

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

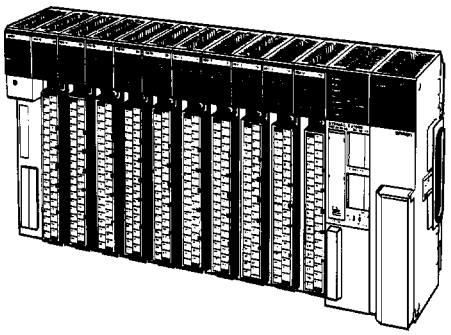
All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

SYSMAC CV-series CV500/CV1000/CV2000 Programmable Controllers

Operation Manual: SFC

Revised May 1993



Notice:

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify warnings in this manual. Always heed the information provided with them.

Caution Indicates information that, if not heeded, could result in minor injury or damage to the product.

DANGER! Indicates information that, if not heeded, could result in loss of life or serious injury.

OMRON Product References

All OMRON products are capitalized in this manual. The word “Unit” is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation “Ch,” which appears in some displays and on some OMRON products, often means “word” and is abbreviated “Wd” in documentation in this sense.

The abbreviation “PC” means Programmable Controller and is not used as an abbreviation for anything else.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

1, 2, 3... 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

© OMRON, 1992

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

TABLE OF CONTENTS

SECTION 1
Outline and Features of SFC 1
1-1 Outline 2
1-2 Features 2
1-3 Application Example 3

SECTION 2
SFC Basics 7
2-1 SFC Terminology 8
2-2 Basic SFC Operation 11
2-3 Memory Areas and Programming Devices 17
2-4 Creating an SFC Program 18

SECTION 3
SFC Programming Elements 19
3-1 Overview 20
3-2 Steps 23
3-3 Initial Steps 24
3-4 Transitions 26
3-5 Action Blocks 28
3-6 Conditional Branching and Joining 36
3-7 Parallel Branching and Joining 38
3-8 Subcharts 40
3-9 SFC Jumps 43
3-10 Interrupt Programs 45

SECTION 4
Controlling Step Status 47
4–1 Step Status 48
4–2 SFC Control Instructions 54
4–3 Program Example 65

SECTION 5
SFC Execution Cycle and Errors 67
5–1 Basic Execution Cycle 68
5–2 Execution Cycle with Multiple Active Steps 68
5–3 Execution Cycle with Subcharts 69
5–4 Power-Up and Restart Execution 71
5–5 User Processing Time 74
5–6 SFC Error Codes 76

SECTION 6
SFC Application Examples 79
6–1 Example 1: Sequential Control 80
6–2 Example 2: Conditional Branching and Joining 81
6–3 Example 3: Using AQs 84
6–4 Example 4: Parallel Branching and Joining 88
6–5 Example 5: Ladder vs SFC 91

Glossary 99
Index 115
OMRON Sales Offices 117
Revision History 119

About this Manual:

This manual describes sequential function chart (SFC) programming for the SYSMAC CV-series Programmable Controllers (PCs). Only the CV500, CV1000, and CV2000 support SFC programming. SFC programs cannot be written to the CVM1. This manual is designed to be used together with two other CV-series PC operation manuals and an installation guide. The entire set of CV-series PC manuals is listed below. Only the basic portions of the catalog numbers are given; be sure you have the most recent version for your area.

Manual	Cat. No.
CV-series PC Installation Guide	W195
CV-series PC Operation Manual: SFC	W194
CV-series PC Operation Manual: Ladder Diagrams	W202
CV-series PC Operation Manual: Host Interface	W205

Actual programming for the CV-series PCs is performed with the CV Support Software (CVSS), for which the following manuals are available. (The guidebook provides a brief introduction and training manual and is not essential to CVSS operation.)

Manual	Cat. No.
The CV Series Getting Started Guidebook	W203
CV Support Software Operation Manual: Basics	W196
CV Support Software Operation Manual: Offline	W201
CV Support Software Operation Manual: Online	W200

Please read this manual completely together with the other CV-series PC and CVSS manuals and be sure you understand the information provide before attempting to install, program, or operate a CV-series PC. The basic content of each section of this manual is outlined below.

Section 1 introduces the basic concept and features of SFC programming and provides a simple example of its application.

Section 2 outlines the terminology, basic elements, and procedure used in writing an SFC program and introduces the PC memory areas and Programming Device capabilities for SFC programs.

Section 3 gives a more in-depth look at SFC programming elements and how they go together to create an SFC program.

Section 4 goes on to show ways to control the execution of SFC programs and describes the ladder-diagram instructions that are used with SFC programs.

Section 5 demonstrates the order in which SFC programs are executed, shows how to calculate program execution time, and provides a list of error codes that are generated for SFC programming errors.

Section 6 provides examples that demonstrate the information provided in earlier sections. The last example shows how a ladder-diagram program can be converted to an SFC program and describes the benefits that this provides.

SECTION 1
Outline and Features of SFC

This section introduces SFC programs and provides an example that compares the differences between SFC programs and ladder-diagram programs. Details on SFC programs are provided in later sections.

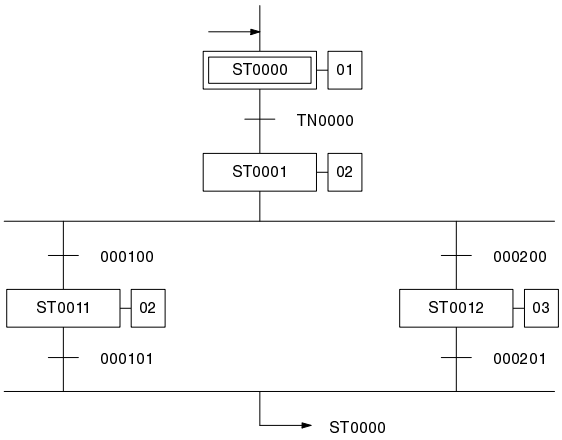
1-1 Outline 2
1-2 Features 2
1-3 Application Example 3

1-1 Outline

Sequential Function Charts, or SFC, is a graphic-oriented programming method developed to provide simple representation of processing sequential steps in control applications.

SFC uses graphic symbols arranged to closely represent the actual sequence of processing, thereby allowing you to program by simply representing the steps of a process and the transition conditions between these steps.

The following illustration shows some of the symbols used in SFC. The boxes represent the processing steps, the lines connecting them represent the processing flow, and the short horizontal lines represent the transition conditions that control processing flow. These symbols are described in more detail later in the manual.



1-2 Features

Improved Design Efficiency

The program is created in the same order as the sequence of processes, thus simplifying program development and allowing greater accuracy. In addition, the time required for programming is greatly reduced.

The entire program can be broken down into progressively more detailed steps, making top-down programming easy and allowing the programming load to be divided among several programmers. The hierarchical and modular structure of the program allows it to be easily reused.

The structured programming and graphic symbols allow programming to proceed without an extensive consideration of hardware so that even beginners can create programs.

The actual control operations within each step can be programmed with ladder diagram instructions basically the same as those used with C-series PCs, allowing partial application of C-series programs. The SFC provides a new method for structuring traditional ladder diagram programs.

Easy Debugging/Maintenance

The graphic-oriented programming method provides visually superior notation. It makes the sequence of steps, the contents of processing, and the conditions for movement between steps visually understandable, and it makes it easy to monitor the movement from one step to another during operation.

Program debugging is simple, as the part of the program corresponding to the process causing trouble can be easily located.

The program can be corrected one section at a time, with little or no influence on other sections. This provides superior maintenance capability.

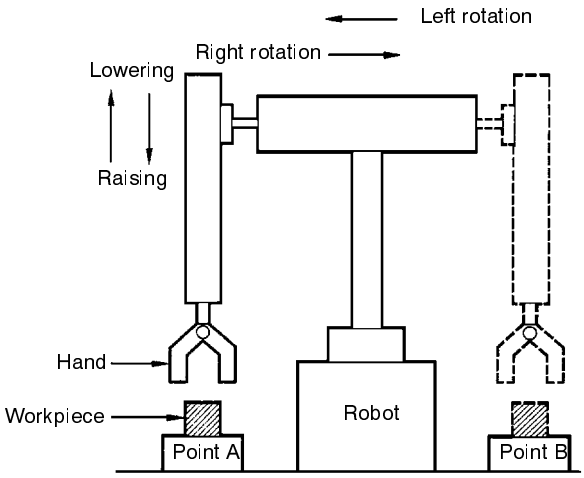
Reduction of Cycle Time

Cycle time is greatly reduced, and processing is thereby speeded up, because only required program steps are executed each cycle.

1-3 Application Example

SFC is well suited to controlling sequential processing. In this simple application example, an SFC program is compared to a traditional ladder diagram program.

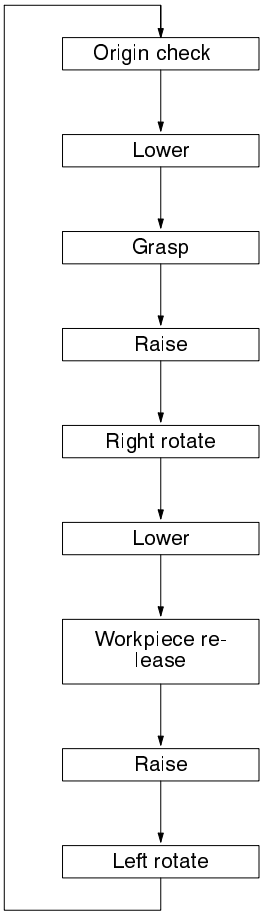
Let's consider an example in which a robot hand, as shown in the illustration below, grasps a workpiece and transfers it from point A to point B.



The robot arm moves as described below. The order of operations is shown on the right.

1, 2, 3...

1. When the start is initiated, a check is made (origin check) to verify that the robot arm is at the origin point and that the hand is open.
2. If the origin check conditions are met, the robot arm is lowered to point A.
3. After the robot arm has been lowered into position, the hand closes and grasps the workpiece.
4. Once the workpiece is securely grasped, the arm is raised.
5. After the arm has been raised, it rotates to the right.
6. When the rotation is complete, the arm is lowered to point B.
7. After the arm has been lowered into position, the hand opens and release the workpiece at point B.
8. The arm is raised again.
9. After the arm has been raised, it rotates to the left and returns to the origin point.

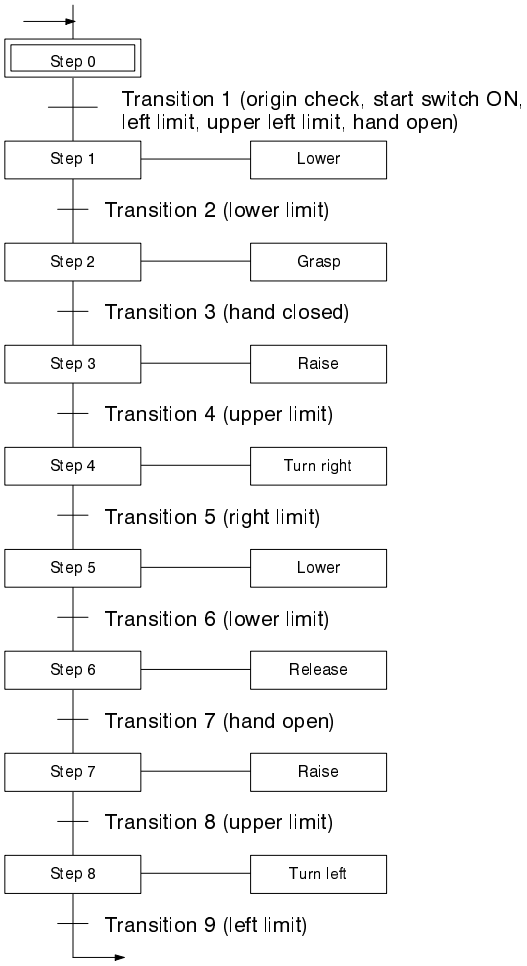


SFC Program

An SFC program is written to represent the actual flow of processes. Each process is written as a separate step. The "actions" that must be carried out for each step are written and steps are separated by the conditions that must be met to move from one step to another.

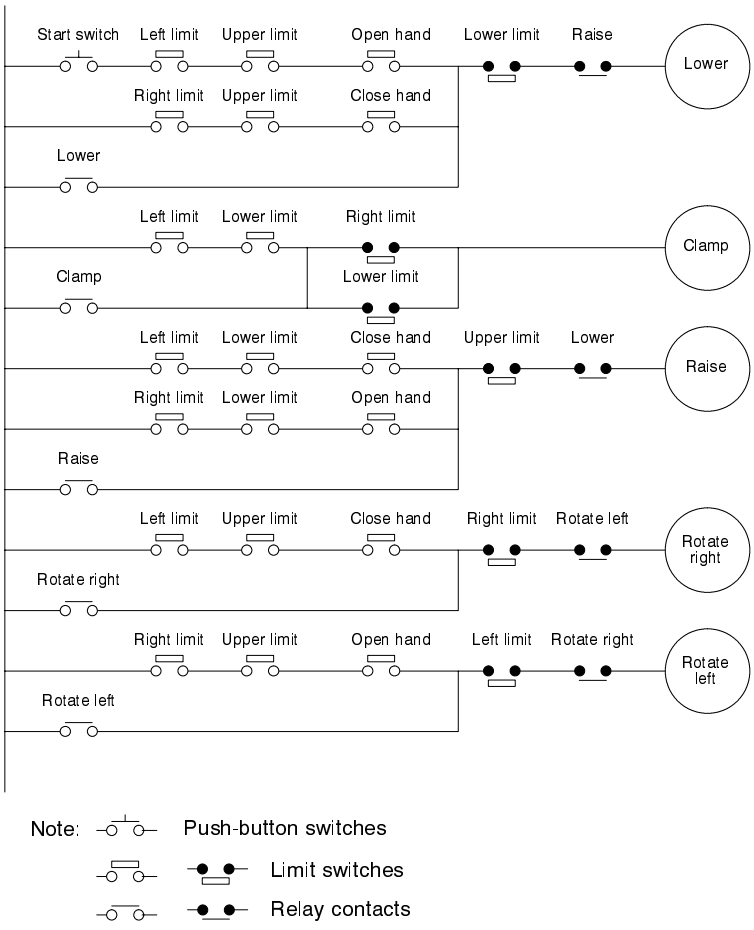
When an SFC program is executed, outputs in previous steps are automatically reset as the program moves from one step to another. This makes SFC ideal for programs that control sequentially executed processes like the one in the above example.

Although this program example uses a separate step for each actuator, multiple processes can be programmed within one step, depending on their complexity.



Ladder Diagram Program

In a ladder program, complex interlocks are required to control execution and the overall flow of processing is not obvious from looking at the program. Only an experienced ladder diagram programmer could readily understand the following example.



SECTION 2
SFC Basics

This section defines basic terminology used to describe SFC programs and then introduces all of the basic elements used to create SFC programs. The way SFC programs operate and the PC memory areas related to SFC programs are also introduced. Finally, an overview of creating an SFC program is provided.

Details on elements in SFC programs are provided in *Section 3 SFC Programming Elements*. Details on controlling step status using ladder-diagram instructions and on the movement of active status through the program are provided in *Section 4 Controlling Step Status* and in *Section 5 SFC Execution Cycle and Errors*. Example SFC programs are provided in *Section 6 SFC Application Examples*.

2-1 SFC Terminology 8
2-2 Basic SFC Operation 11
2-2-1 Sequential Steps 12
2-2-2 Conditional Branching and Conditional Joining 12
2-2-3 Parallel Branching 14
2-2-4 Parallel Joining 14
2-2-5 Status Transfer Conditions 15
2-3 Memory Areas and Programming Devices 17
2-4 Creating an SFC Program 18

2-1 SFC Terminology

Basic SFC terminology is explained below. These terms are explained in more detail later in the manual.

SFC

SFC stands for sequential function chart, and is a method of programming programmable controllers. This method represents an advance over the ladder diagram method, in that it shows the sequence of processes and utilizes the advantages of structured programming. It is highly visually oriented and easy to understand. The method is currently being standardized through the IEC (IEC-SC65A/WG6TF) and JIS.

Sheets

SFC programs are input with the CVSS. If a program is too large, it must be divided into units called sheets. Although smaller programs can be contained in a single sheet, many programs will require multiple sheets. Refer to the CVSS operation manuals for details.

Sequential Processing

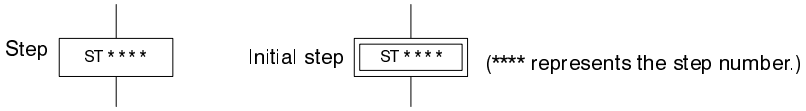
All processing is divided into an ordered series of processes, and each of which is executed in a time sequence in the program. In a ladder program, sequential processing can be represented by step programming instructions, but SFC is superior because steps are visually represented using easy-to-understand symbols.

Steps

A step is a basic element in an SFC program, and each step represents one process. Steps are represented by boxes in the chart, and each step is allocated a step number.

Each step always has a status of either active or inactive. The active status moves from one step to the next step when a transition condition is met. Steps organize the overall structure of the program, and do not themselves perform any control operations. The actual control operations are contained within a step in action blocks, which are explained below.

You can define as “initial steps” the steps that you want to be active when program execution begins.

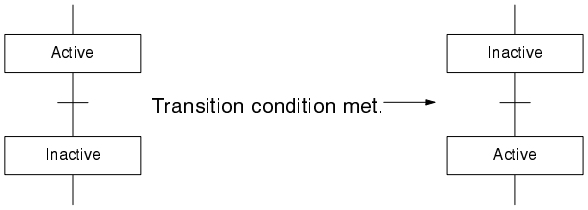


Active and Inactive Status

A step is said to be active when the actions associated with that step are being, or can be, executed. There are three sub-statuses of active status: execute, pause, and halt. (For details, refer to *4–1–3 Subcharts*.) Inactive steps are those in which the actions cannot be executed.

Status Transfers

Status transfer refers to the movement of active status between SFC steps.



Transitions

A transition is a condition which controls the transfer of active status from one step to the next. Only one transition is allowed between any pair of steps and only one step is allowed between any pair of transitions.

The transition moves the active status to the next step when the following three conditions are all met:

• The step above the transition is active.

- The step below the transition is inactive.
- The transition condition is met.

The transition thus serves to control the movement from one step to another.

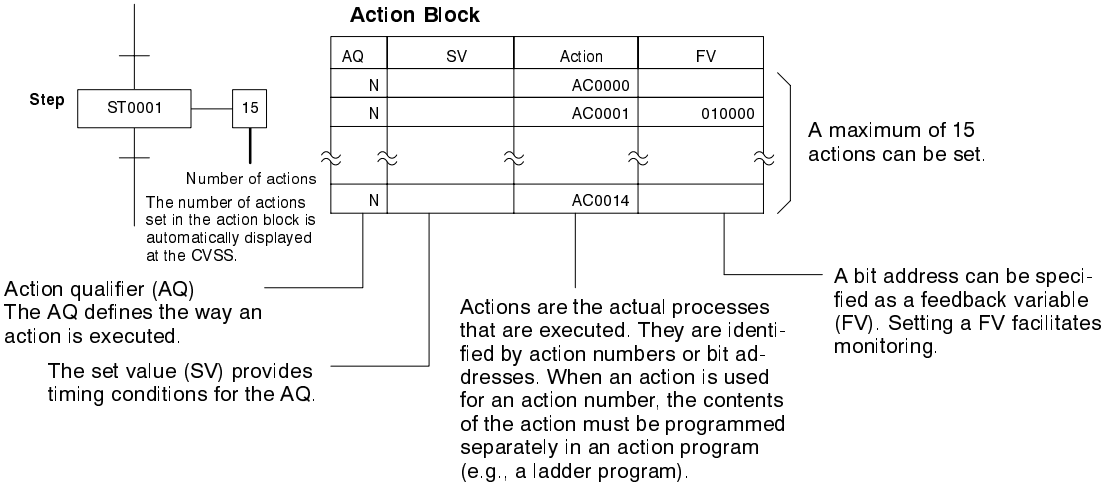
Each transition is identified by a transition number only or by a transition number and a bit address. When only a transition number is used, the transition condition is programmed separately using a ladder program. (Transition programs are explained below.)



Actions

An action is an individual operation contained within a step in an SFC program. Actions are programmed in an action block, which is a list of actions that defines what is to happen in the step.

Each action is identified by either an action number or a bit address. When an action number is used, the contents are programmed separately using a ladder program. When a step becomes active, the actions in that step are executed in order from the top of the action block. You can set from 0 to 15 actions for any one step.



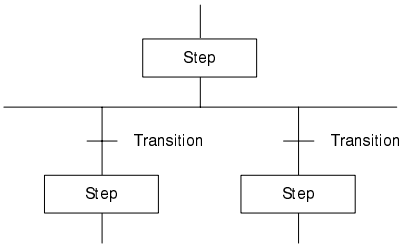
Conditional Branching and Joining

Conditional Branching

A step can be set up so that it branches to more than one step. With conditional branching, each step following the branch has its own transition condition so that program execution can move to any of these steps.

When the step before the branch is active, then the active status will be transferred to one of the steps after the branch when its transition condition is met (assuming it is not already active), but never to more than one step. If two or more transition conditions are met simultaneously, then the step furthest to the left is executed.

A conditional branch is represented in the chart by a single horizontal line.

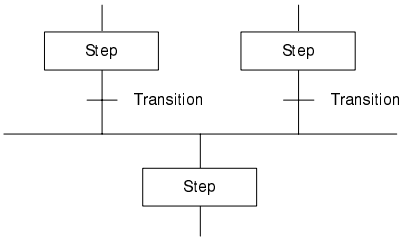


Conditional Joining

Two or more steps can be set up so that they join to one step. With conditional joining, each step before the join has its own transition condition so that program execution can move from either of these steps.

When the step after the join is inactive, then the active status can be transferred to that step if either of the steps before the join is active and its transition condition is met. Unlike parallel joining (described below), where the active status moves simultaneously from two or more steps, here it can be passed on from a single step.

A conditional join is represented in the chart by a single horizontal line.

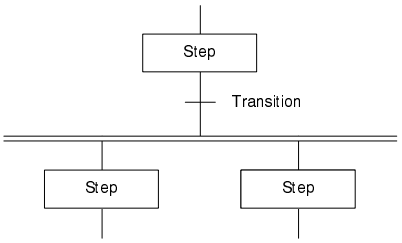


Parallel Branching and Joining

Parallel Branching

A step can be set up so that it branches to more than one step. With parallel branching, there is only one transition condition for all the steps after the branch so that program execution moves to all of these steps simultaneously. If the step before the transition is active and all the branch steps are inactive, then the active status will move simultaneously to all the branch steps when the transition condition is met.

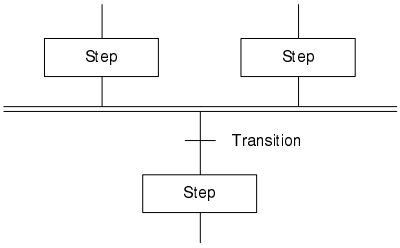
The parallel branch is represented in the chart by two horizontal lines.



Parallel Joining

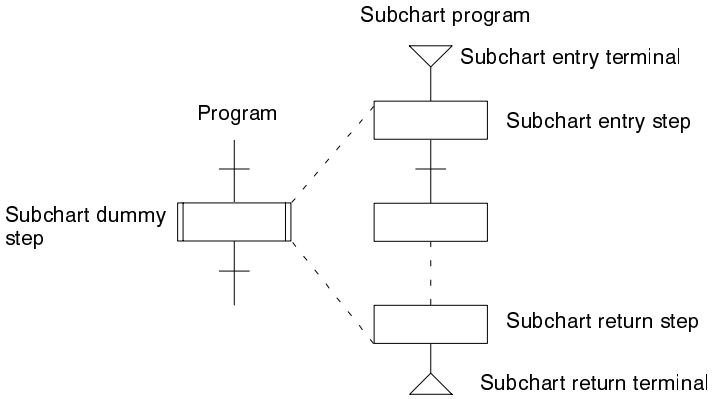
Two or more steps can be set up so that they join to one step. With parallel joining, there is only one transition condition for all the steps before the branch so that program execution moves from all of these steps simultaneously. If the steps before the transition are all active, the step following the transition is inactive, and the transition condition is met, then the active status will move simultaneously from all of the steps before the transition to the step after the transition. This means that all steps which were processed in parallel are synchronized at the parallel join. This feature makes synchronization of processing very simple with SFC.

The parallel join is represented in the chart by two horizontal lines.



Subcharts

Subcharts are similar to the subroutines used by computer programming languages. They function as macros to manage a series of processes as a single step in an SFC program. You can create an independent subchart and call it from several places within a program.



SFC Jumps

The SFC jump is used to move execution of an SFC program to another step in a different section of the program. A jump can be made only from just after a transition to just before a step. In the illustration, the program jumps to step number ST0010. The jump destination is specified with an SFC jump entry arrow.



SFC Control Instructions

SFC control instructions are a group of ladder diagram instructions that can be used in an action program to change the operating status of steps or subcharts in the SFC program. You can use these instructions when you want a program to be executed in a different order than usual. The six SFC control instructions are as follows:

SA(210): . . . . . ACTIVATE STEP
SP(211): . . . . . PAUSE STEP
SR(212): . . . . . RESTART STEP
SF(213): . . . . END STEP
SE(214): . . . . . DEACTIVATE STEP
SOFF(215): . . RESET STEP

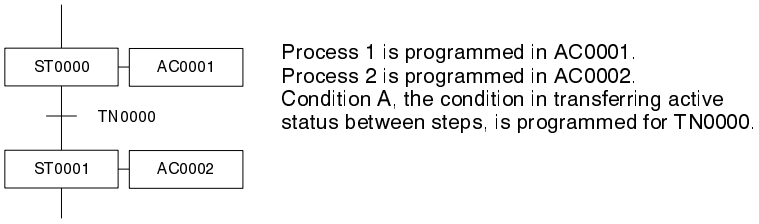
2-2 Basic SFC Operation

This section will explain an SFC program using four basic SFC features: sequential steps, conditional branching, parallel branching, and synchronous steps. The diagrams are only symbolic, for the purpose of explanation, and are not identical to actual CVSS displays.

2-2-1 Sequential Steps

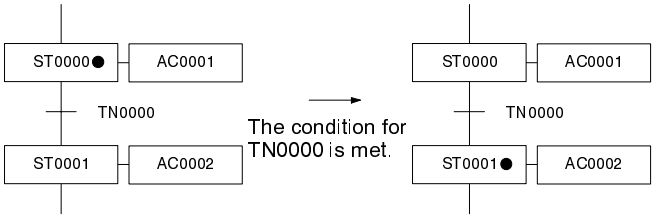
The sequential processing shown below is basic to all SFC programs.

In this example, first step 1 is executed, and then when condition A (the transition) is met, step 1 is stopped and step 2 is executed.



Any step is either active or inactive. Active status moves from one step to another according to the status of the transition conditions and the status of the steps before and after the transitions. When the active status moves, all previous processes are reset.

In the example shown below, active status is transferred from step ST0000 to step ST0001 at the moment that the condition is met for transition TN0000. The black dot indicates the active step.



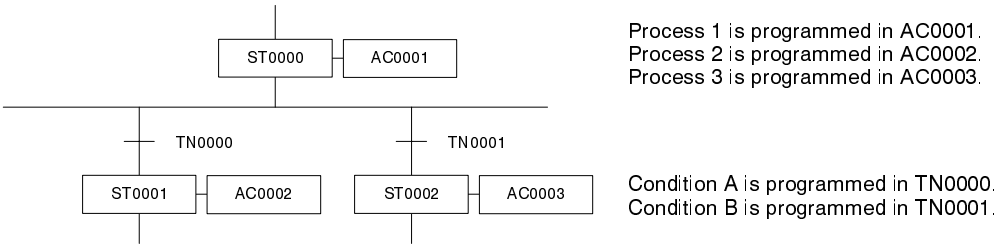
When a step becomes active, it will normally remain active for at least one execution cycle even if the transition condition is met for active status to move to the next step. If you wish, however, you can use an action qualifier (AQ) to prevent execution during the first execution cycle.

Note When active status is transferred from one step to another, the step that becomes inactive is reset, i.e., the actions associated with that step are reset. If you wish you can add the optional AQ symbol H (for hold) to certain AQs (N, P, L, D, and R) to prevent outputs or timers from being reset after execution of an action is complete. In addition, S-group AQs (S, SL, SD, and DS) set an action for continued execution even after the active status of its associated step moves to the next step. Resetting an action means that the bit turns OFF (when the action is a bit address), or that the OUT/OUT NOT instructions are turned OFF and the TIM/TIMH instructions are reset (when the action is an action program).

2-2-2 Conditional Branching and Conditional Joining

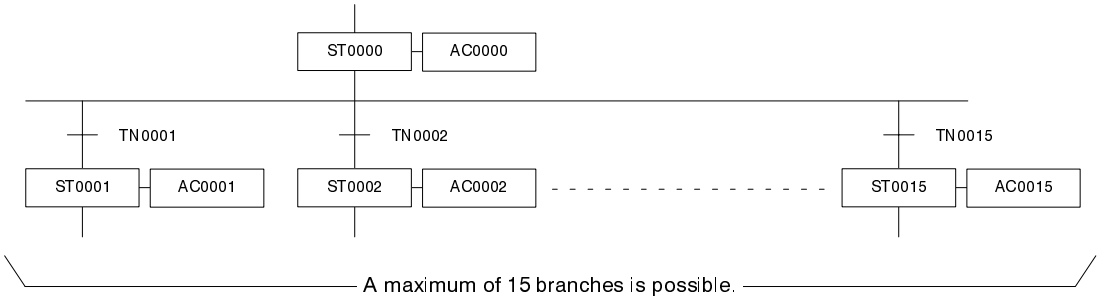
In order for subsequent steps to be selected based on independent conditions, a single horizontal line is used to specify conditional branching as shown in the illustration below. One transition is programmed for each branched step. These transitions set the conditions for transferring active status to one of the branched steps. This kind of branching is called “conditional branching.” In this example,

step 0 (process 1) is being executed. Step 1 (process 2) will be executed if condition A is met, and step 2 (process 3) will be executed if condition B is met.

