng 0 - 10VDC hoặc dòng điện 0 - 20mADC) thành những giá trị số nguyên 16 bit có dấu. Trước hết những giá trị này phải được đổi thành các số thực 32 bit (dấu phẩy động), chẳng hạn theo thuật toán sau:

```
XORD AC0, AC0            //Clear the accumulator.
MOVW AIW0, AC0           //Save the analog value in the accumulator.
LDW>= AC0, 0             //If the analog value is positive,
JMP 0                    //then convert to a real number.
NOT                      //Else,
ORD 16#FFFF0000, AC0     //sign extend the value in AC0.
LBL 0
DTR AC0, AC0             //Convert the 32-bit integer to a real number.
```

Bước tiếp theo là chuẩn hóa về khoảng [0.0 - 1.0] theo phương trình:

N*Norm* = (**N***Raw* / **Span**) + **Offset**

trong đó:

N*Norm* là giá trị đã chuẩn hóa, đại diện cho một đại lượng thật

N*Raw* là giá trị thực chưa chuẩn hóa, đại diện cho một đại lượng thật

Span là hiệu của giá trị lớn nhất có thể có trừ đi giá trị nhỏ nhất có thể có của giá trị chưa chuẩn hóa. Trong S7-200 thường là 32000 - 0 = 32000 đối với các đại lượng không đổi dấu (unipolar) và khi đó **Offset** = 0.0, hay 32000 - (-32000) = 64000 đối với

các đại lượng có thể vừa có giá trị dương vừa có giá trị âm (bipolar) và khi đó **Offset** = 0.5.

Đoạn lệnh sau đây minh họa cho thuật toán này trong trường hợp đại lượng có dấu (bipolar):

```
/R 64000.0, AC0      //Normalize the value in the accumulator
+R 0.5, AC0          //Offset the value to the range from 0.0 to 1.0
MOVR AC0, VD100 //Store the normalized value in the loop TABLE
```

Một cách lô gic chúng ta thấy rằng cần phải có quá trình ngược lại với quá trình trên đối với giá trị ở đầu ra của bộ điều khiển PID. Nghĩa là biến đổi và đưa về thang giá trị thích hợp cho đầu ra từ giá trị đầu ra đã chuẩn hóa trong khoảng 0.0 đến 1.0. Phương trình thuật toán:

R*Scale* = (**M***Norm* - **Offset**) * **Span**

trong đó:

R*Scale* là giá trị thích hợp cho đầu ra, đại diện cho một đại lượng thật

M*Norm* là giá trị đầu ra chuẩn hóa, đại diện cho một đại lượng thật

Span là hiệu của giá trị lớn nhất có thể có trừ đi giá trị nhỏ nhất có thể có của giá trị chưa chuẩn hóa. Trong S7-200 thường là 32000 - 0 = 32000 đối với các đại lượng không đổi dấu (unipolar) và khi đó **Offset** = 0.0, hay 32000 - (-32000) = 64000 đối với các đại lượng có thể vừa có giá trị dương vừa có giá trị âm (bipolar) và khi đó **Offset** = 0.5.

Đoạn lệnh minh họa cho thuật toán:

```
MOVR VD108, AC0 //Move the loop output to the accumulator.
-R 0.5, AC0     //Include this statement only if the value is
                //bipolar.
*R 64000.0, AC0 //Scale the value in the accumulator.
ROUND AC0 AC0   //Convert the real number to a 32-bit integer.
MOVW AC0, AQW0 //Write the 16-bit integer value to the analog
                //output.
```

Chúng ta thường nói về vòng lặp điều khiển thuận khi hệ số khuếch đại dương (Kc > 0) hay vòng lặp điều khiển đảo (nghịch) khi hệ số khuếch đại âm (Kc < 0). Trong trường hợp không có thành phần P (Kc = 0), ta xét dấu của các hệ số Ti và Td.

Các giá trị yêu cầu và giá trị thực tế (biến điều khiển) là những đầu vào của bộ điều khiển PID, do đó các trường tương ứng với chúng trong bảng dữ liệu của PID sẽ không bị thay đổi bởi lệnh này.

Ngược lại trường tương ứng với đầu ra được cập nhật bởi PID. Nó sẽ bị cắt (chặn) nếu vượt ra ngoài khoảng cho phép [0.0 - 1.0].

Nếu có sử dụng thành phần tích phân (I), bias cũng được cập nhật và lại được dùng làm đầu vào cho lần lấy mẫu kế tiếp. Tuy nhiên nó có thể được điều chỉnh trong trường hợp đầu ra bị chặn (vì vượt ra ngoài khoảng [0.0 - 1.0]) theo phương trình sau:

$$\mathbf{MX = 1.0 - (MP_n + MD_n)}$$

khi đầu ra lớn hơn 1.0, hay

$$\mathbf{MX = - (MP_n + MD_n)}$$

khi đầu ra nhỏ hơn 0.0, trong đó:

MX là giá trị bias đã được điều chỉnh

MP_n là giá trị thành phần tỉ lệ (P) của đầu ra ở thời điểm lấy mẫu n

MD_n là giá trị thành phần vi phân (D) của đầu ra ở thời điểm lấy mẫu n

M_n là giá trị của đầu ra ở thời điểm lấy mẫu n

Bằng sự điều chỉnh này, giá trị đầu ra sẽ được đưa về khoảng hợp lệ. Giá trị *bias* cũng bị chặn trong khoảng [0.0 - 1.0] và ghi vào bảng dữ liệu cho lần lấy mẫu tiếp theo sử dụng.

Giá trị bias trong bảng dữ liệu có thể thay đổi được ngay trước khi thực hiện lệnh PID nhưng phải chú ý đây là một số thực nằm trong khoảng [0.0 - 1.0].

Giá trị đại lượng thực tế của lần lấy mẫu trước được lưu lại trong bảng dữ liệu để tính toán thành phần vi phân, không bao giờ được thay đổi giá trị này.

Một bộ điều khiển PID có thể hoạt động ở một trong hai chế độ: Auto hoặc Manual. Thực ra không có chế độ hoạt động nào được xây dựng sẵn cho PID trong S7-200. Sự tính toán chỉ được thực hiện khi có dòng năng lượng (powerflow) đến đầu EN (enable) của bộ PID. PID được xem như hoạt động ở chế độ Auto khi nó thực hiện tính toán một cách tuần hoàn liên tục. Trong trường hợp ngược lại, PID được xem như hoạt động ở chế độ Manual. Vấn đề chúng ta cần xét đến là sự chuyển đổi đảm bảo tính liên tục từ chế độ Manual sang chế độ Auto. Điều đó đòi hỏi đầu ra được tính trong chế độ Manual phải được ghi vào đầu vào ở thời điểm chuyển đổi sang chế độ Auto. Tương tự như cách hoạt động của bộ đếm, CPU sử dụng một bit nhớ để xác định thời điểm chuyển đổi: khi dòng năng lượng thay đổi từ 0 lên 1. Lúc đó CPU sẽ thực hiện một loạt thao tác cần thiết:

- Đặt giá trị yêu cầu bằng giá trị thực tế: **SP***n* = **PV***n*
- Đặt giá trị kế trước của giá trị thực tế: **PV***n-1* = **PV***n*
- Đặt Bias bằng giá trị đầu ra: **MX** = **M***n*

Bit nhớ của một bộ PID có giá trị mặc định là 1 (ON), được đặt khi CPU khởi động hay chuyển từ chế độ STOP sang chế độ RUN. Điều đó cũng có nghĩa là khi bộ PID được thực hiện lần đầu tiên, CPU không nhận biết sự chuyển đổi trạng thái của dòng năng lượng từ 0 lên 1 và do đó không thực hiện các thao tác nêu ở trên.

Lệnh PID là một lệnh đơn giản nhưng rất mạnh trong việc tính toán thuật toán PID. Nếu cần một số tính năng khác, ví dụ như báo động hay những thay đổi đặc biệt, có thể sử dụng các lệnh khác để can thiệp.

Khi chương trình sử dụng được biên dịch, lỗi biên dịch có thể xuất hiện nếu địa chỉ bảng tham số [TBL] hoặc toán hạng [LOOP] của bộ PID vượt ra ngoài phạm vi cho phép (out of range).

Một số phạm vi cho phép không được kiểm tra, vì vậy người lập trình phải chú ý. Chẳng hạn như những giá trị yêu cầu và thực tế phải là các số thực nằm trong khoảng từ 0.0 đến 1.0, cũng như các giá trị thực tế kế trước hay Bias, nếu được sử dụng, không được vượt ra ngoài khoảng [0.0 - 1.0].

Nếu lỗi xuất hiện trong quá trình tính toán thuật toán PID, bit đặc biệt SM1.1 (overflow) sẽ bằng 1 và quá trình tính toán bị dừng lại. Trong những trường hợp như vậy, đầu ra của bộ PID có thể chưa được hoàn thành, vì vậy người lập trình phải chú ý kiểm tra bit đặc biệt này để sử dụng đầu ra một cách hợp lý cũng như điều chỉnh các dầu vào nếu cần thiết.

