

TURKISH REPUBLIC OF NORTHERN CYPRUS

NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

**DEPARTMENT OF ELECTRICAL & ELECTRONIC
ENGINEERING**

DEGREE OF BSc

EE 400 GRADUATION PROJECT

PROGRAMMABLE LOGIC CONTROLLERS

**SUBMITTED BY : Mr. GÜRTÜRK KÖSEOĞLU , 940192
Mr. ERDAL YILDIZ , 950557**

SUBMITTED TO : Mr. ÖZGÜR C.ÖZERDEM

NICOSIA-2000

INDEX

1.LIST OF FIGURES	
1.1. Entrance of the Packing House	1
1.2. Entrance of the Packing House	1
1.3. Pesticide and Fungicide Application Unit	2
1.4. Sizing Unit	2
1.5. Carring Converyor and Stamping and Stabling Unit	3
1.6. Table of Symbol	4
2. INTRIDUCTION	5
3. WHAT IS PLC?	6
4. PLC HISTORY	7
5. GENAREL PHYSICAL BUILD MECHANISM	8
5.1. Compact PLCs	8
5.2. Modular PLC	8
6. INTERNAL STRACTURE OF PLCs	9
6.1. Input Unit	9
6.2. Output Unit	9
6.3. Processing Unit	9
7. ADVANTAGE	10
7.1. Accuracy	10
7.2. Flexibility	10
7.3. Communication	10
7.4. Logic Control of Industrial Automation	10
7.5. Reals and Ladder Logic	10
7.6. System Overvies	11
7.7. CPU Overview	11
7.8. Architecture	11
7.9. Memory Map	11
7.9.1. Data Area	12
7.9.2. Data Objects	12
8. LADDER AND STL PROGRAM	13
9. DESCRIPTION OF OPERATION	37
10. CONCLUSION	73
11. APPENDIXS	74
12. REFERENCES	77

1. LIST OF FIGURES

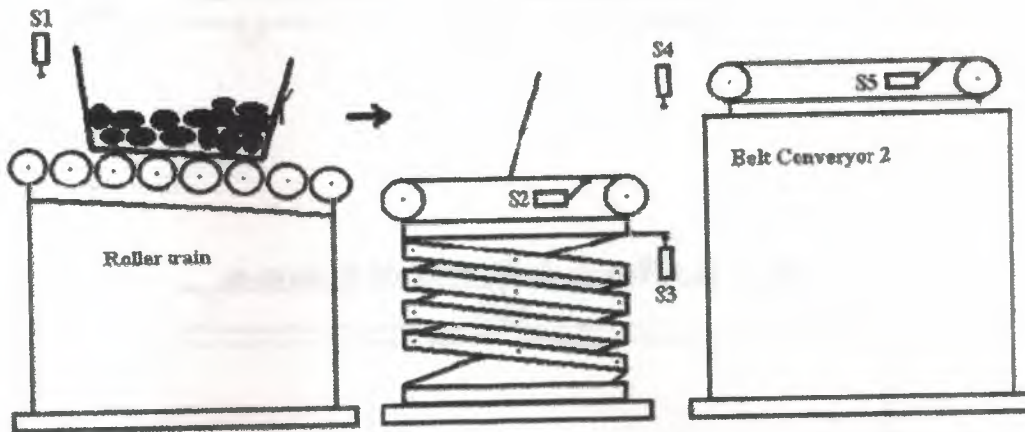


Figure 1.1 : Entrance of the Packing House

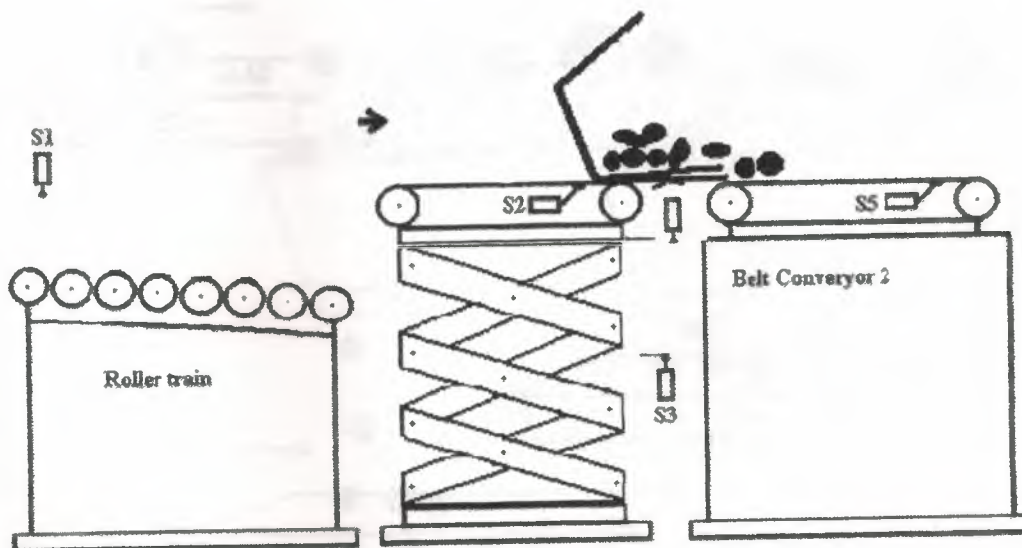


Figure 1.2 : Entrance of the Packing House

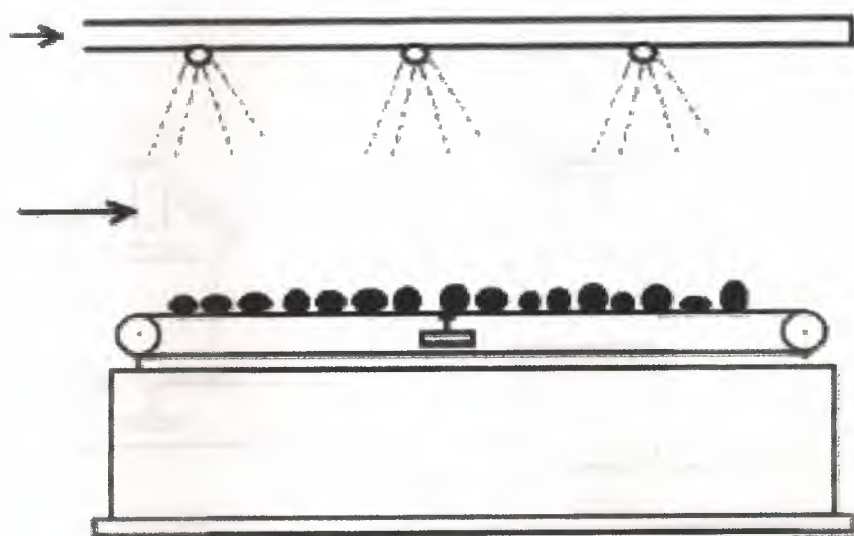


Figure 1.3 : Pesticide and Fungicide Application Unit

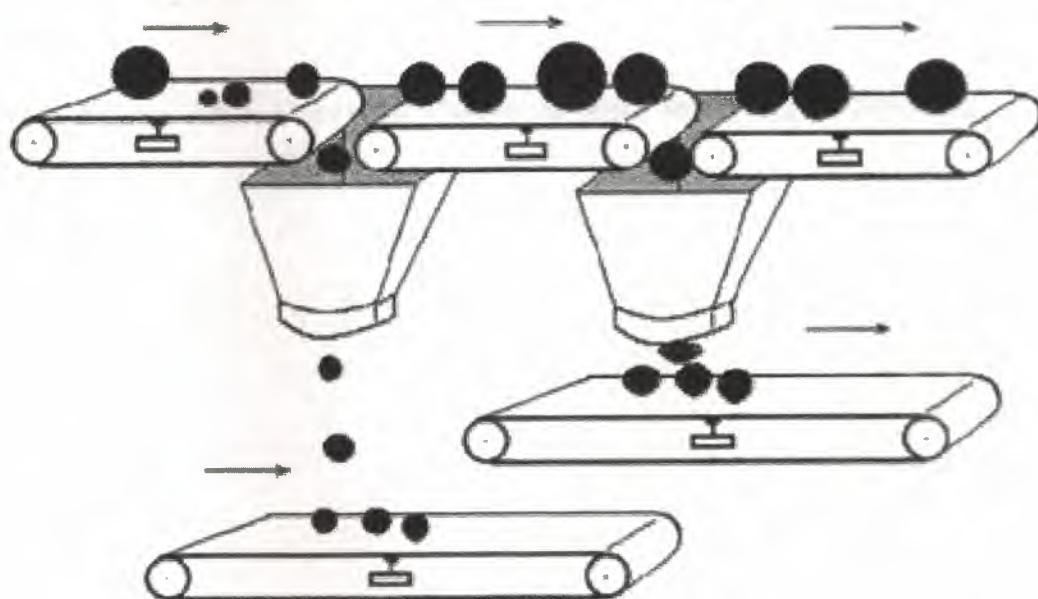


Figure 1.4: Sizing Unit

Stamping Unit

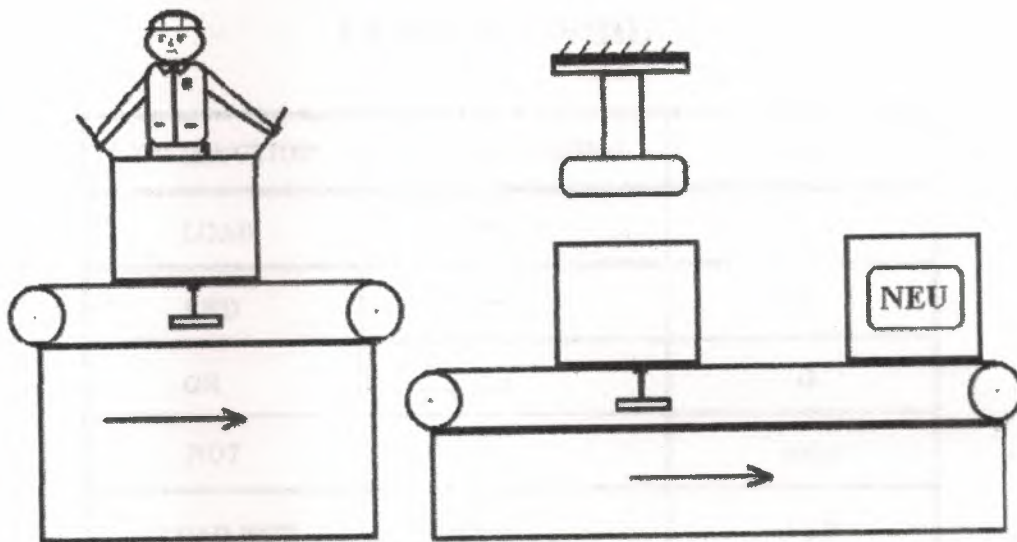


Figure 1.5 : Carrying Conveyor and Stamping & Stabling Unit

Table of Symbol









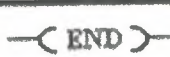
INSTRUCTION	LADDER SEMBOL	SIMATIC S7
LOAD		LD
AND		A
OR		O
NOT		NOT
LOAD NOT		LDN
AND NOT		AN
OR NOT		ON
AND BLOCK		ALD
OR BLOCK		OLD
OUT		=
END		MEND

Figure 1.6

2. INTRODUCTIONS

Now that we understand how inputs and outputs are processed by the PLC, let's look at a variation of our regular outputs. Regular output coils are of course an essential part of our programs but we must remember that they are only true when all instructions before them on the rung are also true.

Think back to the we did a few chapters ago. What would've happened if we couldn't find a "push on/push off" switch? Then we would've had to keep pressing the button for as long as we wanted the bell to sound. (A momentary switch) The latching instructions let us use momentary switches and program the PLC so that when we push one the output turns on and when we push another the output turns off.

Picture the remote control for your TV. It has a button for on and another for off. When I push the on button the TV turns on. When I push the off button the TV turns off. I don't have to keep pushing the on button to keep the TV on. This would be the function of a latching instruction.

The latch instruction is often called a SET or OTL (output latch). The unlatch instruction is often called a RES (reset), OUT (output unlatch) or RST (reset). The diagram below shows how to use them in a program.

3. WHAT IS A PLC ?

A PLC (ie. Programmable Logic Controller) is a device that was invented to replace the necessary sequential relay circuits for machine control. The PLC works by looking at its inputs and depending upon their state, turning on/off its outputs. The user enters a program, usually via software, that gives the desired results.

PLC 's are used in many "real world" applications. If there is industry present, chances are good that there is a PLC present. If you are involved in machining, packaging, material handling, automated assembly or countless other industries you are probably already using them. If you are not, you are wasting money and time. Almost any application that needs some type of electrical control has a need for a PLC.

For example, let's assume that when a switch turns on we want to turn a solenoid on for 5 seconds and then turn it off regardless of how long the switch is on for. We can do this with a simple external timer. But what if the process included 10 switches and solenoids? We would need 10 external timers. What if the process also needed to count how many times the switches individually turned on? We need a lot of external counters.

As you can see the bigger the process the more of a need we have for a PLC. We can simply program the PLC to count its inputs and turn the solenoids on for the specified time.

This site gives you enough information to be able to write programs far more complicated than the simple one above. We will take a look at what is considered to be the "top 20" PLC instructions. It can be safely estimated that with a firm understanding of these instructions one can solve more than 80% of the applications in existence.

4. PLC HISTORY

In the late 1960's PLC 's were first introduced. The primary reason for designing such a device was eliminating the large cost involved in replacing the complicated relay based machine control systems. Bedford Associates (Bedford, MA) proposed something called a Modular Digital Controller (MODICON) to a major US car manufacturer. Other companies at the time proposed computer based schemes, one of which was based upon the PDP-8. The MODICON 084 brought the world's first PLC into commercial production.

When production requirements changed so did the control system. This becomes very expensive when the change is frequent. Since relays are mechanical devices they also have a limited lifetime which required strict adherence to maintenance schedules. Troubleshooting was also quite tedious when so many relays are involved. Now picture a machine control panel that included many, possibly hundreds or thousands, of individual relays. The size could be mind boggling. How about the complicated initial wiring of so many individual devices! These relays would be individually wired together in a manner that would yield the desired outcome.

These "new controllers" also had to be easily programmed by maintenance and plant engineers. The lifetime had to be long and programming changes easily performed. They also had to survive the harsh industrial environment. That's a lot to ask! The answers were to use a programming technique most people were already familiar with and replace mechanical parts with solid-state ones.

In the mid70's the dominant PLC technologies were sequencer state-machines and the bit-slice based CPU. The AMD 2901 and 2903 were quite popular in MODICON and A-B PLC 's. Conventional microprocessors lacked the power to quickly solve PLC logic in all but the smallest PLC 's. As conventional microprocessors evolved, larger and larger PLC 's were being based upon them. However, even today some are still based upon the 2903.(ref A-B 's PLC-3) MODICON has yet to build a faster PLC than their 984A/B/X which was based upon the 2901.