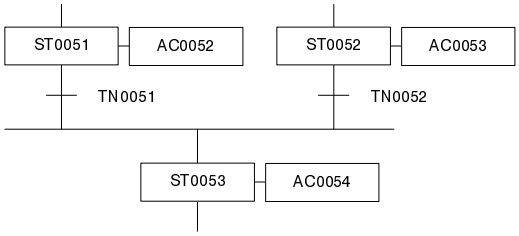


Conditions A and B can be programmed as mutually exclusive conditions so that only one of them is met at any particular time. If there are three or more conditions, then more branches can be programmed (to a maximum of 15).

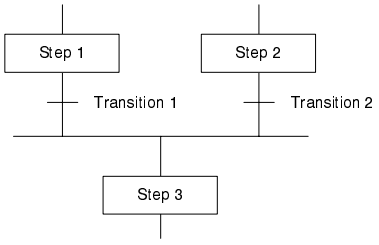
If none of the transition conditions are met, then the active status of the steps will not change. If two or more are met simultaneously, then the transfer condition at the left will be given priority, and the leftmost branch step will become active.



When multiple branched steps are joined into one, it is programmed as shown below. This type of program is called “conditional joining.” A separate transition is necessary, before the join, for each of the branched steps.

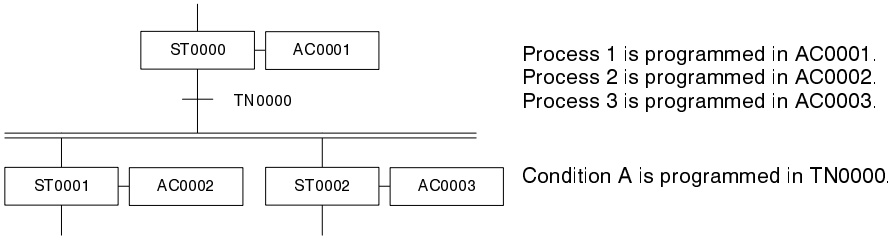


Note If step 1 and step 2 in the following chart become active simultaneously, steps and transitions will be processed in the following order: step 1, transition 1, step 2, transition 2, and back to step 1. When the condition is met for either transition 1 or transition 2, then active status will be transferred from the corresponding step to step 3. If the condition is then met for the other transition, active status will not be transferred from the corresponding step until step 3 returns to inactive status.

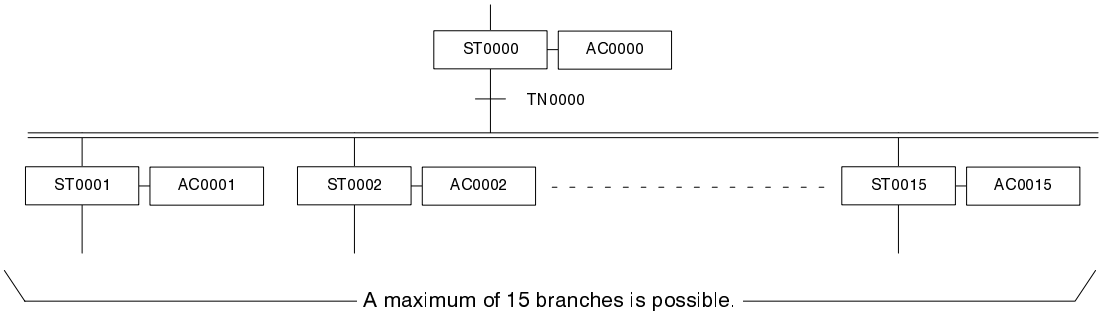


2-2-3 Parallel Branching

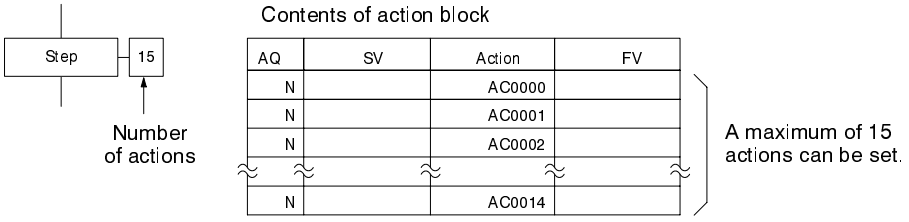
To specify that multiple processes (steps) are to be executed in parallel, use a double horizontal line to indicate parallel branching as shown in the illustration below. The transition (the condition that must be met for active status to move to the parallel steps) must be programmed above the double line. This kind of branching is called "parallel branching." In this example, step 1 (process 2) and step 2 (process 3) are executed simultaneously after step 0 (process 1) is executed and condition A (the transition) is met.



If there are three or more steps to be executed in parallel, then the number of branches can be increased (to a maximum of 15).



There are actually two ways to program parallel execution. In addition to the parallel branching method shown above, you can also program up to 15 actions for one step. By using both of these methods together, you can program as many as 225 actions to be executed simultaneously.

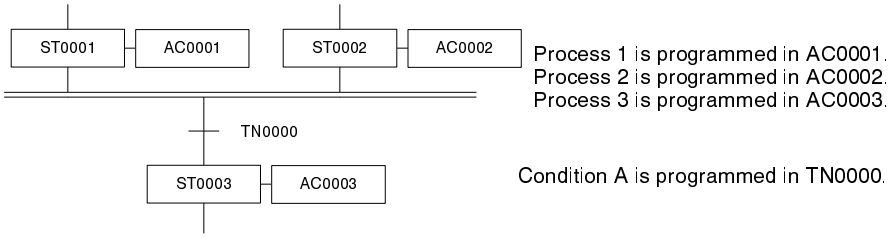


Note Although execution for parallel branching is considered simultaneous, it is, technically speaking, cyclic. Refer to *Section 5 SFC Execution Cycle and Errors*.

2-2-4 Parallel Joining

When you want multiple steps to be executed in parallel, and the program to wait for all of those steps to be completed before moving active status to the next step, then use a double horizontal line to specify parallel joining as shown in the illustration below. The transition (the condition that must be met for active status to move from the parallel steps to the single step) must be programmed below the double line. This kind of branching is called "parallel joining." In this example, steps 1 and 2 (processes 1 and 2) are executed in parallel. When execution of

both of these steps is complete and the transition condition is met, then step 3 (process 3) is executed.



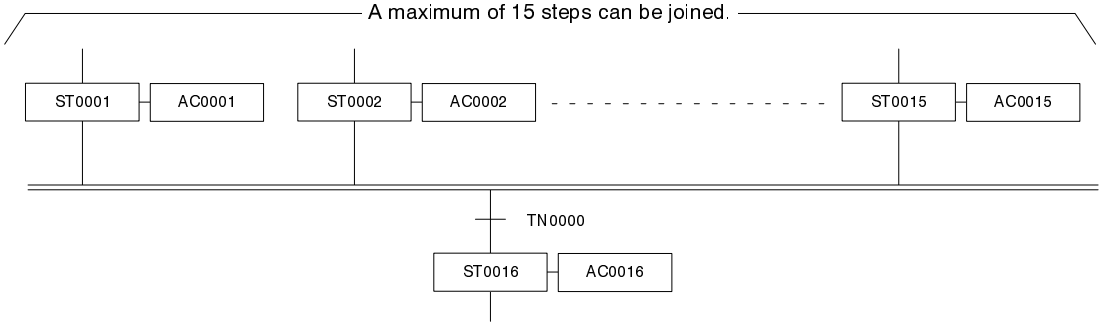
Multiple steps in active status cannot be joined into one unless all of the following three conditions are met:

1, 2, 3...

1. All of the steps immediately above the transition are active.
2. The step just below the transition is inactive.
3. The transition condition is met.

In this example, that would mean that, in order for active status to move from steps ST0001 and ST0002 to step ST0003, steps ST0001 and ST0002 must be active, step ST0003 must be inactive, and the condition must be met for transition TN0000.

If there are three or more parallel steps to be joined, then the number of parallel steps can be increased (to a maximum of 15).



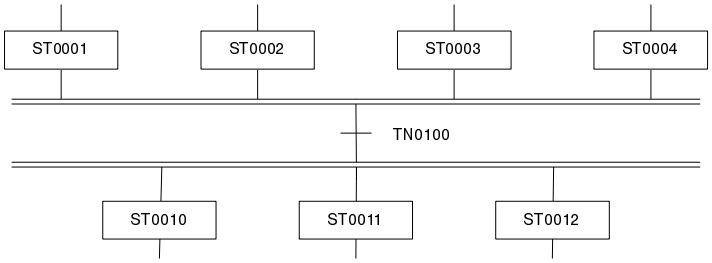
2-2-5 Status Transfer Conditions

In an SFC program, the condition necessary for active status to be transferred from one step to the next step is represented by a transition. The status of the transition alone, however, is not enough: the active or inactive status of the steps above and below the transition must also be considered before you can determine if active status will move. As long as the combinations of steps and transitions in an SFC are programmed according to the rules, the transfer of active status will take place properly. In other words, not only must the transition condition be met, but the step(s) before the transition must be active and the step(s) after it must be inactive.

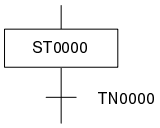
Note Even if the step before the transition is active, it must not be paused or transfer of active status cannot take place. (Active status includes execute, pause, and halt status.) In addition, the transfer of status will not take place for at least one execution cycle after the conditions described above have all been met.

In the example shown below, the transfer of status can take place only if steps ST0001, ST0002, ST0003, and ST0004 are all active, steps ST0010, ST0011, and ST0012 are all inactive, and the condition is met for TN0100. Even after all of

these conditions are met, the transfer of status will not take place for one cycle. For details, refer to *4-1 Step Status*.



Note In the illustration below, there is no step after the transition. If ST0000 is active and the condition is met for TN0000, then the active status will be transferred from ST0000 and simply disappear. This is because there is no active step after the transition. The system treats a non-existent step the same as an inactive step. If there are no other steps with active status, an SFC Stop Error will be generated and operation will be stopped.



2-3 Memory Areas and Programming Devices

PC Memory Affecting SFC

The following tables show the parts of and limits to PC memory that affect SFC operation either directly or indirectly. Refer to the *CV-series PC Operation Manual: Ladder Diagrams* for details on PC memory.

Item	Limit
Number of actions/step	15
Conditional branches from 1 step	15
Conditional joins to 1 step	15
Parallel branches from 1 step	15
Parallel joins to 1 step	15

Name	CV500		CV1000/CV2000	
	Qty	Addresses	Qty	Addresses
Initial steps	31	ST0000 to ST0511	31	ST0000 to ST1023
Steps	512	ST0000 to ST0511	1,024	ST0000 to ST1023
Transitions	512	TN0000 to TN0511	1,024	TN0000 to TN1023
Actions	1,024	AC0000 to AC1023	2,048	AC0000 to AC2047
Subcharts	255	ST0000 to ST0511	511	ST0000 to ST1023
I/O interrupts	32	I/O INT00 to I/O INT31	32	I/O INT00 to I/O INT31
Scheduled interrupts	1	CYCLIC INT0	2	CYCLIC INT0 and CYCLIC INT1
Power off interrupt	1	PWR OFF INT	1	PWR OFF INT
Power on interrupt	1	PWR ON INT	1	PWR ON INT
I/O and work bits	40,896	000000 to 255515	40,896	000000 to 255515
CPU bus link bits	4,096	G00000 to G25515	4,096	G00000 to G25515
Auxiliary bits	8,192	A00000 to A51115	8,192	A00000 to A51115
Timers	512	T0000 to T0511	1,024	T0000 to T1023
Counters	512	C0000 to C0511	1,024	C0000 to C1023

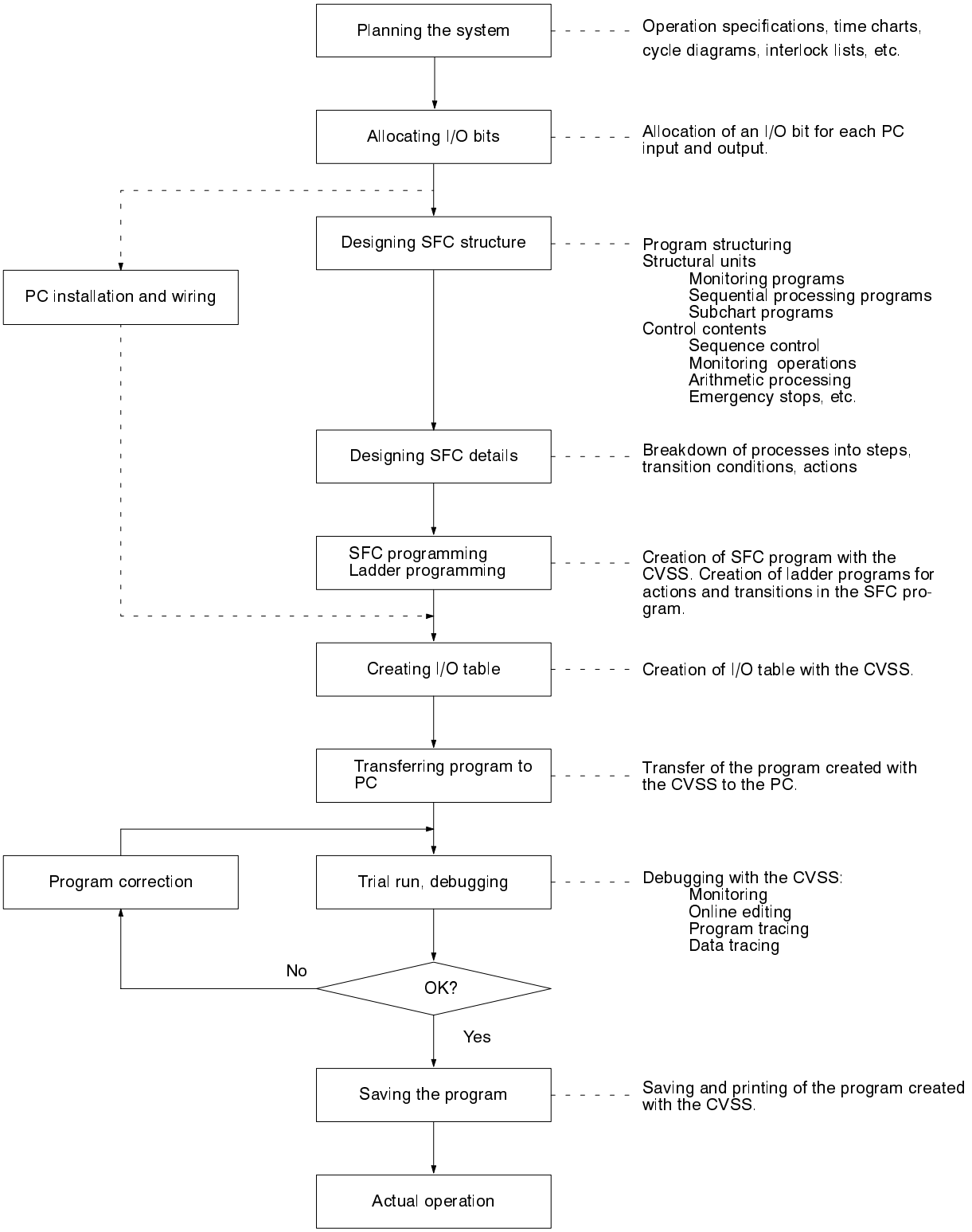
SFC Programming Devices

SFC programming and monitoring can be done with the CVSS. The main operations available are shown in the following table.

Offline operations	Online operations (one sheet or entire program)
Editing actions	Displaying actions
Editing steps/transitions	Transferring sheets from PC
Reading/writing sheets	Changing SFC programming parameters
Saving/retrieving programs	Full-screen monitoring
Setting SFC programming parameters	Part-screen monitoring
Displaying memory usage	Editing SFC programs (with PC in MONITOR, PROGRAM, or DEBUG mode)
Clearing memory	Reading the cycle time
Printing SFC sheets	Clearing memory areas
Checking SFC programs	Running action traces to store SFC execution history
	Debugging by executing individual steps or specified program sections

2-4 Creating an SFC Program

The procedure from SFC programming to operation can be outlined as follows:



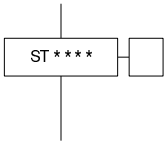
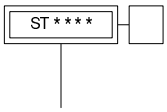
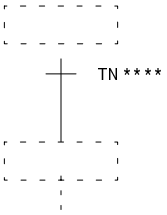
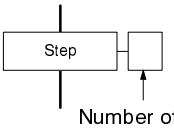
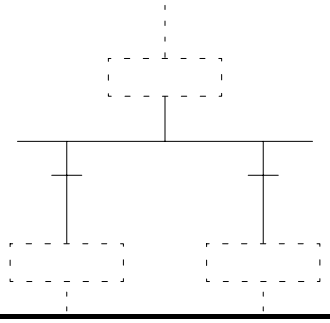
SECTION 3
SFC Programming Elements

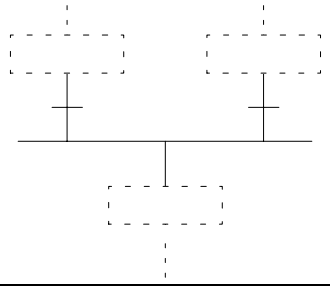
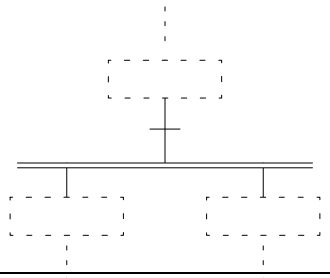
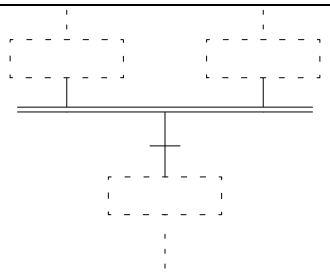
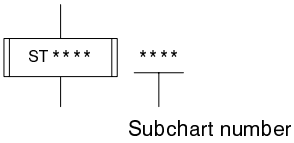
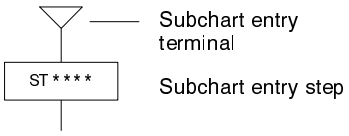
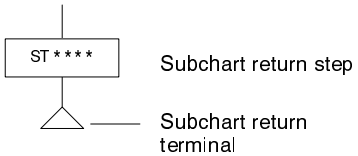
This section provides more detailed explanations of the elements used to create SFC programs. These elements are introduced in *Section 2 SFC Basics*.

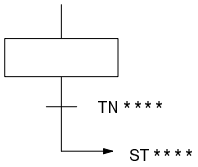
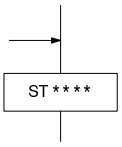
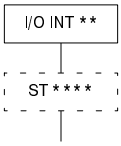
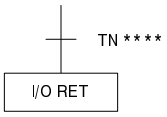
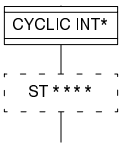
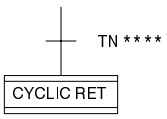
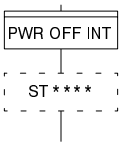
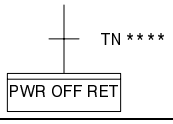
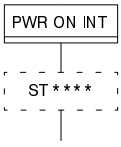
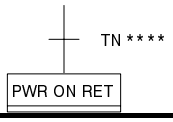
3-1 Overview 20
3-2 Steps 23
3-3 Initial Steps 24
3-4 Transitions 26
3-4-1 Transition Bits 27
3-4-2 Transition Programs 27
3-5 Action Blocks 28
3-5-1 Action Qualifiers 29
3-5-2 AQ Hold Option 31
3-5-3 Action Execution Timing 33
3-5-4 Action Reset Timing 34
3-5-5 AQ and the Execution Cycle 35
3-6 Conditional Branching and Joining 36
3-7 Parallel Branching and Joining 38
3-8 Subcharts 40
3-9 SFC Jumps 43
3-10 Interrupt Programs 45

3-1 Overview

The following table introduces the structural elements used to create SFC programs. These elements are described in more detail in the remainder of section 3.

Name	Symbol	Function	Operand data areas	Page
Step		A step represents a process in the program. Each step is either active or inactive. ST**** represents the step number.	Step number CV500 ST0000 to ST 0511 CV1000/CV2000 ST0000 to ST1023	23
Initial step		The initial steps are the steps that are given active status at the beginning of operation. Every program must have at least one initial step, and can have up to 31. ST**** represents the step number.	Step number CV500 ST0000 to ST0511 CV1000/CV2000 ST0000 to ST1023	24
Transition		A transition is a condition which transfers the active status from one step to the next. Only one transition is allowed between any pair of steps. Each transition is defined as a transition number only or as a transition number and a bit address. TN**** represents the transition number.	Transition number CV500 TN0000 to TN0511 CV1000/CV2000 TN0000 to TN1023 Bit addresses Refer to page 26.	26
Action	 Action block AQ SV Action FV	Actions for a step are set up in an action block. When a step becomes active, actions are executed according to their AQs. A maximum of 15 actions can be input for a single step. **AQ:** Defines the way an action is executed. AQs include: N, P, L, D, R, S, SL, SD, DS, NH, PH, LH, DH, and RH. **SV:** Set values (times) for L, D, SL, SD, DS, LH, and DH AQs. **Action:** An action number indicating an action program or a bit address. **FV:** A bit address.	**SV** 00000 to 65535 (Unit: 0.1/1.0 s) **Action number** CV500 AC0000 to AC1023 CV1000/CV2000 AC0000 to AC2047 **Bit addresses** (Used for actions and FV.) Refer to page 28.	28
Conditional branch		Active status is transferred to the next step for which the transition condition is met. If two or more transition conditions are met simultaneously, the leftmost step is given priority. Up to 15 branches can be made from one step.	---	36

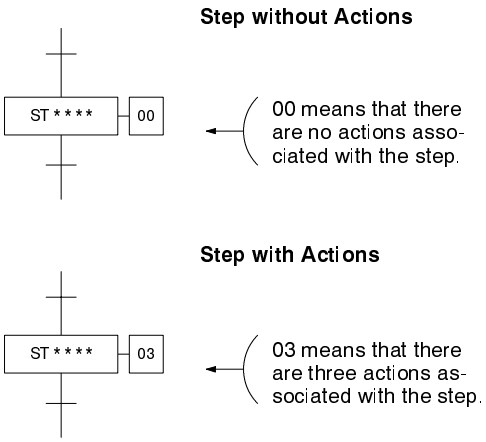
Name	Symbol	Function	Operand data areas	Page
Conditional join		When either of the transition conditions is met, active status is transferred from the step above that transition to the step just after the join (as long as the step above the transition is active and the step just after the join is inactive). Up to 15 transitions can be connected for joining to one step.	---	36
Parallel branch		If the step above the double line is active and the steps below the double line are all inactive, then the active status will move simultaneously to the steps below the double line when the transition condition is met. Up to 15 steps can be connected for branching from one transition.	---	38
Parallel join		If the steps above the double line are all active, the step below the double line is inactive, and the transition condition is met, then the active status will move simultaneously from all of the steps above the double line to the step below the double line. For one transition, up to 15 steps can be connected for joining.	---	
Subchart dummy step		A subchart dummy step is a step for calling a subchart. ST**** represents the dummy step number. The subchart number is displayed to the right of the subchart dummy step symbol. This is the number of the subchart entry step of the subchart to be called.	Step number CV500 ST0000 to ST0511 CV1000/CV2000 ST0000 to ST1023	40
Subchart entry terminal Subchart entry step		Indicate the starting position of the subchart. ST**** represents the subchart entry step number. That number becomes the number of the subchart. Up to 255 subchart entry terminals/steps can be used for the CV500, and up to 511 for the CV1000 or CV2000.	Step number CV500 ST0000 to ST0511 CV1000/CV2000 ST0000 to ST1023	
Subchart return step Subchart return terminal		Indicate the return position of the subchart. ST**** represents the subchart return step number.	Step number CV500 ST0000 to ST0511 CV1000/CV2000 ST0000 to ST1023	

Name	Symbol	Function	Operand data areas	Page
SFC jump		Indicates the jump source in an SFC program. It is written after a transition. ST**** represents the step number of the jump destination.	Step number CV500 ST0000 to ST0511 CV1000/CV2000 ST0000 to ST1023	43
SFC jump entry		Indicates the jump destination in an SFC program. It is written before the step to which the jump is being made.	---	
I/O interrupt entry terminal		Indicates the starting position of an I/O interrupt program. It is connected to a step. ** represents the I/O interrupt number.	I/O interrupt number 00 to 31	45
I/O interrupt return terminal		Indicates the end of the I/O interrupt program. It is connected after a transition.	---	
Scheduled interrupt entry terminal		Indicates the starting position of a scheduled interrupt program. It is connected to a step. ** represents the scheduled interrupt number.	Scheduled interrupt number CV500 0 CV1000/CV2000 0, 1	
Scheduled interrupt return terminal		Indicates the end of the scheduled interrupt program. It is connected after a transition.	---	
Power-off interrupt entry terminal		Indicates the starting position of a power-off interrupt program. It is connected to a step.	---	
Power-off interrupt return terminal		Indicates the end of the power-off interrupt program. It is connected after a transition.	---	
Power-on interrupt entry terminal		Indicates the starting position of a power-on interrupt program. It is connected to a step.	---	
Power-on interrupt return terminal		Indicates the end of the power-on interrupt program. It is connected after a transition.	---	

3-2 Steps

A step in an SFC program represents a process in the real world. Each step consists of from 0 to 15 actions. Actual control operations are programmed within the actions.

Steps are represented in the program by boxes, as shown in the following examples.



Each step is identified by a unique step number. In the above examples, ST**** represents the step number. Each step must have a step number, and each step number can be used only once and must be within the following ranges.

Model	CV500	CV1000/CV2000
Step number	ST0000 to ST0511	ST0000 to ST1023

Each step is either active or inactive. There are actually three different statuses which are considered active status, as shown in the following table.

Status		Meaning
Active	**Execute**	Executing or awaiting execution.
	Pause	Execution temporarily stopped.
	Halt	Execution completed.
Inactive		Action inactive.

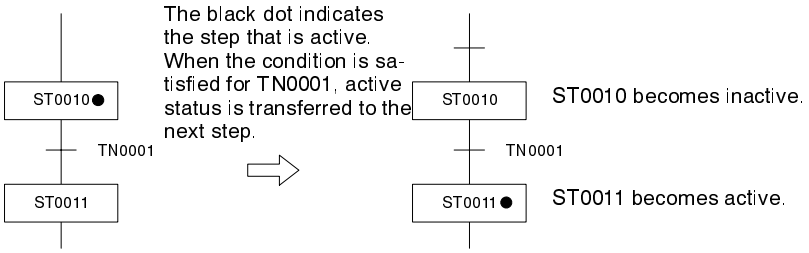
Actions are executed when the step has execute status. They are not executed when the step has inactive, pause, or halt status.

Active status is transferred from one or more steps to the next step(s) when the following three conditions are met. For additional details, refer to the relevant explanations of branching and joining.

- The transition condition is met.
- The step(s) before the transition is active (and not paused).
- The step(s) after the transition is inactive.

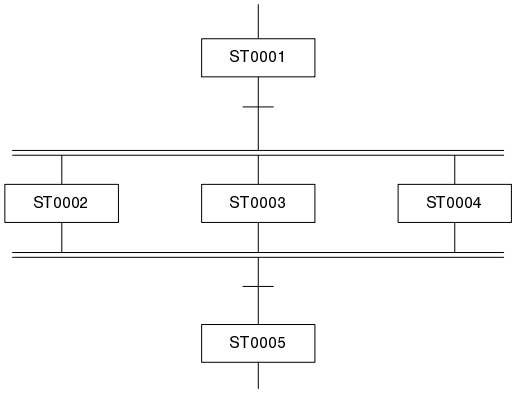
It is possible to program a step with no actions. A step without any actions is called a dummy step. A dummy step causes the program to do nothing until the next transition condition is met.

Application Example



Note

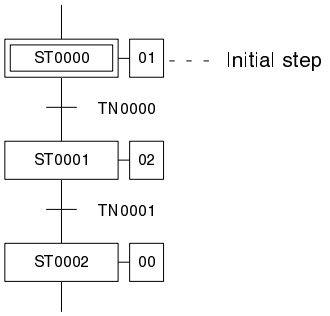
1. Evaluation of transition conditions does not occur until after the actions associated with a step have been executed once, so the actions are executed for at least one execution cycle.
2. When multiple steps are executed in parallel, evaluation of transition conditions is performed immediately after the execution of each step. Active steps are all executed for at least one execution cycle. Even if the transition condition is met, active status transfer takes place from the second cycle onwards.



In this example, ST0002, ST0003, and ST0004 are all assumed to be active. If the transfer condition is met immediately after the execution of ST 0003, then the active status will be transferred to ST0005 without executing ST0004 only if ST0004 has been executed at least once since it went active.