Định dạng bảng các tham số của một bộ PID bao gồm 36 bytes như sau:

Offset	Field	Format	Type	Description
0	Process variable (PV_n)	Double word - real	in	Contains the process variable, which must be scaled between 0.0 and 1.0.
4	Setpoint (SP_n)	Double word - real	in	Contains the setpoint, which must be scaled between 0.0 and 1.0.
8	Output (M_n)	Double word - real	in/out	Contains the calculated output, scaled between 0.0 and 1.0.
12	Gain (K_C)	Double word - real	in	Contains the gain, which is a proportional constant. Can be a positive or negative number.
16	Sample time (T_S)	Double word - real	in	Contains the sample time, in seconds. Must be a positive number.
20	Integral time or reset (T_I)	Double word - real	in	Contains the integral time or reset, in minutes. Must be a positive number.
24	Derivative time or rate (T_D)	Double word - real	in	Contains the derivative time or rate, in minutes. Must be a positive number.
28	Bias (MX)	Double word - real	in/out	Contains the bias or integral sum value between 0.0 and 1.0.
32	Previous process variable (PV_{n-1})	Double word - real	in/out	Contains the previous value of the process variable stored from the last execution of the PID instruction.

Sau đây là một ví dụ cách dùng bộ điều khiển PID:

Một bể nước được dùng để giữ một áp lực cột nước cố định. Nước chảy ra khỏi bể với tốc độ thay đổi không xác định. Để đạt mục đích người ta sử dụng một bơm nước có lưu lượng điều chỉnh được một cách liên tục để bơm nước vào bể.

Giá trị yêu cầu trong ví dụ này là phải giữ mức nước trong bể ở 75%. Giá trị thực tế chính là mức nước đo được, thay đổi từ 0% (khi bể cạn) đến 100% (khi bể đầy). Giá trị xử lý (đầu ra bộ điều khiển PID) là vận tốc bơm, điều chỉnh được từ 0% đến 100% lưu lượng danh định.

Giá trị yêu cầu, không thay đổi, sẽ được ghi trực tiếp vào bảng các tham số của bộ PID. Giá trị thực tế là giá trị không đổi dấu (chỉ dương - unipolar) và là giá trị tương tự đọc vào từ bộ đo mức. Giá trị đầu ra PID cũng là giá trị tương tự, unipolar, dùng để diều khiển tốc độ bơm. Cả hai giá trị tương tự này, đối với S7-200, nằm trong khoảng từ 0 đến 32000.

Ta sử dụng bộ điều khiển PI (chỉ bao gồm thành phần tỉ lệ và tích phân, không chứa thành phần vi phân). Các hằng số điều khiển được tính toán dựa trên những thông số kỹ thuật của hệ điều khiển và có thể điều chỉnh trong quá trình khai thác thực tế. Ở đây ta không đi sâu vào vấn đề này.

Kc = 0.25

Ts = 0.1 s

Ti = 30 min

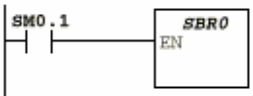
Bơm được điều khiển bằng tay cho đến khi mức nước trong bể đạt 75% thì chuyển sang chế độ tự động và mở van cho nước chảy ra khỏi bể. Đầu vào I0.0 được sử dụng để đổi chế độ: I0.0 = 0 là Manual; I0.0 = 1 là Auto. Khi ở trong chế độ Manual, tốc độ bơm được xác định bởi số thực trong khoảng [0.0 - 1.0] ghi ở VD108.

Chương trình:

LAD | **STL**

MAIN OB1

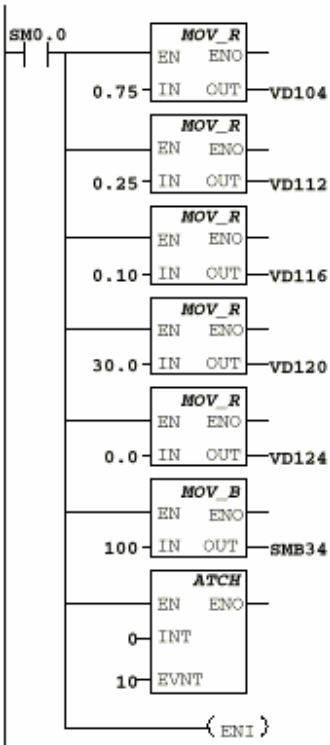
Network 1




```
Network 1
LD     SM0.1        //On the first scan call
CALL   0            //the initialization
                    //subroutine.
```

SUBROUTINE 0

Network 1



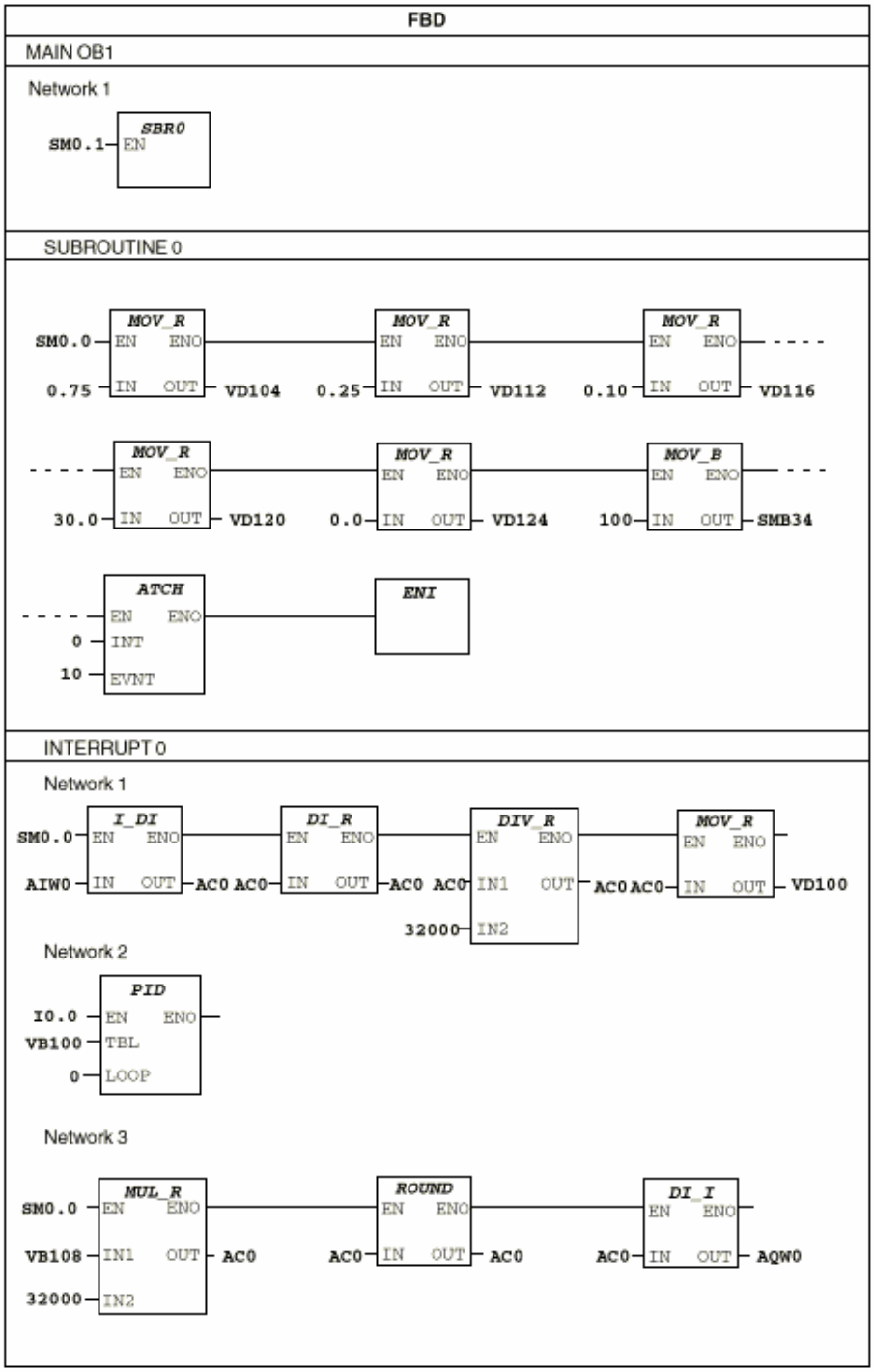

```
Network 1
LD     SM0.0
MOVR   0.75, VD104 //Load the loop setpoint.
                   // = 75% full.
MOVR   0.25, VD112 //Load the loop gain=0.25.
MOVR   0.10, VD116 //Load the loop sample
                   //time = 0.1 seconds.
MOVR   30.0, VD120 //Load the integral time
                   //= 30 minutes.
                   //
MOVR   0.0, VD124  //Set no derivative action.
MOVB   100, SMB34  //Set time interval
                   //(100 ms) for timed
                   //interrupt 0.
ATCH   0, 10       //Set up a timed
                   //interrupt to invoke
                   //PID execution.
ENI                //Enable interrupts.

                   //End of subroutine 0
```

LAD	STL
INTERRUPT 0	

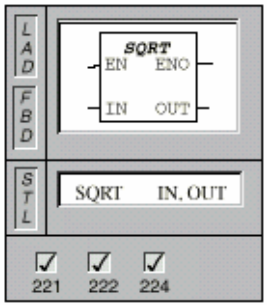
Network 1
SM0.0
I_DI EN ENO, AIW0 IN, OUT AC0
DI_R EN ENO, AC0 IN, OUT AC0
DIV_R EN ENO, AC0 IN1, OUT AC0, 32000 IN2
MOV_R EN ENO, AC0 IN, OUT VD100
	NETWORK 1
	//Convert PV to a
	//normalized real
	//number value - PV is
	//a unipolar input and
	//cannot be negative.
	LD SM0.0
	ITD AIW0, AC0 //Save the unipolar
	//analog value in
	//the accumulator.
	DTR AC0, AC0 //Convert the 32-bit
	//integer to a real
	//number.
	/R 32000.0, AC0 //Normalize the value
	//in the
	//accumulator.
	MOVR AC0, VD100 //Store the normalized
	//PV in the loop TABLE.