Communications abilities began to appear in approximately 1973. The first such system was MODICON 's MODBUS. The PLC could now talk to other PLC 's and they could be far away from the actual machine they were controlling. They could also now be used to send and receive varying voltages to allow them to enter the analog world. Unfortunately, the lack of standardisation coupled with continually changing technology has made PLC communications a nightmare of incompatible protocols and physical networks.

The 80's saw an attempt to standardise communications with General Motor's manufacturing automation protocol (MAP). It was also a time for reducing the size of the PLC and making them software programmable through symbolic programming on personal computers instead of dedicated programming terminals or handheld programmers.

The 90's have seen a gradual reduction in the introduction of new protocols, and the modernisation of the physical layers of some of the more popular protocols that survived the 1980's. The latest standard has tried to merge PLC-programming languages under one international standard. We now have PLC 's that are programmable in function block diagrams, instruction lists, C and structured text all at the same time! PC 's are also being used to replace PLC 's in some applications. The original company who commissioned the MODICON 084 has actually switched to a PC based control system.

5. GENAREL PHYSICAL BUILD MECHANISM

PLC 's are separated into two according to their building mechanisms.

5.1. Compact PLC 's

Compact PLC 's are manufactured such that all units forming the PLC are placed in a case. They are low price PLC with lower capacity. They are usually preferred by small or medium size machine manufacturers. In some types compact enlargement module is present.

5.2. Modular PLC 's

They are formed by combining separate modules (called RACK) together in a board. They can have different memory capacity, I/O numbers, Power Supply up to the necessary limits.

Some examples: SIEMENS S5-115U, KLOCKNER-MOELLER PS316 OMRON C200H.

6. INTERNAL STRUCTURE OF PLC 's:

They have three main units:

1. Input unit
2. Processing unit
3. Output unit

6.1. INPUT UNIT:

Is the that converts the signals coming from the control elements of the system that is going to be controlled into logic levels.

The analog and/or digital signals coming from the sensors or switches showing the systems pressure, humidity, level, etc. enters the PLC through the input unit.

Digital signals are converted to 5V dc by this unit which is the internal voltage level of the device.

The parasitic signals are first filtered by RC passive filters and than they pass through up to coupler that has the property to supply galvanised isolation. As a result of this process the signals are send to input display memory. Analog signals pass through this process the signals are send to input display memory. Analog signals pass through frequency converts in some PLC 's. In this way they gain important noise immunity.

6.2. OUTPUT UNITS:

They are suitably manufactured to successfully control the activators in the system to be controlled. Digital output signals contractor relays, 24V dc NPN or PNP transistors or Tracs, PLC 's output cannot supply large currents. So by digital output relays and by their contactor groups main contactors or windings are operated. In this way unit like motors, heaters, hydraulic values can be operated.

6.3. PROCESSING UNIT:

It is composed of the sub units given below:

- **CPU (Central Processing Unit) :** It is also given the name processing unit. It processes all the input signals according to the user program instruction order and directs the output signals to the related outputs. This process is controlled by a microprocessor. Some times instead of microprocessor a micro controller or microcomputer can also be-used. The difference of these devices from microprocessor is that processor; memory and I/O interfaces are all in one unit. As a memory ROM and RAM is used. Data for Operating System and PLC that cannot be changed are kept in ROM and user program and I/O data are kept in RAM.
- **Program Memory:** It is also defined as user memory it is the memory where the user program is kept. Its capacity is variable according to the instruction number. Eg. 1K instruction = 1024 instruction lines.

7. ADVANTAGES

7.1. ACCURACY

In relay control systems logical knowledge's carries in electro mechanical contactors, they can lose data because of mechanical errors. But PLC 's are microprocessor based system so logical data are carried inside the processor, so that PLC 's are more accurate than relay type of controllers.

7.2. FLEXIBILITY

When there is need of any change in control, relay type of controllers modification are hard, in PLC this change can be made with PLC programmer equipment.

7.3. COMMUNICATION

PLC' s are computer based systems. So that they can transfers their data to another PC or they can take external inputs from another PC, with this specification we can control the system were they are we can effect the system with our PC. With relays this is not possible.

7.4. LOGIC CONTROL OF INDUSTRIAL AUTOMATION

Everyday examples of these systems are machines like dishwashers, clothes washers and dryers, and elevators. In these systems, the outputs tend to be 220vac power signals to motors, solenoids, and indicator lights, and the inputs are DC or AC signals from user interface switches, motion limit switches, binary liquid level sensors, etc. Another major function in these types of controllers is timing.

7.5. REALS AND LADDER LOGIC

In the "old days" (ie. before the 1980's) these types of controllers were implemented with relays. Relays are a technology from the early days of electricity in which an electromagnet activates an electrical switch. When current flows in the coil, electrically, thermally, and mechanically rugged, easy to design with, cheap, and capable of controlling very large currents in their output contacts.

Relays can be thought of as logic gates. For example, if two normally open relays are wired in series, and one end of the resulting output circuit is attached to a voltage source, then the two coils form the inputs of a AND gate: only if current is flowing in BOTH input coils will current flow in the output circuit. A typical application in a washing machine might be to implement the rule that.

The shape of these diagrams invariably led to the name "Ladder diagrams" and "Ladder Logic" to describe them. The term "Relay Ladder Logic" (RLL) describes this logic notation. By including interconnections between the horizontal rungs, it is possible to create latches ("flip-flops") and implement state transitions. Although LL "state machines" get quite complex and are typically not designed with the convenience of finite state machine theory, they have become widely used and supported by technical workers. Because the logic was implemented in physical wiring, it was difficult to change, as new functions were required.

7.6. SYSTEM OVERVIEWS

A typical S7-200 system will include an S7-200 base unit which includes the central processing unit, power supply, and discrete input and output points. Expansion module contains additional input or output points and is connected to the base unit bus connectors. The central processing unit has a built-in communications port for programming or talking with intelligent ASCII devices.

7.7. CPU OVERVIEW

The S7-200 series is a line of small, compact, micro-programmable logic controllers and expansion modules that can be used for a variety of programming applications. There are two types of base units in the S7-200 product line, CPU 212 and CPU 214. Each base unit comes in different models to accommodate the type of power supply, inputs and outputs you require.

7.8. ARCHIECTURE

This section relates to how the S7-200 CPU arranges data and how it executes your program during it's scan cycle.

7.9. MEMORY MAP

The memory space of the S7-200 is divided into five data areas and six data objects. To reference a memory location for use, you must address that location. The addressing conventions allow memory to be accessed as bits, bytes, words and double words. All addresses are zero-based.

Data space is highly flexible, and it allows read and writes access to all memory areas as bits, bytes, words and double words. Data objects are the memory locations that are associated with devices (such as the current value of a counter or the temperature value of an oven). Access to data objects is more restrictive because the data object can be addressed only according to the intended use of that object.

7.9.1. Data Areas

Data memory contains variable memory, and register, and output image register, internal memory bits, and special memory bits. This memory is accessed by a byte bit convention. For example to access bit 3 of Variable Memory byte 25 you would use the address V25.3.

The following table shows the identifiers and ranges for each of the data area memory types:

Area Identifier	Data Area	CPU 212	CPU 214
I	Input	I0.0 to I7.7	I0.0 to I7.7
Q	Output	Q0.0 to I7.7	Q0.0 to Q7.7
M	Internal Memory	M0.0 to M15.7	M0.0 to M31.7
SM	Special Memory	SM0.0 to SM 45.7	SM0.0 to SM 85.7
V	Variable Memory	V0.0 to V1023.7	V0.0 to V4095.7

7.9.2. Data Objects

The S7-200 has six kinds of devices with associated data: timers, counters, analog inputs, analog outputs, accumulators and high-speed counters. Each device has associated data (data objects). For example, the S7-200 has counter devices. Counters have a data value that maintains the current count value. There is also a bit value, which is set when the current value is greater than or equal to the present value. Since there are multiple devices are numbered from 0 to n. the corresponding data objects and object bits are also numbered.

The following table shows the identifiers and ranges for each of the data object memory types:

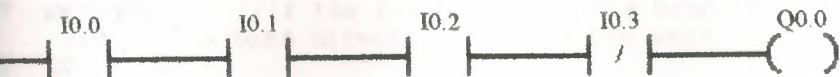
Object Identifier	Object	CPU 212	CPU214
T	Timers	T0 to T63	T0 to T127
C	Counters	C0 to C63	C0 to C127
AI	Analog Input	AIW0 to AIW30	AIW0 to AIW30
AQ	Analog Output	AQW0 to AQW30	AQW0toAQW30
AC	Accumulator Registers	AC0 to AC3	AC0 to AC3
HC	High-speed Counter Current	HC0	HC0 to HC2

PROGRAM TITLE COMMENTS

Press F1 for help and example program

Network 1 If we don't push the stop button and if the second sensor can see the fruit.

NETWORK COMMENT



Network 2 If the fruits com on the band the sifth sensor will see the fruits and band directly starting the work



Network 3 The end of work.



```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1    //If we don't push the stop button and if the second sensor can
               see the fruit.
8  //
9  //NETWORK COMMENT
10 //
11 LD          I0.0
12 A           I0.1
13 A           I0.2
14 AN          I0.3
15 =           Q0.0
16
17 NETWORK 2    //If the fruits com on the band the sifth sensor will see the
               fruits and band directly starting the work
18 LD          I0.4
19 =           Q0.1
20
21 NETWORK 3    //The end of work.
22 MEND

```


PROGRAM TITLE COMMENTS

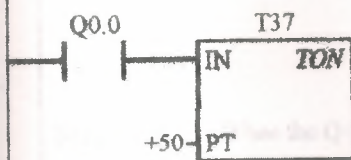
Press F1 for help and example program

Network 1 The bant begins to work after the sensor see's the fruits.

NETWORK COMMENTS



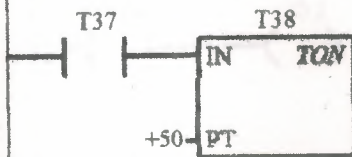
Network 2 After the band works the time counts 5 sec T37



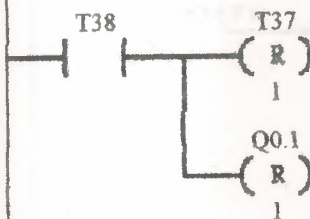
Network 3 When timer's is active it sprays chemical.



Network 4 The timers makes the other timer work T38



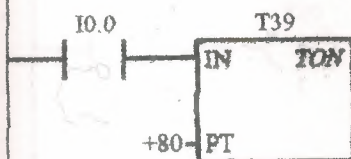
Network 5 When the other timer is active T37 and Q 0.1 resets.



Network 6 When the Q 0.1 doesn't work, T38 resets.



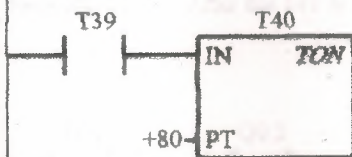
Network 7 If the band is working the timer (T39) counts 8 sec.



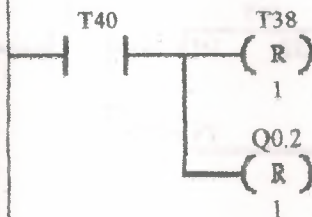
Network 8 After the T39 is active it begins to spray chemical.



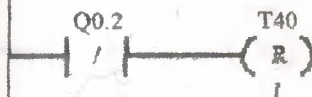
Network 9 The timer makes the other time work (T40)



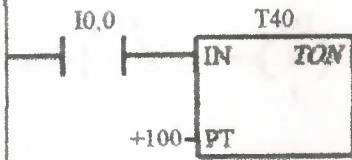
Network 10 When the other time is active T38 and Q 0.2 resets.



Network 11 When the Q0.2 doesn't work T40 resets.



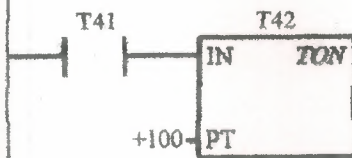
Network 12 If the band is working the timer (T41) counts (10 sec.)



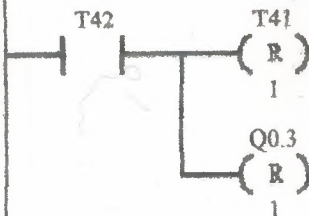
Network 13 After the T41 is active it begins to spray chemical.



Network 14 The timer makes the other time work(T42)



Network 15 When the other time is active T41 and Q0.3 resets.



Network 16 When the Q0.2 doesn't work T42 resets.



Network 17 The end of work



```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1    //The bant begins to work after the sensor see's the fruits.
8  //
9  //NETWORK COMMENTS
10 //
11 LD      I0.0
12 =       Q0.0
13
14 NETWORK 2    //After the band works the time counts 5 sec T37
15 LD      Q0.0
16 TON     T37, +50
17
18 NETWORK 3    //When timer's is active it sprays chemical.
19 LD      T37
20 =       Q0.1
21
22 NETWORK 4    //The timers makes the other timer work T38
23 LD      T37
24 TON     T38, +50
25
26 NETWORK 5    //When the other timer is active T37 and Q 0.1 resets.
27 LD      T38
28 R       T37, 1
29 R       Q0.1, 1
30
31 NETWORK 6    //When the Q 0.1 doesn't work, T38 resets.
32 LDN     Q0.1
33 R       T38, 1
34
35 NETWORK 7    //If the band is working the timer (T39) counts 8 sec.
36 LD      I0.0
37 TON     T39, +80
38
39 NETWORK 8    //After the T39 is active it begins to spray chemical.
40 LD      T39
41 =       Q0.2
42
43 NETWORK 9    //The timer makes the other time work (T40)
44 LD      T39
45 TON     T40, +80
46
47 NETWORK 10   //When the other time is active T38 and Q 0.2 resets.
48 LD      T40
49 R       T38, 1
50 R       Q0.2, 1
51
52 NETWORK 11   //When the Q0.2 doesn't work T40 resets.
53 LDN     Q0.2
54 R       T40, 1
55
56 NETWORK 12   //If the band is working the timer (T41) counts (10 sec.)
57 LD      I0.0
58 TON     T40, +100
59
60 NETWORK 13   //After the T41 is active it begins to spray chemical.
61 LD      T41
62 =       Q0.3
63
64 NETWORK 14   //The timer makes the other time work(T42)
65 LD      T41
66 TON     T42, +100
67
68 NETWORK 15   //When the other time is active T41 and Q0.3 resets.
69 LD      T42
70 R       T41, 1

```

```

71 R      Q0.3, 1
72
73 NETWORK 16 //When the Q0.2 doesn't work T42 resets.
74 LDN    Q0.3
75 R      T42, 1
76
77 NETWORK 17 //The end of work
78 MEND

```

COMMENTS



Example work for...



Example work for...