3-3 Initial Steps

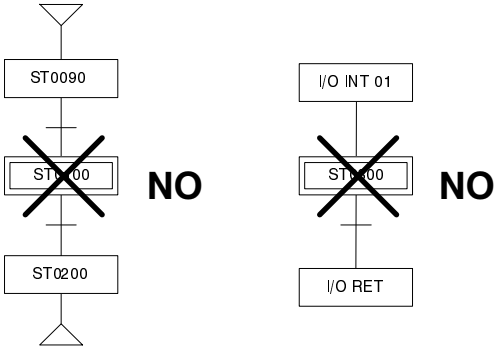
Initial steps are essentially the same as other steps except that they are automatically given active status when the program starts. All steps other than the initial steps are initially inactive. Initial steps can thus be used to program processing that is always to be executed on system startup. Refer to the previous section for general details on steps.



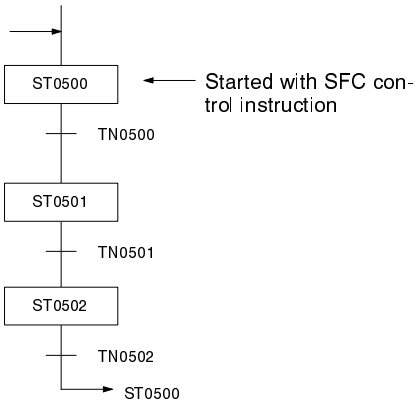
If there are multiple initial steps in one program, then all of them will be active when the program is started. The order of priority for execution will be from top to bottom and then left to right.

If a program occupies more than one sheet, then the order of priority for execution will be from the smaller sheet numbers to the larger. (For details on sheets, refer to the CVSS operation manuals.)

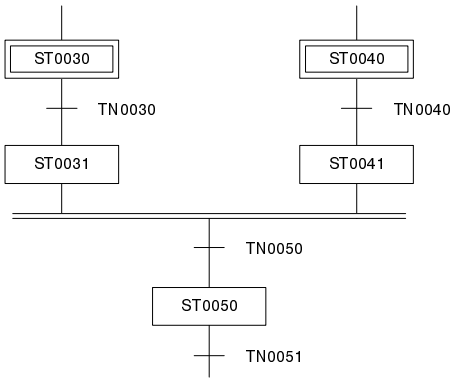
Note 1. An initial step cannot be placed inside of a subchart or an interrupt program.



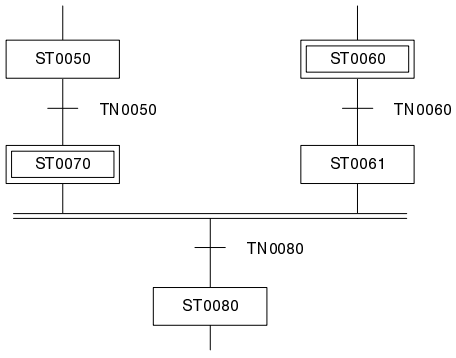
2. You can program a loop with no initial step, but it can only be activated with a SFC control instruction.



Application Examples The following examples demonstrate initial step execution.



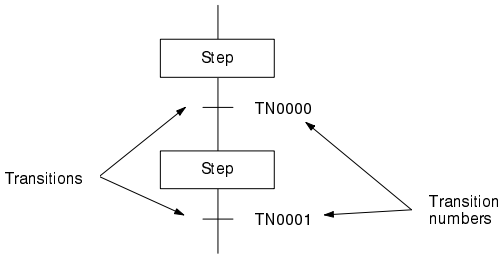
When the above program begins, ST0030 and ST0040 will both be active, ST0030 will be executed before ST0040, and each will be executed for at least one cycle.



When the above program begins, ST0060 and ST0070 will both be active, ST0060 will be executed before ST0070, and each will be executed for at least one cycle.

3-4 Transitions

A transition is the condition which transfers active status from one step to the next. Only one transition is allowed between any pair of steps.



Each transition must be given a transition number, and each transition number can be used only once.

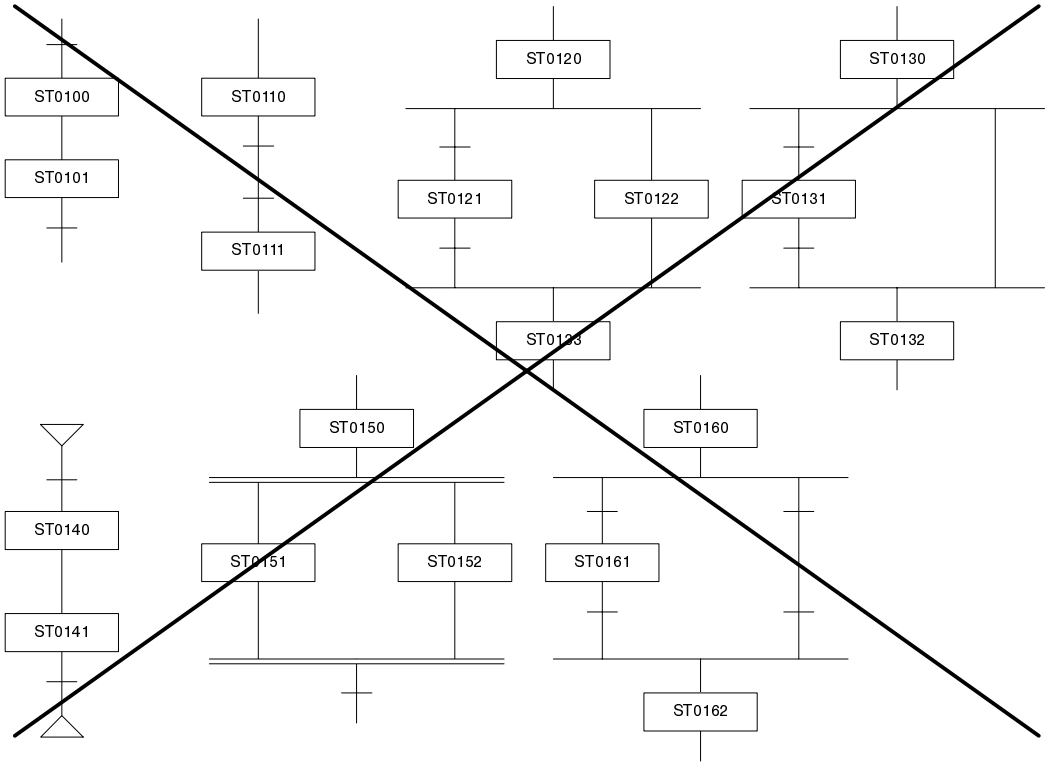
There are two ways to define the condition for transferring active status. It can either be defined using a bit address, or it can be defined using a transition program. A transition program is written as a ladder program. If it is defined using a bit address, then the condition will be ON or OFF depending on the ON/OFF status of the bit. If it is defined using a transition program, then the condition will be on or off depending on the logic status output from the transition program by instructions such as TOUT(202) and TCNT(123). Details on these methods of definition are given below.

The ON/OFF status of a transition can be accessed through its Transition Flag. Each transition has a Transition Flag that is accessed through the transition number in programming.

Note

1. The Transition Flag and the transition program are not reset even after active status is transferred. The Transition Flag thus remains ON, and the outputs and timers in the transition program continue to operate. Be especially careful when using the status of the Transition Flag as an input condition for another operation, or when using a timer in the transition program. The Transition Flag will be turned OFF only when the transition program or bit turn it OFF.
2. Active status is not transferred merely when the transition turns ON. In addition, the step(s) before the transition must be active, and the step(s) after the transition must be inactive.
3. The First Cycle Flag (A50015) cannot be used in a transition program.
4. We strongly recommend that you do not use transition programs to implement control operations. Transition programs should be used only to control program progression between steps. Program design and maintenance will become very difficult if transition programs are used for control operations, which belong in action programs.

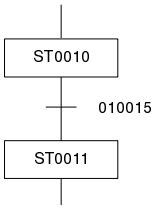
5. A transition must always be placed between two steps. The illustration below shows examples of **incorrect** programs.



3-4-1 Transition Bits

A transition can be defined using a bit so that the status of the bit controls the transition between steps. A transition must be assigned a transition number even when the transition is defined by a bit address.

When a bit address is set, it is displayed on the SFC as shown below (for bit 010015).



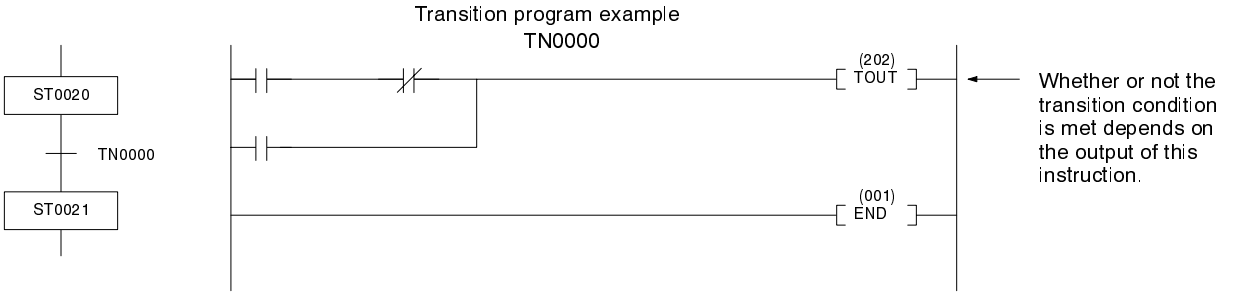
The bit addresses that can be used to define transitions are given in the table below.

Model	CV500	CV1000/CV2000
Transition numbers	TN0000 to TN0511	TN0000 to TN1023
Bit numbers	000000 to 255515 G00000 to G25515 A00000 to A51115 T0000 to T0511 C0000 to C0511 ST0000 to ST0511	000000 to 255515 G00000 to G25515 A00000 to A51115 T0000 to T1023 C0000 to C1023 ST0000 to ST1023

3-4-2 Transition Programs

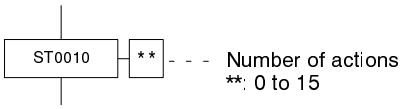
When the transition condition is defined using a transition program, the program is written as a ladder diagram. Either TOUT(202) or TCNT(123) must be in-

cluded in the transition program to output the transition condition. An example of a transition program is shown below.



3-5 Action Blocks

Each step in an SFC program contain a list of the operations that are to be executed for the step. This list is called an action block and the individual operations are called actions. The number of actions in the action block for any step can be displayed to the right of the step number, as shown below.



The action block can contain 0 to 15 actions, each of which contains an action qualifier, a set value, the action itself, and a feedback variable, as shown below.

AQ	SV	Action	FV
N		AC0000	
N		AC0001	010010
N		000015	
etc.			

If there are no actions set for a step, then nothing will be executed even when that step becomes active. A step with no actions is called a dummy step.

The values that are allowed for each field in the actions block are shown below. These are explained individually next.

Model		CV500	CV1000/CV2000
Action qualifier		N, P, L, D, R, S, SL, SD, DS, NH, PH, LH, DH, RH	
Set value	**Constant**	#0000 to #9999	
	Word setting	0000 to 7FFD (00000 to 32765)	
Action	**Action number**	AC0000 to AC1023	AC0000 to AC2047
	Bit address	000000 to 255515 A00000 to A25515 G00800 to G25515	
FV: bit address		000000 to 255515 A00000 to A51115 G00000 to G25515 TN0000 to TN0511 ST0000 to ST0511 T0000 to T0511 C0000 to C0511	000000 to 255515 A00000 to A51115 G00000 to G25515 TN0000 to TN1023 ST0000 to ST1023 T0000 to T1023 C0000 to C1023

Action Qualifiers

The timing of action execution is set with action qualifiers (AQ). You must set an AQ for each action. When a step becomes active, each action in that step is executed according to its AQ setting. AQ are described in detail below.

Set Values

Set values (SV) are related to the timing determined by the AQ, and are not required for every AQ. The following AQs require an SV: L, D, SL, SD, DS, DH, and LH.

Set values can be input as constants (by placing # before the value input) or as a word address (to specify the word containing the SV). When an SV is input as a constant (with #), the value is in BCD. When a word address is input, the contents of that word will be handled in binary.

All of the set values within the same action block are timed by the step timer. Each step has its own step timer. Step timers are incremental timers that are reset and begin counting when the step becomes active, and which retain the present value when the step becomes inactive. If, however, the actions in the step continue to be executed (e.g., for S-group AQs), the step timer continues operating until all execution has been completed.

The PC can be set so that step timers operate in increments of either 0.1 second or 1 second. The increment is set in the PC system settings. The default setting is 0.1 second. (Refer to PC system setting in the CVSS manuals for details.)

Step timers can count up to 6553.5 s (when the unit is 0.1 s) or 65,535 s (when the unit is 1 s). The step timer retains the maximum value if the present value goes beyond the timeable range.

Actions

The action field defines the operation to be performed. If a bit address is used, the status of the bit is controlled according to the other fields. If an action number is used, a separate ladder diagram action program is written for the action and the execution of the action program is controlled by the other fields.

Note If an attempt is made to execute the same action at the same time from more than one step, a duplication error will be generated but the action will be executed. If you wish, you can change the PC system settings so that this error will not be generated. For details, refer to the section on PC system settings in the *CV-series PC Operation Manual: Ladder Diagrams*.

Feedback Variables

A bit address can be specified as a feedback variable (FV) if necessary for program debugging. Setting an FV facilitates monitoring, but has no direct effect on the execution of the action. There is no reason to set an FV except when it is required.

3-5-1 Action Qualifiers

The action qualifier (AQ) defines how an action is to be executed. One AQ must be set for each action in the action block. When a step becomes active, each action in that step is executed according to its AQ setting.

If an action is defined using a bit address, the bit will turn ON and OFF according to the AQ. (When actions are set using bit addresses, “execute” in the table means that the bit turns ON.) If it is defined using an action number, the action program will be executed or not executed according to the AQ.

The following table describes the functions of nine AQs. In addition, five of these AQs (N, P, L, D, and R) can also take the optional AQ symbol H (for hold) to prevent outputs or timers from being reset after execution of an action has been completed. This makes a total of 14 AQs. The five held AQs are covered in detail later.

AQ	Operation	Held	SV
N	Executes the action while the step is active.	NH	---
P	Executes the action once only when the step becomes active.	PH	---
L	After the step becomes active, L executes the action for the time set with the SV or until the step becomes inactive.	LH	Required
D	After the step becomes active, D delays execution for the time set with the SV and then begins execution of the action. If the step becomes inactive before the time set with the SV has elapsed, the action will not be executed.	DH	Required

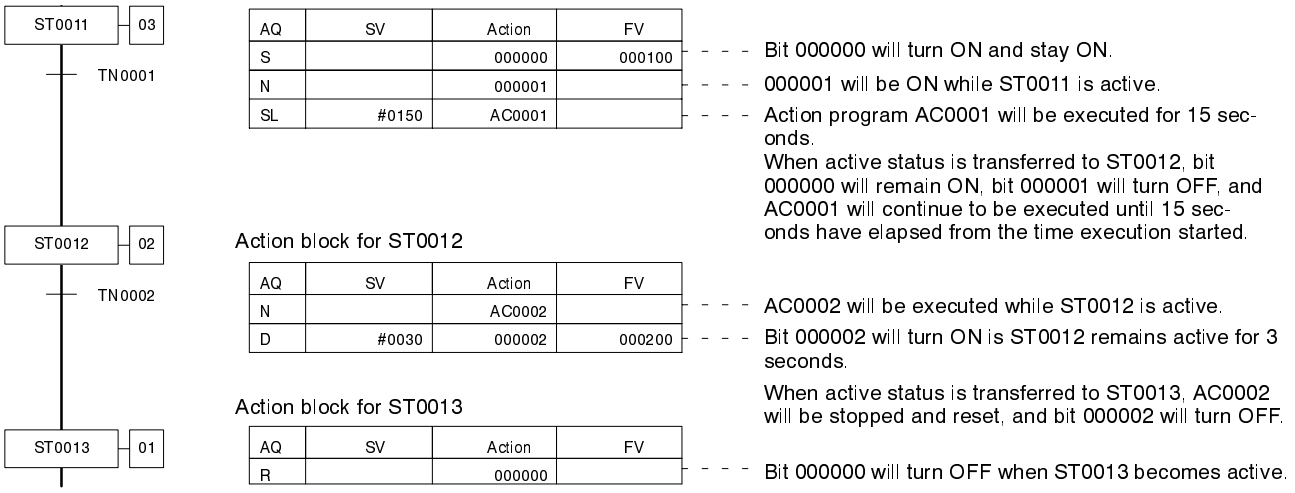
AQ	Operation	Held	SV
R	Stops and resets an action set with S, SL, SD, or DS (see below). If the action is not being executed, then R simply resets it. (Resetting means OUT/OUT NOT are executed with OFF execution conditions, TIM/TIMH are reset, and other timers, counters, and shift registers are held.)	RH	---
S	Begins execution of an action when its step becomes active and continues execution of that action even after the step becomes inactive. Execution is stopped by executing an action for the same bit or action program with an action qualifier of R.	---	---
SL	After the step becomes active, executes the action for the time set with the SV. Unlike L, however, the action will be executed for the specified time even if the step becomes inactive. Execution can be stopped only by executing an action for the same bit or action program with an action qualifier of R.	---	Required
SD	After the step becomes active, SD delays execution for the time set with the SV and then begins execution of the action. Unlike D, however, SD continues execution of that action even after the step becomes inactive. Execution can be stopped by executing an action for the same bit or action program with an action qualifier of R.	---	Required
DS	The function of DS is similar to that of SD. Unlike SD, however, DS will not execute the action if the step becomes inactive before the time set with the SV has elapsed. Once the action is started, however, DS also continues execution of the action even after the step becomes inactive. Execution can be stopped by executing an action for the same bit or action program with an action qualifier of R.	---	Required

Action qualifiers S, SL, SD, and DS are known collectively as S-group AQs. No more than 127 actions with S-group AQs can be executed simultaneously. In addition, the step timer operates during execution of an action with an S-group AQ even after the step itself becomes inactive.

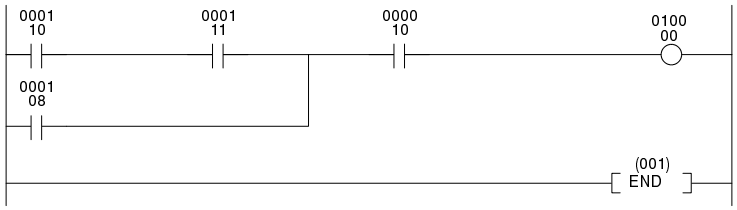
Note An action is normally reset after execution has been completed. If you want an action to be stopped but not reset, then you must add the optional AQ symbol H (for hold) to the AQ for that action. The AQs to which H can be added are N, P, L, D, and R.

Application Example

The following example explains processing for various AQ.



Action Program for AC001



Note All action and transition program must end with the END (001) instruction.

3-5-2 AQ Hold Option

When a step becomes inactive, the actions that are stopped are normally reset. If you wish, however, you can add the optional AQ symbol H (for hold) to the N, P, L, D, and R AQs to prevent outputs or timers from being reset after execution of an action is complete.

Resetting an action means that the action bit is turned OFF (when the action is defined as a bit address) or OUT/OUT NOT instructions are executed with OFF execution conditions (when the action is defined as an action number) and that TIM and TIMH instructions are reset. If the H option is used, the bits in the actions that are normally reset will maintain their status and TIM/TIMH will continue operation.

The five AQs with the optional hold added are NH, PH, LH, DM, and RH. The H option cannot be added to any of the S-group AQs (S, SL, SD, and DS). If you want to hold the status of an S-group action, add the optional H to the R action qualifier that terminates the action.

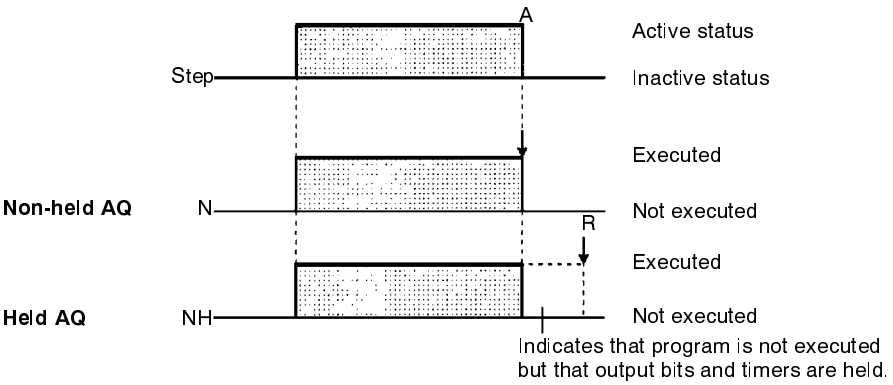
If SOFF(215) is executed, it will be given priority over a held AQ and the action with the held AQ will be reset.

Examples

The following examples demonstrate the difference between AQ with and AQ without the hold option.

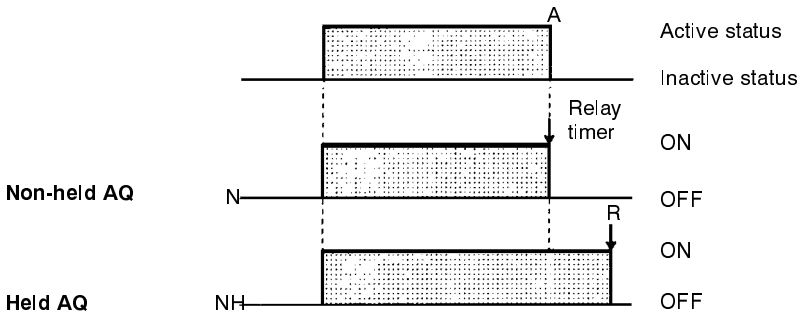
Action Programs

If the AQ is N, then the bits output from the action program will be turned OFF and timers will be reset at point A. If the AQ is NH, then execution of the action program will stop at point A but the bits output from the program will not be turned OFF at that point, and status will be held until an operation is executed to turn them off (e.g., using an action with an R AQ, as shown below). Timers will continue operation from point A and status will be held when time is expired unless another operation is programmed to reset them.



Actions Defined as Bits

If the AQ is N, then the bit will turn OFF at point A. If the AQ is NH, then the bit will not turn OFF at point A and status will be held until they are turned OFF elsewhere in the program (e.g., using an action with an R AQ, as shown below).

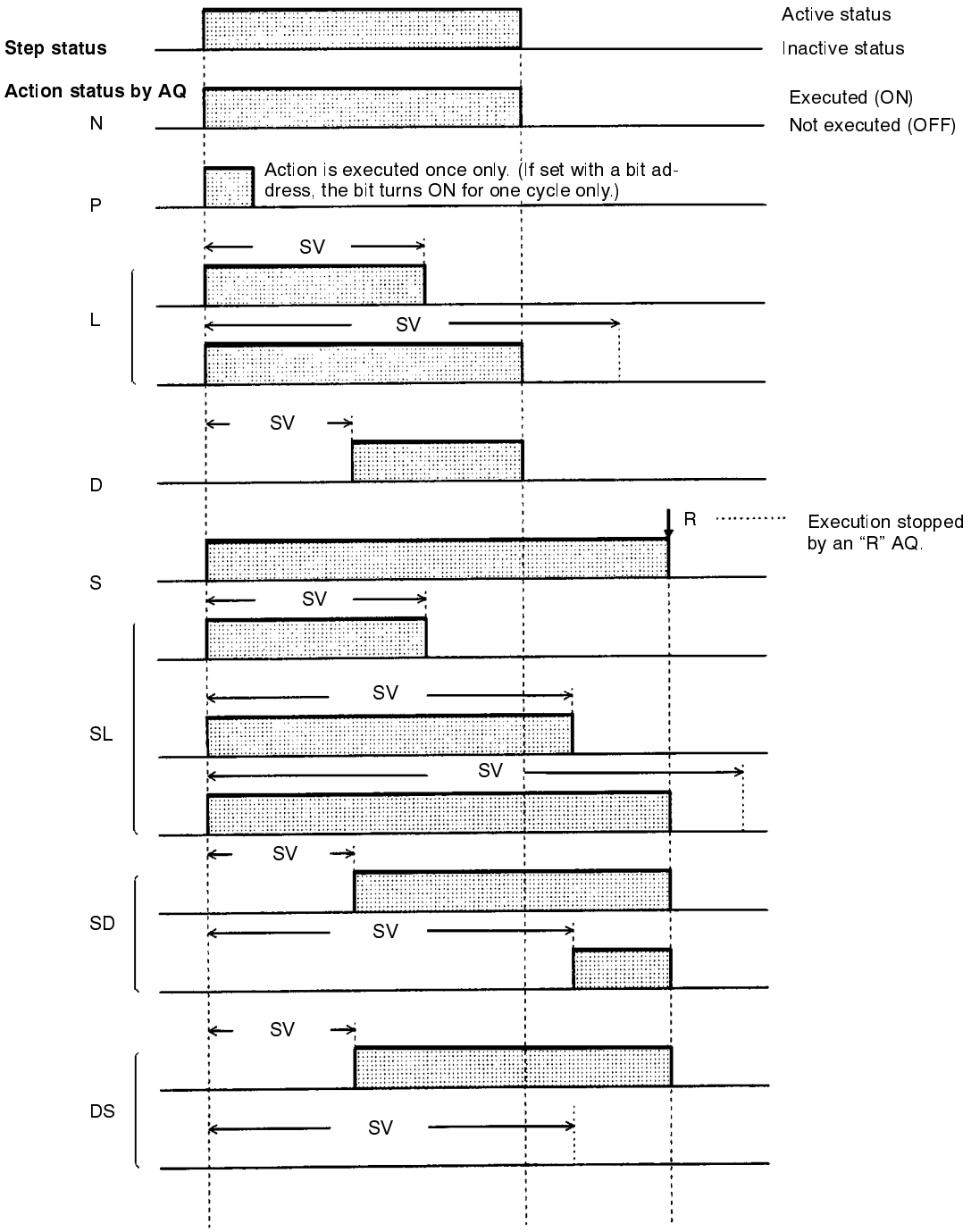


3-5-3 Action Execution Timing

The following diagram shows how the action status changes in relation to the step status according to the various AQ. When the action is defined using a bit address, “execute” and “not execute” in the illustration refer to the ON/OFF status of the bit.

When H is added (NH, PH, LH, DH, and RH), execution of the action is stopped and the output bit status is held after the point in the illustration where “executed” turns to “not executed.”

Use RH when you need to hold the output bit status for S, SL, SD, or DS.

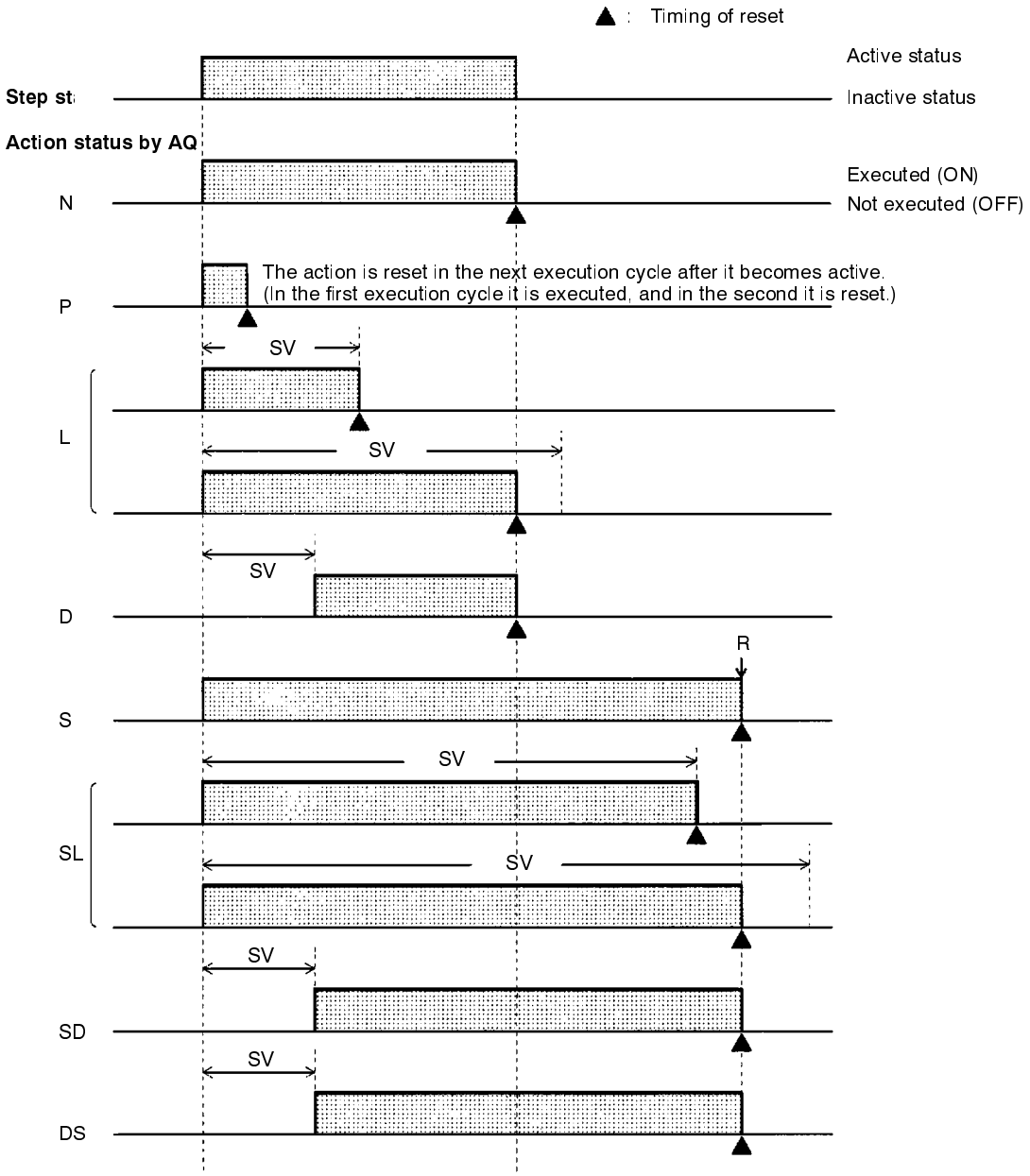


3-5-4 Action Reset Timing

Actions are reset when the step they are in becomes inactive or, for AQ with the hold option, when reset using the R action qualifier. The timing of the reset varies with the AQ which is set for the action. Resetting the action means turning OFF output bits and resetting timers. Operand bits output from the KEEP instruction and counters, however, are not reset.