Network 2
I0.0
PID EN ENO, VB100 TBL, 0 LOOP
	NETWORK 2
	//Execute the loop when
	//placed in auto mode.
	LD I0.0 //When auto mode is
	//entered,
	PID VB100, 0 //invoke PID execution.

Network 3
SM0.0
MUL_R EN ENO, VD108 IN1, OUT AC0, 32000 IN2
ROUND EN ENO, AC0 IN, OUT AC0
DI_I EN ENO, AC0 IN, OUT AQW0
	NETWORK 3
	//Convert M_n to a scaled,
	//sixteen-bit integer.
	//M_n is a unipolar value
	//and cannot be negative.
	LD SM0.0
	MOVR VD108, AC //Move the loop output
	//to the accumulator.
	*R 32000.0, AC0 //Scale the value in
	//the accumulator.
	ROUND AC0, AC0 //Convert the real
	//number value to
	//a 32-bit integer.
	DTI AC0, AQW0 //Write the 16-bit
	//integer value to
	//the analog output.
	
	//end of Interrupt
	Routine 0


FBD
MAIN OB1
Network 1
SM0.1
SBR0
EN
SUBROUTINE 0
SM0.0
MOV_R
EN ENO
0.75
IN OUT
VD104
0.25
VD112
0.10
VD116
30.0
VD120
0.0
VD124
MOV_B
100
SMB34
ATCH
INT
0
EVNT
10
ENI
INTERRUPT 0
Network 1
I_DI
AIW0
AC0
DI_R
DIV_R
IN1
IN2
32000
VD100
Network 2
PID
I0.0
VB100
TBL
LOOP
Network 3
MUL_R
VB108
ROUND
DI_I
AQW0

Lệnh lấy căn số bậc hai (Square Root):



Lệnh này lấy căn số bậc hai của một số thực 32 bit được định địa chỉ ở đầu vào IN, kết quả lưu vào số thực 32 bit được định địa chỉ bởi đầu ra OUT theo phương trình:

$$\sqrt{[IN]} = [OUT]$$

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

+ Bit đặc biệt SM1.1 = 1: lỗi tràn (Overflow) hoặc giá trị đầu vào không hợp lệ (Invalid value).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng 1 nếu kết quả bị tràn.

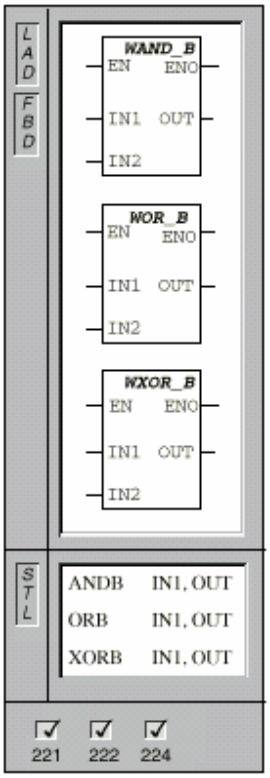
+ SM1.2 (Negative): bằng 1 nếu kết quả là số âm.

Bit đặc biệt SM1.1 dùng để xác định lỗi tràn hoặc giá trị không hợp lệ. Nếu SM1.1 = 1, giá trị các bit SM1.0, SM1.2 không có ý nghĩa và các giá trị đầu vào không thay đổi. Nếu SM1.1 = 0, phép toán hoàn thành với kết quả hợp lệ và các bit SM1.0, SM1.2 phản ảnh trạng thái kết quả theo đúng chức năng của chúng.

Inputs/Outputs	Operands	Data Types
IN	VD, ID, QD, MD, SMD, SD, LD, AC, Constant, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD,AC, *VD, *AC, *LD	REAL

8.8 Các lệnh phép toán lô gic

AND bytes, OR bytes, EXCLUSIVE OR bytes:



Lệnh AND Bytes thực hiện phép toán lô gic AND giữa các bit tương ứng của các byte đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Lệnh OR Bytes thực hiện phép toán lô gic OR giữa các bit tương ứng của các byte đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Lệnh EXCLUSIVE OR Bytes thực hiện phép toán lô gic XOR giữa các bit tương ứng của các byte đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

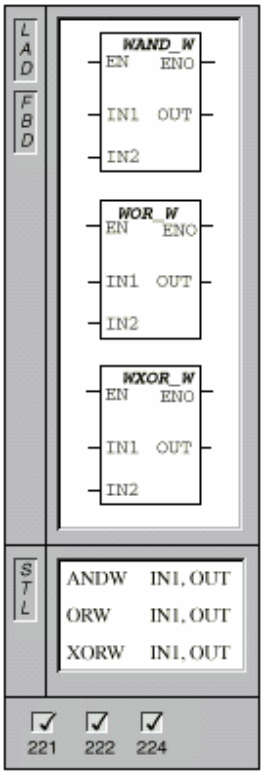
+ Lỗi 0006: địa chỉ gián tiếp.

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB,SB, SMB, LB, AC, *VD, *AC, *LD	BYTE

AND words, OR words, EXCLUSIVE OR words:



Lệnh AND Words thực hiện phép toán lô gic AND giữa các bit tương ứng của các word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Lệnh OR Words thực hiện phép toán lô gic OR giữa các bit tương ứng của các word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Lệnh EXCLUSIVE OR Words thực hiện phép toán lô gic XOR giữa các bit tương ứng của các word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

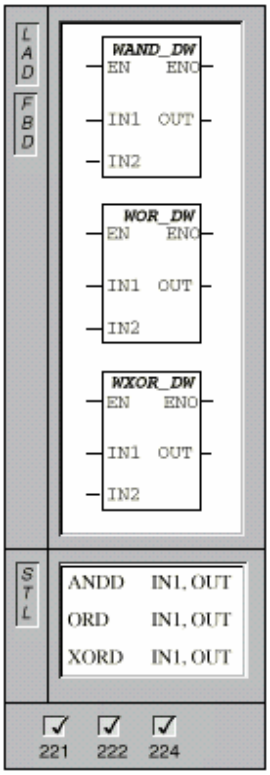
+ Lỗi 0006: địa chỉ gián tiếp.

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

Inputs/Outputs	Operands	Data Types
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, Constant, *VD, *AC, *LD	WORD
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

<u>AND double words, OR double words, EXCLUSIVE OR double words</u>:



Lệnh AND Double words thực hiện phép toán lô gic AND giữa các bit tương ứng của các double word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Lệnh OR Double words thực hiện phép toán lô gic OR giữa các bit tương ứng của các double word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Lệnh EXCLUSIVE OR Double words thực hiện phép toán lô gic XOR giữa các bit tương ứng của các double word đầu vào được định địa chỉ bởi các đầu vào IN1 và IN2, kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

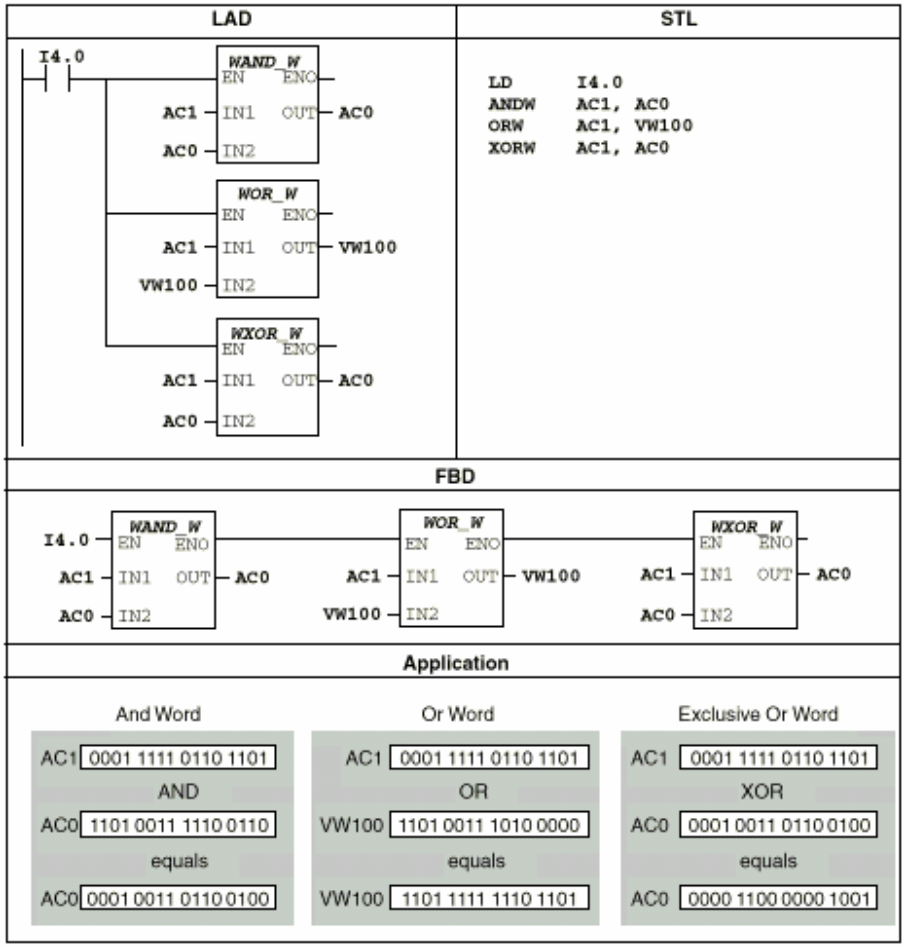
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