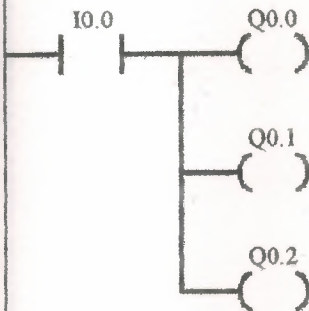


PROGRAM TITLE COMMENTS

Press F1 for help and example program

Network 1 The three bonts begin to work when the twits come on the bant according to there dimension.

NETWORK COMMENTS



Network 2 Sensor see's the fruit it works.



Network 3 Sensor see's the fruit it works.



Network 4 The end of work.

(END)

```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1  //The three bonts begin to work when the twits come on the bant
   according to there dimension.
8  //
9  //NETWORK COMMENTS
10 //
11 LD      IO.0
12 =      Q0.0
13 =      Q0.1
14 =      Q0.2
15
16 NETWORK 2  //Sensor see's the fruit it works.
17 LD      IO.1
18 =      Q0.3
19
20 NETWORK 3  //Sensor see's the fruit it works.
21 LD      IO.2
22 =      Q0.4
23
24 NETWORK 4  //The end of work.
25 MEND

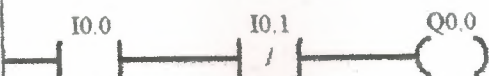
```

PROGRAM TITLE COMMENTS

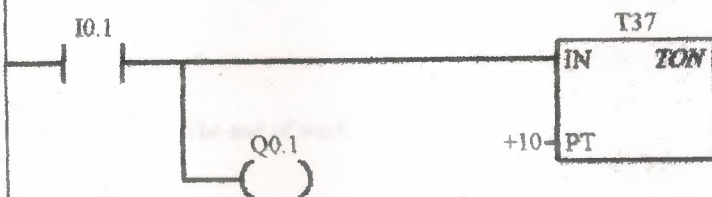
Press F1 for help and example program

Network 1 Sensor see's the fruit and the stemp and seal doesn't work the band should work.

NETWORK COMMENTS



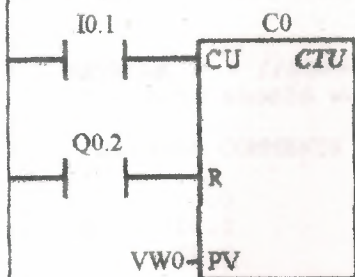
Network 2 When the box is seen, the timer waits 1 sec and the stamp and seel to work.



Network 3 After 1 sec of timer the band to work.



Network 4 The stamped and sealed boxes to be counted.



Network 5 When out of the normal counter the counted by the digital counter.



Network 6 The end of work.



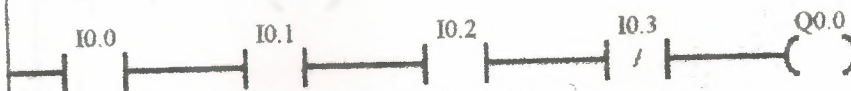
```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1    //Sensor see's the fruit and the stemp and seal doesn't work
               the band should work.
8  //
9  //NETWORK COMMENTS
10 //
11 LD      I0.0
12 AN      I0.1
13 =       Q0.0
14
15 NETWORK 2    //When the box is seen, the timer waits 1 sec and the stamp and
               seel to work.
16 LD      I0.1
17 TON     T37, +10
18 =       Q0.1
19
20 NETWORK 3    //After 1 sec of timer the band to work.
21 LD      T37
22 =       Q0.2
23
24 NETWORK 4    //The stemped and sealed boxes to be counted.
25 LD      I0.1
26 LD      Q0.2
27 CTU     C0, VW0
28
29 NETWORK 5    //When out of the normal counter the conted by the digital
               counter.
30 LD      C0
31 =       Q0.3
32
33 NETWORK 6    //The end of work.
34 MEND

```


PROGRAM TITLE COMMENTS

Press F1 for help and example program



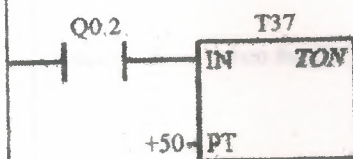
Network 2 If the fruit com on the band. If the sensor will see the fruits and the band directly starting the work.



Network 3 The bant begins to work after sensor seen's the fruits.



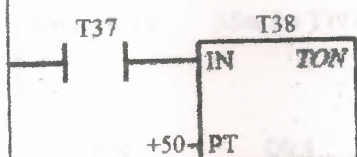
Network 4 After the band works the time counts 5 sec T37.



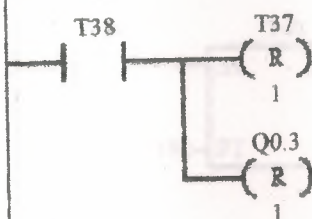
Network 5 When the timer is active it sprays chemical.



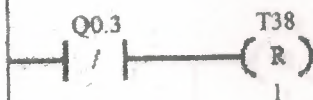
Network 6 The timer makes the other timer work T38.



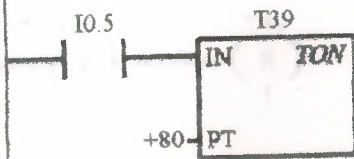
Network 7 When the other timer is active T37 and Q0.3 resets.



Network 8 when the Q0.3 doesn't work T38 resets.



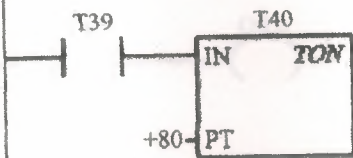
Network 9 If the band is working the timer (T38) counts 8 sec.



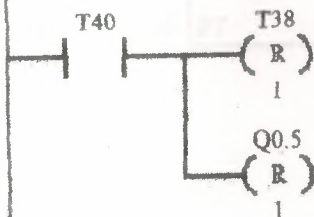
Network 10 After the T39 is active it begins to spray chemical.



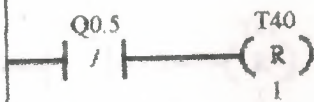
Network 11 The timer makes the other time work(T40)



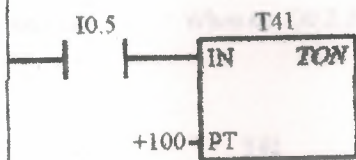
Network 12 When the other time is active T38 and Q0.05 resets.



Network 13 When the Q0.5 doesn't work T40 resets.



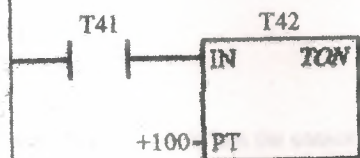
Network 14 If the band is working the timer (T41) counts (10 sec).



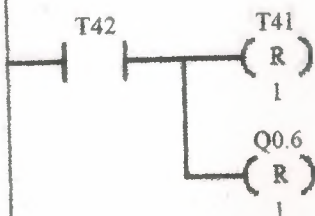
Network 15 After the T41 is active it begins to spray chemical.



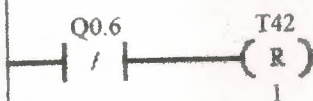
Network 16 The timer makes the other time work(T42).



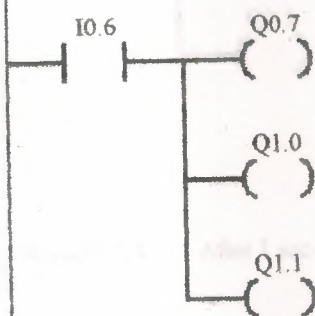
Network 17 When the other time is active T41 and Q0.6 resets.



Network 18 When the Q0.2 doesn't work T42 resets.



Network 19 The three bant begin to work when the fruits come on the bant according to there dimension.



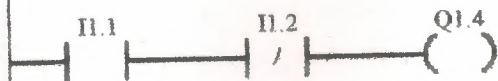
Network 20 When the sensor see's the fruit it works.



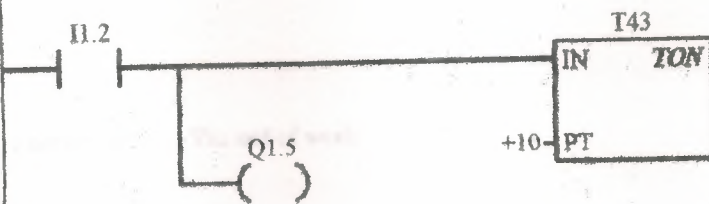
Network 21 When the sensor see's fruit it works.



Network 22 Sensor see's the fruit it works and the stemp and seal doesn't work then the band should work.



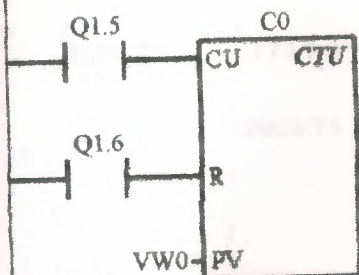
Network 23 When the box is seen, the timer waits 1 sec and the stamp and seal to work.



Network 24 After 1 sec of timer the band to work.



Network 25 The stamped and sealed boxes to be counted.



Network 26 When out the normal counter by the digital counter.



Network 27 The and of work.

(END)

```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1  //If we don't push the stop button and if the sensor can see
   the fruit.
8  //
9  //NETWORK COMMENTS
10 //
11 LD      I0.0
12 A       I0.1
13 A       I0.2
14 AN      I0.3
15 =       Q0.0
16
17 NETWORK 2  //If the fruit com on the band. if the sensor will see the
   fruits and the band directly starting the work.
18 LD      I0.4
19 =       Q0.1
20
21 NETWORK 3  //The bant begins to work after sensor seen's the fruits.
22 LD      I0.5
23 =       Q0.2
24
25 NETWORK 4  //After the band works the time counts 5 sec T37.
26 LD      Q0.2
27 TON     T37, +50
28
29 NETWORK 5  //When the timer is active it sprays chemical.
30 LD      T37
31 =       Q0.3
32
33 NETWORK 6  //The timers makes the other timer work T38.
34 LD      T37
35 TON     T38, +50
36
37 NETWORK 7  //When the other timer is active T37 and Q0.3 resets.
38 LD      T38
39 R       T37, 1
40 R       Q0.3, 1
41
42 NETWORK 8  //when the Q0.3 doesn't work T38 resets.
43 LDN     Q0.3
44 R       T38, 1
45
46 NETWORK 9  //If the band is working the timer (T38) counts 8 sec.
47 LD      I0.5
48 TON     T39, +80
49
50 NETWORK 10 //After the T39 is active it begins to spray chemical.
51 LD      T39
52 =       Q0.4
53
54 NETWORK 11 //The timer makes the other time work(T40)
55 LD      T39
56 TON     T40, +80
57
58 NETWORK 12 //When the other time is active T38 and Q0.5 resets.
59 LD      T40
60 R       T38, 1
61 R       Q0.5, 1
62
63 NETWORK 13 //When the Q0.5 doesn't work T40 resets.
64 LDN     Q0.5
65 R       T40, 1
66
67 NETWORK 14 //If the band is working the timer (T41) counts (10 sec).
68 LD      I0.5

```



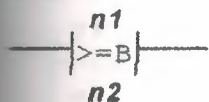
```

69 TON      T41, +100
70
71 NETWORK 15 //After the T41 is active it begins to spray chemical.
72 LD      T41
73 =      Q0.6
74
75 NETWORK 16 //The timer makes the other time work(T42).
76 LD      T41
77 TON      T42, +100
78
79 NETWORK 17 //When the other time is active T41 and Q0.6 resets.
80 LD      T42
81 R      T41, 1
82 R      Q0.6, 1
83
84 NETWORK 18 //When the Q0.6 doesn't work T42 resets.
85 LDN     Q0.6
86 R      T42, 1
87
88 NETWORK 19 //The three bant begin to work when the fruits come on the bant
      according to there dimension.
89 LD      I0.6
90 =      Q0.7
91 =      Q1.0
92 =      Q1.1
93
94 NETWORK 20 //When the sensor see's the fruit it works.
95 LD      I0.7
96 =      Q1.2
97
98 NETWORK 21 //When the sensor see's fruit it works.
99 LD      I1.0
100 =     Q1.3
101
102 NETWORK 22 //Sensor see's the fruit it works and the stemp and seal
      doesn't work then the band should work.
103 LD      I1.1
104 AN      I1.2
105 =      Q1.4
106
107 NETWORK 23 //When the box is seen, the timer waits 1 sec and the stamp and
      seal to work.
108 LD      I1.2
109 TON      T43, +10
110 =      Q1.5
111
112 NETWORK 24 //After 1 sec of timer the band to work.
113 LD      T37
114 =      Q1.6
115
116 NETWORK 25 //The stemped and sealed boxes to be counted.
117 LD      Q1.5
118 LD      Q1.6
119 CTU      C0, VW0
120
121 NETWORK 26 //When out the normal counter by the digital counter.
122 LD      C0
123 =      Q1.7
124
125 NETWORK 27 //The and of work.
126 MEND

```

Compare Byte Greater Than Or Equal Contact

Symbol:



Operands:

n1, n2 (unsigned byte):

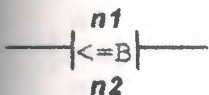
VB, IB, QB,
MB, SMB, AC,
Constant, *VD,
*AC

Description of operation:

The Compare Byte Greater Than or Equal Contact is closed when the byte value stored at address n1 is greater than or equal to the byte value stored at address n2. Power flows through the contact when closed.

Compare Byte Less Than Or Equal Contact

Symbol:



Operands:

n1, n2 (unsigned byte):

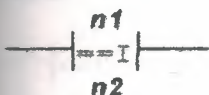
VB, IB, QB,
MB, SMB, AC,
Constant, *VD,
*AC

Description of operation:

The Compare Byte Less Than or Equal Contact is closed when the byte value stored at address n1 is less than or equal to the byte value stored at address n2. Power flows through the contact when closed.

Compare Integer Equal Contact

Symbol:



Operands:

n1, n2 (signed integer word):

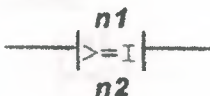
VW, T, C, IW, QW,
MW, SMW, AC,
AIW, Constant, *VD, *AC

Description of operation:

The Compare Integer Equal Contact is closed when the signed integer word value stored at address n1 is equal to the signed integer word value stored at address n2. Power flows through the contact when closed.

Compare Integer Greater Than Or Equal Contact

Symbol:



Operands:

n1, n2 (signed integer word):

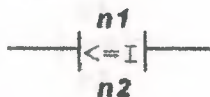
VW, T, C, IW, QW, MW,
SMW, AC, AIW, Constant,
*VD, *AC

Description of operation:

The Compare Integer Greater Than or Equal Contact is closed when the signed integer word value stored at address n1 is greater than or equal to the signed integer word value stored at address n2. Power flows through the contact when closed.

Compare Integer Less Than Or Equal Contact

Symbol:



Operands:

n1, n2 (signed integer word):

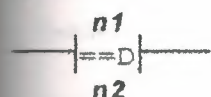
VW, T, C, IW, QW, MW,
SMW, AC, AIW, Constant,
*VD, *AC

Description of operation:

The Compare Integer Less Than or Equal Contact is closed when the signed integer word value stored at address n1 is less than or equal to the signed integer word value stored at address n2. Power flows through the contact when closed.