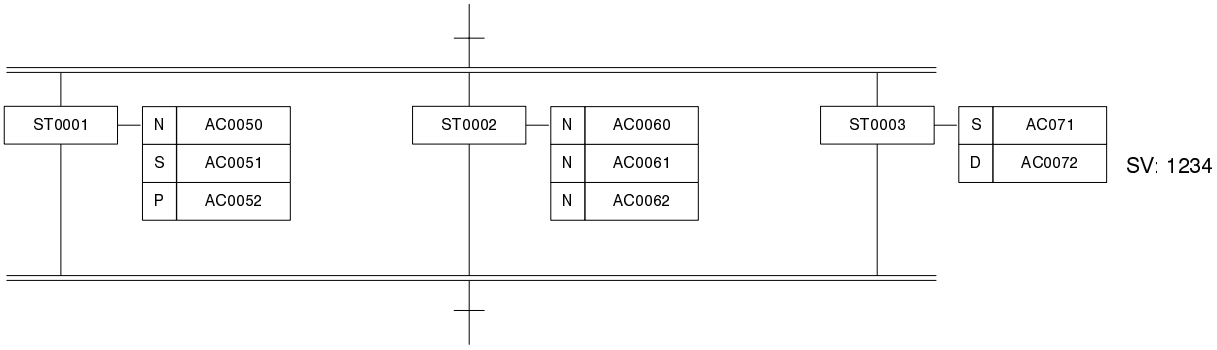
The following timing charts show when actions are reset in relation to the step status for the various AQ.

Reset Timing According to AQ (Action Defined by Program)



3-5-5 AQ and the Execution Cycle

The following SFC program will be used to illustrate the relationship between AQ and the execution cycle.

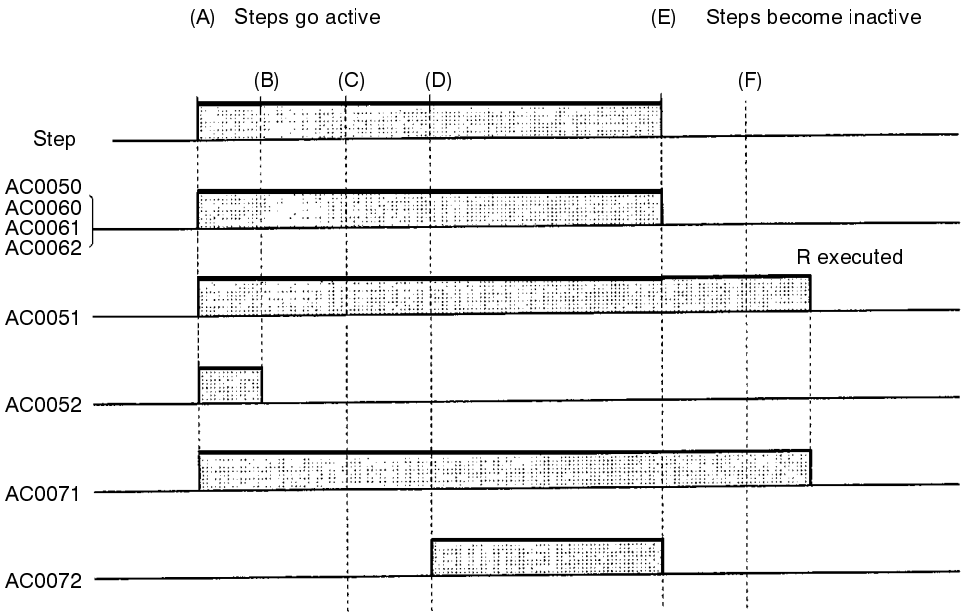


The following table and timing chart shows the operations performed in each execution cycle from the time that steps ST0001, ST0002, and ST0003 becomes active. This chart assumes that cyclic refreshing is used. Operations other than those shown in the illustration have been omitted. The numbers in the table represent the order of execution within each execution cycle. The following items affect execution order.

1, 2, 3...

1. Actions with S-group AQs are executed at the end of the execution cycle.
2. When the AQ is P, the second execution cycle only resets the action.
3. Outputs from a step are reset during the cycle after the step becomes inactive provided neither S-group AQs (S, SL, SD, and DS) nor hold-option AQs (NH, PH, DH, and LH) will be reset.
4. The conditions for transfer of active status are examined upon completion of execution of each step. When the conditions are met for active status to be transferred, then the currently active step(s) become inactive and are reset. In a case of parallel joining such as in this example, execution may be stopped between simultaneous active steps depending on when the conditions for transfer are met. For example, if execute status is transferred when execution of ST0001 is completed, only ST0001 (and not ST0002 and ST0003) will be executed in that execution cycle.

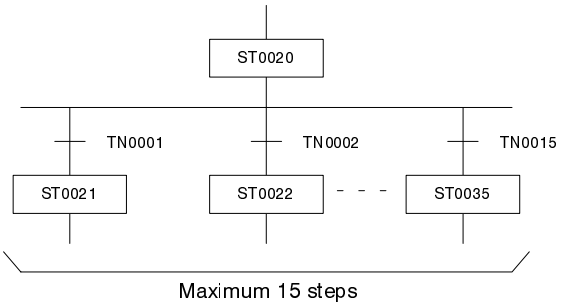
	Action number	AC0050	AC0051	AC0052	AC0060	AC0061	AC0062	AC0071	AC0072	I/O
	AQ	N	S (Note 1)	P	N	N	N	S (Note 1)	D	refresh
A	First cycle	1	6	2	3	4	5	7	---	8
B	Second cycle	1	6	2 (Note 2)	3	4	5	7	---	8
C	Third cycle on	1	5	---	2	3	4	6	---	7
D	Cycle after lapse of AC0072 delay time	1	6	---	2	3	4	7	5	8
E	Cycle after steps become inactive (Note 3)	1	6	---	2 (Note 4)	3 (Note 4)	4 (Note 4)	7	5 (Note 4)	8
F	Further cycles	---	1	---	---	---	---	2	---	3



3-6 Conditional Branching and Joining

Conditional Branching

With conditional branching, one step is branched to two or more steps and active status is transferred to the first step after the transition for which the condition is met. Conditional branching is shown below. A single step can be conditionally branched into a maximum of 15 steps.



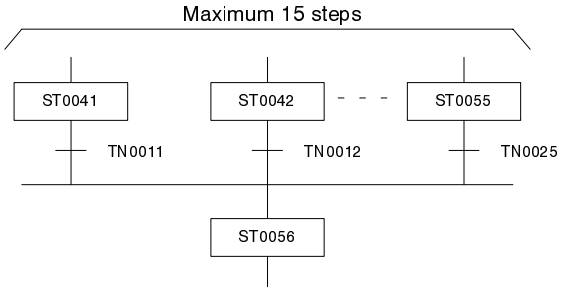
If the step before the transition is active and at least one step after the transition is inactive, active status is transferred from the step before the branch to the first inactive step after that branch whose transition condition is met. If the transition conditions for two or more inactive steps are met simultaneously, active status will move to the leftmost step.

Active status cannot be transferred to more than one step after the branch. It is conceivable, however, that two steps could be active at the same time if one of the steps is active from a previous cycle and active status is then transferred from the step before the branch to another step after the branch.

Conditional Joining

With conditional joining, two or more branched steps are joined into one step, and active status is passed from a step just before the transition for which the

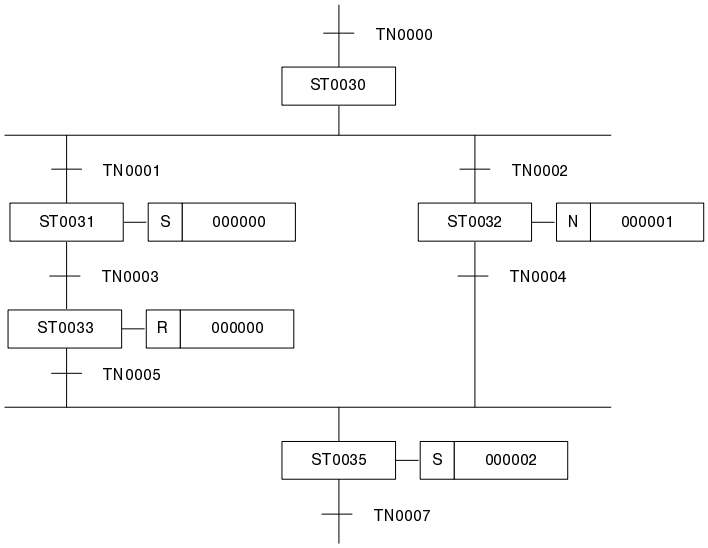
transition condition is met. Up to 15 steps can be conditionally joined to form a single step. Conditional joining is shown in the following example.



Active status will move from any active step before the branch to the step after the branch provided the transition condition is met and the step after the branch is inactive.

Application Example

The following example illustrates conditional branching and joining.



If ST0030 is active, ST0031 and ST0032 are inactive, and the condition for TN0001 is met, then active status will be transferred from ST0030 to ST0031 and bit 000000 will turn ON. If the condition for TN0002 is met before the condition for TN0001, then active status will be transferred to ST0032 and bit 000001 will turn ON. If the conditions are met for TN0001 and TN0002 simultaneously, then active status will be transferred to ST0031.

If ST0031 is active, ST0033 is inactive, and the condition for TN0003 is met, then active status will be transferred from ST0031 to ST0033 and bit 000000 will be reset.

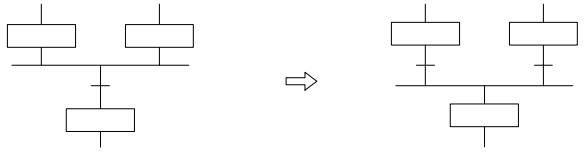
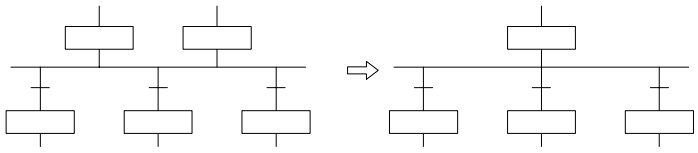
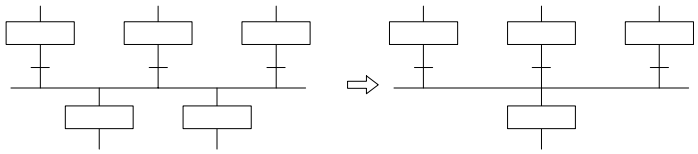
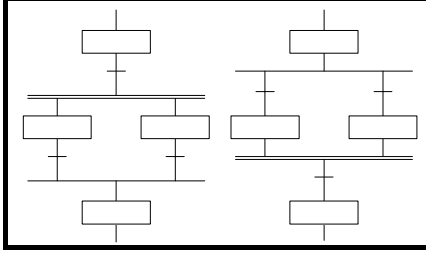
If ST0032 is active, ST0035 is inactive, and the condition for TN0004 is met, then active status will be transferred from ST0032 to ST0035. As this happens, bit 000001 will turn OFF and bit 000002 will turn ON.

If ST0033 is active, ST0035 is inactive, and the condition for TN0005 is met, then active status will be transferred from ST0033 to ST0035.

Note If ST0030 is active, ST0031 and ST0032 are inactive, and the conditions are never met for either TN0001 or TN002, then ST030 will remain active indefinitely and program execution will not advance.

Programming Errors

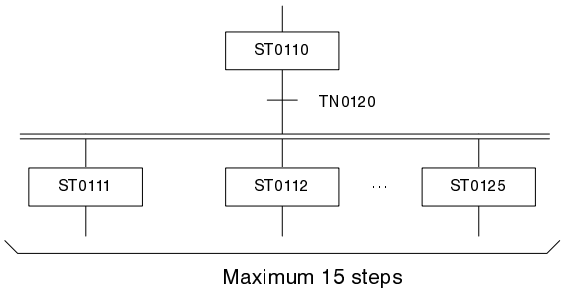
The most common types of conditional branching/joining programming errors are shown below along with corrections where possible.

Incorrect program	Correct program	Reason
		For conditional branching, the transitions must be placed after the branching line.
		For conditional joining, the transitions must be placed before the joining line.
		You can only branch from one step at a time.
		The steps being joined can be joined only into a single step.
	Do not combine conditional and parallel branching/joining.	Parallel branching and conditional joining cannot be combined. Likewise, conditional branching and parallel joining cannot be combined.

3-7 Parallel Branching and Joining

Parallel Branching

With parallel branching, one step is branched into two or more steps and active status is transferred to all of the branched steps simultaneously when the transition condition is met. A maximum of 15 steps can be branched from a single step. An example of parallel branching is shown below.

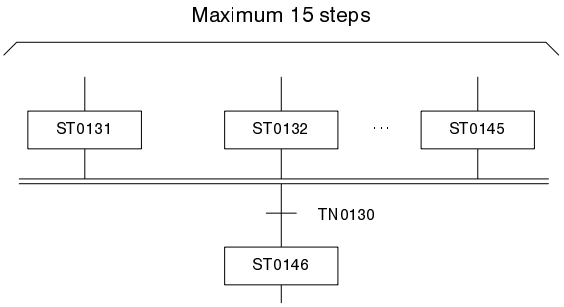


Active status is transferred when the step before the double line (the parallel branching line) is active, all of the steps after the double line are inactive, and the transition condition is met. When the active status is transferred, all of the steps just after the double line become active simultaneously and are executed in order from left to right.

If the maximum 15 actions is programmed for each of 15 parallel steps, you can have as many as 225 actions executed simultaneously.

Parallel Joining

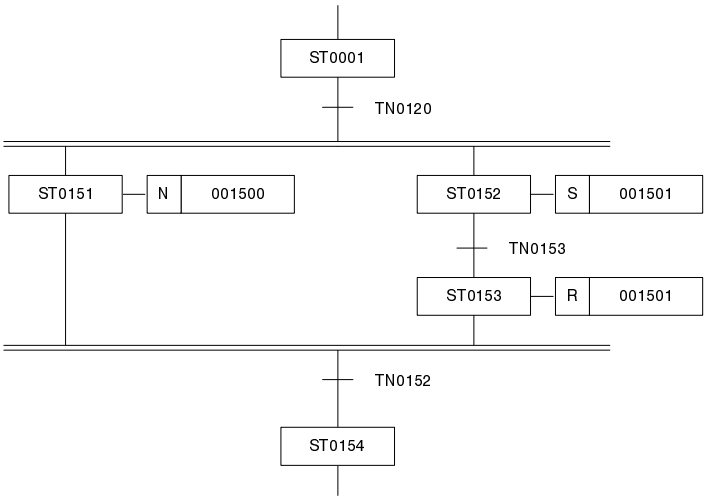
With parallel joining, two or more branched steps are joined into one step and active status is transferred from all of the steps simultaneously when the transition condition is met. A maximum of 15 steps can be joined into a single step. An example of parallel joining is shown below.



Active status is transferred when all of the steps above the double line (the parallel joining line) are active, the step below the line is inactive, and the transition condition is met.

Application Example

The following example illustrates parallel branching and joining.



If ST0001 is active, ST0151 and ST0152 are inactive, and the condition for TN0120 is met, then ST0151 and ST0152 will become active simultaneously and bits 001500 and 001501 will turn ON.

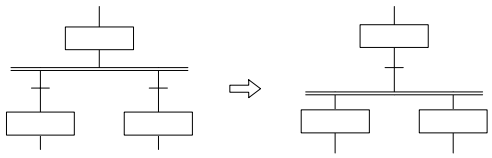
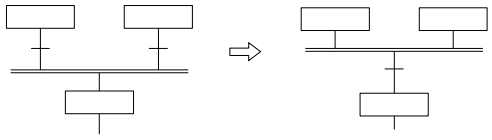
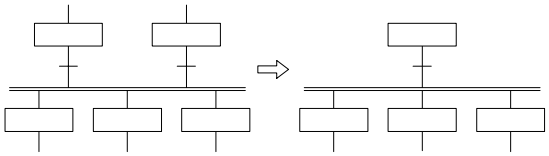
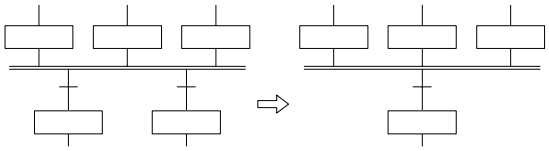
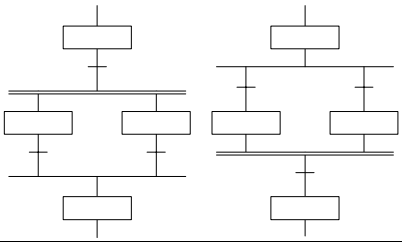
If the condition for TN0153 is met and ST0153 is inactive, then active status will be transferred to ST0153 and bit 001501 will be reset.

When ST0151 and ST0153 are both active, ST0154 is inactive, and the condition for TN0152 is met, then active status will be transferred to ST0154.

Note Even if ST0151 becomes active and TN0152 turns ON (i.e., the condition is met), active status will not be transferred to ST0154 unless ST0153 is also active and ST0154 is inactive.

Programming Errors

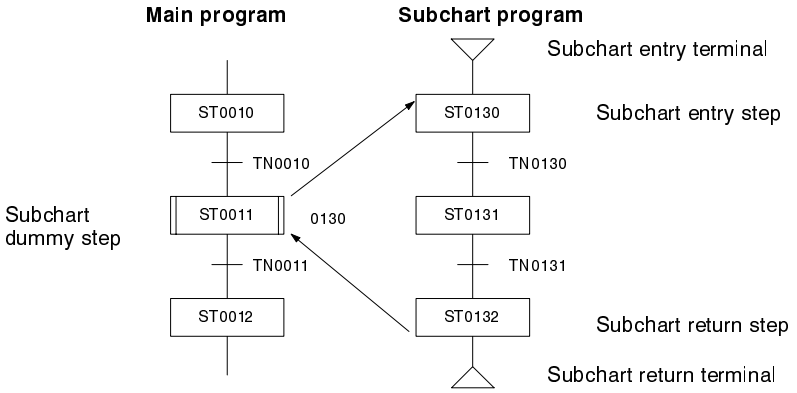
The most common types of parallel branching/joining programming errors are shown below along with corrections where possible.

Incorrect program	Correct program	Reason
		For parallel branching, the transition must be placed before the branching line.
		For parallel joining, the transition must be placed after the joining line.
		You can only branch from one step at a time.
		The steps being joined can be joined into only a single step.
	Do not combine conditional and parallel branching/joining.	Parallel branching and conditional joining cannot be combined. Likewise, conditional branching and parallel joining cannot be combined.

3-8 Subcharts

The main program can call a subchart by using a subchart dummy step. Subcharts are essentially the same as subroutines. The subchart starts with an entry terminal and entry step and ends with a one or more return steps and return terminals. Multiple return steps/terminals could be used to return from branches within the subchart.

You can call other subcharts from within one subchart, i.e., subcharts can be nested. There are no limits to the number of nesting levels. You can write up to 255 subcharts with the CV500 and up to 511 with the CV1000 or CV2000. An example of a subchart program and the step calling it is shown below.



In the example, ST0011 is the subchart dummy step and ST0130 is the subchart entry step. The number written inside of the subchart dummy step is the dummy step number, and the number written outside of that step is the number of the subchart that is being called (i.e., the number of the subchart entry step).

The subchart dummy step cannot have any actions of its own.

The same subchart can be called from multiple places within a program. If its execution has not been completed, however, when it is called again, the later step will wait until the execution is complete and the call will be made when the first step calling the subchart has finished.

The active/inactive status of steps within a subchart depends on the status of the subchart dummy step that calls that subchart. (For details, refer to the *CV-series PC Operation Manual: Ladder Diagrams*.) Furthermore, active status cannot pass from the subchart dummy step to the next step until the return step in that subchart becomes active or until all of the steps in that subchart become inactive (i.e., until subchart operations are complete). In addition, the active status will not be passed from the subchart dummy step unless the transition condition is met and the step after the subchart dummy step is inactive.

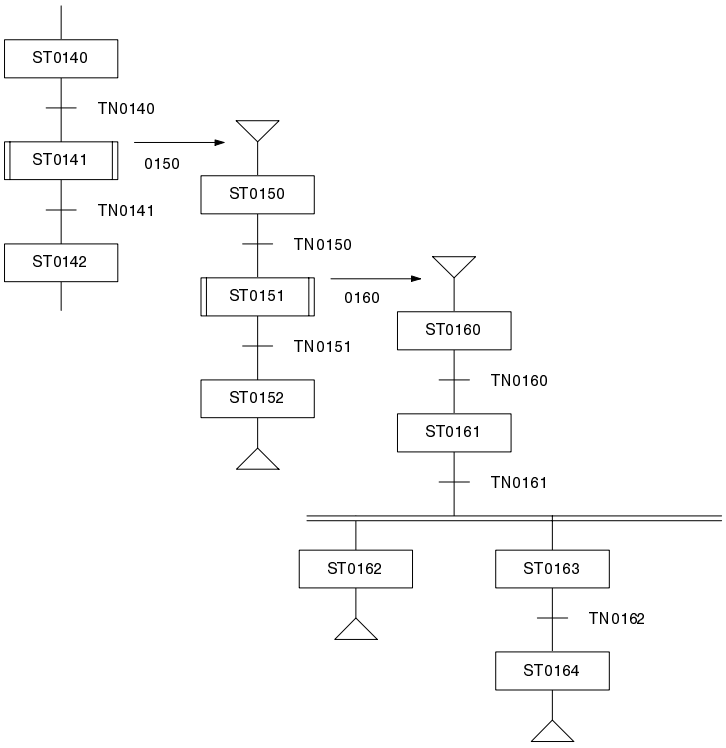
Note When a subchart return step becomes active and the subchart is finished, any active steps remaining within the subchart will also be terminated. That is to say, a subchart can have more than one return step, and execution of the subchart will be completed when any one of them becomes active.

In the example, status transfer will take place as shown in the table below.

Condition	ST0010	ST0011	ST0012	ST0130	ST0131	ST0132
---	**ACTIVE**	Inactive	Inactive	Inactive	Inactive	Inactive
TN0010 ON	Inactive	**ACTIVE**	Inactive	**ACTIVE**	Inactive	Inactive
TN0130 ON	Inactive	**ACTIVE**	Inactive	Inactive	**ACTIVE**	Inactive
TN0131 ON	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	**ACTIVE**
TN0011 ON	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	Inactive

Note The operations shown in the table will be executed over the course of several execution cycles.

Application Example



In this example, status transfer will take place as shown in the table below.

Condition	ST0140	ST0141	ST0142	ST0150	ST0151	ST0152	ST0160	ST0161	ST0162	ST0163	ST0164
---	**ACTIVE**	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive
TN0140 ON	Inactive	**ACTIVE**	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive
TN0150 ON	Inactive	**ACTIVE**	Inactive	Inactive	**ACTIVE**	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	Inactive
TN0160 ON	Inactive	**ACTIVE**	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	Inactive
TN0161 ON	Inactive	**ACTIVE**	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	**ACTIVE**	**ACTIVE**	Inactive
TN0151 ON	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	Inactive	Inactive
TN0141 ON	Inactive	Inactive	**ACTIVE**	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive

Precautions

Observe the following precautions when programming subcharts.

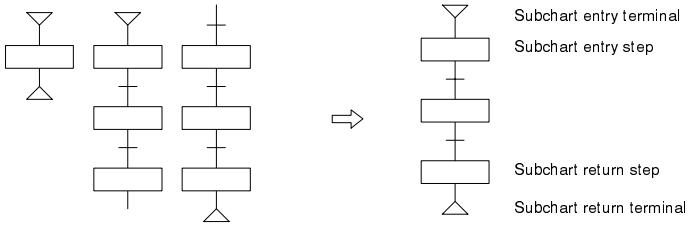
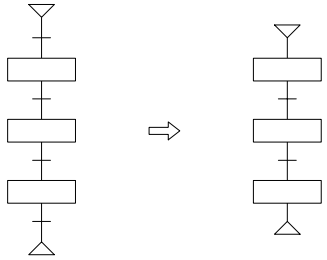
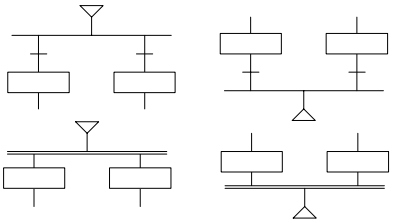
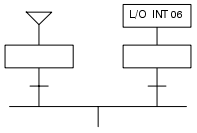
1, 2, 3...

1. The following steps cannot be used as or in place of subchart dummy steps.
   - Initial steps
   - Subchart entry steps
   - Subchart return steps
   - Interrupt entry terminals
   - Interrupt return terminals
2. A subchart cannot be called simultaneously from multiple subchart dummy steps. If a subchart is called while it is already being executed, the second call will wait until execution of the subchart has been completed.
3. If the program calls a subchart that does not exist, an fatal SFC error will be generated and program execution will stop.
4. When a subchart is called by a dummy step within another subchart, execution of the nested subchart will stop when the dummy step that called it becomes inactive. That is to say, any active steps remaining in the nested subchart will not be executed after a return step within that subchart becomes active, and active status is returned to the dummy step.

5. A subchart can only be called by a dummy step. When you want to use a SFC control instruction to activate a subchart, activate the dummy step that calls that subchart.
6. Initial steps cannot be placed inside of a subchart.

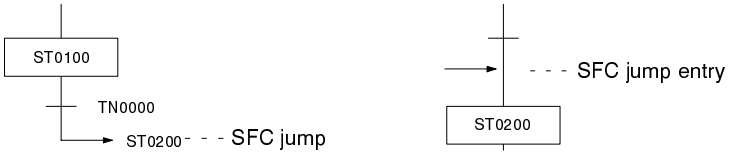
Programming Errors

The most common types of subchart programming errors are shown below along with corrections where possible.

Incorrect program	Correct program	Reason
		A subchart entry terminal and a subchart entry step must be combined at the beginning of a subchart. Likewise, a subchart return terminal and a subchart return step must be combined at the end of a subchart.
		A transition may not be placed between a subchart entry terminal and a subchart entry step or between a subchart return step and a subchart return terminal.
	Do not branch from or join before terminals.	Branching may not be programmed immediately after a subchart entry terminal, and joining may not be programmed immediately before a subchart return terminal.
	Do not join subcharts and interrupt programs.	A subchart and an interrupt program cannot be joined. In addition, two or more subcharts cannot be connected together and a subchart cannot be connected to the main program.

3-9 SFC Jumps

The SFC jump is used to move execution of an SFC program to another step at a different location in the program. A jump can be made only from just after a transition to just in front of a step. The jump destination is specified with an SFC jump entry. An example of an SFC jump is shown below.

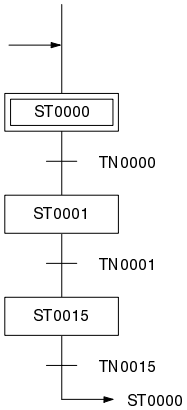


In the illustration, the program jumps from step ST0100 to step ST0200. The jump source (just after the transition on the left) and the jump destination (just after the step on the right) are indicated by arrows.

The jump destination step becomes active when the jump is made, and then becomes inactive when the condition is met for transition immediately following it and active status is passed to the next step.

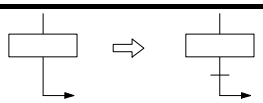
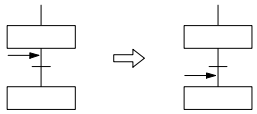
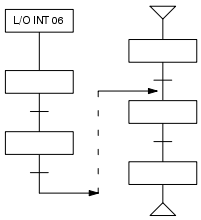
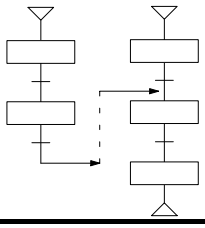
A jump cannot be made off of the sheet when programming on the CVSS. For details on SFC sheets, refer to the CVSS operation manuals.

You can create a closed loop with a jump, as in this example.