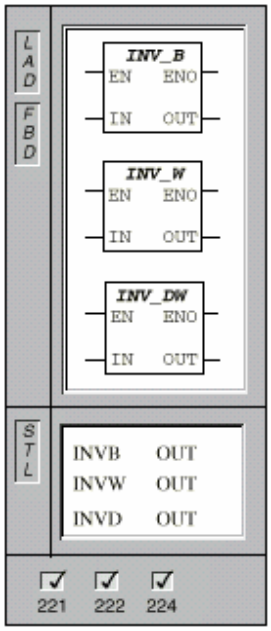
Inputs/Outputs	Operands	Data Types
IN1, IN2	VD, ID, QD, MD, SMD, AC, LD, HC, Constant, *VD, *AC, SD, *LD	DWORD
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DWORD

Ví dụ các lệnh lô gic:

And, Or, and Exclusive Or Instructions Example



<u>INVERT byte, INVERT word, INVERT double word</u>:



Lệnh INVERT byte thực hiện phép đảo kiểu thứ nhất (đảo từng bit một, 0 thành 1, 1 thành 0) một byte được định địa chỉ bởi đầu vào IN, kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Lệnh INVERT word thực hiện phép đảo kiểu thứ nhất (đảo từng bit một, 0 thành 1, 1 thành 0) một word được định địa chỉ bởi đầu vào IN, kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Lệnh INVERT double word thực hiện phép đảo kiểu thứ nhất (đảo từng bit một, 0 thành 1, 1 thành 0) một double word được định địa chỉ bởi đầu vào IN, kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

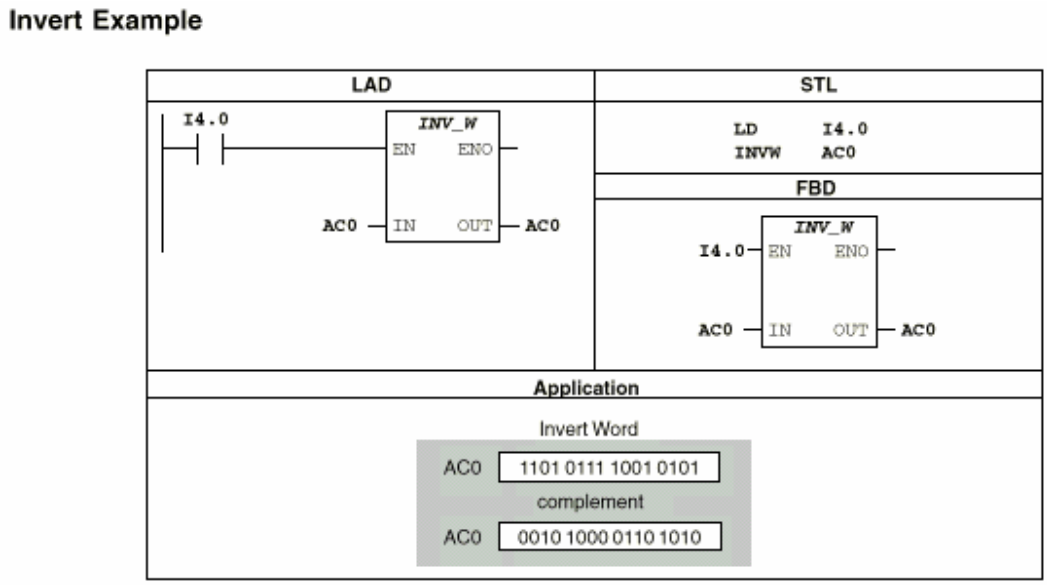
+ Lỗi 0006: địa chỉ gián tiếp.

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

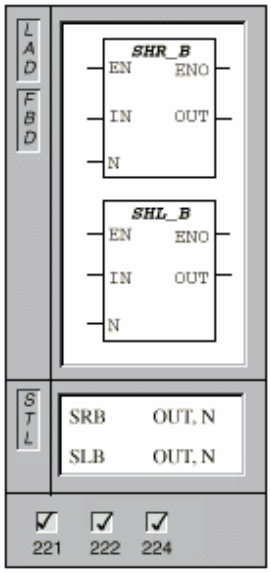
Invert...	Inputs/Outputs	Operands	Data Types
Byte	IN	VB, IB, QB, MB,SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
	OUT	VB, IB, QB, MB, SB, SMB, LB, AC,*VD, *AC, *LD	BYTE
Word	IN	VW, IW, QW, MW, SW, SMW, T, C, AIW, LW, AC, Constant, *VD, *AC, *LD	WORD
	OUT	VW, IW, QW, MW,SW, SMW, T, C, LW, AC, *VD, *AC, *LD	WORD
Double Word	IN	VD, ID, QD, MD, SD, SMD, LD, HC, AC, Constant, *VD, *AC, *LD	DWORD
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD

Ví dụ:



8.9 Các lệnh dịch (shift) và quay (rotate) nội dung các ô nhớ

Các lệnh dịch (shift) nội dung một byte:



Những lệnh này ghi dịch (shift) nội dung một byte được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Shift Right Byte) hoặc sang trái (Shift Left Byte), kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Phép ghi dịch (shift) điền giá trị 0 (OFF) vào các bit đã bị dịch đi. Như vậy nếu số lần dịch [N] lớn hơn 8, thực tế chỉ cần dịch tối đa 8 lần vì sau đó kết quả chắc chắn bằng 0.

Nếu số lần dịch [N] lớn hơn 0, bit cuối cùng trong byte bị dịch ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép dịch bằng 0.

Các lệnh ghi dịch một byte đều xem các byte là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

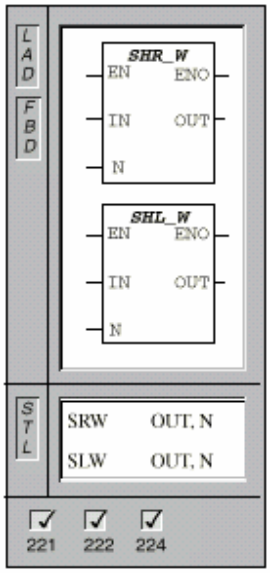
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng bit cuối cùng bị dịch ra ngoài.

Inputs/Outputs	Operands	Data Types
IN, OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE
N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE

<u>Các lệnh dịch (shift) nội dung một word</u>:



Những lệnh này ghi dịch (shift) nội dung một word được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Shift Right Word) hoặc sang trái (Shift Left Word), kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Phép ghi dịch (shift) điền giá trị 0 (OFF) vào các bit đã bị dịch đi. Như vậy nếu số lần dịch [N] lớn hơn 16, thực tế chỉ cần dịch tối đa 16 lần vì sau đó kết quả chắc chắn bằng 0.

Nếu số lần dịch [N] lớn hơn 0, bit cuối cùng trong word bị dịch ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép dịch bằng 0.

Các lệnh ghi dịch một word đều xem các word là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

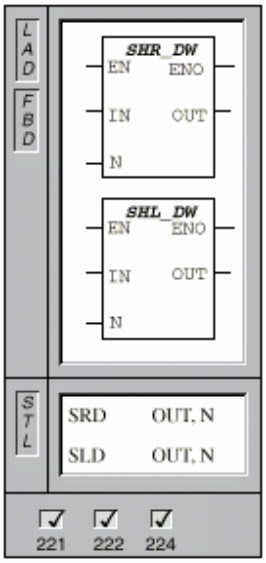
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng bit cuối cùng bị dịch ra ngoài.

Inputs/Outputs	Operands	Data Types
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, Constant, *VD, *AC, *LD	WORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

<u>Các lệnh dịch (shift) nội dung một double word</u>:



Những lệnh này ghi dịch (shift) nội dung một double word được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Shift Right Double word) hoặc sang trái (Shift Left Double word), kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Phép ghi dịch (shift) điền giá trị 0 (OFF) vào các bit đã bị dịch đi. Như vậy nếu số lần dịch [N] lớn hơn 32, thực tế chỉ cần dịch tối đa 32 lần vì sau đó kết quả chắc chắn bằng 0.

Nếu số lần dịch [N] lớn hơn 0, bit cuối cùng trong double word bị dịch ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép dịch bằng 0.