Compare Double Integer Equal Contact

Symbol:



Operands:

n1, n2 (signed
integer double word):

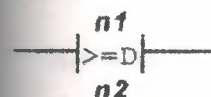
VD, ID, QD,
MD, SMD, AC,
HC, Constant,
*VD, *AC

Description of operation:

The Compare Double Integer Equal Contact is closed when the double word value stored at address n1 is equal to the double word value stored at address n2. Power flows through the contact when closed.

Compare Double Integer Greater Than Or Equal Contact

Symbol:



Operands:

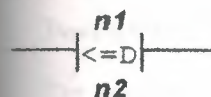
n1, n2 (signed
integer double word): VD, ID, QD, MD, SMD, AC,
HC, Constant, *VD, *AC

Description of operation:

Compare Double Integer Greater Than Or Equal Contact is closed when the double word value stored at address n1 is greater than or equal to the double word value stored at address n2. Power flows through the contact when closed.

Compare Double Integer Less Than Or Equal Contact

Symbol:



Operands:

n1, n2 (signed
integer double word):

VD, ID, QD,
MD, SMD, AC,
HC, Constant,
*VD, *AC

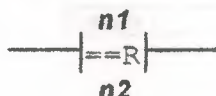
Description of operation:

The Compare Double Integer Less Than Or Equal Contact is closed when the double word value stored at address n1 is less than or equal to the double word value stored at address n2. Power flows through the contact when closed.

Compare Real Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

n1, n2 (real): VD, ID, QD, MD, SMD, AC,
HC, Constant, *VD, *AC

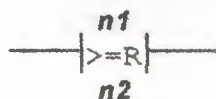
Description of operation:

The Compare Real Equal Contact is closed when the real value stored at address n1 is equal to the real value stored at address n2. Power flows through the contact when closed.

Compare Real Greater Than Or Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

n1, n2 (Dword): VD, ID, QD, MD, SMD, AC,
HC, Constant, *VD, *AC

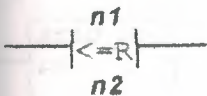
Description of operation:

Compare Real Greater Than Or Equal Contact is closed when the real value stored at address n1 is greater than or equal to the real value stored at address n2. Power flows through the contact when closed.

Compare Real Less Than Or Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

n1, n2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

Description of operation:

The Compare Real Less Than Or Equal Contact is closed when the real value stored at address n1 is less than or equal to the real value stored at address n2. Power flows through the contact when closed.

Invert Power Flow Contact

Symbol:



Operands:

(none)

Description of operation:

The NOT (Invert Power Flow) contact changes the state of power flow. If power flow reaches the Not contact, then it stops. When power flow does not reach the Not contact, it sources power flow.

Positive Transition Contact

Symbol:



Operands:

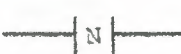
(none)

Description of operation:

The Positive Transition Contact allows power to flow for one scan, for each off-to-on transition.

Negative Transition Contact

Symbol:



Operands:

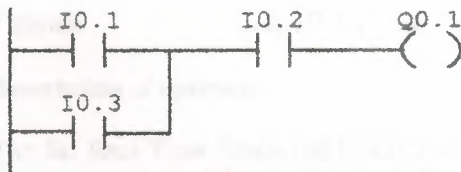
(none)

Description of operation:

The Negative Transition Contact allows power to flow for one scan, for each on-to-off transition.

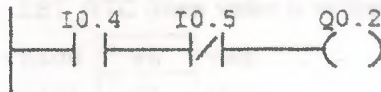
Ladder Contact Examples

Network 1



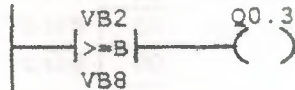
When I0.1 or I0.3 is on and I0.2 is on then output Q0.1 is turned on.

Network 2



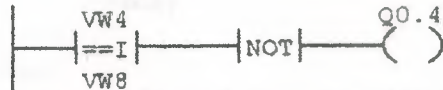
When I0.4 is on and I0.5 is not on, then output Q0.2 is turned on.

Network 3



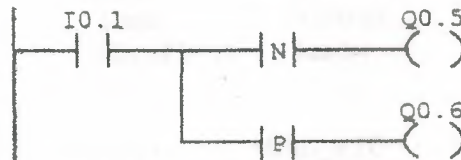
When VB2 is greater than or equal to VB8, then output Q0.3 is turned on.

Network 4



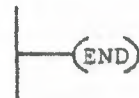
When VB4 equals VB8, then output Q0.4 is turned off (Note: The NOT instruction can be used to create a Not Equal comparison.)

Network 5



When I0.1 transitions from on to off, then output Q0.5 is turned on for one scan cycle. When I0.1 transitions from off to on, then Q0.6 is turned on for one scan.

Network 6

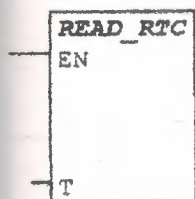


End of the main user program.

Read Real Time Clock

Note: Real Time Clock instructions are supported by the CPU 214 only.

Symbol:



Operands:

T (byte): VB, IB, QB, MB, SMB, *VD, *AC

Description of operation:

The Read Real Time Clock (READ_RTC) box reads the current time and date from the clock and loads it in an 8-byte buffer (T).

Example Memory Data Starting at VB400:

READ_RTC (Clock is read)

VB400	95	Year
VB401	03	Month
VB402	24	Day
VB403	08	Hour
VB404	00	Minute
VB405	00	Second
VB406	00	
VB407	06	Day of Week

24-Mar-95
8:00:00
Friday

Note:

The time of day clock initializes the following date and time after extended power outages or memory has been lost:

Date: 01-Jan-90
Time: 00:00:00
Day of Week Sunday

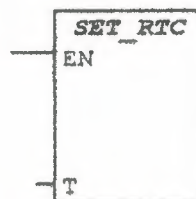
Note:

Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. If you do this and the clock instruction is executing when the interrupt that also executes the clock instruction occurs, then the clock instruction in the interrupt routine is not executed. SM4.5 is then set, indicating that two simultaneous accesses to the clock were attempted.

Set Real Time Clock

Note: Real Time Clock instructions are supported by the CPU 214 only.

Symbol:



Operands:

T (byte): VB, IB, QB, MB, SMB, *VD, *AC

Description of operation:

The Set Real Time Clock (SET_RTC) box writes the current time and date loaded in an 8-byte buffer (T) to the clock.

Example Memory Data Starting at VB400:

SET_RTC (New value is written to clock)

VB400	96	Year
VB401	03	Month
VB402	24	Day
VB403	08	Hour
VB404	00	Minute
VB405	00	Second
VB406	00	
VB407	06	Day of Week

24-Mar-96
8:00:00
Friday

Note:

The time of day clock initializes the following date and time after extended power outages or memory has been lost:

Date: 01-Jan-90
Time: 00:00:00
Day of Week Sunday

Note:

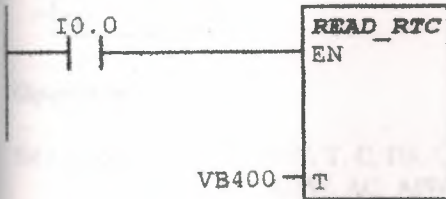
Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. If you do this and the clock instruction is executing when the interrupt that also executes the clock instruction occurs, then the clock instruction in the interrupt routine is not executed. SM4.5 is then set, indicating that two simultaneous accesses to the clock were attempted.

Real-time Examples

Clock Instruction

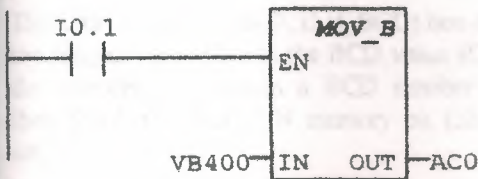
Network 1

When I0.0 is on, the clock is read and the value is stored in the buffer, starting at VB400.



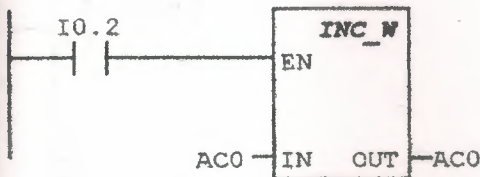
Network 2

When I0.1 is on, the year value (95) from the first byte of VB400 is moved to AC0.



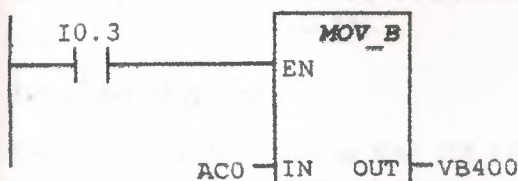
Network 3

When I0.2 is on, the year value in AC0 is incremented by 1.



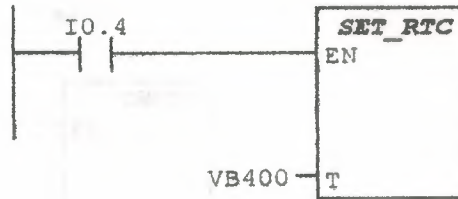
Network 4

When I0.3 is on, the new year value (96) is stored in VB400.



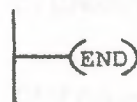
Network 5

When I0.4 is on, the new year value is written to the clock.



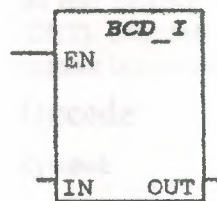
Network 6

End of the main user program.



BCD to Integer

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

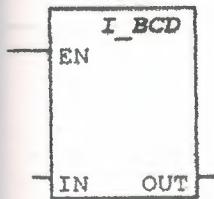
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Convert BCD to Integer (BCD_I) box converts the BCD value (IN) to an integer value (OUT). If the input value contains an invalid BCD digit, the BCD/BIN memory bit (SM1.6) is set.

Integer to BCD

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

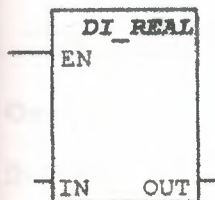
Description of operation:

The Convert Integer to BCD (I_BCD) box converts the integer value (IN) to the BCD value (OUT). If the conversion produces a BCD number greater than 9999, the BCD/BIN memory bit (SM1.6) is set.

Integer Double Word to Real

Note: CPU 214 only.

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

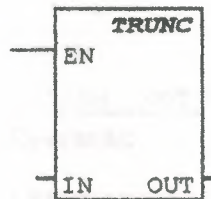
Description of operation:

The Integer Double Word to Real (DI_REAL) instruction converts a 32-bit signed integer (IN) into a 32-bit real number (OUT).

Truncate

Note: CPU 214 only.

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

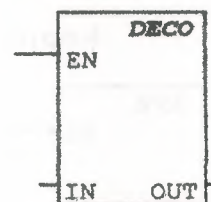
OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Truncate (TRUNC) instruction converts a 32-bit real number (IN) into a 32-bit signed integer (OUT). Only the whole number portion of the real number is converted (round-to-zero).

Decode

Symbol:



Operands:

IN (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

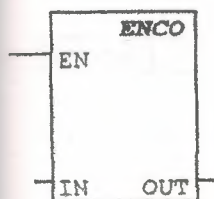
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

Description of operation:

The Decode (DECO) box sets the bit in the output word (OUT) that corresponds to the bit number represented by the least-significant nibble (LSN) of the input byte (IN). All other bits of the output word are set to 0.

Encode

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

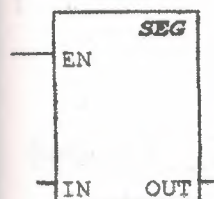
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

Description of operation:

The Encode (ENCO) box writes the bit number (bit #) of the least-significant bit set of the input word (IN) into the least-significant nibble (LSN) of the output byte (OUT).

Segment

Symbol:



Operands:

IN (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

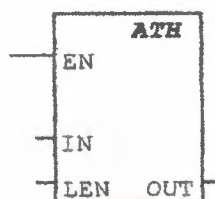
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

Description of operation:

The Segment (SEG) box generates a bit pattern (OUT) that illuminates the segments of a seven-segment display. The illuminated segments represent the character in the least-significant digit of the input byte (IN).

ASCII to Hex

Symbol:



Operands:

LEN (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

IN (byte): VB, IB, QB, MB, SMB, *VD, *AC

OUT (byte): VB, IB, QB, MB, SMB, *VD, *AC

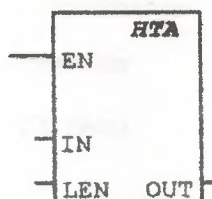
Description of operation:

The ASCII to HEX (ATH) box converts the ASCII string of length LEN, starting with the character IN, to hexadecimal digits starting at the location OUT. The maximum length of the ASCII string is 255 characters.

Legal ASCII characters are the hexadecimal values 30-39, and 41-46. If an illegal ASCII character is encountered, the conversion is terminated, and the NOT_ASCII memory bit (SM1.7) is set.

Hex to ASCII

Symbol:



Operands:

LEN (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

IN (byte): VB, IB, QB, MB, SMB, *VD, *AC

OUT (byte): VB, IB, QB, MB, SMB, *VD, *AC

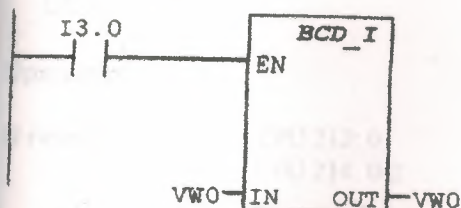
Description of operation:

The HEX to ASCII (HTA) box converts the hexadecimal digits, starting with the input byte IN, to an ASCII string starting at the location OUT. The number of hexadecimal digits to be converted is specified by length LEN. The maximum number of the hexadecimal digits that can be converted is 255.

Ladder Conversion Instruction Examples

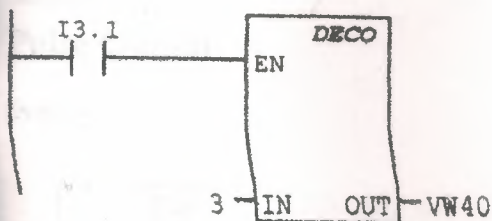
Network 1

When I3.0 is on, the Binary Coded Decimal value in VW0 is converted to an integer value.



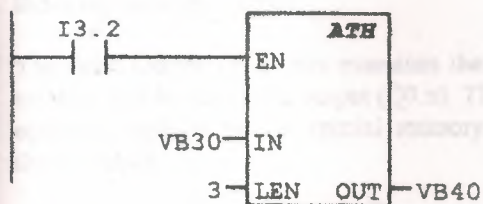
Network 2

When I3.1 is on, 3 is decoded and the corresponding bit of VW40 is set.