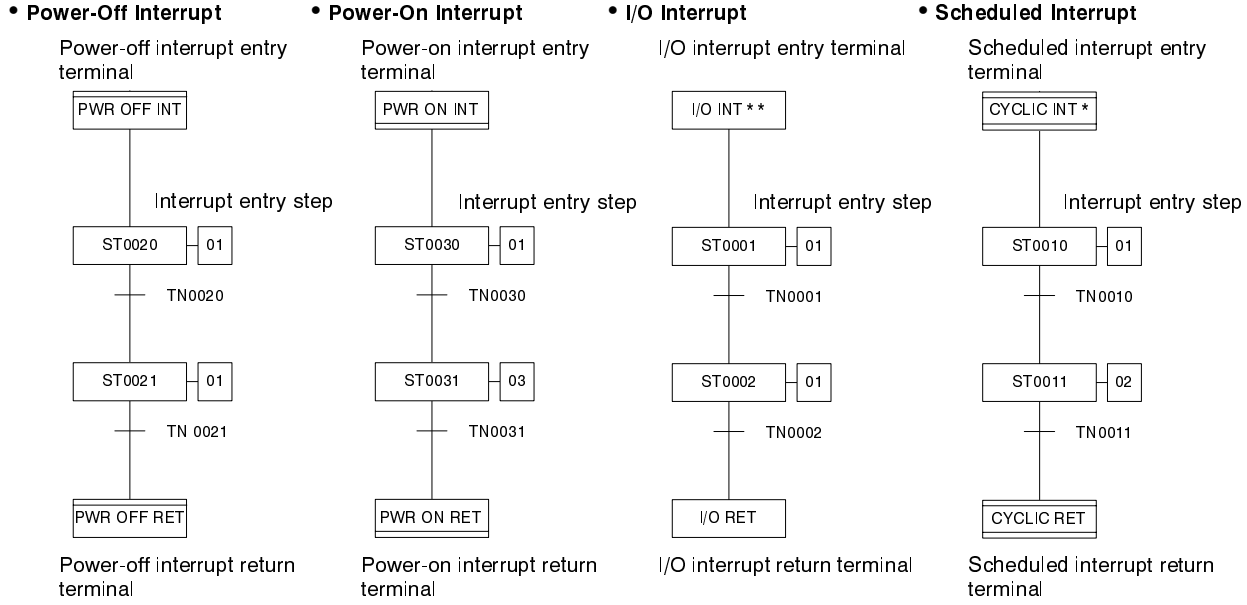
Programming Errors

The most common types of jump programming errors are shown below along with corrections where possible.

Incorrect program	Correct program	Reason
		The jump must be programmed immediately after a transition.
		The jump destination must be programmed immediately before a step.
	Do not jump into or out of interrupt programs.	A jump cannot be made from or to an interrupt program from any other program (the main program, a subchart program, or another interrupt program). Jumps can be made within a single interrupt program.
	Do not jump into or out of subcharts.	A jump cannot be made from or to a subchart program from any other program (the main program, another subchart program, or an interrupt program). Jumps can be made within a single subchart program.

3-10 Interrupt Programs

Interrupt programs can be written separately from the main program to provide processing for special purposes. The various interrupt programs, shown below, allow special processing for power interruptions (both before and after the interruptions), for I/O interrupt signals, and for scheduled processing (processing repeated at a specific interval). Each interrupt program contains one entry terminal and entry step plus one or more return terminals. Multiple return terminals may be used if there is branching within the interrupt program.



Interrupt entry terminals and return terminals are not steps and cannot contain actions.

When an interrupt program is executed, the main program execution is discontinued until the interrupt program execution is completed.

The following table shows the number of each type of interrupt program that can be written.

Type of Interrupt		Qty.	Interrupt no.
I/O		32	00 to 31
Scheduled	CV500	1	0
	CV1000/CV2000	2	0, 1
Power-off		1	---
Power-on		1	---

Interrupt masks, clearance of interrupt causes, and settings of scheduled interrupt spaces are all executed with interrupt control instructions. For details on execution and control of interrupt programs, refer to the *CV-series PC Operation Manual: Ladder Diagrams.*

Precautions

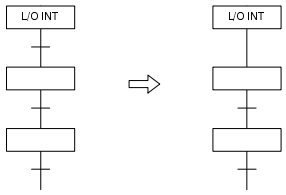
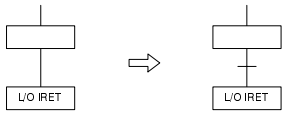
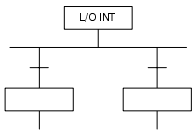
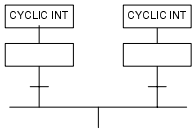
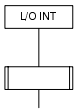
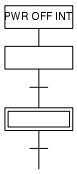
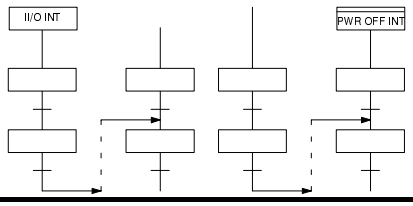
Observe the following precautions when programming interrupts.

1, 2, 3...

1. No transition can be inserted between the entry terminal and the entry step.
2. The interrupt entry step cannot be used as a subchart dummy step.
3. If interrupt program execution time is too long, there may be a execution cycle overrun error.

Programming Errors

The most common types of interrupt programming errors are shown below along with corrections where possible.

Incorrect program	Correct program	Reason
		A transition cannot be placed directly after an entry terminal.
		A transition must be placed directly before a return terminal.
	Include a step within the interrupt program.	You cannot write an interrupt program without any steps.
	Do not branch from an entry terminal.	An interrupt program cannot be branched directly after the entry terminal.
	Do not joint interrupt programs.	You cannot join two or more interrupt programs. Furthermore, you cannot join the main program and an interrupt program, or a subchart and an interrupt program.
	Do not use an entry step to call a subchart.	An interrupt entry step cannot be used as a subchart dummy step.
	Do not place an initial step in an interrupt program.	An initial step cannot be placed inside of an interrupt program.
	Do not jump out of an interrupt program.	A jump cannot be made from or to an interrupt program from another type of program. Jumps can be made within an interrupt program.

SECTION 4
Controlling Step Status

This section provides a review of step status and then explains how to use the ladder-diagram instructions called SFC control instructions to artificially control step status and thus control the execution flow of the SFC program. This type of control can be used in normal SFC program flow or to allow for special processing such as that for emergency situations. The other ladder diagram instructions related to SFC programs are also described. Examples using SFC control instructions are provided in *Section 6 SFC Application Examples*.

4–1 Step Status 48
4–1–1 Controlling Steps 48
4–1–2 SFC Control Instructions and Status Changes 50
4–1–3 Subcharts 51
4–1–4 SFC Control Instructions and Interrupt Programs 52
4–1–5 Precautions 52
4–2 SFC Control Instructions 54
4–2–1 Introduction 54
4–2–2 ACTIVATE STEP - SA(210) 55
4–2–3 PAUSE STEP - SP(211) 56
4–2–4 RESTART STEP - SR(212) 57
4–2–5 END STEP - SF(213) 58
4–2–6 DEACTIVATE STEP - SE(214) 59
4–2–7 RESET STEP - SOFF(215) 60
4–2–8 TRANSITION OUTPUT - TOUT(202) 61
4–2–9 TRANSITION COUNTER - TCNT(123) 62
4–2–10 READ STEP TIMER -TSR(124) 63
4–2–11 WRITE STEP TIMER - TSW(125) 64
4–3 Program Example 65

4–1 Step Status

Step status can be broadly divided into active and inactive. Active status is further divided into execute, pause, and halt status. Step status is controlled either by the normal flow and conditions of the SFC program, as described in the previous section, or by using the SFC control instructions, which are explained later in this section.

A step does not normally go to pause or halt status. In order to place a step in pause or halt status, you must use an SFC control instruction.

Caution SFC control instructions forcibly change the status of steps. When using these instructions, be sure to carefully consider their effects on program execution and on other steps in the program.

Execute Status

In execute status, actions are either being executed or waiting to be executed (e.g., due to a set value for the AQ). When a step goes into active status, it normally goes into execute status first. When the status of a step changes from inactive state to execute (active) state, the step time will be reset and started.

Pause Status

Actions in steps that have been paused are temporarily not executed. A step in pause status is active, but actions are not executed and active status is not transferred. Actions with S-group action qualifiers (S, SL, SD, and DS), however, continue to be executed even while the step is paused.

The status of a step is held when it goes to pause status, and execution starts from the beginning of the step when pause status is cleared. The step timer and TIM and TIMH instructions in paused actions continue timing even while paused.

Halt Status

Actions in steps that have been halted are no longer executed even though the step is still active. Actions with S-group action qualifiers (S, SL, SD, and DS), however, continue to be executed even while the step is halted. Active status is transferred from a step in halt status when the transition condition is met.

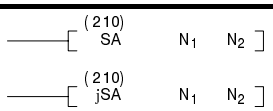
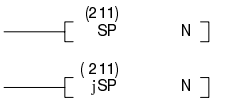
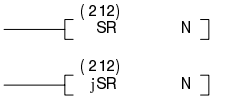
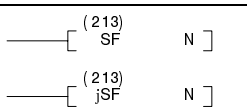
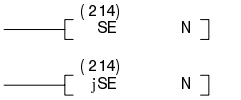
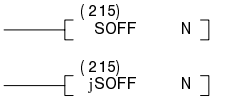
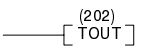
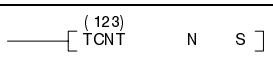
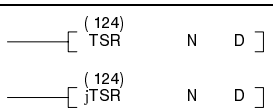
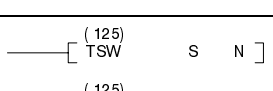
The step timer and TIM and TIMH instructions in halted actions continue timing even while halted.

Inactive Status

When a step becomes inactive, only actions with S-group action qualifiers (S, SL, SD, and DS) continue to be executed. While S-group actions are being executed, the step timer and TIM and TIMH instructions in the S-group actions continue timing.

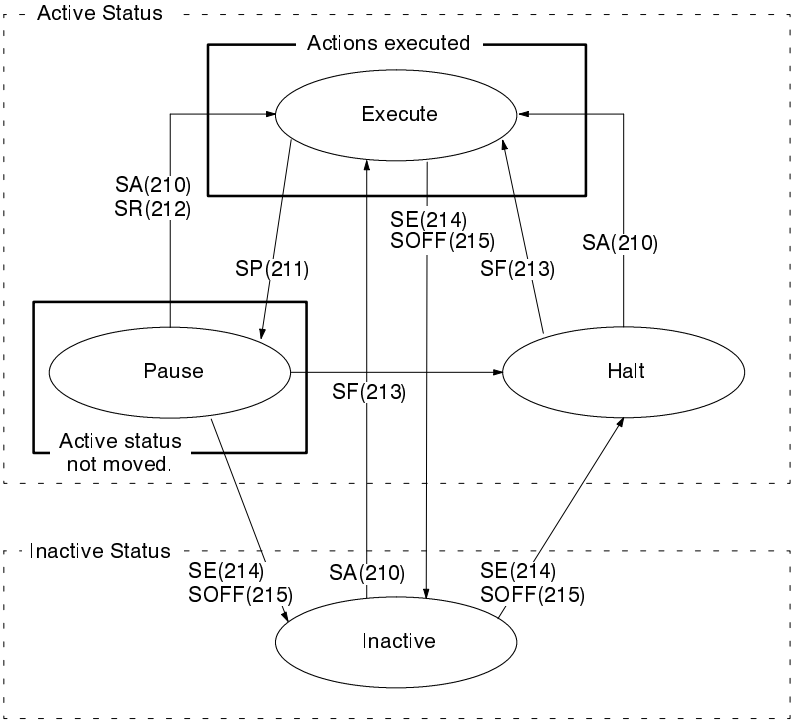
4–1–1 Controlling Steps

SFC control instructions are a group of instructions that can be used in action programs to change the step status (and thus indirectly subchart status) in an SFC program. The SFC control instructions are outlined in the following table. Details are provided later in section 4.

Mnemonic (function code) and name	Symbol(s)	Operation	Operand data areas	Page
SA(210) ACTIVATE STEP		Activates a step or subchart to start the execution of actions. N_1: Step number N_2: Subchart number	Step number: N_1 CV500 0000 to 0511 CV1000/CV2000 0000 to 1023 Subchart number: N_2 CV500 0000 to 0511 CV1000/CV2000 0000 to 1023	55
SP(211) PAUSE STEP		Changes the status of a step or subchart from execute to pause. N: Step number	Step number: N_1 CV500 0000 to 0511 CV1000/CV2000 0000 to 1023	56
SR(212) RESTART STEP		Changes the status of a step or subchart from pause to execute. N: Step number		
SF(213) END STEP		Changes the status of a step or subchart from execute or pause to halt. N: Step number		58
SE(214) DEACTIVATE STEP		Changes a step or subchart from active (execute, pause, or halt) to inactive status. N: Step number		59
SOFF(215) RESET STEP		Resets a step or subchart (regardless of its status) to inactive status. Also resets actions with or without hold-option AQs. N: Step number	Step number: N CV500 0000 to 0511 CV1000/CV2000 0000 to 1023	59
TOUT(202) TRANSITION OUTPUT		Outputs the result of a transition program to the Transition Flag.	---	61
TCNT(123) TRANSITION COUNTER		Computes the number of times that a transition program is executed, and turns ON the Transition Flag when the preset count is reached. N: Counter number S: Number of times executed	Counter number CV500 0000 to 0511 CV1000/CV2000 0000 to 1023	62
TSR(124) READ STEP TIMER		Reads the present value of the step timer and writes to a designated word. N: Step number D: Destination for PV	Step number: N CV500 0000 to 0511 CV1000/CV2000 0000 to 1023	63
TSW(125) WRITE STEP TIMER		Changes the present value of the step timer from a designated word. S: New PV N: Step number		64

4–1–2 SFC Control Instructions and Status Changes

The SFC control instructions that directly affect step status are shown in the following illustration. The arrows indicate the status changes that are possible using these instructions.

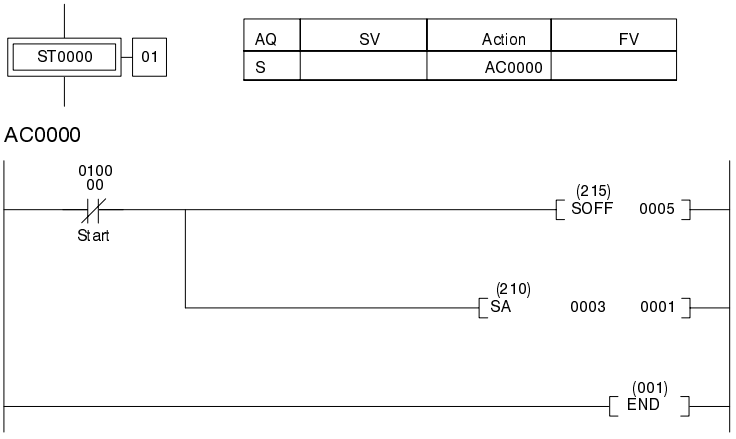


Note

1. When you want to activate a subchart with SA(210), execute SA(210) on the subchart dummy step that calls the subchart. If subcharts are nested, SA(210) cannot be executed unless all of the subchart dummy steps in the levels above the target subchart are in execute status.
2. When most SFC control instructions are executed on a subchart dummy step, they act on all of the *active* steps in the subchart. SOFF(215), however, acts on *all* of the steps in the subchart, not only on those steps that are active.

Application Example

In the following example, bit 010000, which starts operation, is always checked (AQ = S) in AC0000 of initial step ST0000. If the bit turns OFF, ST0005 is made inactive, all of its actions are reset, and ST0003 of subchart ST0001 is changed to execute status. (In order for ST0003 to go to execute status, however, subchart 0001 must already be in execute status.)



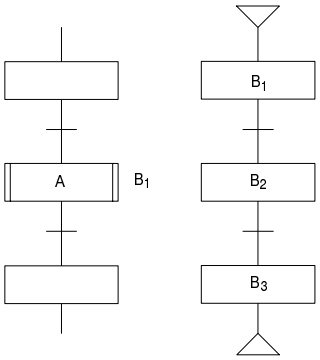
4–1–3 Subcharts

Step Status

Steps within a subchart take their status from the status of the dummy step that calls the subchart. The following table shows the status that are possible for subchart steps depending on the status of the dummy step.

Dummy step status	Statuses possible for subchart steps			
	Execute	Pause	Halt	Inactive
Execute	Yes	Yes	Yes	Yes
Pause	No	Yes	No	Yes
Halt	No	No	Yes	Yes
Inactive	No	No	No	Yes

For example, if the subchart dummy step A is halted, steps B_1, B_2, and B_3 of the subchart called from A can be in either halt or inactive status only.



SFC Control Instructions

The status of steps in a subchart can be changed by executing SFC control instructions using the dummy step that calls the subchart as the operand. If all of the steps in a subchart are made inactive by SE(214) or SOFF(215), then the execution of that subchart will be completed and active status will pass from the subchart dummy step when the transition condition is met. The following table shows the instructions that can be executed for each possible subchart dummy step status.

Dummy step status	SA	SP	SR	SF	SE	SOFF
Execute	Yes	Yes	Yes	Yes	Yes	Yes
Pause	No	No	No	No	Yes	Yes
Halt	No	No	No	No	Yes	Yes
Inactive	No	No	No	No	No	Yes*

* Reset only

For example, if the subchart dummy step is halted, steps B_1, B_2, and B_3 of the example subchart above can be in either halt or inactive status, and therefore the only instructions that can be executed are SE(214) and SOFF(215).

Caution Do not create programs like the ones described below, as they cannot be run.

- A program that designates a step that is not included in the subchart designated by SA(210).
- A program where a subchart designated by an instruction is undefined.
- A program where a subchart designated by SA(210) does not exist.
- A program where an instruction does not match the status of a step (e.g., where SP(211) is used on an inactive step).
- A program where a step designated by SA(210) is in an interrupt program.

4–1–4 SFC Control Instructions and Interrupt Programs

The following table shows which SFC control instructions can be used from within interrupt programs. No SFC control instructions can be used to control steps within interrupt programs.

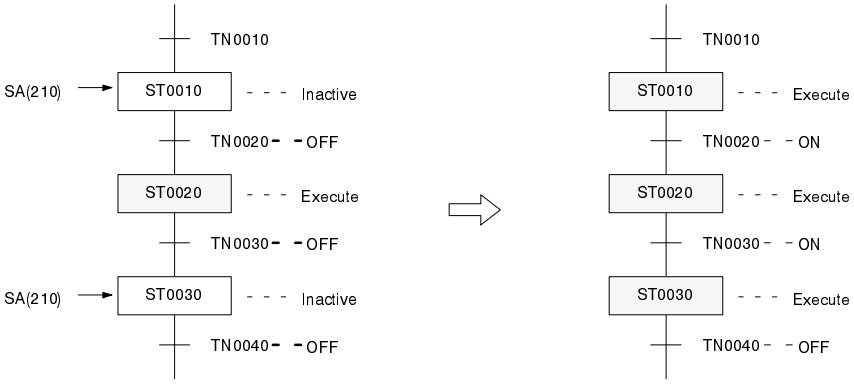
Instruction	Execution from interrupt program	Execution toward interrupt program
SA	Not possible.	Not possible.
SP	Possible.	
SR		
SF		
SE	Possible only from power-on	
SOFF	interrupt program.	

4–1–5 Precautions

The following examples demonstrate some problems that can occur when using SFC control instructions to artificially alter step status.

Consecutive Steps

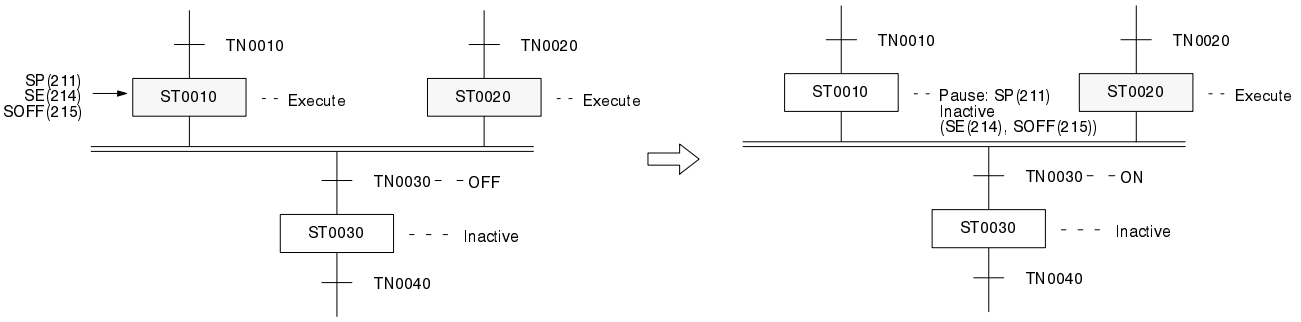
Be cautious of making consecutive steps active simultaneously. When ST0020 in the example below is in execute status and ST0010 and ST0030 are inactive, consider the restrictions in movement of active status if you change ST0010 and ST0030 to execute status with SA(210).



Even if the condition for TN0020 is met (i.e., TN0020 turns ON), step status cannot be transferred until ST0020 becomes inactive. Similarly, even if the condition for TN0030 is met, step status cannot be transferred until ST0030 becomes inactive. Active status can be moved by making ST0020 or ST0030 inactive by means of SE(214) or SOFF(215), or by waiting for TN0040 to turn ON and for active status to be transferred to the step after ST0030.

Parallel Joining: Example 1

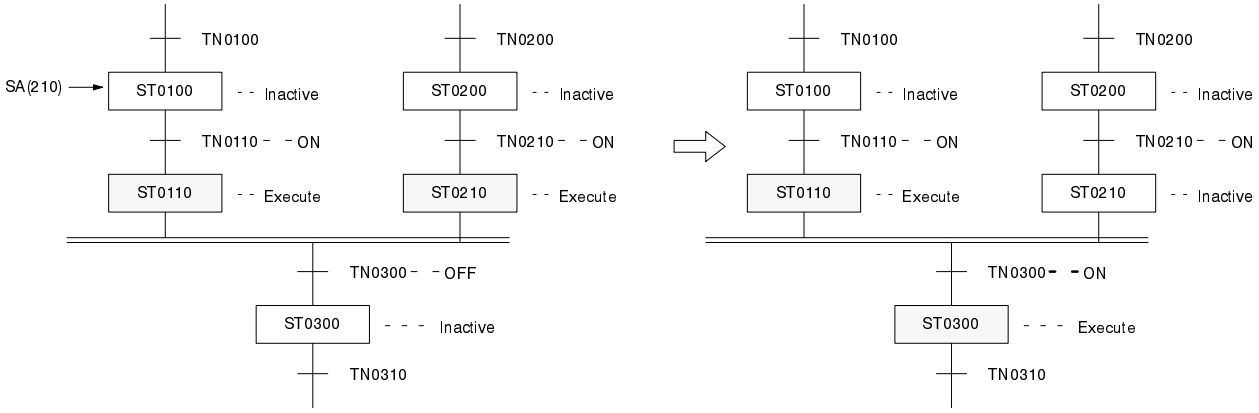
When the steps just before a parallel join (ST0010 and ST0020 in this example) are in execute status, consider the restrictions in movement of active status if you change ST0010 to either pause or inactive status with SP(211), SE(214), or SOFF(215).



If ST0010 is paused or inactive, active status cannot be transferred to ST0030 even if TN0030 turns ON. To enable active status to be transferred, change ST0010 to execute status with SR(212) or SF(213) if ST0010 is paused, or with SA(210) if ST0010 is paused or inactive.

Parallel Joining: Example 2

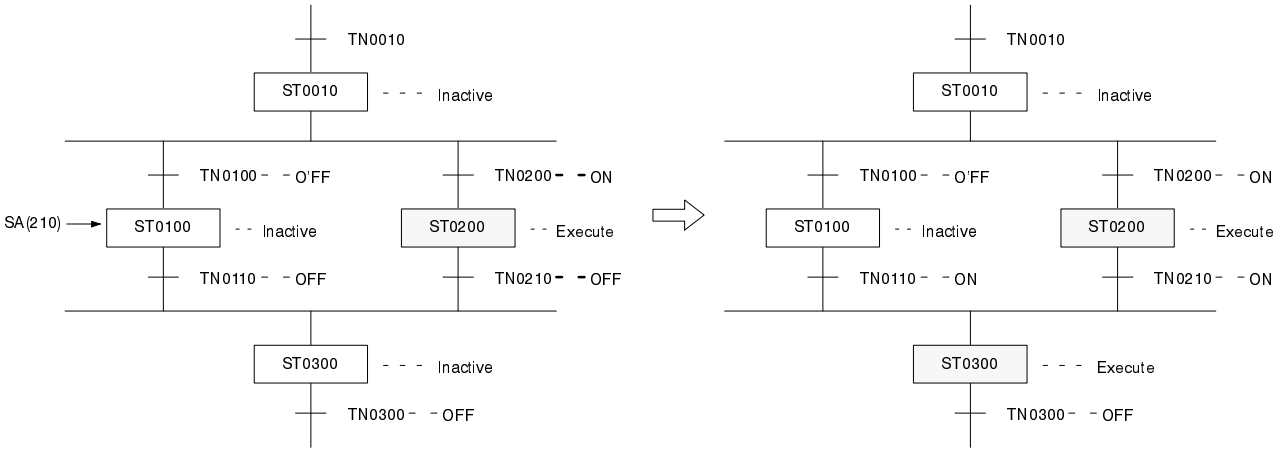
When the steps just before a parallel join (ST0110 and ST0210 in this example) are in execute status, consider the restrictions in movement of active status if you change ST0100 to execute status with SA(210).



When TN0300 turns ON, active status will be passed from ST0110 and ST0210 to the ST0300. If ST0100 has been changed to execute status with SA(210), active status will then be transferred to ST0110 when TN0110 turns ON, and ST0110 will remain in execute status until turned off with an SFC control instruction or until the next time the parallel section of the program is entered.

Conditional Joining

When a step just before a conditional join (ST0200 in this example) is in execute status, consider the restrictions in movement of active status if you change ST0100 to execute status with SA(210).



If ST0100 is changed to execute status with SA(210) and then the transition conditions are met for ST0100 and ST0200 simultaneously (i.e., TN0110 and

TN0210 turn ON simultaneously), the transition on the left will be given priority and active status will pass from ST0100 to ST0300. ST0200 will remain in execute status, therefore, when ST0100 becomes inactive and ST0300 goes to execute status. In this situation, you can either have ST0200 wait for active status to be passed from ST0300 to the next step (at which point active status would be passed from ST0200 to ST0300), or you can change ST0200 to inactive status with SE(214) or SOFF(215).

4–2 SFC Control Instructions

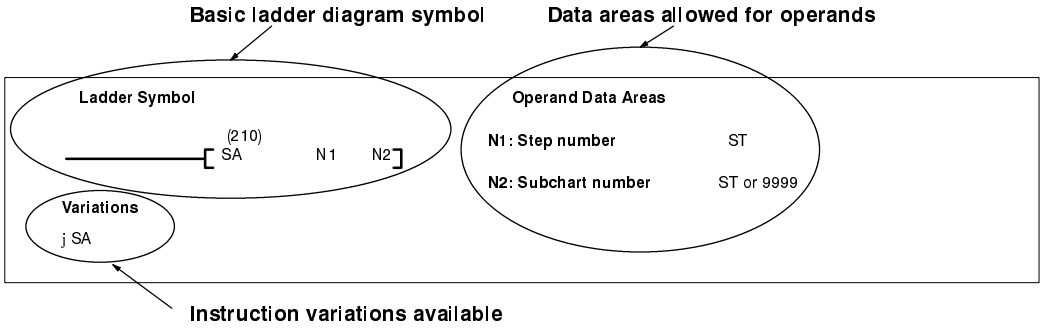
4–2–1 Introduction

The remainder of section 4 describes the ladder-diagram instructions that are related to SFC programs including SFC control instructions, which are used to control step status, and the instructions used to output transition conditions from transition programs (TOUT(202) and TCNT(123)).

Refer to the *CV-series PC Operation Manual: Ladder Diagrams* for details on ladder-diagram programs and data areas.

Layout

Each instruction is introduced with a frame that shows the basic form of the instruction, the variations of the instruction, and the data areas that can be used for each operand, as shown in the following illustration.



Basic Ladder Symbol

The ladder symbol shows how the instruction will appear in a program. The function code (210) is provided above the mnemonic (SA) and the operands are provided to the right (N_1 and N_2). The ladder symbol is the same for any of the variations of the instruction except that the mnemonic changes.

Operand Data Areas

The data areas that can be used for each instruction are listed. The actual operand will be a number, such as a word address, a bit address, an indirect address, or a constant, depending on the requirements of the instruction and the needs of the program. The abbreviations and addresses used for the data areas are listed in the following table.

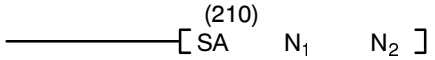
Abbreviation	Area	PC	Range
CIO	CIO Area (Core I/O)	CV500	Words: CIO 0000 to CIO 2427; Bits: CIO 000000 to CIO 242715
		CV1000/ CV2000	Words: CIO 0000 to CIO 2555; Bits: CIO 000000 to CIO 255515
TR	Temporary Relay Area	Both	TR0 to TR7 (bits only)
G	CPU Bus Link Area	Both	Words: G000 to G255; Bits: G00000 to G25515
A	Auxiliary Area	Both	Words: A000 to A511; Bits: A00000 to A51115
TN	Transition Area	CV500	TN0000 to TN0511 (Transition Flags)
		CV1000/ CV2000	TN0000 to TN1023 (Transition Flags)
ST	Step Area	CV500	ST0000 to ST0511 (Step Flags)
		CV1000/ CV2000	ST0000 to ST1023 (Step Flags)

Abbreviation	Area	PC	Range
T	Timer Area	CV500	T0000 to T0511 (PV or Completion Flag)
		CV1000/ CV2000	T0000 to T1023 (PV or Completion Flag)
C	Counter Area	CV500	C0000 to C0511 (PV or Completion Flag)
		CV1000/ CV2000	C0000 to C1023 (PV or Completion Flag)
DM	DM Area	CV500	D00000 to D08191 (words only)
		CV1000/ CV2000	D00000 to D24575 (words only)
	EM Area	CV1000/ CV2000	E00000 to E32765 for each bank (words only)
IR	Index registers	Both	IR0 to IR2 (words only)
DR	Data registers	Both	DR0 to DR2 (words only)

Variations The alternate forms of the instruction are listed here.