Các lệnh ghi dịch một double word đều xem các double word là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

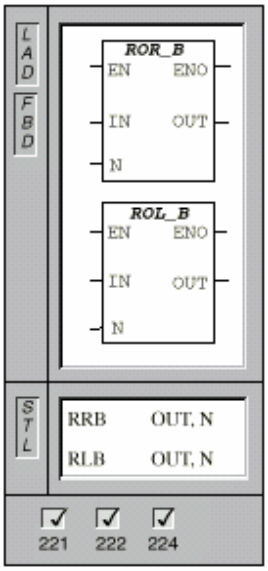
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng bit cuối cùng bị dịch ra ngoài.

Inputs/Outputs	Operands	Data Types
IN	VD, ID, QD, MD, SD, SMD, LD, AC, HC, Constant, *VD, *AC, *LD	DWORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD

Các lệnh quay (rotate) nội dung một byte:



Những lệnh này quay (rotate) nội dung một byte được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Rotate Right Byte) hoặc sang trái (Rotate Left Byte), kết quả lưu vào byte được định địa chỉ bởi đầu ra OUT.

Phép quay (rotate) điền giá trị bit cuối cùng (bị quay ra ngoài) vào bit đầu tiên. Như vậy nếu số lần quay [N] lớn hơn 8, thực tế chỉ cần quay một số lần bằng số dư trong phép chia [N] cho 8 vì sau đó quá trình sẽ được lặp lại. Do đó số lần quay thực tế chỉ nằm trong khoảng từ 0 đến 7. Nếu số lần quay bằng 0 (hay số lần quay chia hết cho 8), phép quay không được thực hiện. Trong trường hợp phép quay được thực hiện, bit cuối cùng trong byte bị quay ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép quay bằng 0.

Các lệnh ghi quay một byte đều xem các byte là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

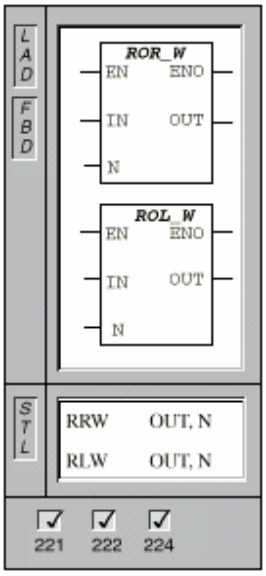
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng bit cuối cùng bị quay ra ngoài.

Inputs/Outputs	Operands	Data Types
IN	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *AC, *LD	BYTE
N	VB, IB, QB, MB, SMB, SB, LB, AC, Constant, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *AC, *LD	BYTE

Các lệnh quay (rotate) nội dung một word:



Những lệnh này quay (rotate) nội dung một word được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Rotate Right Word) hoặc sang trái (Rotate Left Word), kết quả lưu vào word được định địa chỉ bởi đầu ra OUT.

Phép quay (rotate) điền giá trị bit cuối cùng (bị quay ra ngoài) vào bit đầu tiên. Như vậy nếu số lần quay [N] lớn hơn 16, thực tế chỉ cần quay một số lần bằng số dư trong phép chia [N] cho 16 vì sau đó quá trình sẽ được lặp lại. Do đó số lần quay thực tế chỉ nằm trong khoảng từ 0 đến 15. Nếu số lần quay bằng 0 (hay số lần quay chia hết cho 16), phép quay không được thực hiện. Trong trường hợp phép quay được thực hiện, bit cuối cùng trong word bị quay ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép quay bằng 0.

Các lệnh ghi quay một word đều xem các word là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

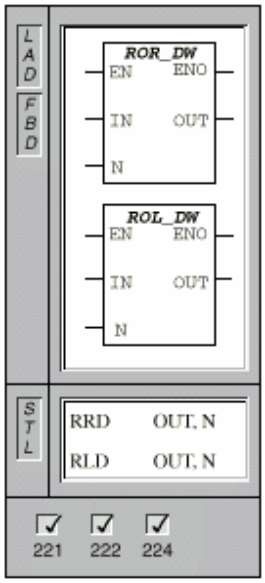
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

+ SM1.1 (Overflow): bằng bit cuối cùng bị quay ra ngoài.

Inputs/Outputs	Operands	Data Types
IN	VW, T, C, IW, MW, SMW, AC, QW, LW, AIW, Constant, *VD, *AC, SW, *LD	WORD
N	VB, IB, QB, MB, SMB, LB, AC, Constant, *VD, *AC, SB, *LD	BYTE
OUT	VW, T, C, IW, QW, MW, SMW, LW, AC, *VD, *AC, SW, *LD	WORD

Các lệnh quay (rotate) nội dung một double word:



Những lệnh này quay (rotate) nội dung một double word được định địa chỉ bởi đầu vào IN đi [N] lần (định bởi toán hạng N), mỗi lần một bit sang phải (Rotate Right Double word) hoặc sang trái (Rotate Left Double word), kết quả lưu vào double word được định địa chỉ bởi đầu ra OUT.

Phép quay (rotate) điền giá trị bit cuối cùng (bị quay ra ngoài) vào bit đầu tiên. Như vậy nếu số lần quay [N] lớn hơn 32, thực tế chỉ cần quay một số lần bằng số dư trong phép chia [N] cho 32 vì sau đó quá trình sẽ được lặp lại. Do đó số lần quay thực tế chỉ nằm trong khoảng từ 0 đến 31. Nếu số lần quay bằng 0 (hay số lần quay chia hết cho 32), phép quay không được thực hiện. Trong trường hợp phép quay được thực hiện, bit cuối cùng trong double word bị quay ra ngoài sẽ được ghi vào bit đặc biệt SM1.1 (overflow). Bit đặc biệt SM1.0 (zero) sẽ có giá trị 1 (ON) nếu kết quả cuối cùng sau phép quay bằng 0.

Các lệnh ghi quay một double word đều xem các double word là những số không dấu (unsigned).

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

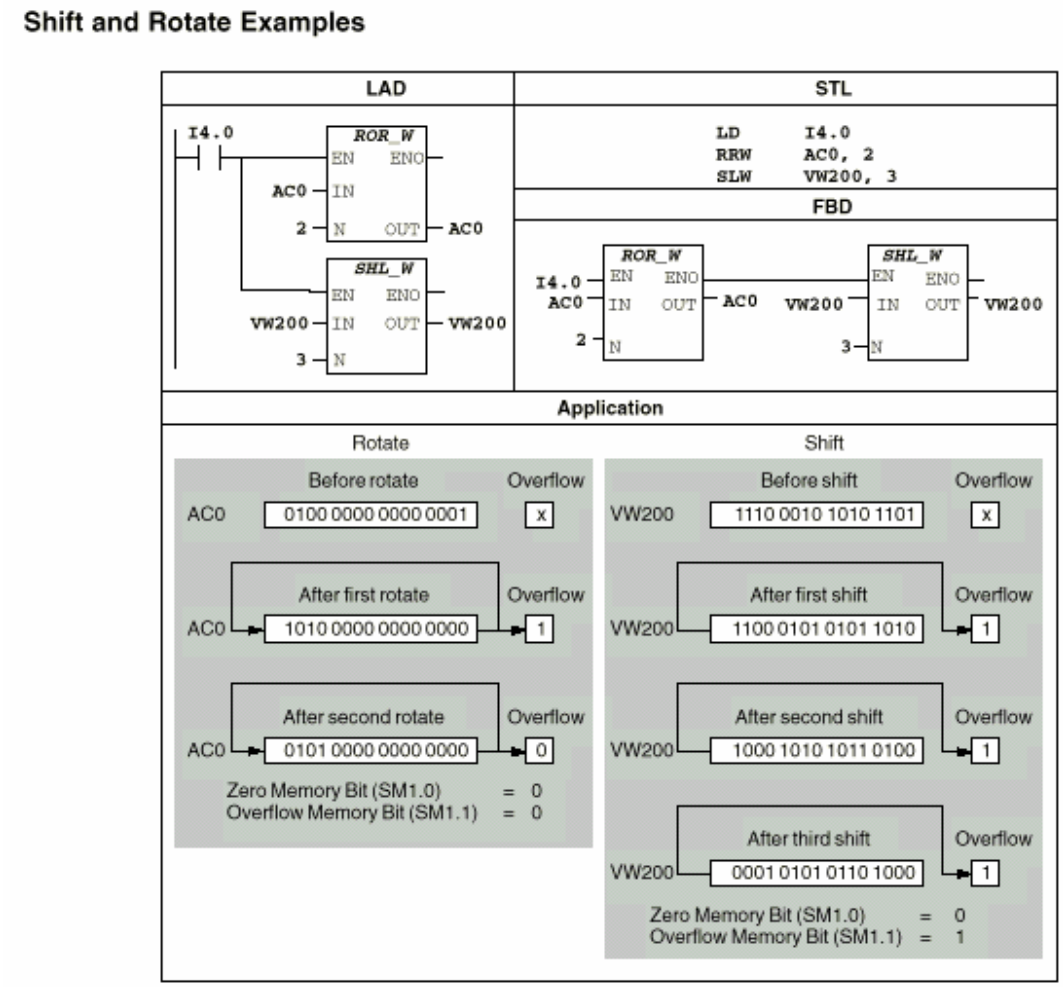
Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.0 (Zero): bằng 1 nếu kết quả bằng 0.