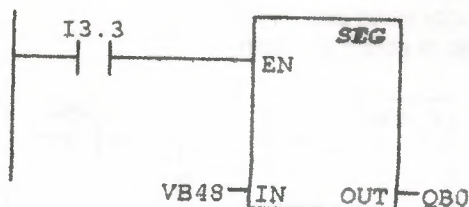
Network 3

When I3.2 is on, the 3-character ASCII string starting with the character at VB30 is converted to hexadecimal digits starting at VB40.



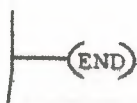
Network 4

When I3.3 is on, a bit pattern is generated at QB0 that illuminates the segments of the character represented by VB48.



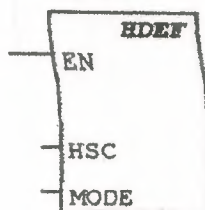
Network 5

End of the main user program.



HSC Definition

Symbol:



Operands:

HSC (byte): CPU 212: 0
CPU 214: 0-2

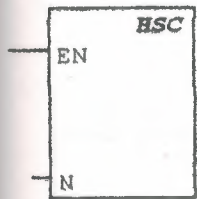
MODE (byte): CPU 212: 0
CPU 214: 0 (HSC0), 0-11 (HSC1-2)

Description of operation:

When the High-speed Counter Definition (HDEF) box is enabled, the referenced counter (HSC) is assigned a high-speed counter type or MODE. Only one HDEF box may be used per counter.

High Speed Counter

Symbol:



Operands:

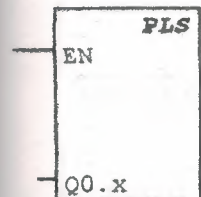
N (word): CPU 212: 0
CPU 214: 0-2

Description of operation:

When the High-speed Counter (HSC) box is enabled, the state of the HSC special memory bits are examined. The HSC operation defined by the special memory bits is then invoked. The parameter N specifies the High-speed Counter number.

Pulse Output

Symbol:



Operands:

Q0.x (word): CPU 214: 0-1

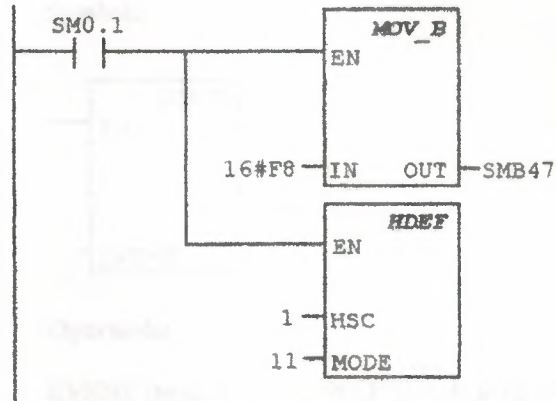
Description of operation:

The Pulse Output (PLS) box examines the special memory bits for that pulse output (Q0.x). The pulse operation defined by the special memory bits is then invoked.

Ladder High-speed Operation Instruction Examples

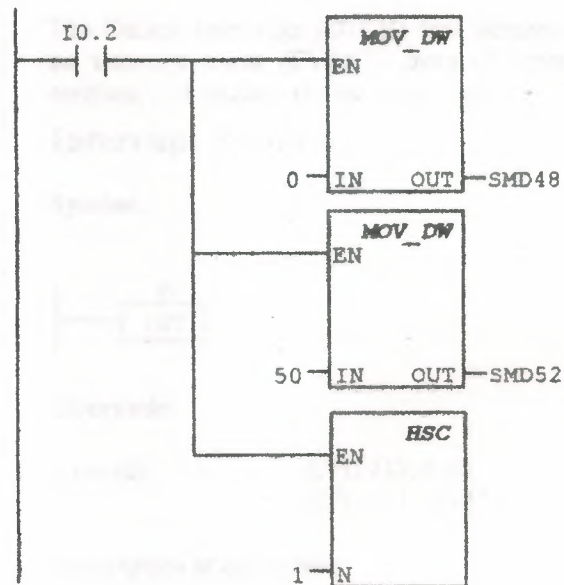
Network 1

On the first scan, the counter is enabled. Initial direction is set to count up. Start and reset inputs are set to active high. 4x mode is set.



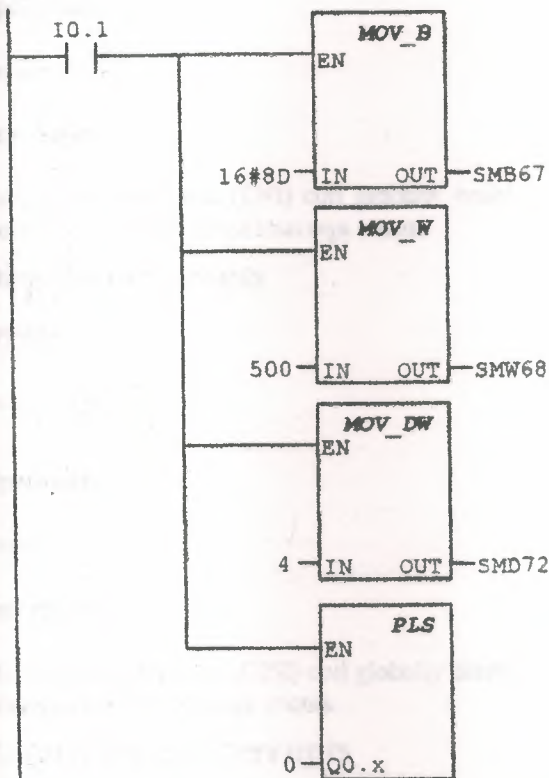
Network 2

When I0.2 is on, the current value of HSC1 is cleared and its preset value is set to 50.



Network 3

When I0.1 is on, the Pulse Train Output control byte is set up, and the PTO operation is invoked: cycle time 500ms, pulse count 4, PLS 0 → Q0.0.



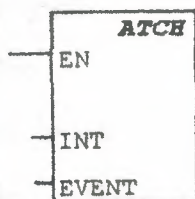
Network 4

End of the main user program.

(END)

Attach Interrupts

Symbol:



Operands:

INT (byte): CPU 212: 0-31
CPU 214: 0-127

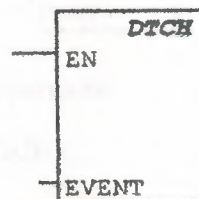
EVENT (byte): CPU 212: 0, 1, 8-10, 12
CPU 214: 0-20

Description of operation:

The Attach Interrupts (ATCH) box associates an interrupt event (EVENT) with an interrupt routine number (INT), and enables the interrupt event.

Detach Interrupts

Symbol:



Operands:

EVENT (byte): CPU 212: 0, 1, 8-10, 12
CPU 214: 0-20

Description of operation:

The Detach Interrupts (DTCH) box disassociates an interrupt event (EVENT) from all interrupt routines, and disables the interrupt event.

Interrupt Routine

Symbol:



Operands:

n (word): CPU 212: 0-31
CPU 214: 0-127

Description of operation:

The Interrupt Routine (INT) label marks the beginning of the interrupt routine (n). The maximum number of interrupts supported by the CPU 212 is 32, and by the CPU 214, 128.

Enable Interrupts

Symbol:



Operands:

(none)

Description:

The Enable Interrupts (ENI) coil globally enables processing of all attached interrupt events.

Disable Interrupts

Symbol:



Operands:

(none)

Description:

The Disable Interrupts (DISI) coil globally disables processing of all interrupt events.

Return from Interrupts

Symbol:



Conditional Return from

Interrupts



Unconditional Return from

Interrupts

Operands:

(none)

Description:

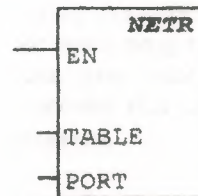
The Conditional Return from Interrupts (RETI) coil returns from an interrupt based upon the condition of the preceding logic.

The Unconditional Return from Interrupts (RETI) coil must be used to terminate each interrupt routine.

Network Read

Note: CPU 214 only.

Symbol:



Operands:

TABLE: VB, MB, *VD, *AC

PORT: Constant
(CPU 214: 0)

Description of operation:

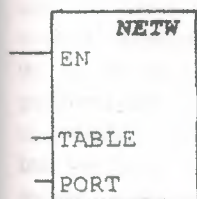
The Network Read (NETR) instruction initiates a communication operation to gather data from a remote device through the specified port (PORT), as defined in the description table (TABLE).

You can use the NETR instruction to read up to 16 bytes of information from a remote station, and use the NETW instruction to write up to 16 bytes of information to a remote station. A maximum of eight NETR and NETW instructions may be activated at any one time. For example, you can have four NETR and four NETW instructions, or two NETR and six NETW instructions.

Network Write

Note: CPU 214 only.

Symbol:



Operands:

TABLE: VB, MB, *VD, *AC

PORT: Constant
(CPU 214: 0)

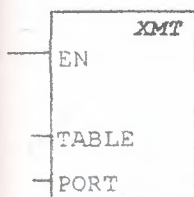
Description of operation:

The Network Write (NETW) instruction initiates a communication operation to write data to a remote device through the specified port (PORT), as defined in the description table (TABLE).

You can use the NETR instruction to read up to 16 bytes of information from a remote station, and use the NETW instruction to write up to 16 bytes of information to a remote station. A maximum of eight NETR and NETW instructions may be activated at any one time. For example, you can have four NETR and four NETW instructions, or two NETR and six NETW instructions.

Transmit

Symbol:



Operands:

TABLE (byte): VB, IB, QB, MB, SMB, *VD, *AC

PORT (byte): 0

Description of operation:

The Transmit (XMT) box invokes the transmission of the data buffer (TABLE). The first entry in the data buffer specifies the number of bytes to be transmitted. PORT specifies the communication port to be used for transmission. It must always be 0.

Data Sharing with Interrupt Events

Because interrupt events are asynchronous to the main user-program, they can occur at any point during execution of the main user-program. When the main program and an interrupt routine share data, you must understand the nature of the problems that can arise and how to avoid such problems.

Data-sharing problems can occur in situation where a sequence of operations are performed in the main program on data stored in a memory location shared by the main program and an interrupt routine. If an intermediate result is stored in the shared memory location, then an interrupt event occurring before the sequence is complete will cause the interrupt routine to be executed with invalid data, or it will corrupt an intermediate value in the main program.

The situations described above apply whether you write your programs in STL or LAD. If you write your programs in LAD, you should also be aware that many LAD instructions produce a sequence of STL instructions. If the LAD instruction is located in the main program and is operating on data stored in a shared memory location, an interrupt event can occur between the execution of the STL instructions, altering intermediate values and making it appear that the LAD instruction executed incorrectly. For techniques to avoid problems with data sharing, see Programming Techniques for Data Sharing.

Programming Techniques for Data Sharing

The following programming techniques should be followed to avoid problems with data sharing between your main program and interrupt routines. These techniques either restrict the way access is made to shared memory locations, or they make instruction sequences using shared memory locations uninterruptible. The appropriate technique depends upon the size of the data being shared (simple elements such as a byte, word, or double-word variable or complex elements such as multiple variables) and the programming language (STL or LAD).

If the shared data is a single byte, word, or double-word variable and your program is written in STL, then make sure that intermediate or temporary values are not stored in shared memory locations. A shared location should be accessed in the main program only as the initial source value or the final destination value in a sequence of operations.

If the shared data is a single byte, word, or double-word variable and your program is written in LAD, then access shared memory locations using a Move instruction. If the main program performs one or more operations on a data value provided by an interrupt routine, the Move instruction must be used to move the data value from the shared memory location to a non-shared memory location or to an accumulator. If the main program performs one or more operations on data in order to provide a value to an interrupt routine, then the last operation must be a Move instruction that moves the data value from an accumulator or non-shared memory location to the shared memory location. Other instructions in the sequence must not directly access the shared memory location.

If the shared data is composed of related bytes, words, or double-words whose values must agree: for example, the pressure and temperature of a gas in a tank, then the interrupt disable/enable instructions, DISI and ENI, must be used to control interrupt routine execution. At the point in your main program (STL or LAD) where operations on shared memory locations are to begin, interrupts must be disabled. Once all actions affecting shared locations are complete, interrupts must be re-enabled. During the time that interrupts are disabled, interrupt routines cannot execute and access shared memory locations.

Interrupt Event Priority Table

Interrupt Description (By group priority)	Event #	In-Group Priority	Supported in CPU 21
Comm. (Highest Priority)			
Receive interrupt	8	0	Y
Transmit complete interrupt	9	0*	Y
Discrete (Middle Priority)			
Rising edge, I0.0**	0	0	Y
Rising edge, I0.1	2	1	
Rising edge, I0.2	4	2	
Rising edge, I0.3	6	3	
Falling edge, I0.0**	1	4	Y
Falling edge, I0.1	3	5	
Falling edge, I0.2	5	6	
Falling edge, I0.3	7	7	
HSC0 CV=PV** (current value = preset value)	12	0	Y
HSC1 CV=PV (current value = preset value)	13	8	
HSC1 direction input changed	14	9	
HSC1 external reset	15	10	
HSC2 CV=PV (current value = preset value)	16	11	
HSC2 direction input changed	17	12	
HSC2 external reset	18	13	
PLS0 pulse count complete interrupt	19	14	
PLS1 pulse count complete interrupt	20	15	
Timed (Lowest Priority)			
Timed interrupt 0	10	0	Y
Timed interrupt 1	11	1	

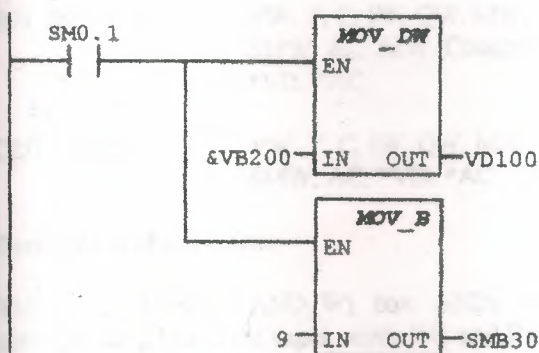
* Since communication is inherently half-duplex, both transmit and receive are the same priority.

**If event 12 (HSC0 CV=PV) is attached to an interrupt, then neither event 0 nor event 1 can be attached to interrupts. Likewise, if either event 0 or 1 is attached to an interrupt, then event 12 cannot be attached to an interrupt.

Ladder Interrupt / Communication Instruction Examples

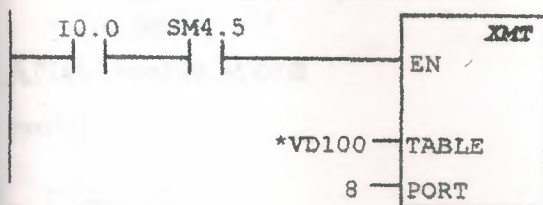
Network 1

On the first scan, create a pointer to the data to be transmitted. Select freeport mode, 9600 baud, no parity, 8 bits per character. SMB30 is the freeport control byte.