4–2–2 ACTIVATE STEP - SA(210)

Ladder Symbol



Variations

j SA

Operand Data Areas

N_1: Step number	ST
N_2: Subchart number	ST or 9999

Description SA(210) changes a designated step or subchart to execute status and starts execution of actions. A step activated by SA(210) will be executed as the last active step in the execution cycle for the first cycle after the step goes active.

To designate a step in a subchart, specify the step number of the dummy step calling the subchart as the subchart number, N_2, and then designate the number of the step with N_1.

To designate a subchart, designate the step number of the dummy step calling the subchart as the step number, N_1 , and designate a subchart number of 9999.

To designate a step not in a subchart, designate the step number and designate a subchart number of 9999.

Note SA(210) can neither be executed from an interrupt program nor for steps in an interrupt program.

Normal Steps

The specific results of executing SA(210) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Status does not change, but status will not be transferred to the next step for one execution cycle after execution of SA(210).

Pause Changes the step status from pause to execute. The output status that was in effect at the time the status was changed from execute to pause will be continued, and execution will begin from the top of the step.

Halt Changes the step status from halt to execute. The output status that was in effect at the time the status was changed from execute to pause will be continued, and execution will begin from the top of the designated step.

Inactive Changes the step status from inactive to execute. Execution will begin from the top action of the designated step. The step timer will be reset and started.

Subchart Dummy Steps

The specific results of executing SA(210) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute SA(210) does not change the subchart dummy step itself. All the steps in a subchart that are active when SA(210) is executed go to execute status. The output status that was in effect at the time the status was changed from execute to pause or halt will be continued, and execution will begin from the action at the top of the step.

Pause Changes the subchart dummy step and the steps in the subchart that are in pause status to execute status. The output status that was in effect at the time the status was changed from execute to pause will be continued, and execution will begin from the top.

Halt Changes the subchart dummy step, and the steps in the subchart that are halted, to execute status. The output status that was in effect at the time the status was changed from execute to halt will be continued, and execution will begin from the top.

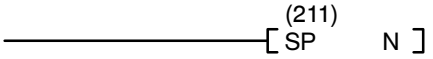
Inactive Changes the subchart dummy step, and the steps in the subchart that are inactive, to execute status. SA(210) also places the designated subchart entry step in execute status, and starts operation from the top action. The step timer will be reset and started.

Flags ER (A50003): Turns ON when the step designated by N_1 is undefined.
Turns ON when the subchart designated by N_2 is undefined.
Turns ON when the subchart designated by N_2 is not being executed.

Note A50003 will be ON only in the above case. When A50003 is ON, nothing will be executed.

4–2–3 PAUSE STEP - SP(211)

Ladder Symbol



Operand Data Area

N: Step number ST

Variations

j SP

Description SP(211) changes the status of a step or subchart from execute to pause. To designate a subchart, designate the step number of the dummy step calling the subchart as the step number, N.

In pause status, the execution of actions with N, P, L, and D action qualifiers is stopped, but the present values of TIM and TIMH instructions that have been started continue to operate. The present values of other timers and counters, as well as other outputs, are maintained. Step timers continue to operate, so be careful when using L and D action qualifiers.

Note SP(211) cannot be executed for steps in an interrupt program.

Normal Steps Status

The specific results of executing SP(211) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Changes step status from execute to pause. Execution of actions will be paused, but output status will be maintained and the step timer will continue. When

SP(211) is executed on the current step, execution of the actions in that step will be paused beginning with the next execution cycle.

Pause, Halt, Inactive Step status does not change.

Subchart Dummy Steps

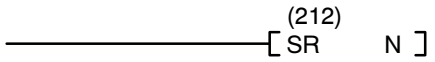
The specific results of executing SP(211) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute Changes the status of the subchart dummy step from execute to pause. All active steps in the designated subchart (including those in halt status) will be placed in pause status. Execution of actions will be paused, but output status will be maintained and the step timer will continue. If SP(211) is executed for a subchart from within the same subchart, the actions within the only step containing SP(211) will be paused beginning with the next execution cycle.

Pause, Halt, Inactive Step status does not change.

4–2–4 RESTART STEP - SR(212)

Ladder Symbol



Operand Data Area

N: Step number ST

Variations

j SR

Description SR(212) changes the status of a step or subchart from pause to execute.

Note SR(212) cannot be executed for steps in an interrupt program.

Normal Steps Status

The specific results of executing SR(212) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Step status does not change.

Pause Changes the step status from pause to execute. Execution will be resumed from the beginning of the step, and output status will continue with the status that existed at the time the step was changed to pause status.

Halt, Inactive Step status does not change.

Subchart Dummy Steps

The specific results of executing SR(212) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute Step status does not change.

Pause Changes the status of the subchart dummy step and all steps in the subchart that are in pause status from pause to execute. Execution will be resumed from the beginning of the subchart, and output status will continue with the status that existed at the time the dummy step was changed to pause status.

Halt, Inactive Step status does not change.

4–2–5 END STEP - SF(213)

Ladder Symbol	Operand Data Area
(213) SF N	**N: Step number** ST

Variations

j SF

Description SF(213) changes the status of a step or subchart from execute or pause to halt.

In halt status, the execution of actions with N, P, L, and D action qualifiers is stopped, but the present values of TIM and TIMH instructions that have been started continue to operate. The present values of other timers and counters, as well as other outputs, are maintained. Step timers continue to operate, so be careful when using L and D action qualifiers.

Note SF(213) cannot be executed for steps in an interrupt program.

Normal Steps Status

The specific results of executing SF(213) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Changes the step status from execute to halt. Execution of actions will be stopped, but output status will be maintained and the step timer will continue counting. When SF(213) is executed on the current step, execution of the actions in that step will be stopped beginning with the next execution cycle.

Pause Changes the step status from pause to halt.

Halt, Inactive Step status does not change.

Subchart Dummy Steps

The specific results of executing SF(213) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute Changes the status of the subchart dummy step from execute to halt. All active steps in the designated subchart (including those in pause status) will be changed to halt status. Execution of actions will be stopped, but output status will be maintained and the step timer will continue counting. If SF(213) is executed for a subchart from within the same subchart, the actions within only the step containing SF(213) will be stopped beginning with the next execution cycle.

Pause Changes the status of the subchart dummy step from pause to halt. All paused steps in the designated subchart will be changed to halt status. Execution of actions will be stopped, but output status will be maintained and step timers will continue.

Halt, Inactive Step status does not change.

4–2–6 DEACTIVATE STEP - SE(214)

Ladder Symbol	Operand Data Area
(214) SE N	**N: Step number** ST

Variations

j SE

Description

SE(214) changes the status of a step or subchart from active (execute, pause or halt) to inactive status.

SE(214) does not necessarily result in transfer of active status from one step to another. When SE(214) is executed, it simply makes the designated step or subchart inactive.

Note SE(214) cannot be executed from any interrupt program other than a power-on interrupt program. In addition, SE(214) cannot be executed for steps in any interrupt program (including a power-on interrupt program).

Normal Steps Status

The specific results of executing SE(214) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Changes the step status from execute to inactive. Execution of actions will be stopped and the actions will be reset. Actions with the optional hold action qualifier, however, will be stopped but not reset. In addition, execution will be continued for actions with S-group AQs. The step timer will continue. If SE(214) is executed on the current step, step execution will be stopped directly after execution of the instruction.

Pause Changes the step status from pause to inactive. Execution of actions will be stopped and the actions will be reset. Actions with the optional hold action qualifier, however, will be stopped but not reset. In addition, execution will be continued for actions with S-group AQs. The step timer will continue.

Halt Changes the step status from halt to inactive. The step's actions will be reset. Actions with the optional hold action qualifier, however, will not be reset. In addition, execution will be continued for actions with S-group AQs. The step timer will continue.

Inactive Step status does not change.

Subchart Dummy Steps

The specific results of executing SE(214) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute Changes status of subchart dummy step from execute to inactive, and makes inactive all of the steps in the subchart that were active at the time the instruction was executed. Actions will be stopped and reset. Actions with the optional hold action qualifier, however, will be stopped but not reset. In addition, execution will be continued for actions with S-group AQs. Step timers will continue. If SE(214) is executed for a subchart from within the same subchart, step execution will be stopped directly after execution of the instruction.

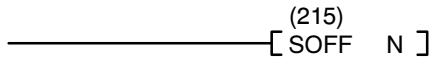
Pause Changes status of subchart dummy step from pause to inactive, and makes inactive all of the steps in the subchart that were active at the time the instruction was executed. Actions will be stopped and reset. Actions with the optional hold action qualifier, however, will be stopped but not reset. In addition, execution will be continued for actions with S-group AQs. Step timers will continue.

Halt Changes status of subchart dummy step from halt to inactive, and makes inactive all of the steps in the subchart that were active at the time the instruction was executed. Actions will be reset. Actions with the optional hold action qualifier, however, will not be reset. In addition, execution will be continued for actions with S-group AQs. Step timers will continue.

Inactive Step status does not change.

4–2–7 RESET STEP - SOFF(215)

Ladder Symbol



Operand Data Area

N: Step number ST

Variations

j SOFF

Description SOFF(215) makes a designated step or subchart inactive and resets all of the actions in the step.

SOFF(215) does not necessarily result in transfer of active status from one step to another. When SOFF(215) is executed, it simply makes the designated step or subchart inactive.

Note SOFF(215) cannot be executed from any interrupt program other than a power-on interrupt program. In addition, SE(214) cannot be executed for steps in any interrupt program (including a power-on interrupt program).

Normal Steps Status

The specific results of executing SOFF(215) for steps in each step status are described below (for normal steps inside or outside of subcharts).

Execute Changes the step status from execute to inactive. Execution of all actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and the actions will be reset. The step timer will also be stopped. If SOFF(215) is executed on the current step, step execution will be stopped directly after execution of the instruction.

Pause Changes the step status from pause to inactive. Execution of all actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and the actions will be reset. The step timer will also be stopped.

Halt Changes the step status from halt to inactive. All actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and reset. The step timer will be stopped.

Inactive Execution of actions with S-group AQs will be stopped and the actions will be reset. Actions with the optional hold action qualifier will be reset. The step timer will also be stopped.

Subchart Dummy Steps

The specific results of executing SOFF(215) for steps in each step status are described below (for dummy steps controlling subcharts).

Execute Changes the status of the subchart dummy step from execute to inactive and makes all of the steps in the subchart inactive. Execution of all actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and the actions will be reset. Step timers will also be stopped. If SOFF(215) is executed for a subchart from within the same subchart, step execution will be stopped directly after execution of the instruction.

Pause Changes the status of the subchart dummy step from pause to inactive and makes all of the steps in the subchart inactive. Execution of all actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and the actions will be reset. Step timers will also be stopped.

Halt Changes the status of the subchart dummy step from halt to inactive and makes all of the steps in the subchart inactive. Execution of all actions (including actions with the optional hold action qualifier and actions with S-group AQs) will be stopped and the actions will be reset. The step timer will also be stopped.

Inactive Resets all actions of steps in the subchart. Execution of Actions with the optional hold action qualifier and actions with S-group AQs will also be stopped and the actions will be reset. Step timers will also be stopped.

4–2–8 TRANSITION OUTPUT - TOUT(202)

Ladder Symbol



Description TOUT(202) outputs the result of a transition condition operation to the Transition Flag to control the transition condition. If TOUT(202) is executed with an ON execution condition, the Transition Flag will be turned ON. If TOUT(202) is executed with an OFF execution condition, the Transition Flag will be turned OFF.

TOUT(202) cannot be used outside of a transition program, and writing to the Transition Flag area cannot be achieved with any instructions other than TOUT(202) and TCNT(123) (see next section).

If TOUT(202) and/or TCNT(123) are used more than once in one transition program, the status of the Transition Flag will be controlled by the last TOUT(202) or TCNT(123) in the program.

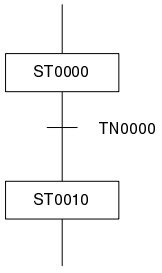
The ON/OFF status of a Transition Flag determined by TOUT(202) is held until the next execution of TOUT(202), even if the step status changes.

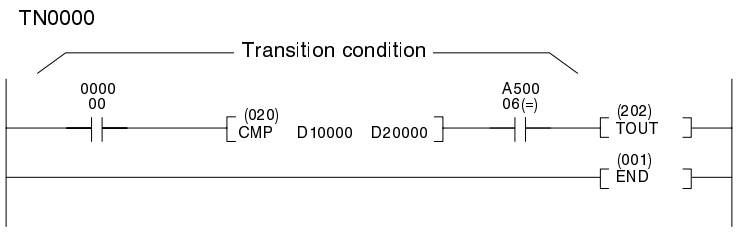
Note

1. A50015 (First Cycle Flag) cannot be used within a transition program.
2. The following three conditions must be met in order for active status to be transferred from one step to another:
   1) The transition condition (i.e., the Transition Flag) must be ON.
   2) The step(s) before the transition must be active (execute or halt, but not pause).
   3) The step(s) after the transition must be inactive.

Example In the following example, if CIO 000000 is ON and D100000 and D20000 have the same content, the Equals Flag, A50006, will turn ON, and TOUT(202) will turn ON the Transition Flag. If CIO 00000 is OFF or D100000 and D20000 have different contents, TOUT(202) will turn OFF the Transition Flag.

If the Transition Flag turns ON, ST0000 is active (but not paused), and ST0010 is inactive, then all of the transition conditions are met and active status will be transferred from ST0000 to ST0010.





4–2–9 TRANSITION COUNTER - TCNT(123)

Ladder Symbol	Operand Data Areas
(123) TCNT N S	**N: Counter number** C **S: No. of executions** CIO, G, A, T/C, #, DM, DR, IR

Limitations

The contents of word S must be in BCD.

Description

TCNT(123) turns a corresponding Transition Flag ON or OFF according to the number of times the transition program is executed.

The counter is started the first time TCNT(123) is executed with an ON execution condition after it is reset and the present value is incremented starting with the second execution. Therefore, the actual number of execution will be S + 1.

When the present value reaches the value set for S, the Transition Flag and the Counter Flag will turn ON.

The status of the transition counter is maintained even when step status is changed, so the transition counter should normally be reset using CNR(236) in the previous step.

If TOUT(202) and/or TCNT(123) are used more than once in one transition program, the status of the Transition Flag will be controlled by the last TOUT(202) or TCNT(123) in the transition program.

TCNT(123) cannot be used outside of a transition program and writing to the Transition Flag area cannot be achieved with any instructions other than TOUT(202) (see previous section) and TCNT(123).

Note

1. A50015 (First Cycle Flag) cannot be used within a transition program.
2. The following three conditions must be met in order for active status to be transferred from one step to another:
   1) The transition condition (i.e., the Transition Flag) must be ON.
   2) The step(s) before the transition must be active (execute or halt, but not pause).
   3) The step(s) after the transition must be inactive.

Precautions

The same counter numbers are used by CNT, CNTR(012), and TCNT(123). Do not use the same number to define more than one counter, regardless of the counter instruction used.

The transition counter counts each time the transition program is executed. When using a parallel join, be particularly careful of the number that is set, because the transition program will be executed each time the steps just above the transition are executed.

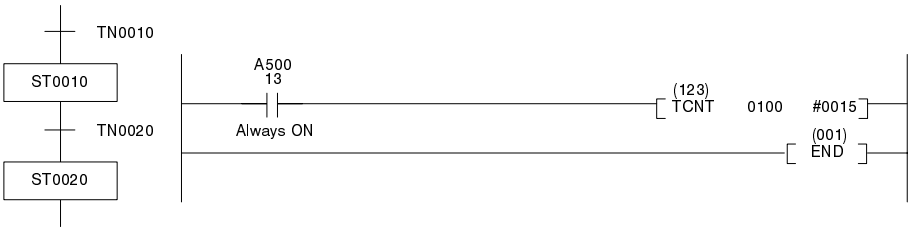
Flags

ER (A50003): ON when the content of N is not a counter number.
ON when the content of S is not BCD data.
On when the content of *DM or *EM word is not BCD.

Note A50003 will be ON only in the above case. When A50003 is ON, nothing will be executed.

Example

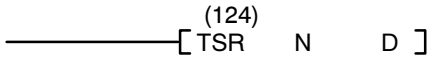
When the program for TN0020 is executed for the first time, counter C0100 will be started. After that, the transition program will be counted each time it is executed. As a result, the Completion Flag for C0100 will turn ON when the program has been executed 1 + 15 times, turning ON the Transition Flag for TN0020. From the 17th execution onwards, C0100 and the Transition Flag will remain ON until reset in a subsequent execution cycle.



Note Here, CNR(236) would be used in the ST0010 action block to reset counter C0100 with A50015 (First Cycle Flag).

4–2–10 READ STEP TIMER -TSR(124)

Ladder Symbol



Variations

j TSR

Operand Data Areas

N: Step number	CIO, G, A, T/C, #, DM, DR, IR
D: Destination	CIO, G, A, T/C, DM, DR, IR

Description

When executed with an ON execution condition, TSR(124) reads the present value (PV) of the step timer for the specified step and stores it as a binary value in the word designated by D. When executed with an OFF execution condition, TSR(124) does nothing.

Step Timers

Step timers time SFC steps and are used for operations such as measuring AQ timing when the AQ uses a set value to control action execution. Each step has its own step timer. Step timers are incremental timers that are reset and begin counting when the step becomes active, and which retain the present value when the step becomes inactive. If, however, the actions in the step continue to be executed (e.g., for S-group AQs), the step timer continues operating until all execution has been completed.

The PC can be set so that step timers operate in increments of either 0.1 second or 1 second. The increment is set in the PC system settings. The default setting is 0.1 second. Step timers can count up to 6,553.5 s (when the unit is 0.1 s) or 65,535 s (when the unit is 1 s). The step timer retains the maximum value if the present value goes beyond the timeable range.

Step timer values are stored and handled as binary data.

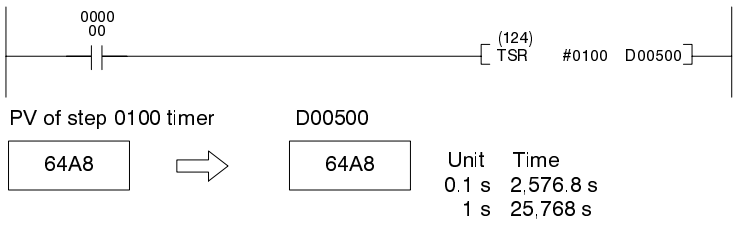
Flags

ER (A50003): ON when the N is set outside of the range.
On when the content of *DM or *EM word is not BCD.

Note A50003 will be ON only in the above case. When A50003 is ON, nothing will be executed.

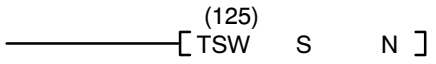
Example

When CIO 000000 is ON in the following example, the present value of the step timer for step ST0100 will be output to D00500 in binary data. This is unrelated to the active/inactive status of the step.



4–2–11 WRITE STEP TIMER - TSW(125)

Ladder Symbol



Variations

j TSW

Operand Data Areas

S: Source	CIO, G, A, T/C, DM, DR, IR
N: Step number	CIO, G, A, T/C, #, DM, DR, IR

Limitations

The contents of S must be set with binary data.

Description

TSW(125) is used to change the present value (PV) of the step timer. If TSW(125) is executed with an ON execution condition, the contents of word S is written as the present value for the step timer for step N. When executed with an OFF execution condition, TSW(125) does nothing.

Refer to the previous instruction for details on step timers.

Flags

ER (A50003): ON when the N is set outside of the range.
ON when the content of *DM or *EM word is not BCD.

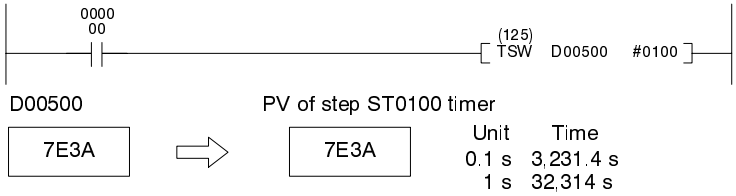
Note A50003 will be ON only in the above case. When A50003 is ON, nothing will be executed.

Precautions

Step timers are used to control the execution timing of actions with certain AQs. Be careful when making changes with TSW(125).

Example

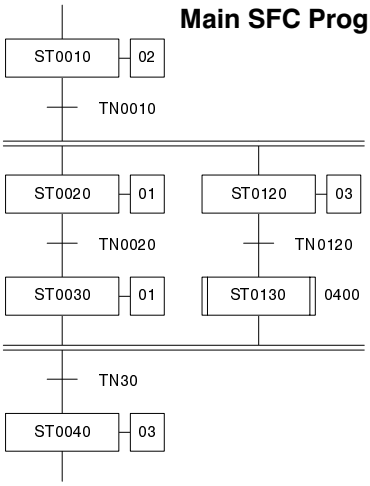
When CIO 000000 is ON in the following example, the contents of D00500 ($7E3A) will be written as the present value for the step timer of step ST0100. This is unrelated to the active/inactive status of the step.



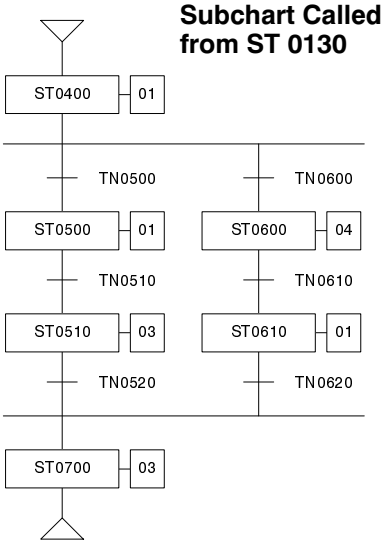
4–3 Program Example

The operation of the SFC control instructions used in the following example is described after the program.

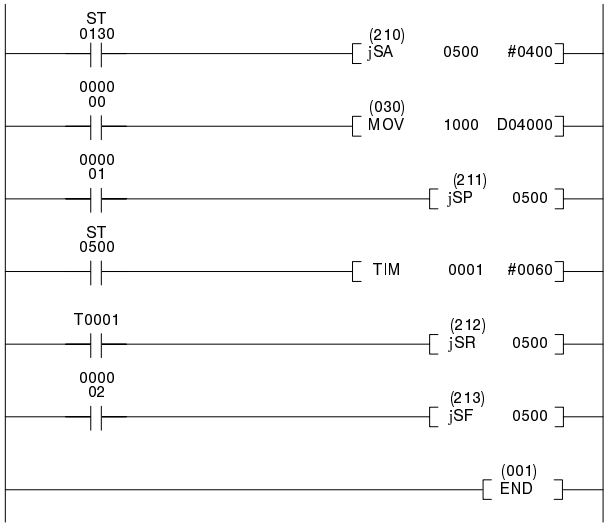
Main SFC Program



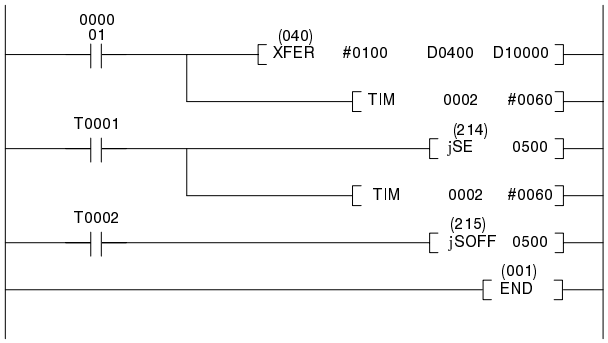
Subchart Called from ST 0130



Action Program for AC0020 (ST0020)



Action Program for AC0500 (ST0500)



Action Block for ST0020

AQ	SV	Action	FV
N	********	AC0020	********

Action Block for ST0030

AQ	SV	Action	FV
N	********	AC0030	********

Action Block for ST0500

AQ	SV	Action	FV
S	********	AC0050	********

SA(210) in AC0020 When subchart ST0400 (i.e., the subchart with initial step ST0400) is active, SA(210) changes the status of ST0500 in the subchart to execute status. (Subchart ST0400 is executed while ST0130 is active.)

SP(211) in AC0020 SP(211) changes the status of ST0500 to pause.

SR(212) in AC0020 SR(212) changes the status of ST0500 back from pause to execute.

SF(213) in AC0020 SF(213) changes the status of ST0500 to halt. Execution of action AC0500 will continue because it has an "S" AQ.

SE(214) in AC0500 SE(214) changes the status of ST0500 to inactive. Execution of action AC0500 will continue because it has an "S" AQ.

SOFF(215) in AC0500 SOFF(215) changes the status of ST0500 to inactive. The action in the step is stopped and reset even though it has an S-group AQ.

SECTION 5
SFC Execution Cycle and Errors

This sections describes how the status of steps and the action qualifiers of actions determines the order of SFC program execution. The methods used to compute user program execution time and error codes are also provided.

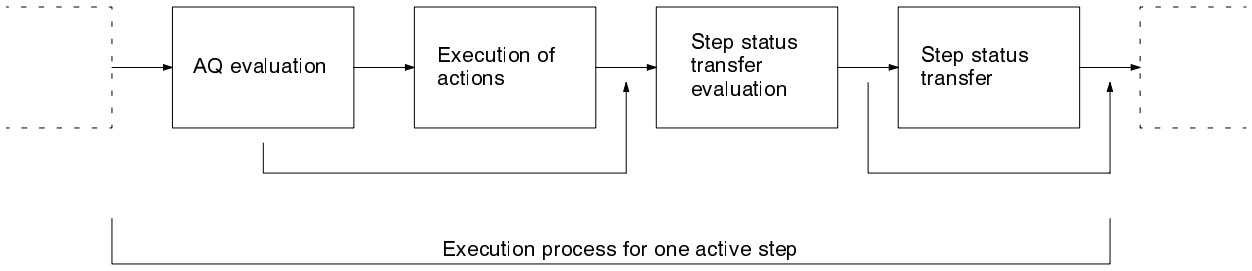
5−1 Basic Execution Cycle 68
5−2 Execution Cycle with Multiple Active Steps 68
5−3 Execution Cycle with Subcharts 69
5−4 Power-Up and Restart Execution 71
5−4−1 Power-Up Execution 71
5−4−2 Restart Continuation 71
5−4−3 Items Retained and Not Retained with Restart Continuation 73
5−5 User Processing Time 74
5−6 SFC Error Codes 76
5−6−1 Fatal SFC Errors 76
5−6−2 Non-fatal SFC Errors 76
5−6−3 SFC Program Precautions 77

5–1 Basic Execution Cycle

In an SFC program, multiple steps can become active simultaneously, and multiple actions can be executed simultaneously according to their action qualifiers (AQs). Furthermore, transition programs are also executed. Actually, all of these processes only appear to be executed simultaneously. In reality, they are executed in sequential order as a part of the overall execution cycle.

Step Execution Process

In an SFC execution cycle, the processes shown in the illustration below are executed for all active steps. Nothing is executed for inactive steps.



AQ Evaluation

The action qualifiers are evaluated to determine whether or not each action is to be executed.

Execution of Actions

All the actions whose AQ evaluation indicates execution are executed in the order in which they are listed in the action block. If an action is defined using a bit address, rather than with a program, the bit is turned ON and OFF with the timing of action execution.

Note If there is a loop in an action and the action does not end, the next action will not be executed and active status will not be transferred to the next step. If this occurs, there will be a cycle time overrun and operation will be stopped.

Status Transfer Evaluation

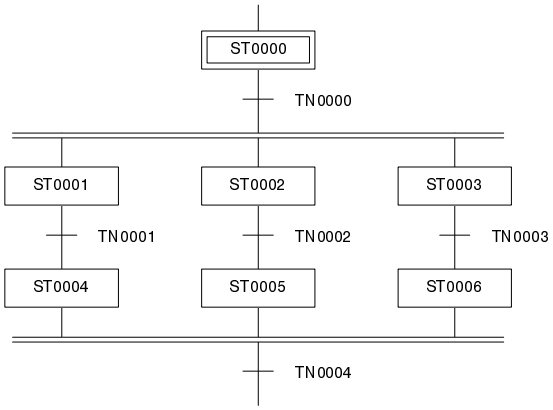
The transfer conditions are evaluated to determine whether or not active status is to be transferred from a given step to the next step. The transition program following the step is executed or the status of the bit defined as the transition condition is examined. When there are multiple steps active at the same time (as, for example, in cases of parallel joining), the transition program after these steps is executed once following each active step. In other words, the same transition program may be executed a number of times within the same cycle.

Step Status Transfer

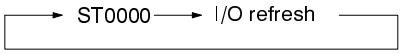
When it is determined that active status will be transferred to the next step, the currently active step will be made inactive and the next step will be made active.

5–2 Execution Cycle with Multiple Active Steps

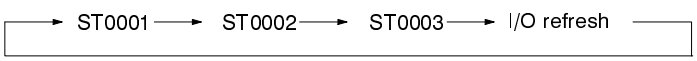
When there are multiple steps active at one time, the execution process described above will be performed for each active step. The order of execution will depend on the positions of the steps and the order of priority for initial steps. For details on the order of priority for initial step execution, refer to page 24, *Initial Steps*.



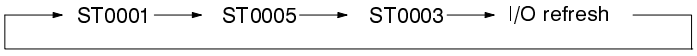
In the SFC program shown above, the following execution cycle will be repeated if only ST0000 is active. If the I/O refresh is set to cyclic refresh, the refresh will be executed at the end of each cycle.