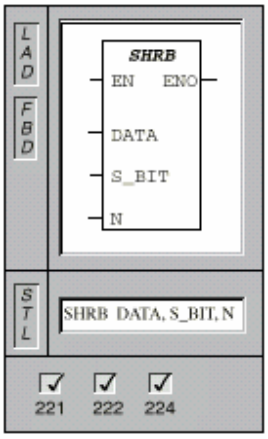
+ SM1.1 (Overflow): bằng bit cuối cùng bị quay ra ngoài.

Inputs/Outputs	Operands	Data Types
IN	VD, ID, QD, MD, SMD, LD, AC, HC, Constant, *VD, *AC, SD, *LD	DWORD
N	VB, IB, QB, MB, SMB, LB, AC, Constant, *VD, *AC, SB, *LD	BYTE
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DWORD

Ví dụ sử dụng các phép dịch và quay:



Lệnh dịch một thanh ghi các bit (Shift Register Bit):



Lệnh này dịch (shift) nội dung một khối các bit liên tiếp đi một bit, với bit đầu tiên bị dịch đi được thay thế bằng giá trị bit được trỏ đến bởi toán hạng DATA và bit cuối cùng bị dịch ra ngoài sẽ được ghi vào bit đặc biệt SM1.1. Khối các bit liên tiếp này được xác định với bit đầu tiên (bit thấp nhất) có địa chỉ định bởi toán hạng S_BIT và có độ dài bằng giá trị tuyệt đối của toán hạng [N]. Điều đó có nghĩa [N] là một số có dấu, dấu của [N] xác định chiều dịch chuyển: [N] dương thì dịch lên còn [N] âm thì dịch xuống.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0006: địa chỉ gián tiếp.

+ Lỗi 0091: toán hạng vượt quá giới hạn cho phép.

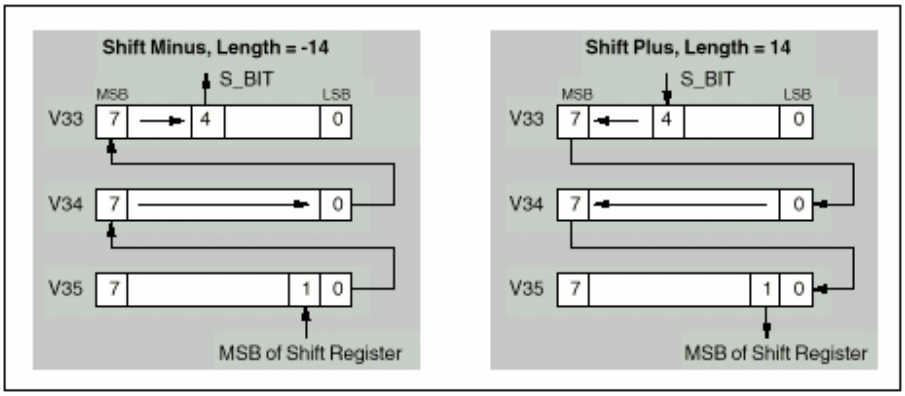
+ Lỗi 0092: lỗi trường số (count field).

Những bit nhớ đặc biệt có nội dung bị ảnh hưởng bởi lệnh này:

+ SM1.1 (Overflow): bằng bit cuối cùng bị quay ra ngoài.

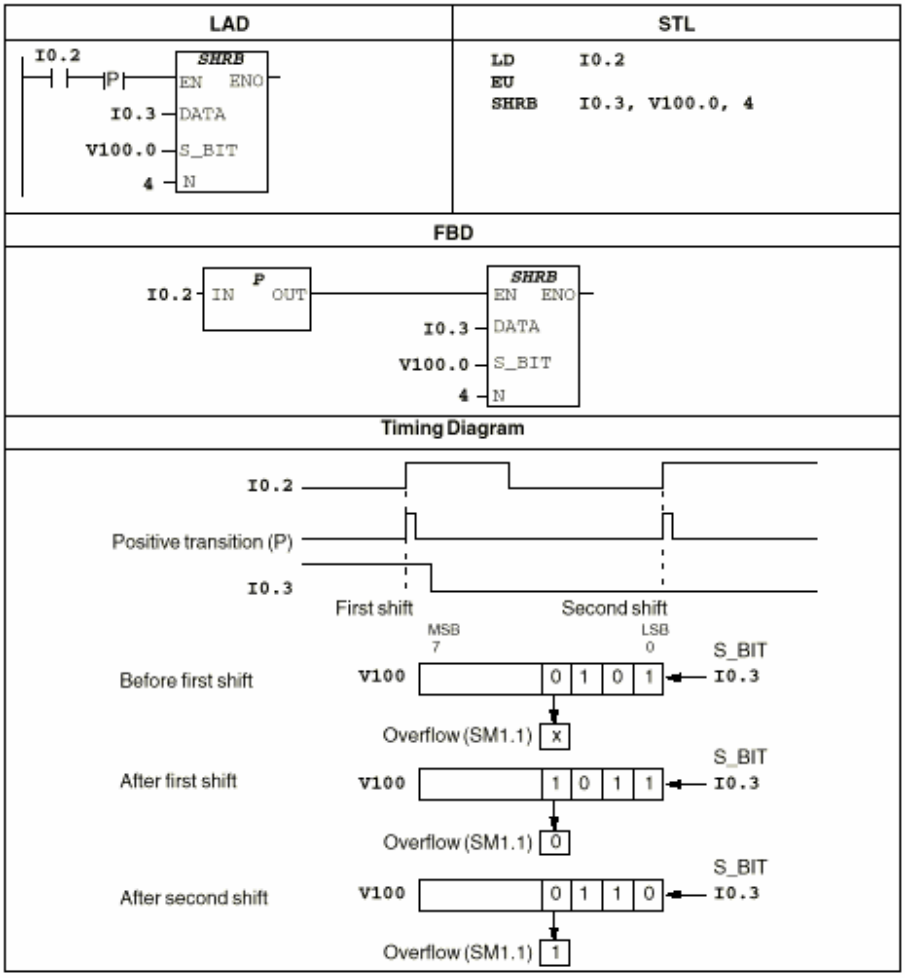
Inputs/Outputs	Operands	Data Types
DATA, S_BIT	I, Q, M, SM, T, C, V, S, L	BOOL
N	VB, IB, QB, MB, SMB, LB, AC, Constant, *VD, *AC, SB, *LD	BYTE

Minh họa lệnh này với khối các bit bắt đầu từ V33.4 và có độ dài 14 bit:

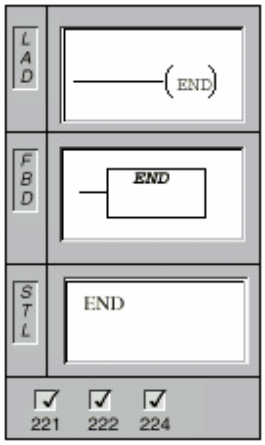


Một ví dụ khác:

Shift Register Bit Example



8.10 Các lệnh điều khiển chương trình

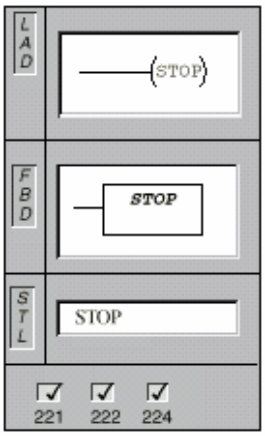


Lệnh END:

Lệnh END có điều kiện dùng để kết thúc chương trình chính khi thỏa mãn điều kiện trước nó.

Lệnh END không có toán hạng, không được sử dụng trong các chương trình con hay trong các chương trình xử lý ngắt.

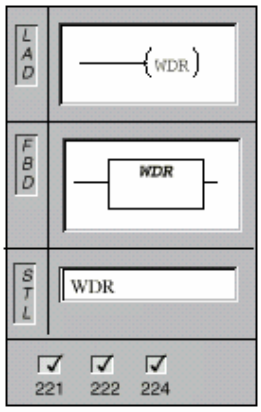
Phần mềm STEP 7 Micro / Win 32 tự động thêm lệnh END không điều kiện (lệnh END không có bất cứ điều kiện nào trước nó) vào cuối mỗi chương trình chính.



Lệnh STOP:

Lệnh STOP dừng chương trình đang được thực hiện ngay lập tức bằng cách chuyển CPU từ chế độ hoạt động (RUN) sang chế độ STOP.

Nếu lệnh STOP được thực hiện từ một chương trình xử lý ngắt thì chương trình xử lý ngắt ấy sẽ bị kết thúc ngay đồng thời tất cả những ngắt đang chờ được xử lý (nếu có) cũng đều bị hủy. Tuy nhiên CPU vẫn xử lý nốt những lệnh còn lại trong vòng quét của chương trình chính khi bị ngắt và chỉ dừng chương trình ở cuối vòng quét bằng cách chuyển chế độ từ RUN sang STOP.



Lệnh WATCHDOG RESET:

Lệnh này khởi động lại đồng hồ canh hệ thống (System Watchdog). Điều đó cho phép kéo dài thời gian thực hiện vòng quét mà không bị lỗi "watchdog".