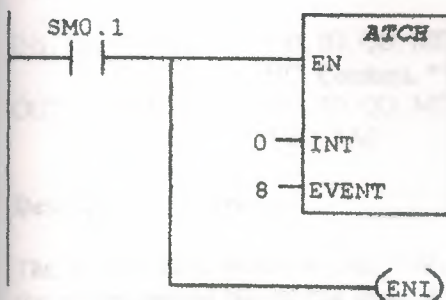
Network 2

When I0.0 and SM4.5 are both on, the message in the buffer (pointed to by VD100) is transmitted. SM4.5 is on when the transmitter is idle.



Network 3

Assign receive interrupt event 8 to interrupt routine 0, and enable the routine.



Network 4

End of main ladder program.



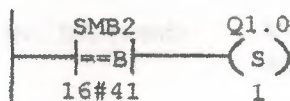
Network 5

Begin interrupt routine 0.



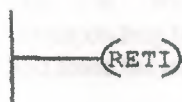
Network 6

Compare received character in special memory byte SMB2 with capital letter "A". If character is "A", Q0.1 is set.



Network 7

Return from interrupt to main program.



Horizontal Lines

In ladder logic, horizontal lines represent wires connecting elements in series.

All lines in a network must be connected to valid elements.

All networks must terminate in a coil or a box.

Vertical Lines

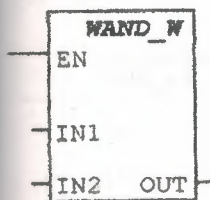
In ladder logic, vertical lines represent wires connecting to parallel branches.

All lines in a network must be connected to valid elements.

All networks must terminate in a coil or a box.

AND Word

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The AND Word (WAND_W) box ANDs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

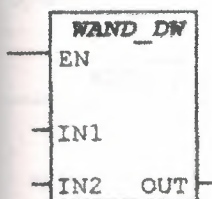
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

AND Double Word

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The AND Double Word (WAND_DW) box ANDs the corresponding bits of the input double words

IN1 and IN2, and loads the result (OUT) in a double word.

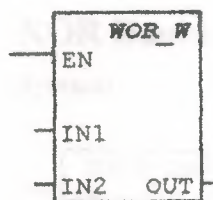
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

OR Word

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The OR Word (WOR_W) box ORs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

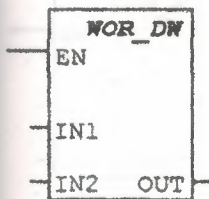
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

OR Double Word

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The OR Double Word (WOR_DW) box ORs the corresponding bits of the input double words IN1 and IN2, and loads the result (OUT) in a double word.

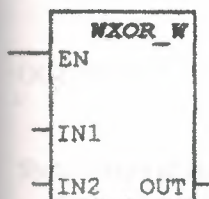
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

XOR Word

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Exclusive OR Word (WXOR_W) box XORs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

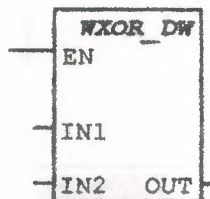
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

XOR Double Word

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Exclusive OR Double Word (WXOR_DW) box XORs the corresponding bits of the input double words IN1 and IN2, and loads the result (OUT) in a double word.

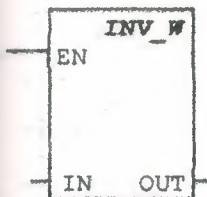
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Invert Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

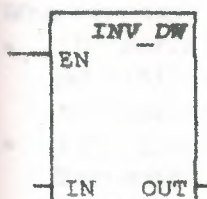
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Invert Word (INV_W) box takes the ones complement of the input word value (IN) and loads the result in a word value (OUT).

Invert Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

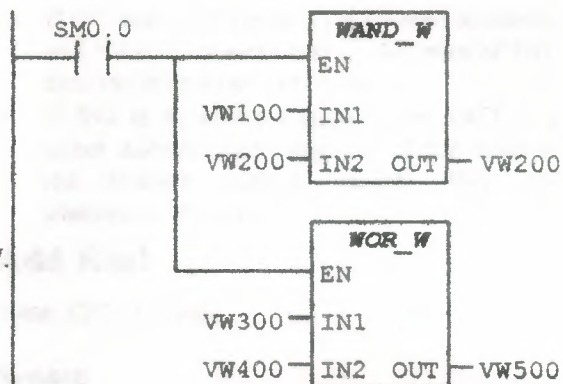
Description of operation:

The Invert Double Word (INV_DW) box takes the ones complement of the input double word value (IN) and loads the result in a double word value (OUT).

Ladder Logical Operations Examples

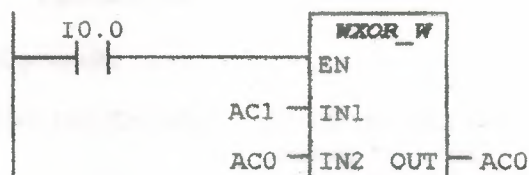
Network 1

Every scan, AND VW100 and VW200 together and store the result in VW200. Also, OR VW300 and VW400 together and store the result in VW500.



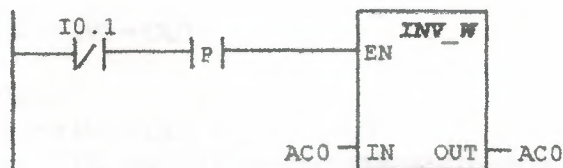
Network 2

When I0.0 is on, "XOR" AC1 and AC0 together and store the result in AC0.



Network 3

When I0.1 transitions from off to on, invert AC0 (ones complement) and store it in AC0.



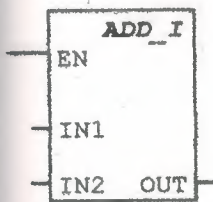
Network 4

End of main user program.



Add Integer

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Add Integer (ADD_I) box adds two 16-bit integers (IN1, IN2), and produces a 16-bit result (OUT), as is shown in the equation:

$$IN1 + IN2 = OUT$$

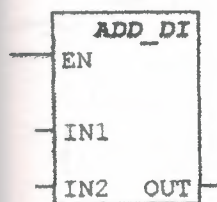
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Add Double Integer

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Add Double Integer (ADD_DI) box adds two 32-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

$$IN1 + IN2 = OUT$$

Note:

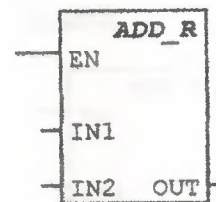
When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Add Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:

The Add Real (ADD_R) box adds two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

$$IN1 + IN2 = OUT$$

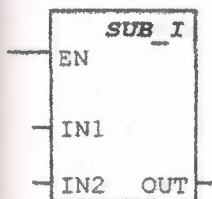
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Integer

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Subtract Integer (SUB_I) box subtracts two 16-bit integers (IN1, IN2), and produces a 16-bit result (OUT), as is shown in the equation:

$$IN1 - IN2 = OUT$$

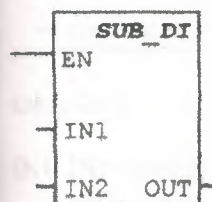
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Double Integer

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Subtract Double Integer (SUB_DI) box subtracts two 32-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

$$IN1 - IN2 = OUT$$

Note:

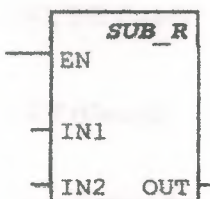
When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:

The Subtract Real (SUB_R) box subtracts two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

$$IN1 - IN2 = OUT$$

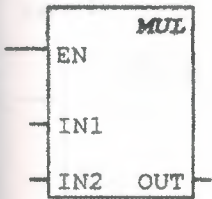
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Multiply Integer

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Multiply Integer (MUL) box multiplies two 16-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

$$IN1 * IN2 = OUT$$

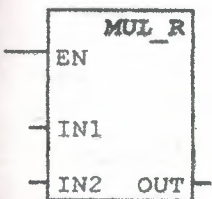
Note:

Some overlapping input and output operands are invalid.

Multiply Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:

The Multiply Real (MUL_R) box multiplies two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

$$IN1 * IN2 = OUT$$

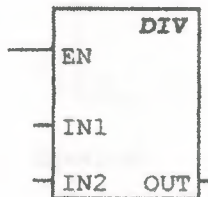
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Divide Integer

Symbol:



Operands:

IN1, IN2 (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

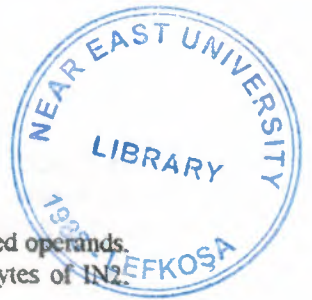
Description of operation:

The Divide Integer (DIV) box divides two 16-bit integers (IN1, IN2), and produces a 32-bit result (OUT) composed of a 16-bit quotient and a 16-bit remainder, as is shown in the equation:

$$IN1 / IN2 = OUT$$

Notes:

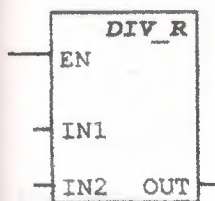
- Some overlapping input and output operands are invalid.
- The 32-bit result (OUT) cannot be the same as the second input (IN2).



Divide Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:

The Divide Real (DIV_R) box divides two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number quotient (OUT), as is shown in the equation:

$$IN1 / IN2 = OUT$$

Note:

When $IN1 \neq 0$ and $IN2 \neq 0$:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

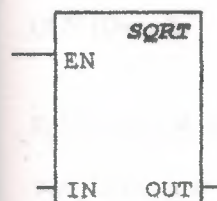
Note:

$IN2 = OUT$ is not valid for Ladder programming.

Square Root Real

Note: CPU 214 only.

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

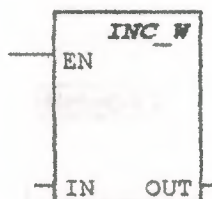
Description of operation:

The Square Root of Real Numbers (SQRT) box takes the square root of a 32-bit real number (IN) and produces a 32-bit real number result (OUT), as is shown in the equation:

$$\sqrt{IN} = OUT$$

Increment Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

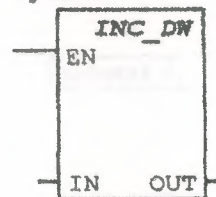
Description of operation:

The Increment Word (INC_W) box adds 1 to the input word value (IN) and loads the result in a word value (OUT), as is shown in the equation:

$$IN + 1 = OUT$$

Increment Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

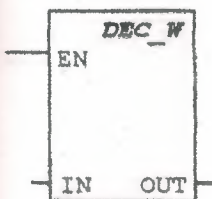
Description of operation:

The Increment Double Word (INC_DW) box adds 1 to the input double word value (IN) and loads the result in a double word value (OUT), as is shown in the equation:

$$IN + 1 = OUT$$

Decrement Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

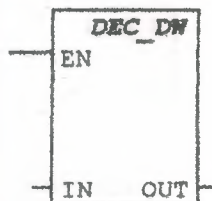
Description of operation:

The Decrement Word (DEC_W) box subtracts 1 from the input word value (IN) and loads the result in a word value (OUT), as is shown in the equation:

$$IN - 1 = OUT$$

Decrement Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

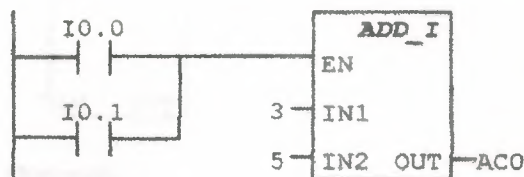
The Decrement Double Word (DEC_DW) box subtracts 1 from the input double word value (IN) and loads the result in a double word value (OUT), as is shown in the equation:

$$IN - 1 = OUT$$

Math/Inc/Dec Examples

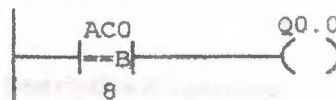
Network 1

When I0.0 or I0.1 is on then AC0 equals the sum of IN1 and IN2.



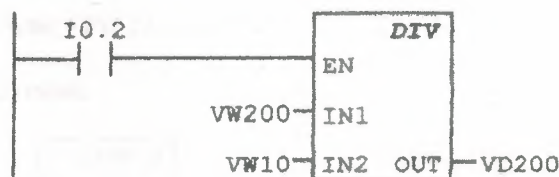
Network 2

If AC0 equals 8, turn on Q0.0.



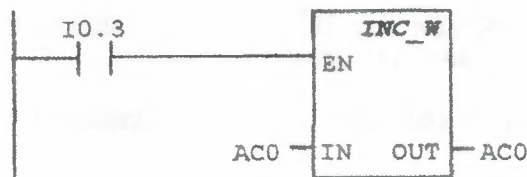
Network 3

VW200 is divided by VW10. The quotient is put in VW202, and the remainder is put in VW200. (Note: VD200 contains VW200 and VW202.)



Network 4

When I0.3 is on, then the value in AC0 is incremented by 1 and stored in AC0.



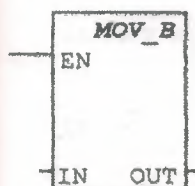
Network 5

End of the main user program.



Move Byte

Symbol:



Operands:

IN (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

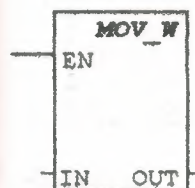
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

Description of operation:

The Move Byte (MOV_B) box moves the input byte (IN) to the output byte (OUT). The input byte is not altered by the move.

Move Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

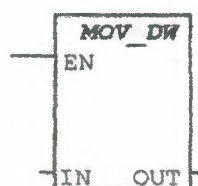
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

Description of operation:

The Move Word (MOV_W) box moves the input word (IN) to the output word (OUT). The input word is not altered by the move.

Move Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC, &VB, &IB, &QB, &MB, &T, &C

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

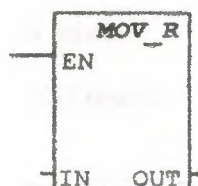
Description of operation:

The Move Double Word (MOV_DW) box moves the input double word (IN) to the output double word (OUT). The input double word is not altered by the move.

Move Real

Note: CPU 214 only.

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

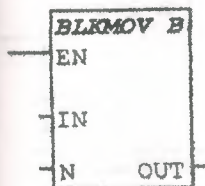
OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Move Real (MOV_R) box moves a 32-bit real input double word (IN) to the output double word (OUT). The input double word is not altered by the move.

Block Move Byte

Symbol:



Operands:

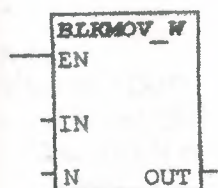
IN (byte): VB, IB, QB, MB, SMB *VD, *AC
 OUT (byte): VB, IB, QB, MB, SMB, *VD, *AC
 N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Block Move Byte (BLKMOV_B) box moves the number of bytes specified (N), from the input array starting at IN, to the output array starting at OUT. N has a range of 1 to 255.