When the condition for TN0000 is met, active status is transferred from ST0000 to ST0001, ST0002, and ST0003 simultaneously. After the status has been transferred, the step that has become inactive will be left out of the cycle, and the steps that have newly become active will be executed instead. These newly active steps will be executed in order from left to right in the SFC program. In the example above, ST0000 would be left out and ST0001, ST0002, and ST0003 would be executed. The order of execution would be as follows:



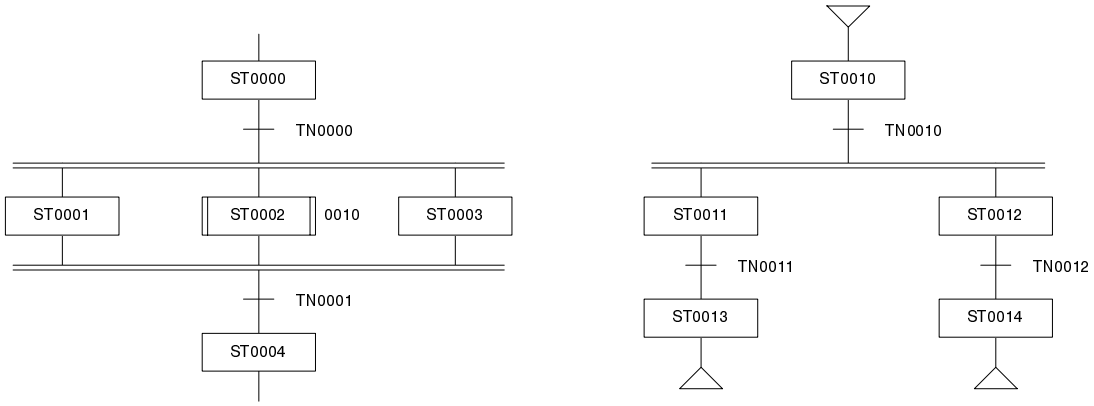
If the condition for TN0002 is then met, and active status is transferred from ST0002 to ST0005, the execution cycle will be as follows:



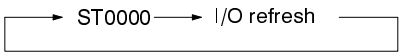
As can be seen from these examples, the execution cycle of an SFC follows a fixed order.

5–3 Execution Cycle with Subcharts

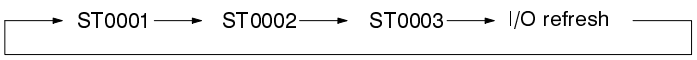
The following example illustration shows a case where a subchart is executed as part of the cycle. Actions have been omitted in the illustration.



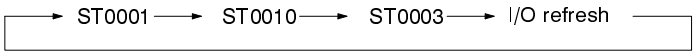
In this SFC program, if only ST0000 is active, the execution cycle will be as shown below. If the I/O refresh is set to cyclic refresh, the refresh will be executed at the end of each cycle.



When the condition for TN0000 is met and active status is transferred from ST0000 to ST0001, ST0002, and ST0003, the execution cycle will be as follows:

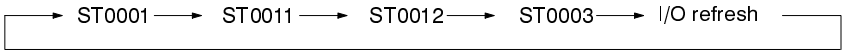


In this case, ST0002 is a subchart dummy step, so subchart 0010 will be called and step ST0010 will also go to active status. The execution order of the active steps in the subchart will thus occupy the position of ST0002 in the cycle.

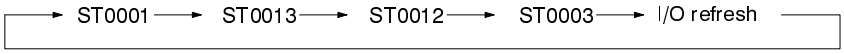


The active status of the subchart dummy step will not be transferred until execution of the subchart has been completed; so, even if TN0001 turns ON, ST0004 will not become active until subchart 0010 has been completely executed. Evaluation of TN0001, however, will be conducted with each cycle while the subchart is being executed.

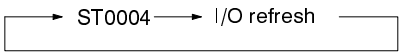
When TN0010 turns ON, the active status within the subchart will be transferred from ST0010 to ST0011 and ST0012. At that time, the execution cycle will be as follows:



If TN0011 then turns ON and the active status within the subchart is transferred from ST0011 to ST0013, the execution cycle will be as follows:

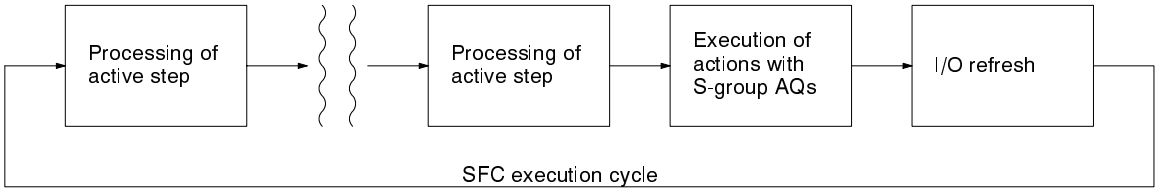


ST0013 is the subchart return step, so TN0001 (which follows the subchart dummy step, ST0002) will now become effective. When TN0001 turns ON, active status will be transferred from ST0001, ST0002, and ST0003 to ST0004. At this point, the execution of the subchart will be complete and ST0014 will not be executed. When the subchart is finished, all of the actions within the subchart, including ST0012, go to inactive status. The execution cycle will then be as follows:



Actions with S-group AQs

Actions with S-group AQs (S, SL, SD, and DS) are executed once each cycle, after all active steps have been processed. Therefore, if a step has an action with an S-group AQ, that action will not be executed during step processing. Rather, it will be executed together with other S-group actions at the end of the cycle. All actions other than those with S-group AQs are executed during step processing.



Even when actions are defined using bit addresses, they turn ON and OFF with the execution timing described above. Therefore, when actions with S-group AQs are defined using bit addresses, the bits are turned ON or OFF at the end of each cycle.

Up to 127 actions with S-group AQs can be executed within the same cycle. Any actions exceeding 127 will not be executed, and a non-fatal SFC error will be generated.

5–4 Power-Up and Restart Execution

5–4–1 Power-Up Execution

If a power-on interrupt program exists and the restart continuation setting is made before the power is cut, then when the power is turned back on, execution will commence from the power-on interrupt program.

When operation is started under normal conditions, the program will be executed from all of the initial steps in the program as described in *3-3 Initial Steps*. All steps other than the initial steps will start out inactive.

5–4–2 Restart Continuation

There may be occasions when the program will be interrupted due to a power failure or other cause, and you will want to resume operation from the point where the program left off. At such times you can use the restart continuation setting.

In order to execute restart continuation, you must follow the procedure outlined below. The mode when the power is cut must be either RUN or MONITOR.

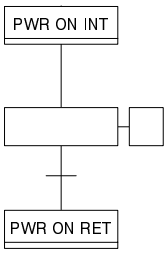
1, 2, 3...

1. Turn ON the Restart Continuation Bit (A00011).
2. Turn ON the IOM Hold Bit (A00012).
3. In the PC system settings, set both of the above bits (A00011 and A00012) to retain status at power-up.
4. In the PC system settings, set the mode at the time of power-up to either RUN or MONITOR.
5. Create a power-off interrupt program. (For details on how to create a power-off interrupt program, refer to the CVSS operation manuals.)
6. In the PC system settings, set the power-off interrupt to “enable.”

For details on restart continuation, refer to the *CV-series PC Operation Manual: Ladder Diagrams.*

Power-On Interrupt Program

A power-on interrupt program is executed at the beginning of the program only when restart continuation is enabled. The power-on interrupt program is written as shown in the illustration below.



When using a step control instruction to select the step that is to execute initialization and resume operation, write the instruction in this power-on interrupt program.

Basic Operation Example

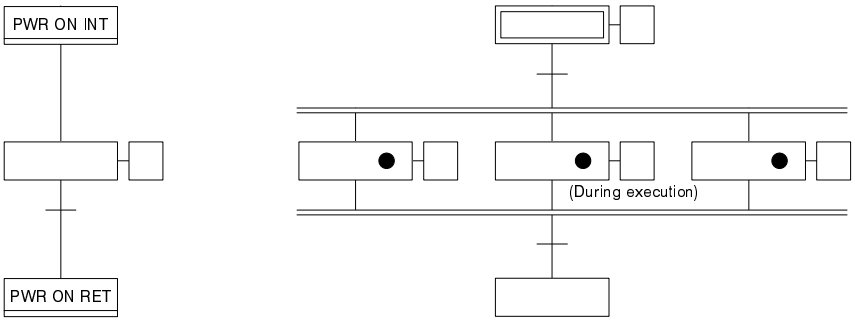
If the restart continuation setting is made before the power is cut, then when the power is turned back on, execution will commence from the power-on interrupt

program. If the power-on interrupt program finishes, or if there is no power-on interrupt program, then execution will resume with the next instruction after the last one that was completed at the time of the power failure.

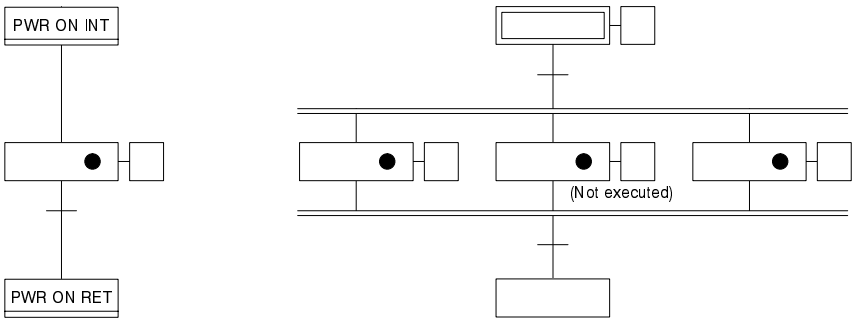
If the active status of steps has been changed by a step control instruction in the power-on interrupt program, then execution will commence from the beginning of the currently active steps.

The example illustration below shows status changes for restart continuation. The black dots indicate the steps that are active.

Status Before Power Failure

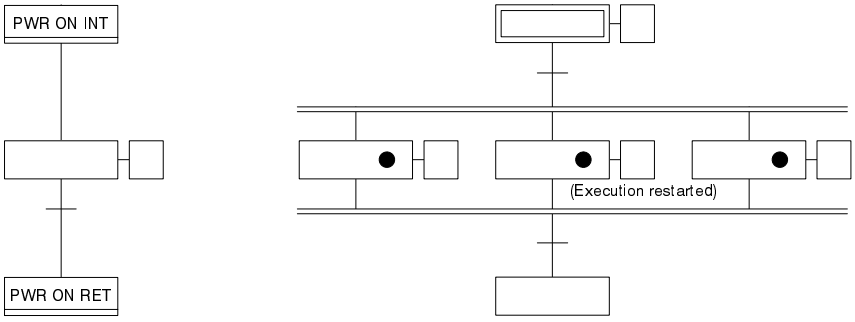


Status when Powered Up Again



The steps that were active just prior to the power failure remain active, but their programs will not be executed. The power-on interrupt program will be executed instead.

Status after Completion of Power-On Interrupt Program



Execution will be resumed with the instruction after the last one that was executed at the time of the power failure. If there is a power failure during execution of an I/O interrupt or scheduled interrupt program, execution will be resumed from that point in the program.

5–4–3 Items Retained and Not Retained with Restart Continuation

Items Retained

The following are retained after a power interruption if restart continuation is designated.

- Active status of steps
- Execution status of interrupt programs
- Causes of interrupt programs accepted before power interruption (except for power-on interrupt)
- Program execution position
- Interlock status
- I/O memory status
- Forced set/reset status (along with A00013 and PC system settings)
- PV of timer/counter
- PV of step timer
- AQ (L, D, SL, SD, DS) lapse time
- Error history information
- Trace execution status (If Trace Execution at Power-Up is set with a PC system setting, then the PC system setting is given priority and the trace is made the initial start.)
- Data saved by CCS(173)
- Index register contents
- The Expansion DM bank number (Even when EMBC(171) is used in a power-off interrupt program, the status just prior to execution of the power-off interrupt program is restored.)

Items Not Retained

The following are not retained after a power interruption if restart continuation is designated. The status of any of the first eight items required after a power interrupt must be stored in held memory using the power-off interrupt program and then restored using the power-on interrupt program.

- Error status (Error history information is retained.)
- I/O memory access prohibit status from Peripheral Devices programmed with IOSP(187) and I/O Control Unit, I/O Interface Unit, and Remote I/O Slave Unit display data programmed with IODP(189)
- CPU Bus Service Disable Bits (word A015)
- Peripheral Service Disable Bit, Host Link Service Disable Bit, and I/O Refresh Disable Bit (word A017)
- Registered message data
- I/O interrupt mask status
- Scheduled interval set time
- Data register contents
- Execution status of differential monitor
- Execution time count status
- Execution status of pause monitor

Also, execution of the following instructions may not be properly completed depending on the timing of the power failure:

SEND(192), RECV(193), CMND(194), READ(190), WRIT(191), FILR(180), FILW(181), FILP(182), FLSP(183)

5–5 User Processing Time

The time required for processing of a user program will vary depending on factors such as the number of active steps, the number of transitions evaluated, and the number of actions executed in each step. This section explains how processing time for a user program is calculated.

The execution time for a user program can be found with the following formula.

User program execution time = (A) + (B) + (C), where

(A) = Processing time for active steps

(B) = Processing time for transition evaluation

(C) = Processing time for action execution

The values of (A) through (C) can be found as follows:

(A) Active Steps

This is the time required in a cycle for processing steps with execute status (not executed for steps pause or halt status even if the steps are active). It is arrived at by means of the following equation:

(A) = Constant 25 ms + (10 ms) x number of active steps

(B) Transition Evaluation

This is the total processing time involved in evaluating transitions in a cycle. The processing time for individual transitions will vary depending on how the transitions are defined (bits or transition programs) and how many times they are evaluated. The processing time for one transition is as follows:

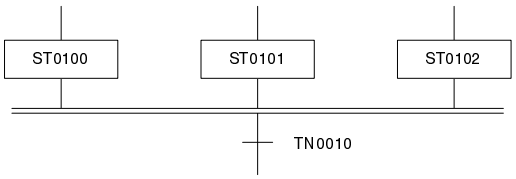
Transition bits: 8 ms

Transition programs: Constant 36 ms + instruction processing time

For details on instruction processing time, refer to the table of instruction execution times in the *CV-series PC Operation Manual: Ladder Diagrams*.

Be careful to take into account the fact that, when a parallel join is used, a single transition will be evaluated several times in one cycle. In other words, transition evaluation is performed individually for each active step after which a transition is connected. This process is repeated for transitions connected after multiple active steps. The transitions will be evaluated for as many times as the number of active steps before it.

In the following example, if ST0100, ST0101, and ST0102 are all active, then TN0010 will be evaluated three times each cycle.



(C) Action Execution

This is the time required in a cycle to process actions. The processing time required for individual actions will vary depending on how the actions are defined (bits or action programs). The processing time for one action is as follows:

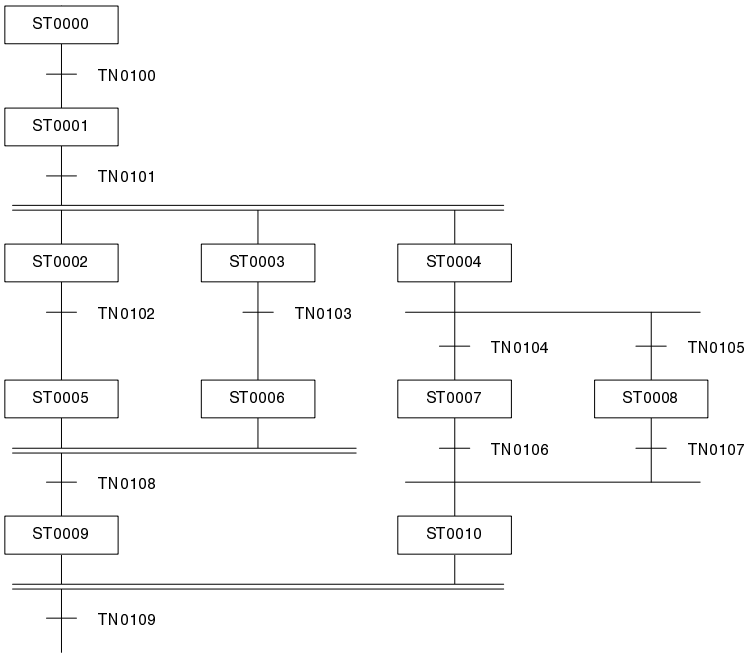
Action bits: 6 ms

Action programs: Constant 35 ms + instruction processing time

For details on instruction processing time, refer to the table of instruction execution times in the *CV-series PC Operation Manual: Ladder Diagrams*.

Depending on the AQ, the numbers shown above may increase by several microseconds.

Example: Calculating User Program Execution Time



ST0000 Action Block

S		AC0000	

ST0004 Action Block

N		00100	
S		AC1000	

ST0005 Action Block

N		00200	
N		AC2000	

ST0006 Action Block

N		00300	

In this program, let us suppose that ST0004, ST0005, and ST0006 are currently active (execute status), and that execution of AC0000 was begun earlier when ST0000 was active. The transitions that will be evaluated in this cycle are TN0104, TN0105, and TN0108. TN0108 will be evaluated twice in the cycle (i.e., once each for ST0005 and ST0006).

The contents of TN0104, TN0105, and TN0108 are as follows:

TN0104: Bit address
TN0105: Ladder program
TN0108: Bit address

The total processing times for execution of the action and transition programs (ladder diagrams) are assumed to be as follows:

AC0000 . . 2,000 ms
AC1000 . . 1,500 ms
AC2000 . . 3,000 ms
TN0105 . . . . . 30 ms

The total time required for program execution can then be calculated as follows:

(A) Active Steps Constant 25 ms + (10 ms) x 3 = 55 ms

(B) Transition Evaluation 8 ms for TN0104 + (constant 36 ms + 30 ms) for TN0105 + (8 ms for TN0108 x 2) = 90 ms

(C) Action Execution

35 ms (constant) + 2,000 ms = 2,035 ms . . . [AC0000]
35 ms (constant) + 1,500 ms = 1,535 ms . . . [AC1000]
35 ms (constant) + 3,000 ms = 3,035 ms . . . [AC2000]
6 ms x 3 = 18 ms . . . . [bits 000100, 000200, and 000300]

Total 6,623 ms

User program processing time = (A) + (B) + (C)

= 55 ms + 90 ms + 6,623 ms

= 6,768 ms

Note

1. If active status is transferred during the course of a cycle, the execution time for that portion of the program will be lengthened. The same applies when an interrupt program is executed.
2. If only action programs and transition programs are used, use the following equation:

   User processing time = Constant 35 ms + instruction processing time

5–6 SFC Error Codes

5–6–1 Fatal SFC Errors

If a fatal SFC error occurs, the SFC Fatal Error Flag (A40107) will turn ON and one of the following error codes (in BCD) will be set in word A414.

Code	Error	Content	Possible correction
0001	No active step	There are no active steps.	Check the program.
0002	No subchart	There is no subchart entry step for the subchart dummy step, or the subchart number is incorrect.	Check and then re-transfer the program.
0003	No initial step	There are no initial steps in the program.	Create an initial step and then re-transfer the program.
0007	No action set	There is no action program set, or the bit address for the action is incorrect.	Check and then re-transfer the program.
0008	No transition set	There is no transition program set, or the bit address for the transition is incorrect.	
0014	Interrupt return error	An interrupt return terminal was executed outside of an interrupt program.	Check the program.
0015	Subchart return error	A subchart return step was executed outside of a subchart program.	
0004 to 0006, 0009 to 0013	Memory error	Memory contents are incorrect.	Re-transfer the program.

5–6–2 Non-fatal SFC Errors

If a non-fatal SFC error occurs, the SFC Non-fatal Error Flag (A40211) will turn ON and one of the following error codes (in BCD) will be set in word A418.

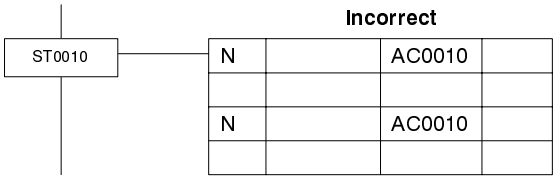
Code	Error	Content	Possible correction
0001	Overlapping action execution	The program attempted to execute the same action from more than one step at the same time.	Check the program. If you wish, you can make a PC system setting so that overlapping action execution will not generate an error. With the default setting, this non-fatal SFC error is generated.
0002	S-group AQ overuse (Note)	The program attempted to simultaneously execute more than 127 actions with S-group AQs. Any actions that exceed the maximum allowable 127 will not be executed.	Check the program.
0003	Overlapping S-group AQ execution (Note)	The program attempted multiple execution of the same action having an S-group AQ.	Check the program.

Note S-group AQs include S, SL, SD, and DS. Up to 127 S-group AQ actions can be executed simultaneously.

5–6–3 SFC Program Precautions

If FILP(182) is executed in the first cycle when a step becomes active, an overlapping action execution error will be generated.

An action program cannot be created which uses the same action number more than once in the same step. The following action block is thus incorrect.



SECTION 6
SFC Application Examples

This section provides examples that illustrate some specific applications of SFC programs. The last example also describes the differences between ladder diagrams and SFC programs and shows how one example program can be converted.

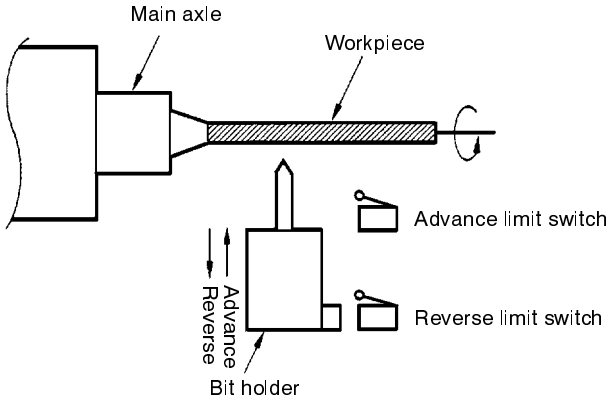
6−1 Example 1: Sequential Control 80
6−2 Example 2: Conditional Branching and Joining 81
6−3 Example 3: Using AQs 84
6−4 Example 4: Parallel Branching and Joining 88
6−5 Example 5: Ladder vs SFC 91

6–1 Example 1: Sequential Control

This example shows how to use SFC programming for sequential processing.

Operation

In this example, the industrial tool shown in the illustration is used to cut workpieces.



The following operation is performed:

1, 2, 3...

1. When the start switch is turned ON, the program first checks to be sure that the bit holder is in the reverse position (at the reverse limit switch). Then it begins rotating the main axle. Two seconds later it advances the bit holder.
2. When the bit holder reaches the advance position (the advance limit switch), it stops for one second and is then reversed.
3. When the bit holder reaches the reverse position (the reverse limit switch), it stops. The rotation of the main axle also stops.

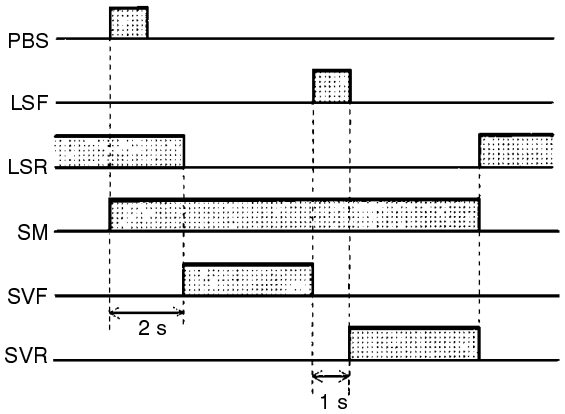
I/O Devices

I/O devices are allocated to I/O bits as shown below. I prefixes indicate inputs; Q prefixes, outputs. (These prefixes are used on CVSS displays.) The symbols are used in the action blocks to indicate the bits that are controlled by each action.

Name	Symbol	Allocation
Start switch	PBS	I000001
Advance limit switch	LSF	I000002
Reverse limit switch	LSR	I000003
Main axle motor	SM	Q000101
Bit holder advance cylinder	SVF	Q000102
Bit holder reverse cylinder	SVR	Q000103

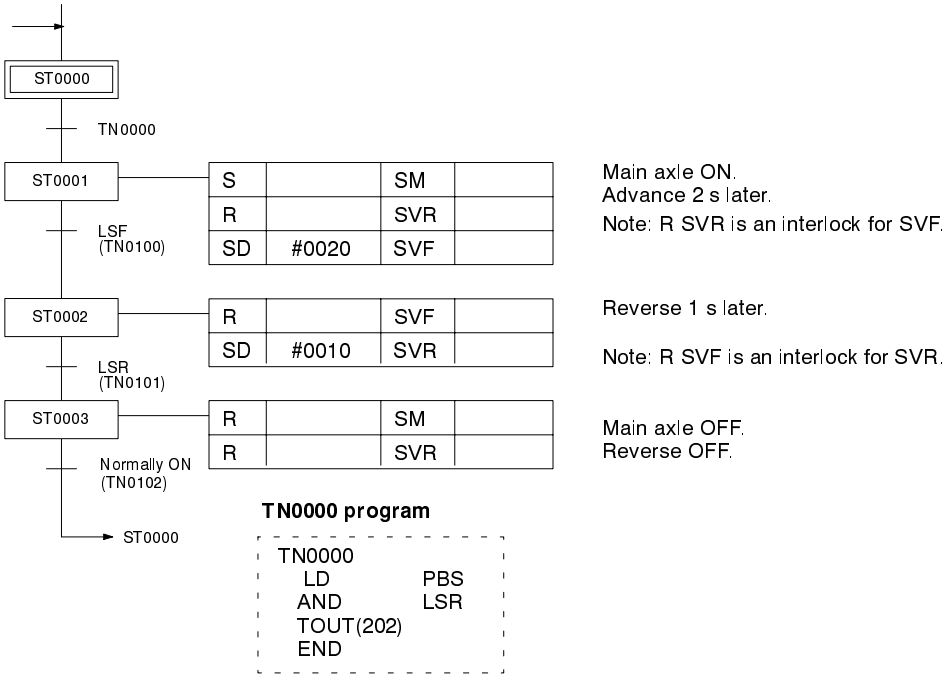
Timer Chart

Operation timing is illustrated below.



SFC Program

The following program shows how the above operation can be programmed as sequential steps. A transition number is required even when a transition is defined as a bit. In the chart, these transitions are enclosed in parentheses. The transition program is shown as a mnemonic list.

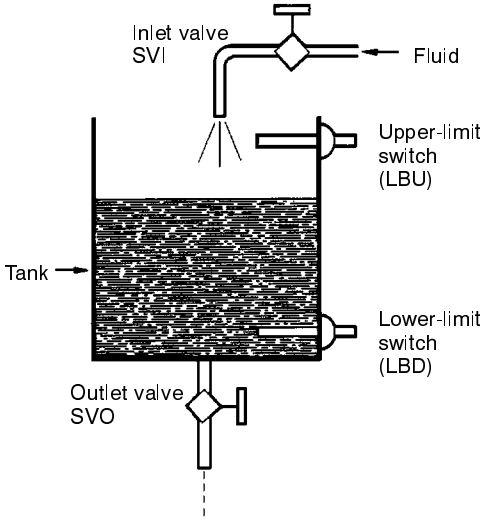


6–2 Example 2: Conditional Branching and Joining

This example shows how to use conditional branching and joining together with a subchart to provide for emergency processing and to switch between manual and automatic operation.

Operation

In this example the fluid in a tank rises until it reaches the upper limit, and then it is discharged.



The following operation is performed:

1, 2, 3...

1. During automatic operation, when the fluid level falls below the lower limit, fluid is added until it passes the upper limit.
2. During automatic operation, when the fluid level rises above the upper limit, fluid is discharged until it passes the lower limit.

3. During manual operation, both the inlet valve and the outlet valve will be open while their respective push-button switches are being pushed.
4. If the emergency stop switch is pushed, both the inlet valve and the outlet valve will be closed. Operation can be restarted with the start switch.

I/O Devices

I/O devices are allocated to I/O bits as shown below. I prefixes indicate inputs; Q prefixes, outputs. (These prefixes are used on CVSS displays.) The symbols are used in the program to indicate the bits that are controlled.

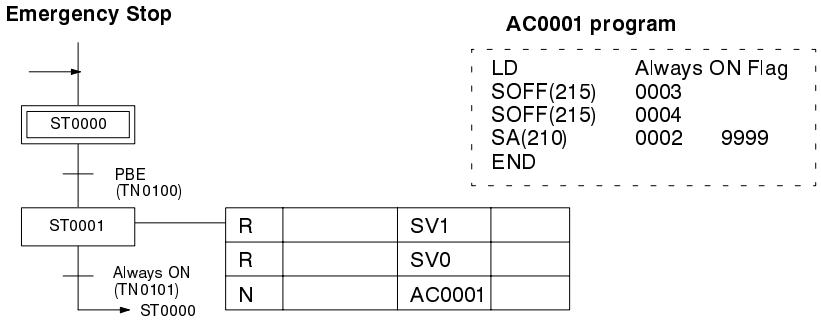
Name	Symbol	Allocation
Emergency stop switch (push-button)	PBE	I001001
Automatic operation switch	CSA	I001002
Manual operation switch	CSM	I001003
Start switch (push-button)	PBS	I001004
Manual inlet valve switch	PBI	I001005
Manual outlet valve switch	PBO	I001006
Upper-limit switch	LBU	I001007
Lower-limit switch	LBD	I001008
Inlet	SVI	Q002001
Outlet	SVO	Q002002

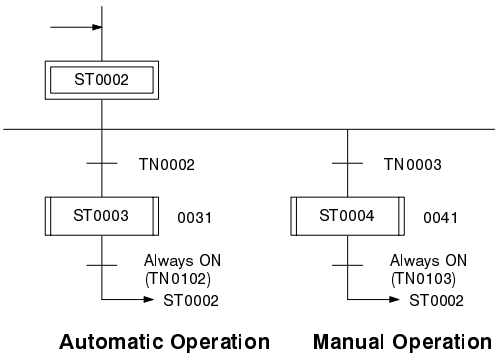
Main SFC Program

The following program shows how the above operation can be programmed. A transition number is required even when a transition is defined as a bit. In the chart, these transitions are enclosed in parentheses. The action and transition programs are shown as mnemonic lists.