Chú ý cẩn thận khi sử dụng lệnh này vì khi nó nằm trong các vòng lặp (không kết thúc vòng quét) hay khi nó kéo dài vòng quét sẽ ảnh hưởng tới hệ thống, chẳng hạn như việc thực thi các tính năng:

- Truyền thông (trừ chế độ FreePort)
- Cập nhật các đầu vào ra (trừ những lệnh truy xuất trực tiếp)
- Cập nhật "Forcing"

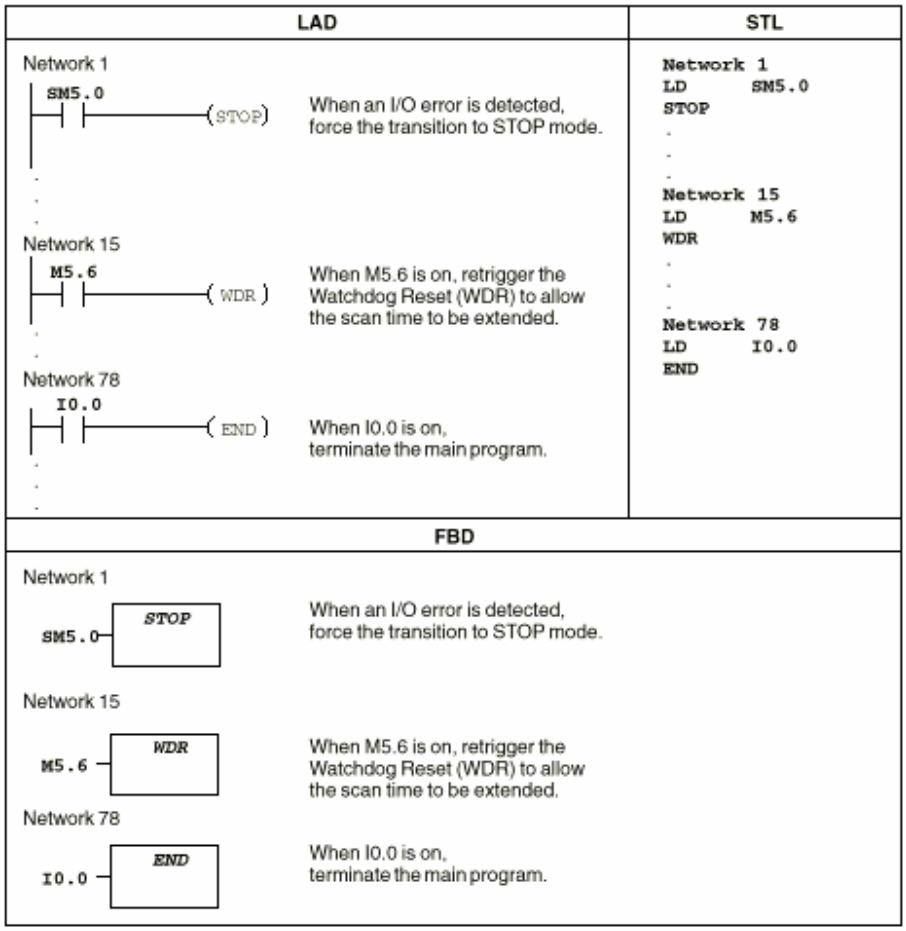
- Cập nhật các bit đặc biệt, như SM0, SM5 đến SM29
- Chẩn đoán lỗi Run-Time
- Các bộ định thời có độ phân giải 10 ms và 100ms hoạt động sai lệch (đặc biệt khi thời gian vòng quét vượt quá 25s)
- Lệnh STOP không được thực hiện khi ở trong chương trình xử lý ngắt

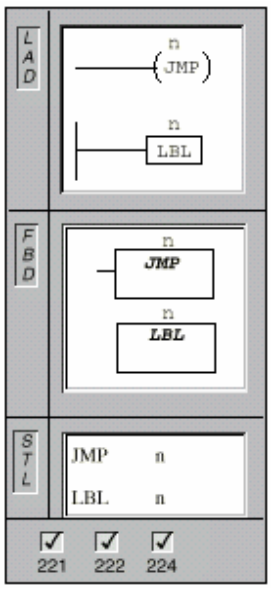
Nếu thời gian vòng quét có thể vượt quá 300ms, hoặc có thể có ngắt kéo dài vòng quét lên quá 300ms, ta phải dùng lệnh WDR.

Việc chuyển công tắc của CPU sang vị trí STOP sẽ dừng chương trình trong vòng 1.4 giây.

Ví dụ sử dụng những lệnh nêu trên:

Stop, End, and WDR Example





<u>Lệnh nhảy (JUMP) và nhãn (LABEL)</u>:

Lệnh nhảy (Jump to Label) rẽ nhánh chương trình đến một đoạn lệnh được đánh dấu bằng một nhãn. Khi một lệnh nhảy được thực hiện, đỉnh ngăn xếp luôn luôn có giá trị 1.

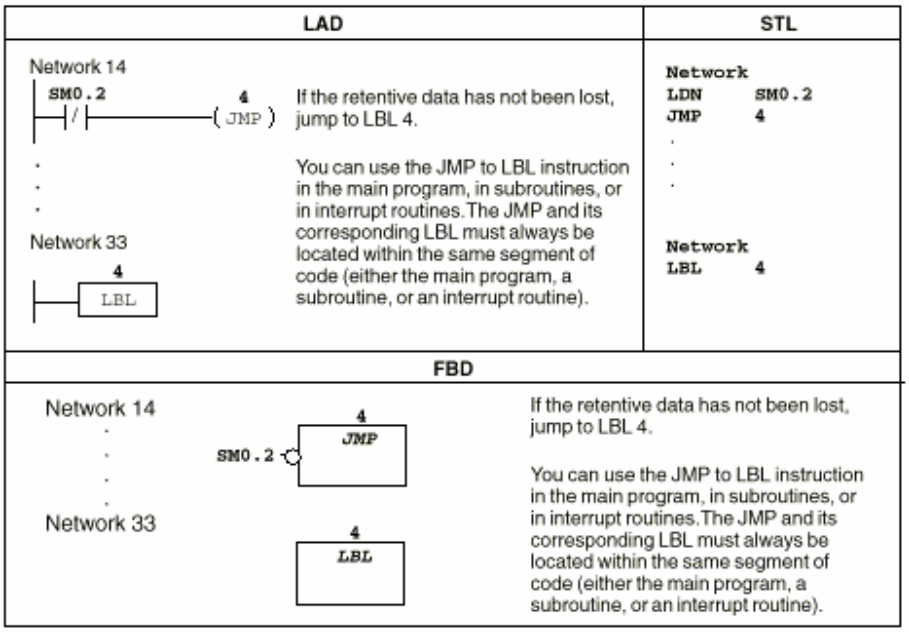
Nhãn dùng để đánh dấu vị trí cho các lệnh nhảy.

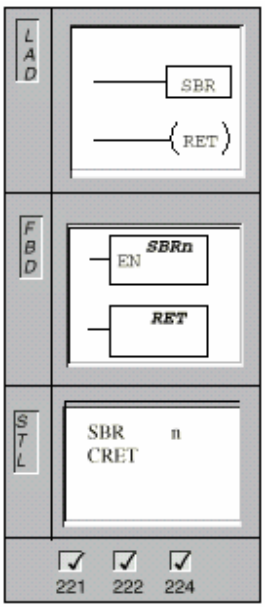
Cả hai lệnh trên có toán hạng là một số nguyên trong khoảng từ 0 đến 255 (số nhãn). Đối với CPU 212 chỉ được từ 0 đến 63.

Lệnh nhảy chỉ được phép rẽ nhánh chương trình đến một nhãn hoặc ở cùng trong chương trình chính, hoặc ở cùng trong một chương trình con hay chương trình xử lý ngắt.

Ví dụ:

Jump to Label Example





Chương trình con:

Lệnh gọi (CALL) một chương trình con chuyển quyền điều khiển đến cho chương trình con đó. S7-200 có thể gọi một chương trình con có hoặc không có tham số. Trong STEP 7 Micro / Win 32, ta thêm chương trình con vào chương trình từ Menu chính **Edit > Insert > Subroutine**.

Lệnh kết thúc chương trình con (Return) có điều kiện kết thúc việc thực hiện chương trình con đó và trở về chương trình chính khi thỏa mãn điều kiện trước nó.

Một khi việc thực hiện một chương trình con kết thúc, quyền điều khiển được chuyển về cho lệnh kế tiếp lệnh gọi chương trình con ấy.

Toán hạng của lệnh gọi chương trình con chính là định danh của chương trình con, là một số nguyên trong khoảng từ 0 đến 255.

Những lỗi có thể được gây nên bởi lệnh này (ENO = 0):

+ Bit đặc biệt SM4.3 = 1: lỗi Run - Time.

+ Lỗi 0008: số lần gọi chương trình con vượt quá con số cho phép.

STEP 7 Micro / Win 32 tự động gắn lệnh kết thúc và trở về từ chương trình con (RET) vào cuối mỗi chương trình con được thêm vào.