Block Move Word

Symbol:



Operands:

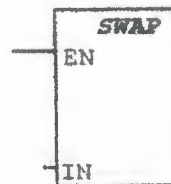
IN (word): VW, T, C, IW, QW, MW, SMW, AIW, *VD, *AC
 OUT (word): VW, T, C, IW, QW, MW, SMW, AQW, *VD, *AC
 N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Block Move Word (BLKMOV_W) box moves the number of words specified (N), from the input array starting at IN, to the output array starting at OUT. N has a range of 1 to 255.

Swap

Symbol:



Operands:

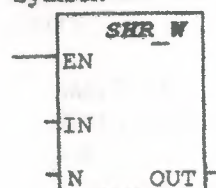
IN (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Swap Byte box exchanges the most-significant byte with the least-significant byte of the word (IN).

Shift Right Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
 N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
 OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Shift Right Word (SHR_W) box shifts the word value (IN) right by the shift count (N), and loads the result in the output word (OUT).

SM1.0 (zero) = 1 if OUT = 0
 SM1.1 (overflow) = 1 if last bit shifted out = 0

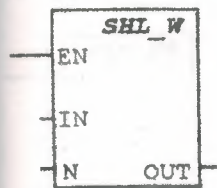
Note:

When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Left Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Shift Left Word (SHL_W) box shifts the word value (IN) left by the shift count (N), and loads the result in the output word (OUT).

SM1.0 (zero) = 1 if OUT = 0

SM1.1 (overflow) = 1 if last bit shifted out = 0

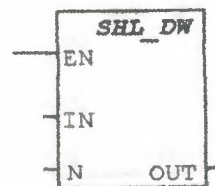
Note:

When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Left Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Shift Left Double Word (SHL_DW) box shifts the double word value (IN) left by the shift count (N), and loads the result in the output double word (OUT).

SM1.0 (zero) = 1 if OUT = 0

SM1.1 (overflow) = 1 if last bit shifted out = 0

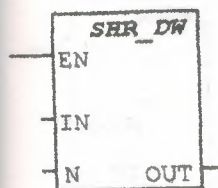
Note:

When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Right Double Word

Symbol:



Operands:

IN (Dword): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Shift Right Double Word (SHR_DW) box shifts the double word value (IN) right by the shift count (N), and loads the result in the output double word (OUT).

SM1.0 (zero) = 1 if OUT = 0
SM1.1 (overflow) = 1 if last bit shifted out = 0

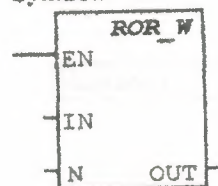
Note:

When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Rotate Right Word

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Rotate Right Word (ROR_W) box rotates the word value (IN) right by the shift count (N), and loads the result in the output word (OUT).

SM1.0 (zero) = 1 if OUT = 0
SM1.1 (overflow) = 1 if last bit rotated = 0

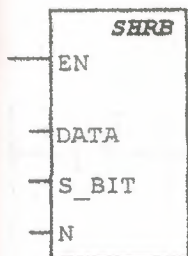
Note:

When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Register Bit

Symbol:



Operands:

DATA, S_BIT (bit): I, Q, M, SM, T, C, V

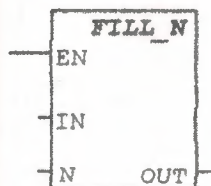
N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Shift Register Bit (SHRB) instruction shifts the value of DATA into the shift register. S_BIT specifies the least-significant bit of the shift register. N specifies the length of the shift register and the direction of the shift (shift plus = N, shift minus = -N).

Fill Memory

Symbol:



Operands:

IN (word): VW, T, C, IW, QW, MW, SMW, AIW, Constant, *VD, *AC

OUT (word): VW, T, C, IW, QW, MW, SMW, AQW, *VD, *AC

N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

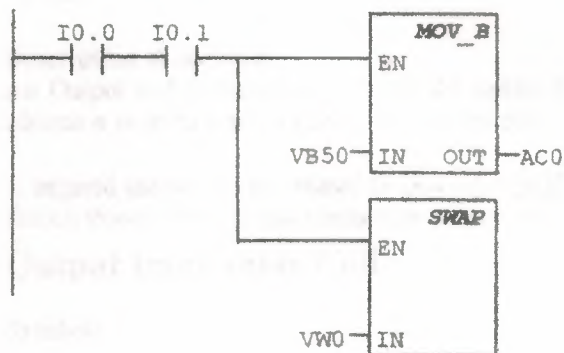
Description of operation:

The Fill Memory Box (FILL_N) fills the memory starting at the output word (OUT) with the word input pattern (IN) for the number of words specified by N. N has a range of 1 to 255.

Move / Shift / Rotate / Fill Examples

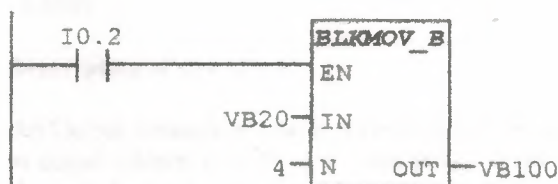
Network 1

When I0.0 and I0.1 are on then move VB50 to AC0, and swap the most significant byte (MSB) of VW0 with the LSB of VW0.



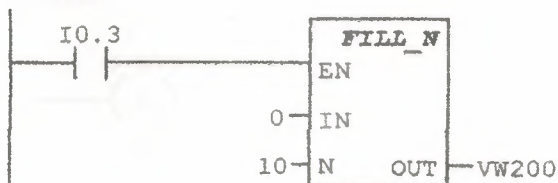
Network 2

When I0.2 is on then move VB20-VB23 to VB100-VB103.



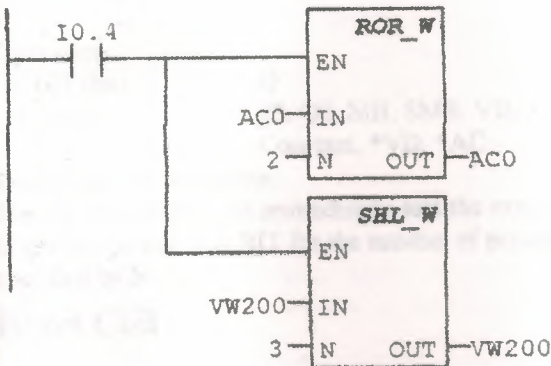
Network 3

When I0.3 is on then fill VW200-VW218 with 0's.

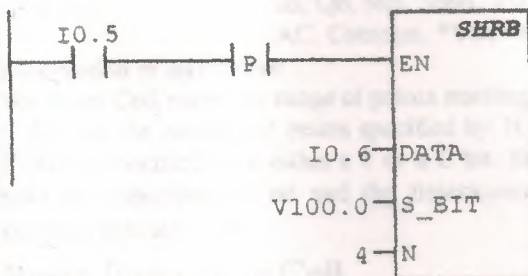


Network 4

When I0.4 is on, then the word value in AC0 is rotated right twice and stored in AC0, and the word value in VW200 is shifted left 3 times and stored in VW200.

**Network 5**

Upon every 0 to 1 transition of I0.5, the value of I0.6 is shifted into the shift register starting at V100.0 and of length 4.

**Network 6**

Main end of the user program.

**Output**

Symbol:



Operands:

n (bit): I, Q, M, SM, T, C, V

Description of operation:

An Output coil is turned on and the Bit stored at address *n* is set to 1 when power flows to the coil.

A negated output can be created by placing a **NOT** (Invert Power Flow) contact before an output coil.

Output Immediate Coil

Symbol:



Operands:

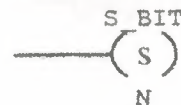
n (bit): Q

Description of operation:

An Output Immediate Coil is turned on and the Bit at output address *n* is set to 1 when power flows to the coil. An update of the addressed image register output Bit and also the corresponding physical output Bit occurs immediately after the coil is scanned without waiting for scan cycle completion.

Set

Symbol:



Operands:

S_BIT (bit): I, Q, M, SM, T, C, V

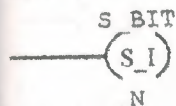
N (byte): IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

Description of operation:

The Set Coil sets the range of points starting at S_BIT for the number of points specified by N.

Set Immediate Coil

Symbol:



Operands:

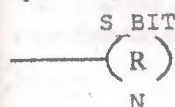
S_BIT (bit): Q
N (byte): IB, QB, MB, SMB, VB, AC,
Constant, *VD, *AC

Description of operation:

The Set Immediate Coil immediately sets the range of points starting at S_BIT for the number of points specified by N.

Reset Coil

Symbol:



Operands:

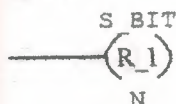
S_BIT (bit): I, Q, M, SM, T, C, V
N (byte): IB, QB, MB, SMB, VB,
AC, Constant, *VD, *AC

Description of operation:

The Reset Coil resets the range of points starting at S_BIT for the number of points specified by N. If S_BIT is specified to be either a T or a C bit, then both the timer/counter bit and the timer/counter current value are reset.

Reset Immediate Coil

Symbol:



Operands:

S_BIT (bit): Q
N (byte): IB, QB, MB, SMB,
VB, AC, Constant, *VD,
*AC

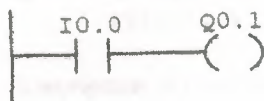
Description of operation:

The Reset Immediate Coil immediately resets the range of points starting at S_BIT for the number of points specified by N.

Ladder Output Coil Examples

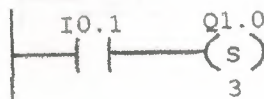
Network 1

When I0.0 is on, then output Q0.1 is turned on.



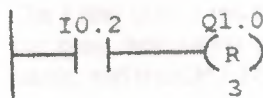
Network 2

When I0.1 is on, then outputs Q1.0, Q1.1 and Q1.2 are set (turned on). These outputs will remain on, even if I0.1 is turned off, until they are reset.



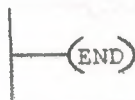
Network 3

When I0.2 is turned on, then outputs Q1.0, Q1.1 and Q1.2 are reset (turned off)



Network 4

End of the main user program.



End

Symbols:

—(END) Conditional End

└(END) Unconditional End

Operands:

(none)

Description of operation:

The Conditional End coil terminates the main user program based on the condition of the preceding logic.

The Unconditional End coil must be used to terminate the user program.

Stop

Symbol:

—(STOP)

Operands:

(none)

Description of operation:

The Stop coil terminates execution of the user program by causing a transition to the stop mode.

Watchdog Reset

Symbol:

—(WDR)

Operands:

(none)

Description of operation:

The Watchdog Reset (WDR) coil allows the watchdog timer to be retriggered. This extends the time the scan takes without getting a watchdog error.

Jump

Symbol:

—ⁿ(JMP)

Operands:

n: CPU 212: 0-63
CPU 214: 0-255

Description of operation:

The Jump to Label (JMP) coil performs a branch to the specified label (*n*) within the program.

Label

Symbol:

└ⁿ
LBL

Operands:

n: CPU 212: 0-63
CPU 214: 0-255

Description of operation:

The Label (LBL) instruction marks the location of the jump destination (*n*). The CPU 212 allows 64 labels, and the CPU 214 allows 256.

Call

Symbol:

—ⁿ(CALL)

Operands:

n: CPU 212: 0-15
CPU 214: 0-63

Description of operation:

The Subroutine Call (CALL) coil transfers control to the subroutine (*n*).

Subroutine

Symbol:



Operands:

n: CPU 212: 0-15
CPU 214: 0-63

Description of operation:

The Subroutine (SBR) label marks the beginning of the subroutine (n). The CPU 212 supports 16 subroutines, and the CPU 214 supports 64.

Return

Symbols:

 Conditional Return from Subroutine

 Unconditional Return from Subroutine

Operands:

(none)

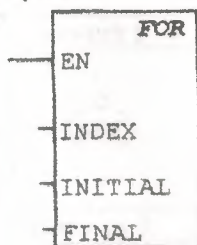
Description of operation:

The Conditional Return from Subroutine coil may be used to terminate a subroutine, based on the condition of the preceding logic.

The Unconditional Return from Subroutine coil must be used to terminate each subroutine.

For

Symbol:



Operands:

INDEX (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

INITIAL (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

FINAL (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

Description of operation:

The FOR box executes the code between the FOR and the NEXT. You must specify the current loop count (INDEX), the starting value (INITIAL), and the ending value (FINAL). If the starting value is greater than the final value, the loop is not executed. After each execution of the instructions between the FOR and the NEXT instruction, the INDEX value is incremented and the result is compared to the final value. If the INDEX is greater than the final value, the loop is terminated. For example, given an INITIAL value of 1, and a FINAL value of 10, the instructions between the FOR and the NEXT are executed 10 times with the INDEX value incrementing 1, 2, 3, . . . 10.

Next

Symbol:



Operands:

(none)

Description of operation:

The NEXT coil marks the end of the FOR loop, and sets the top of stack to 1.

No Operation

Symbol:



Operands:

n: 0-255

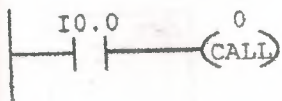
Description of operation:

The No Operation (NOP) coil has no effect on the user program execution. The operand n is a number from 0-255.

Ladder Program Control Examples

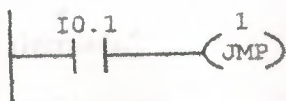
Network 1

When I0.0 is on, execute Subroutine 0.



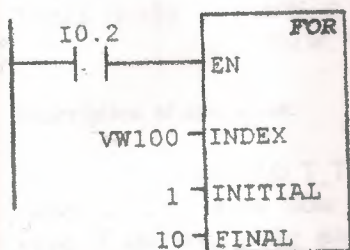
Network 2

When I0.1 is on, jump to Label 1.



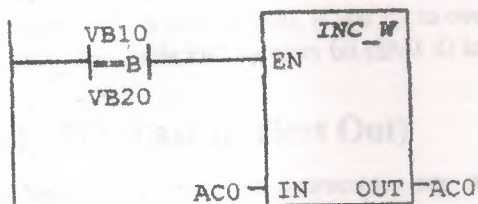
Network 3

When I0.2 is on, execute the For/Next loop 10 times.



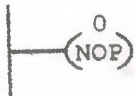
Network 4

If VB10 = VB20, then increment AC0 by 1.



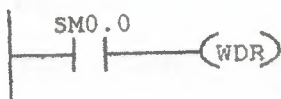
Network 5

This network does nothing.



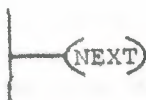
Network 6

SM0.0 is always on, therefore the Watchdog Timer is extended to allow a longer scan.



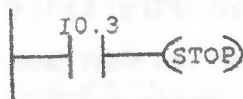
Network 7

This is the end of the For/Next loop.



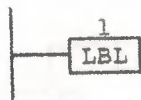
Network 8

If I0.3 comes on, then the CPU goes to Stop mode.



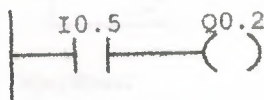
Network 9

The Jump in Network #2 jumps to this location.