Một chương trình con có thể được gọi từ trong một chương trình con, hiện tượng này gọi là Nesting. Độ sâu của Nesting tối đa là 08 lần gọi. Việc gọi đến một chương trình con từ chính nó (đệ qui - Recursion) không bị cấm nhưng người lập trình phải thật sự cẩn trọng với cách dùng này.

Khi gọi một chương trình con, CPU lưu lại toàn bộ ngăn xếp, ghi giá trị 1 vào đỉnh ngăn xếp và 0 vào các giá trị còn lại của ngăn xếp rồi chuyển quyền điều khiển cho chương trình con. Khi việc thực hiện một chương trình con hoàn tất, ngăn xếp được phục hồi lại trạng thái trước đó và quyền điều khiển được chuyển về cho chương trình đã gọi. Lưu ý những thanh ghi đa năng (Accumulators) không được lưu hay phục hồi trong các quá trình trên.

Việc gọi một chương trình con với tham số được thực hiện thông qua việc định nghĩa cho chương trình con một bảng tham số cục bộ. Mỗi tham số bao gồm tên tham số (tối đa 08 ký tự), kiểu biến (vào, ra hay tạm thời) và kiểu dữ liệu (Bool, Byte, INT, ...). Mỗi chương trình con có thể có nhiều nhất 16 tham số.

Kiểu biến của tham số xác định tham số vào cho chương trình con (IN), vừa vào vừa ra (IN_OUT) hay là tham số ra từ chương trình con (OUT). Cụ thể như sau:

- Tham số dạng vào (IN) được truyền đến cho chương trình con: Nếu tham số là địa chỉ trực tiếp (ví dụ VB10), nội dung ô nhớ ở địa chỉ ấy sẽ được truyền vào cho chương

trình con; Nếu tham số là địa chỉ gián tiếp (ví dụ *AC1), nội dung ô nhớ được trỏ đến sẽ được truyền vào cho chương trình con; Nếu tham số là hằng số (ví dụ 16#9A8B) hay là một địa chỉ (ví dụ &VB100), hằng số hay địa chỉ ấy sẽ được truyền vào cho chương trình con.

- Tham số dạng vào - ra (IN_OUT): chương trình con sử dụng số liệu từ địa chỉ xác định bởi tham số này đồng thời xuất dữ liệu cũng ra địa chỉ ấy. Hiển nhiên rằng tham số dạng này không thể là một hằng số (như 16#1234) hay địa chỉ (như &VB100).
- Tham số dạng ra (OUT): chương trình con xuất dữ liệu ra địa chỉ này. Tham số dạng này không thể là một hằng số (như 16#1234) hay địa chỉ (như &VB100).
- Tham số cục bộ (TEMP): là những tham số được chương trình con sử dụng chỉ trong phạm vi chương trình con này.

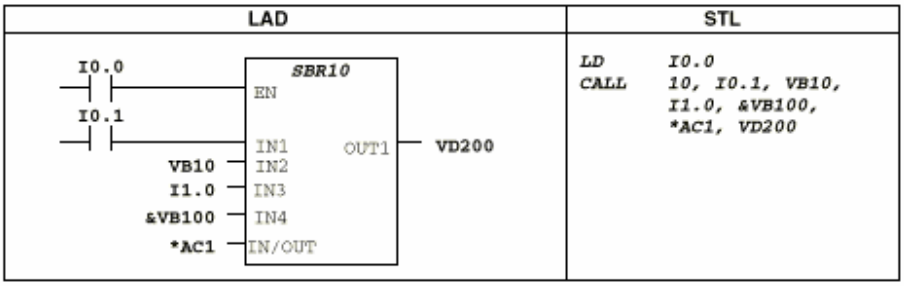
	Name	Var. Type	Data Type	Comment
	EN	IN	BOOL	
L0.0	IN1	IN	BOOL	
LB1	IN2	IN	BYTE	
LB2.0	IN3	IN	BOOL	
LD3	IN4	IN	DWORD	
LW7	IN/OUT1	IN/OUT	WORD	
LD9	OUT1	OUT	DWORD	
		TEMP		

Để thêm vào một tham số cho một chương trình con, trong bảng các tham số ở đầu chương trình con (hình phía trên) đặt con trỏ vào kiểu biến ta muốn thêm (IN, IN/OUT, OUT hay TEMP), nhấn phím phải chuột và chọn **Insert > Row below** để thêm vào một tham số mới ở vị trí dưới con trỏ với dạng tham số thích hợp.

Kiểu dữ liệu của tham số xác định kích thước cũng như định dạng của nó:

- Kiểu dòng năng lượng (Boolean Power Flow): được xem là kiểu bit lô gic nhưng chỉ có thể là dạng vào (IN) và phải được khai báo trước tất cả các kiểu khác (như những tham số EN và IN1 trong ví dụ trên).
- Kiểu bit lô gic (Boolean): đại diện cho một bit, có thể là dạng ra (OUT) hoặc vào (IN), như IN3.
- Kiểu Byte, Word, DWord: tham số ra hoặc vào, 1, 2 hay 4 bytes đại diện cho các số không dấu (unsigned).
- Kiểu Int, DInt: tham số ra hoặc vào, 2 hay 4 bytes đại diện cho các số nguyên có dấu (signed).
- Kiểu Real: tham số ra hoặc vào, đại diện cho các số thực dấu phẩy động 4 bytes (theo chuẩn IEEE).

Một ví dụ gọi chương trình con với các tham số được khai báo như trên:



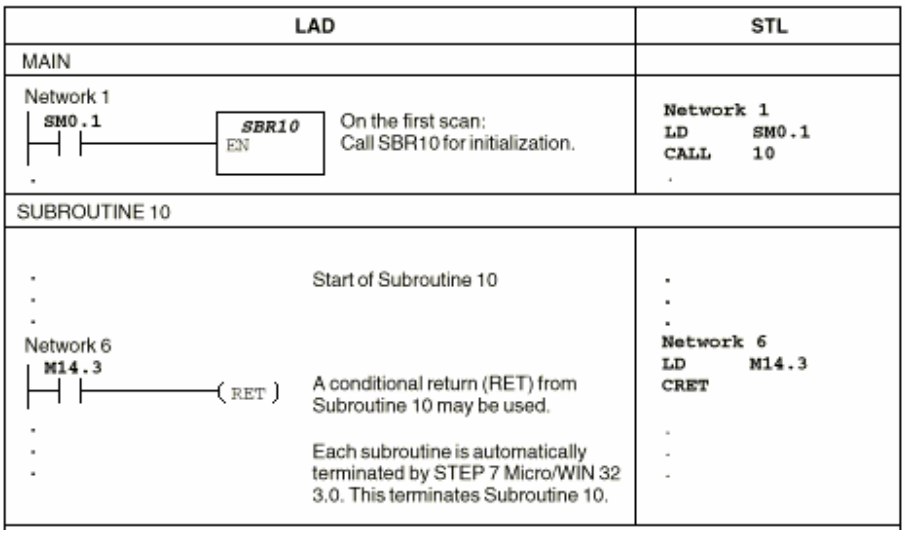
Trong ví dụ trên, tham số IN4 = &VB100 được chứa vào một từ kép (double word unsigned). Nếu gán cho tham số một giá trị là hằng số, 16#1234 chẳng hạn thì phải xác định kiểu dữ liệu cho nó bằng cách viết DW#16#1234.

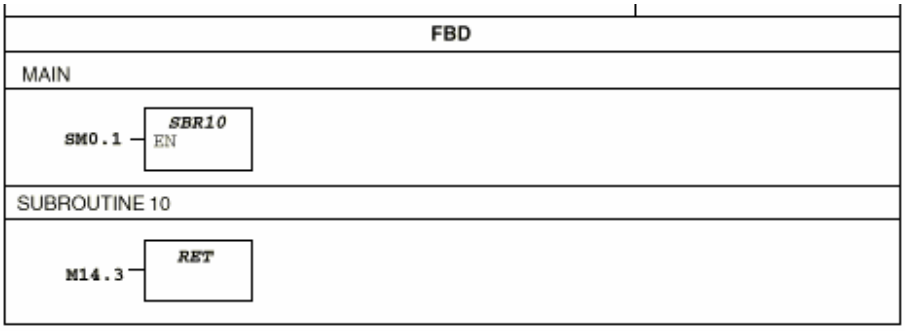
Khi một chương trình con được gọi, nó bao gồm một vùng dữ liệu cục bộ chứa các tham số (được đánh địa chỉ như cột đầu tiên của bảng các tham số). Những tham số dạng vào sẽ được sao chép vào vùng dữ liệu cục bộ này trước khi chương trình con thực hiện và những tham số dạng ra lại được sao chép ra từ vùng ấy sau khi việc thực hiện chương trình con hoàn thành. Lưu ý chương trình con không kiểm tra kiểu dữ liệu nên người lập chương trình phải chú ý sử dụng đúng kiểu đã khai báo.

Tất nhiên thứ tự các tham số cũng phải phù hợp như đã khai báo (đặc biệt trong STL): đầu tiên là dạng vào (IN) rồi đến các dạng vào - ra (IN/OUT) và dạng ra (OUT).

Ví dụ sử dụng chương trình con:

Subroutine, and Return from Subroutine Example





9. TẬP LỆNH IEC 1131-3

10. PHỤ LỤC