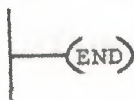
Network 10

When I0.5 is on, turn on Q0.2.



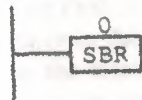
Network 11

End of the main user program.



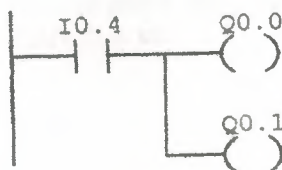
Network 12

Start of Subroutine 0.



Network 13

If I0.4 is on, then turn on Q0.0 and Q0.1.



Network 14

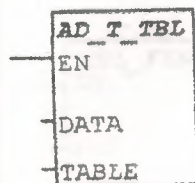
End of Subroutine 0.



Add to Table

Note: Table and Find instructions are supported by the CPU 214 only.

Symbol:



Operands:

DATA (word): VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

TABLE (word): VW, T, C, IW, QW, MW, SMW, *VD, *AC

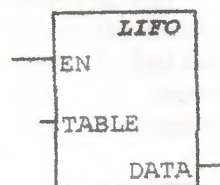
Description of operation:

The Add To Table (AD_T_TBL) box adds word values (DATA) to the table (TABLE). The first value of the table is the maximum table length (TL). The second value is the entry count (EC) that specifies the number of entries in the table. New data are added to the table after the last entry. Each time new data are added to the table, the entry count (EC) is incremented. If you try to overfill the table, the Table Full memory bit (SM1.4) is set.

LIFO (Last In First Out)

Note: Table and Find instructions are supported by the CPU 214 only.

Symbol:



Operands:

TABLE (word): VW, T, C, IW, QW, MW, SMW, *VD, *AC

DATA (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

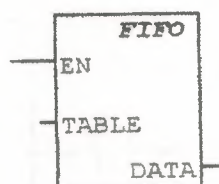
Description of operation:

The Last In First Out (LIFO) box removes the last entry in the table (TABLE), and outputs the value to the location (DATA). The entry count (EC) in the table is decremented for each instruction execution. If you try to remove an entry from an empty table, the Table Empty memory bit (SM1.5) is set.

FIFO (First In First Out)

Note: Table and Find instructions are supported by the CPU 214 only.

Symbol:



Operands:

TABLE (word): VW, T, C, IW, QW, MW, SMW, *VD, *AC

DATA (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

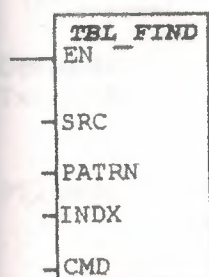
Description of operation:

The First In First Out (FIFO) box removes the first entry in the table (TABLE), and outputs the value to the location (DATA). All other entries of the table are shifted up one location. The entry count (EC) in the table is decremented for each instruction execution. If you try to remove an entry from an empty table, the Table Empty memory bit (SM1.5) is set.

Find Table

Note: Table and Find instructions are supported by the CPU 214 only.

Symbol:



Operands:

SRC (word): VW, T, C, IW, QW, MW, SMW, *VD, *AC

PATRN (word): VW, T, C, IW, QW, MW, SMW, AIW, AC, Constant, *VD, *AC

INDX (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

CMD: 1-4

Description of operation:

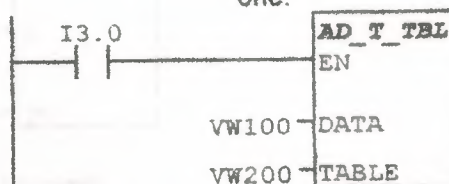
The Find Table (TBL_FIND) box searches the table (SRC), starting with the table entry specified by INDX, for the data value (PATRN) that matches the criteria (CMD). The CMD parameter is given a numeric value 1-4 that corresponds to =, <, >, and >, respectively.

If a match is found, the INDX points to the matching entry in the table. If a match is not found, the INDX has a value equal to the entry count. To find the next matching entry, the INDX must be incremented before invoking the TBL_FIND again.

Ladder Table / Find Instruction Examples

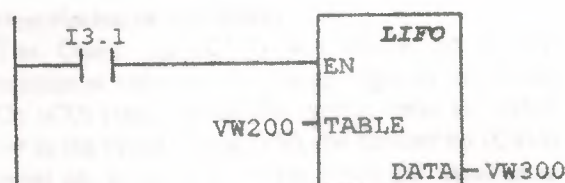
Network 1

When I3.0 is on, the value VW100 is added to the table starting at VW200. The EC (entry count) is incremented by one.



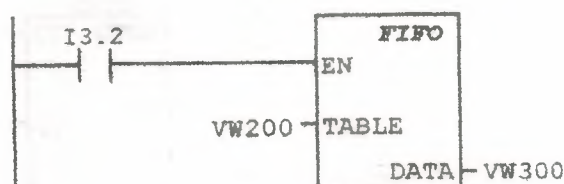
Network 2

When I3.1 is on, the last data value of the table starting at VW200 is output to the data location VW300. The EC is decremented by one.



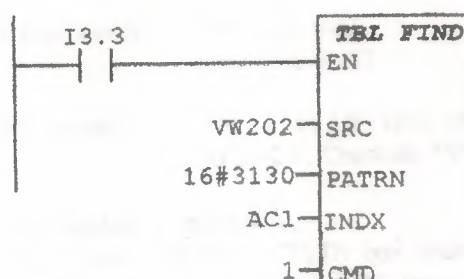
Network 3

When I3.2 is on, the first data value of the table starting at VW200 is output to the data location VW300. The EC is decremented by one.



Network 4

When I3.3 is on, the table VW202 is searched for a value equal to 3130 Hex.



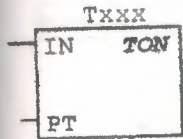
Network 5

End of the main user program.



Timer – On Delay

Symbol:



Operands:

Txx (word): CPU 212: 32-63
CPU 214: 32-63, 96-127

PT (word): VW, T, C, IW, QW, MW, SMW,
AC, AIW, Constant, *VD, *AC

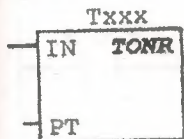
Description of operation:

The On-Delay Timer (TON) box times up to the maximum value when the enabling Input (IN) comes on. When the current value (Txxx) is \geq the Preset Time (PT), the timer bit turns on. It resets when the enabling input goes off. Timing stops upon reaching the maximum value.

	CPU 212/214	CPU 214
1 ms	T32	T96
10 ms	T33-T36	T97-T100
100 ms	T37-T63	T101-T127

Timer – Retentive On Delay

Symbol:



Operands:

Txxx (word): CPU 212: 0-31
CPU 214: 0-31, 64-95

PT (word): VW, T, C, IW, QW, MW, SMW,
AC, AIW, Constant, *VD, *AC

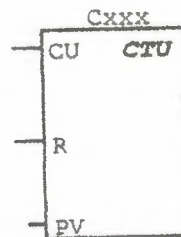
Description of operation:

The Retentive On Delay Timer (TONR) box times up to the maximum value when the enabling Input (IN) comes on. When the current value (Txxx) is \geq the Preset Time (PT), the timer bit turns on. Timing stops when the enabling input goes off, or upon reaching the maximum value.

	CPU 212/214	CPU 214
1 ms	T0	T64
10 ms	T1-T4	T65-T68
100 ms	T5-T31	T69-T95

Count Up

Symbol:



Operands:

Cxxx (word): CPU 212: 0-63
CPU 214: 0-127

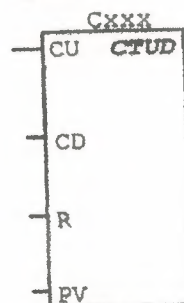
PV (word): VW, T, C, IW, QW, MW, SMW,
AC, AIW, Constant, *VD, *AC

Description of operation:

The Count Up (CTU) box counts up to the maximum value on the rising edges of the Count Up (CU) input. When the current value (Cxxx) is \geq to the Preset Value (PV), the counter bit (Cxxx) turns on. It resets when the Reset (R) input turns on. It stops counting upon reaching the maximum value (32,767).

Count Up / Down

Symbol:



Operands:

Cxxx (word): CPU 212: 0-63
CPU 214: 0-127

PV (word): VW, T, C, IW, QW, MW, SMW,
AC, AIW, Constant, *VD, *AC

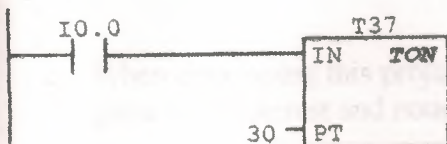
Description of operation:

The Count Up/Down (CTUD) box counts up on rising edges of the Count Up (CU) input. It counts down on the rising edges of the Count Down (CD) input. When the current value (Cxxx) is \geq to the Preset Value (PV), the counter bit (Cxxx) turns on. It stops counting up upon reaching the maximum value (32,767), and stops counting down upon reaching the minimum value (-32,768). It resets when the Reset (R) input turns on.

Ladder Timer / Counter Examples

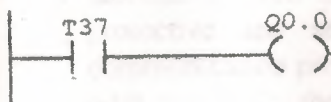
Network 1

When I0.0 is on then the timer will start. After 3 seconds (30 X 100ms) T37 bit will come on.



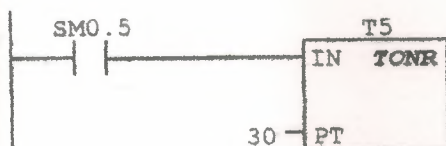
Network 2

When Timer 37 reaches its preset, turn on Q0.0.



Network 3

When SM0.5 (1 sec. clock pulse, .5 sec. on and .5 sec. off) is ON, then the timer will time. The T5 bit will come on after 6 seconds.



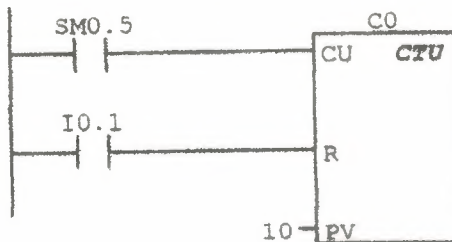
Network 4

When Timer 5 reaches its preset, turn on Q0.1.



Network 5

By using SM0.5 (1 sec. clock pulse) the counter will count pulses and turn on the C0 bit when a count of 10 is reached. I0.0 resets the counter.



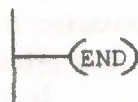
Network 6

When C0 reaches its preset, turn on Q0.2.



Network 7

End of the main user program.



CONCLUSION

When developing this project we see that PLC the individual's life easier which it has gained our interest and notice.

With the information observed from our lecturer and our researchers for this topic PLC, is a convenient tool with a wide range of useful ways to be used. Such examples can be mentioned several machines can be used at the same time, easy adjustments from the PLC programme can be made within a few minutes by the keyboard, installed PLC programmes can be controlled or checked before within the office and laboratory, even the PLC programmes for firm can be made at home. It is very protective and safe for the workers which they are protected from danger, communication programmes of PLC's within each other or within operation can happen with the PLC; the developed industries have constructed the productivity, security, establishment security fast productivity, quality, and we can see that PLC is a very cheap programme that can be fundamentally used

APPENDIXS

Compare Byte Greater Than Or Equal Contact (37)
Compare Byte Less Than Or Equal Contact (37)
Compare Integer Equal Contact (37)
Compare Integer Greater Than Or Equal Contact (37)
Compare Integer Less Than Or Equal Contact (37)
Compare Double Integer Equal Contact (38)
Compare Dounle Integer Greater Than Or Equal Contact (38)
Compare Double Integer Less Than Or Equal Contact (38)
Compare Real Equal Contact (38)
Compare Real Greater Than Or Equal Contact (38)
Compare Real Less Than Or Equal Contact (39)
Invert Power Flow Contact (39)
Positive Transition Contact (39)
Negative Transition Contact (39)
Ladder Contact Examples (39)
Read Real Time Clock (40)
Set Real Time Clock (40)
Real-time Clock Instruction Examples (41)
BCD to Integer (41)
Integer to BCD (42)
Integer Double Word to Real (42)
Truncate (42)
Decode (42)
Encode (43)
Segment (43)
ASCII to Hex (43)
Hex to ASCII (43)
Ladder Conversion Instruction Examples (44)
HSC Definition (44)
High Speed Counter (45)
Pulse Output (45)
Ladder High-speed Operation Instruction Examples (45)
Attach Interrupts (46)
Detach Interrupts (46)
Interrupt Routine (46)
Enable Interrupts (47)
Disable Interrupts (47)
Return from Interrupts (47)
Network Read (47)
Network Write (48)
Transmit (48)
Data Sharing with Interrupt Events (48)
Programming Techniques for Data Sharing (48)
Interrupt Event Priority Table (49)

Ladder Interrupt / Communication Instruction Examples (50)
 Horizontal Lines (50)
 Vertical Lines (50)
 AND Word (51)
 AND Double Word (51)
 OR Word (51)
 OR Double word (52)
 XOR Word (52)
 XOR Double Word (52)
 Invert Word (53)
 Invert Double Word (53)
 Ladder Logical Operations Examples (53)
 Add Integer (54)
 Add Double Integer (54)
 Add Real (54)
 Subtract Integer (55)
 Subtract Double Integer (55)
 Subtract Real (55)
 Multiply Integer (56)
 Multiply Real (56)
 Divide Integer (56)
 Divide Real (57)
 Square Root Real (57)
 Increment Word (57)
 Increment Double Word (57)
 Decrement Word (58)
 Decrement Double Word (58)
 Math/Inc/Dec Examples (58)
 Move Byte (59)
 Move Word (59)
 Move Double Word (59)
 Move Real (59)
 Block Move Byte (60)
 Block Move Word (60)
 Swap (60)
 Shift Right Word (60)
 Shift Left Word (61)
 Shift Left Double Word (61)
 Shift Right Double Word (62)
 Rotate Right Word. (62)
 Shift Register Bit (63)
 Fill Memory (63)
 Move / Shift / Rotate / Fill / Examples (63)
 Output (64)
 Output Immediate Coil (64)
 Set (64)
 Set Immediate Coil (65)
 Reset Coil (65)
 Reset Immediate Coil (65)
 Ladder Output Coil Examples (65)

End (66)
Stop (66)
Watchdog Reset (66)
Jump (66)
Label (66)
Call (66)
Subroutine (67)
Return (67)
For (67)
Next (67)
No Operation (67)
Ladder Program Control Examples (68)
Add to Table (69)
LIFO (Last In First Out) (69)
FIFO (First In First Out) (69)
Find Table (70)
Ladder Table / Find Instruction Examples (70)
Timer – On Delay (71)
Timer – Retentive On Delay (71)
Count Up (71)
Count Up / Down (71)
Ladder Timer / Counter Examples (72)

REFERENCES

1. SIMATIC S7-200 and Industrial Automation,
İTÜ Electric & Electronic Department, July 1998
Doç.Dr. Salman KURTULAN
2. PLC, November 1999
Richard BALDRY
3. Programmable Logic Controllers, June 1999
Hugh JACK