The main program is divided into two. Both parts of the program are started with initial steps when power is turned on.

The first part of the program uses ST0001 for emergency stop processing, and the system will be completely reset if PBE is turned ON. In the section part of the program, ST0031 is a subchart entry step number for automatic operation, and ST0041 is a subchart entry step number for manual operation. Both of these program sections are shown below.



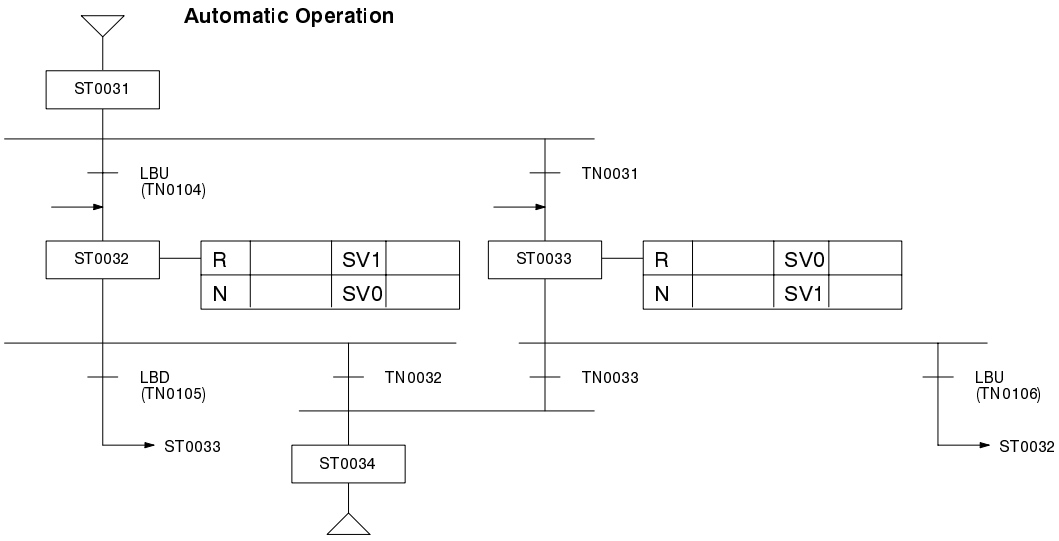



```
TN0002
  LD NOT    PBE
  AND       PBS
  AND       CSA
  AND NOT   CSM
  TOUT(202)
  END(001)
```

```
TN0003
  LD NOT    PBE
  AND       CSM
  AND NOT   CSA
  TOUT(202)
  END(001)
```

SFC Subcharts

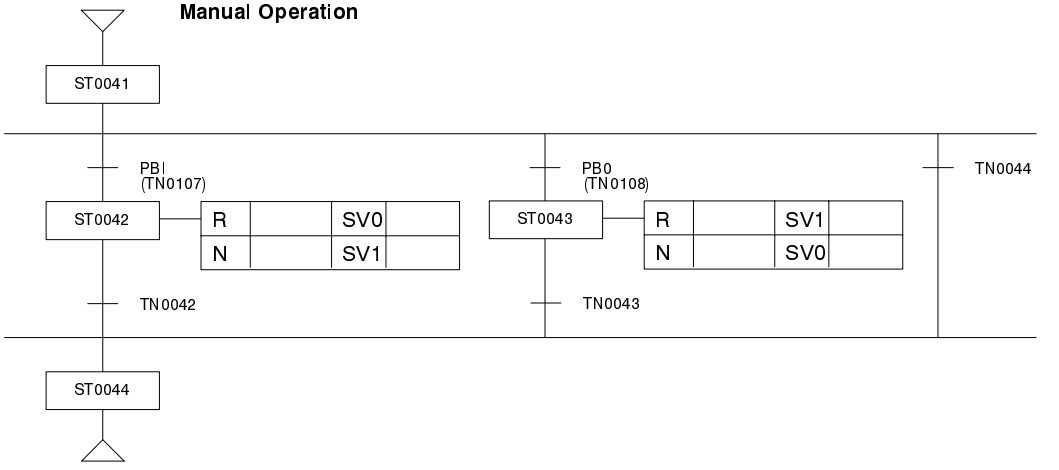
Here are the subcharts called by the second part of the main program. Because conditional branching is used, only one of these will become active at a time.




```
TN0031
  LD NOT    LBU
  TOUT(202)
  END(001)
```

```
TN0032
  LD        CSM
  OR NOT    PBS
  TOUT(202)
  END(001)
```

```
TN0033
  LD        CSM
  OR NOT    PBS
  TOUT(202)
  END(001)
```




```
TN0042
  LD NOT    PBI
  TOUT(202)
  END(001)
```

```
TN0043
  LD NOT    PBO
  TOUT(202)
  END(001)
```

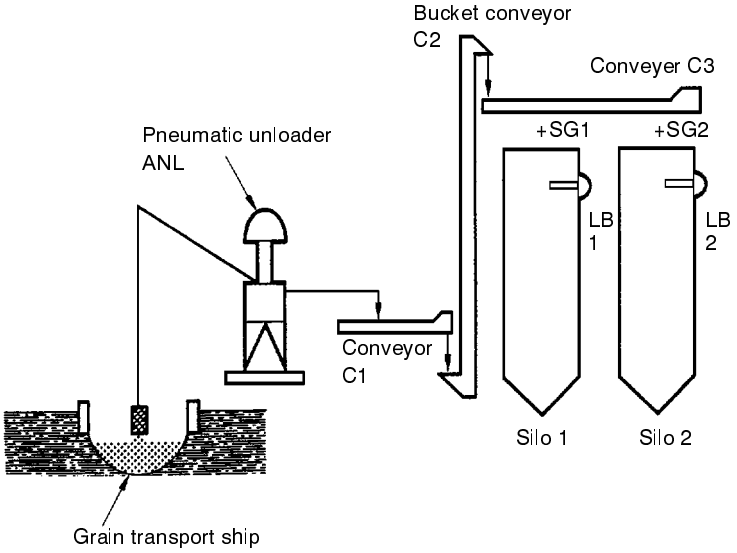
```
TN0044
  LD NOT    PBI
  AND NOT   PBO
  TOUT(202)
  END(001)
```

6–3 Example 3: Using AQs

This example shows how to use various action qualifiers to control processing.

Operation

In this example, the program controls the sequenced starting and stopping of three conveyors as grain is unloaded from a ship and carried to two silos.



The following operation is performed:

1, 2, 3...

1. Grain type A is loaded into silo 1, and grain type B is loaded into silo 2.
2. When the grain level reaches the upper limit, the loading stops.

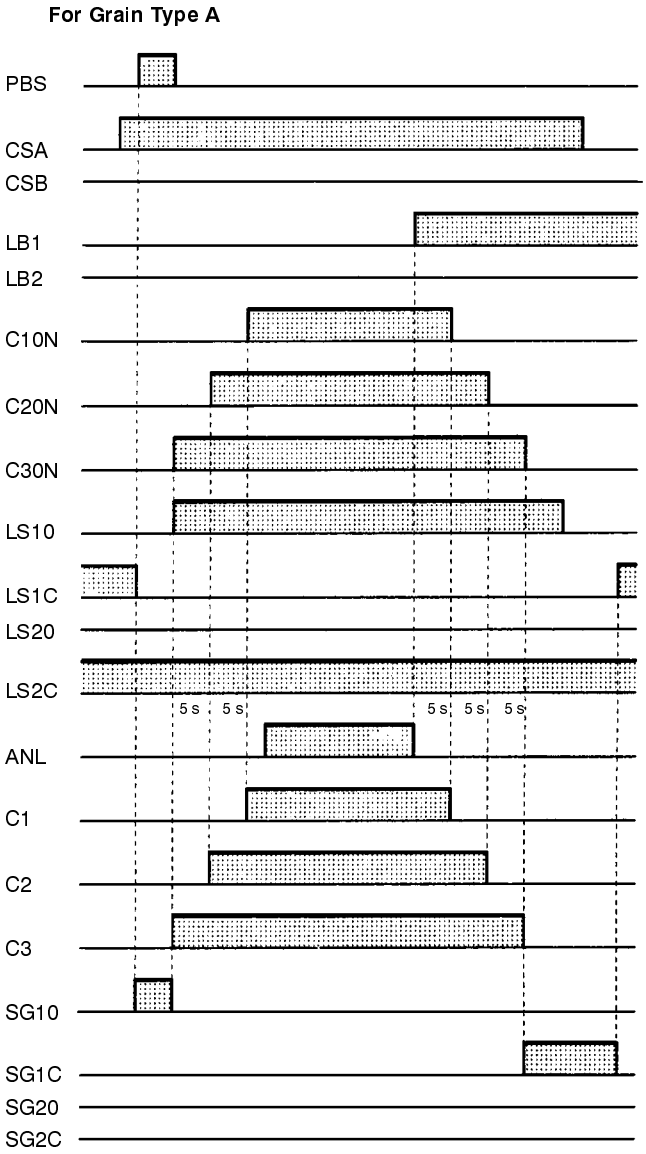
I/O Devices

I/O devices are allocated to I/O bits as shown below. I prefixes indicate inputs; Q prefixes, outputs. (These prefixes are used on CVSS displays.) The symbols are used in the program to indicate the bits that are controlled.

Name	Symbol	Allocation
Start switch	PBS	I003001
Grain type A setting switch	CSA	I003002
Grain type B setting switch	CSB	I003003
Upper limit, no. 1 silo	LB1	I003004
Upper limit, no. 2 silo	LB2	I003005
Run confirm signal, conveyor C1	C1ON	I003006
Run confirm signal, conveyor C2	C2ON	I003007
Run confirm signal, conveyor C3	C3ON	I003008
Slide gate SG1 open LS	LS1O	I003009
Slide gate SG1 closed LS	LS1C	I003010
Slide gate SG2 open LS	LS2O	I003011
Slide gate SG2 closed LS	LS2C	I003012
Pneumatic unloader operation	ANL	Q004001
Conveyor C1 operation	C1	Q004002
Conveyor C2 operation	C2	Q004003
Conveyor C3 operation	C3	Q004004
Slide gate SG1 open	SG1O	Q004005
Slide gate SG1 closed	SG1C	Q004006
Slide gate SG2 open	SG2O	Q004007
Slide gate SG2 closed	SG2C	Q004008

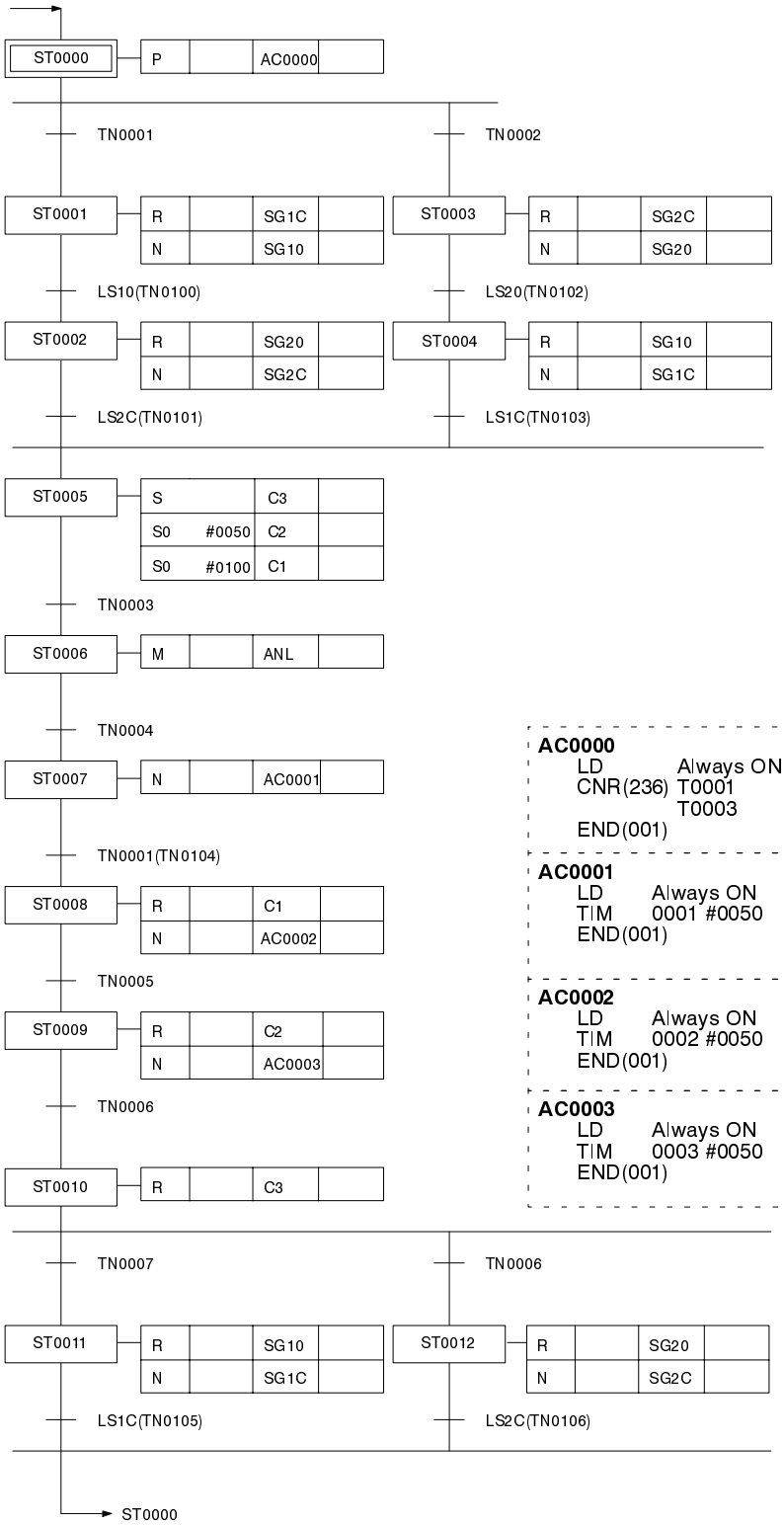
Time Chart

Operation timing is illustrated below.



SFC Program

The following program shows how the above operation can be programmed. A transition number is required even when a transition is defined as a bit. In the chart, these transitions are enclosed in parentheses. The action and transition programs are shown as mnemonic lists.



- ST0000 Initial step
- ST0001 Opens slide gate for grain type A: ON
- ST0002 Closes slide gate for other grain type (B): ON
- ST0003 Opens slide gate for grain type B: ON
- ST0004 Closes slide gate for other grain type (A): ON
- ST0005 Starts conveyors C3, C2, and C1 in that order.
- ST0006 Starts unloader
- ST0007 Stops unloader at upper limit level. Sets timer T0001 (5 s).
- ST0008 Stops conveyor C1 when time is up for timer T0001. Sets timer T0002 (5 s).
- ST0009 Stops conveyor C2 when time is up for timer T0002. Sets timer T0003 (5 s).
- ST0010 Stops conveyor C3 when time is up for timer T0003.
- ST0011 Closes slide gate SG1: ON
- ST0012 Closes slide gate SG2: ON

```
AC0000
    LD          Always ON
    CNR(236)    T0001
                T0003
    END(001)
```

```
AC0001
    LD      Always ON
    TIM     0001 #0050
    END(001)
```

```
AC0002
    LD      Always ON
    TIM     0002 #0050
    END(001)
```

```
AC0003
    LD      Always ON
    TIM     0003 #0050
    END(001)
```

```
TN0001
    LD          PBS
    AND         CSA
    AND NOT     LB1
    TOUT(202)
    END(001)
```

```
TN0002
    LD          PBS
    AND         CSB
    AND NOT     LB2
    TOUT(202)
    END(001)
```

```
TN0003
    LD      C3ON
    AND     C2ON
    AND     C1ON
    TOUT(202)
    END(001)
```

```
TN0004
    LD      CSA
    AND     LB1
    LD      CSB
    AND     LB2
    OR LD
    TOUT(202)
    END(001)
```

```
TN0005
    LD          T0002
    AND NOT     C1ON
    TOUT(202)
    END(001)
```

```
TN0006
    LD          T0003
    AND NOT     C2ON
    TOUT(202)
    END(001)
```

```
TN0007
    LD          CSA
    AND NOT     C3ON
    TOUT(202)
    END(001)
```

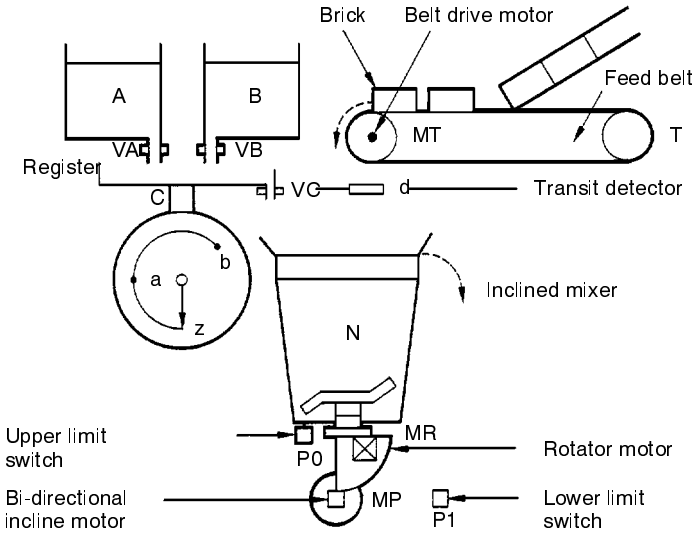
```
TN0008
    LD          CSB
    AND NOT     C3ON
    TOUT(202)
    END(001)
```

6–4 Example 4: Parallel Branching and Joining

This example shows how to use parallel processing.

Operation

In this example a mixer is controlled.



The following operation is performed:

1, 2, 3...

1. Register C measures fluid A to point “a” and fluid B from point “a” to point “b,” and then the fluid is poured into mixer N.
2. Two bricks are dropped into mixer N from conveyor T.
3. Mixer N is turned for fixed time t_1.
4. Mixer N is inclined and the composite material is poured out.

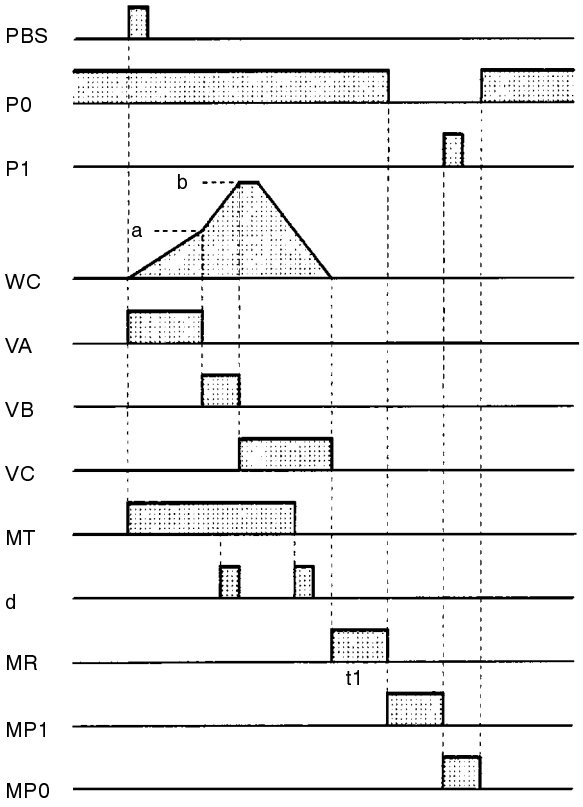
I/O Devices

I/O devices are allocated to I/O bits as shown below. I prefixes indicate inputs; Q prefixes, outputs. (These prefixes are used on CVSS displays.) The symbols are used in the program to indicate the bits that are controlled.

Name	Symbol	Allocation
Start switch	PBS	I004001
Brick detector	d	I004002
Mixer upper limit switch	P0	I004003
Mixer lower limit switch	P1	I004004
Volume data	WC	0050
Fluid A outlet valve	VA	Q006001
Fluid B outlet valve	VB	Q006002
Fluid outlet valve	VC	Q006003
Conveyor motor	MT	Q006004
Mixer rotator motor	MR	Q006005
Mixer incline right	MP1	Q006006
Mixer incline left	MP0	Q006007

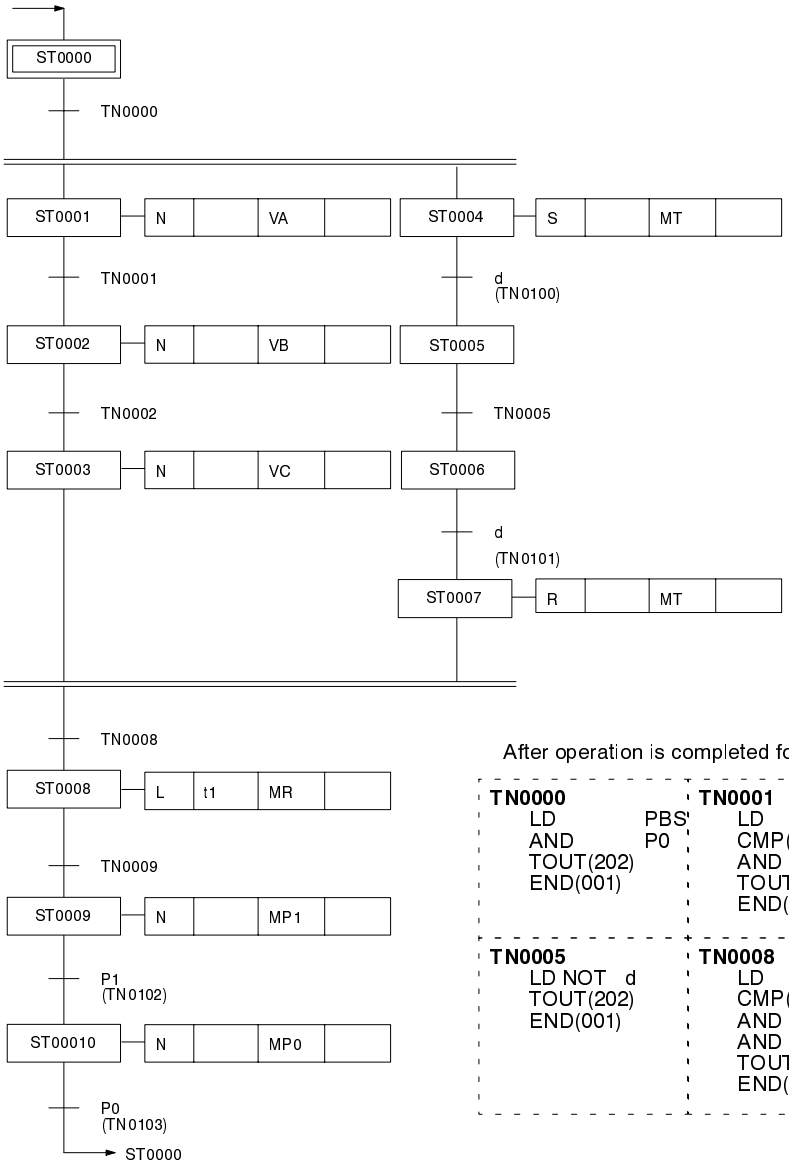
Time Chart

Operation timing is illustrated below.



SFC Programming

The following program shows how the above operation can be programmed. A transition number is required even when a transition is defined as a bit. In the chart, these transitions are enclosed in parentheses. The action and transition programs are shown as mnemonic lists.



- ST0000 Initial step
- ST0001 Opens outlet valve for fluid A.
- ST0004 Turns conveyor motor ON
- ST0002 Opens outlet valve for fluid B if weight is “a.”
- ST0005 Goes to next step upon brick detection.
- ST0003 Opens outlet valve for fluid C if weight is “b”.
- ST0006 Goes to next step when brick detection turns OFF.
- ST0007 Turns conveyor OFF upon second brick detection.
- ST0008 Mixes for t1 time when weight data goes to 0 and brick detection turns OFF.
- ST0009 Inclines mixer to right when mixing stops.
- ST0010 Returns mixer to upper limit after incline reaches lower limit.

After operation is completed for one cycle, it returns to the start and is repeated.

TN0000
```
LD          PBS
AND         P0
TOUT(202)
END(001)
```

TN0001
```
LD          Always ON
CMP(020)    WC a
AND NOT <   Flag
TOUT(202)
END(001)
```

TN0002
```
LD          Always ON
CMP(020)    WC b
AND NOT <   Flag
TOUT(202)
END(001)
```

TN0005
```
LD NOT      d
TOUT(202)
END(001)
```

TN0008
```
LD          Always ON
CMP(020)    WC 0
AND         = Flag
AND NOT     d
TOUT(202)
END(001)
```

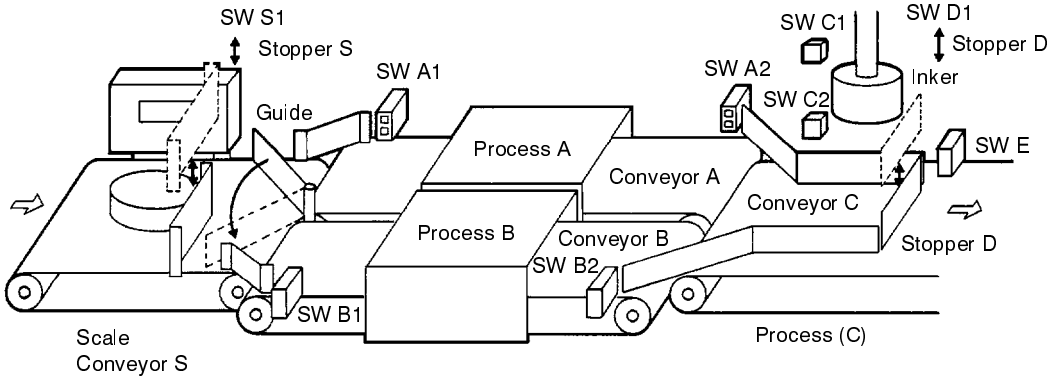
TN0009
```
LD NOT      MR
TOUT(202)
END(001)
```

6–5 Example 5: Ladder vs SFC

This example shows how to convert a program from ladder diagram form to SFC form. The application is described first; the programs and conversion are given later.

System

The program switches between two processes depending on the weight of the product. After being marked at the end, the products are moved on to the next process.



I/O Devices and I/O Bits

The following outlines the overall process and the allocation of I/O bits.

The overall process is started when input bit 000000 turns ON.

Scale

Conveyor S is started, stopper S is lowered, and the product is stopped at a fixed position.

The product is weighed.

Conveyor S is started. (output bit 000300)

Stopper S is lowered. (output bit 000200)

Stopper S is raised. (output bit 000201)

Stopper S reaches the upper limit SW S1. (input bit 000010)

The product is weighed. (output bit 00100)

The weighing result is output in grams. (input word 0004)

Guide

The weighing result is received and the products shunted to conveyor A or conveyor B.

Guide moved to conveyor A (output bit 000202)

or guide moved to conveyor B (output bit 000203)

Conveyor A started (output bit 000301)

or conveyor B started (output bit 000302)

SW A1

Detects when product passes. (input bit 000001)

SW B1

Detects when product passes. (input bit 000002)

SW A2

Detects when product passes. (input bit 000003)

SW B2

Detects when product passes. (input bit 000004)

Inker

Conveyor C is started, stopper D is lowered, and the product is stopped at a fixed position.

The product is marked.

Conveyor C is started. (output bit 000303)

Stopper D is lowered. (output bit 000204)

Stopper D is raised. (output bit 000205)

Slide moved down (output bit 000206)

Slide moved up (output bit 000207)

SW C1

Detects upper limit of inker slide. (input bit 000005)

SW C2

Detects lower limit of inker slide. (input bit 000006)

Stopper D reaches upper limit SW D1. (input bit 000011)

SW E

Detects when product passes. (input bit 000007)

Operation

The following operation is performed:

1, 2, 3...

1. When input bit 000000 turns ON, conveyor S is started and stopper S is lowered.
2. Conveyor S is run for 5 seconds, and the product is moved to the position of stopper S.
3. The scale is started and the product is weighed (with the results input to word 0004), and compared to a standard of 300 g.
4. If the weight is 300 g or more, the product is moved to process A. (Conveyor A is started.) If the weight is less than 300 g, the product is moved to process B. (Conveyor B is started.)
5. When either SW A2 or SW B2 turns ON, conveyor C is started and stopper D is lowered.
6. Conveyor C is run for 5 seconds, and the product is moved to the position of stopper D.
7. The inker is operated and the product is marked.
8. When the marking is complete, conveyor C is restarted and the product is carried off.

Program Conversion

This application is written in the form of a ladder diagram starting on the following page, and in the form of an SFC program on pages after the ladder diagram. We have written the same application both as a ladder program and as an SFC program, and we will compare the features of each.

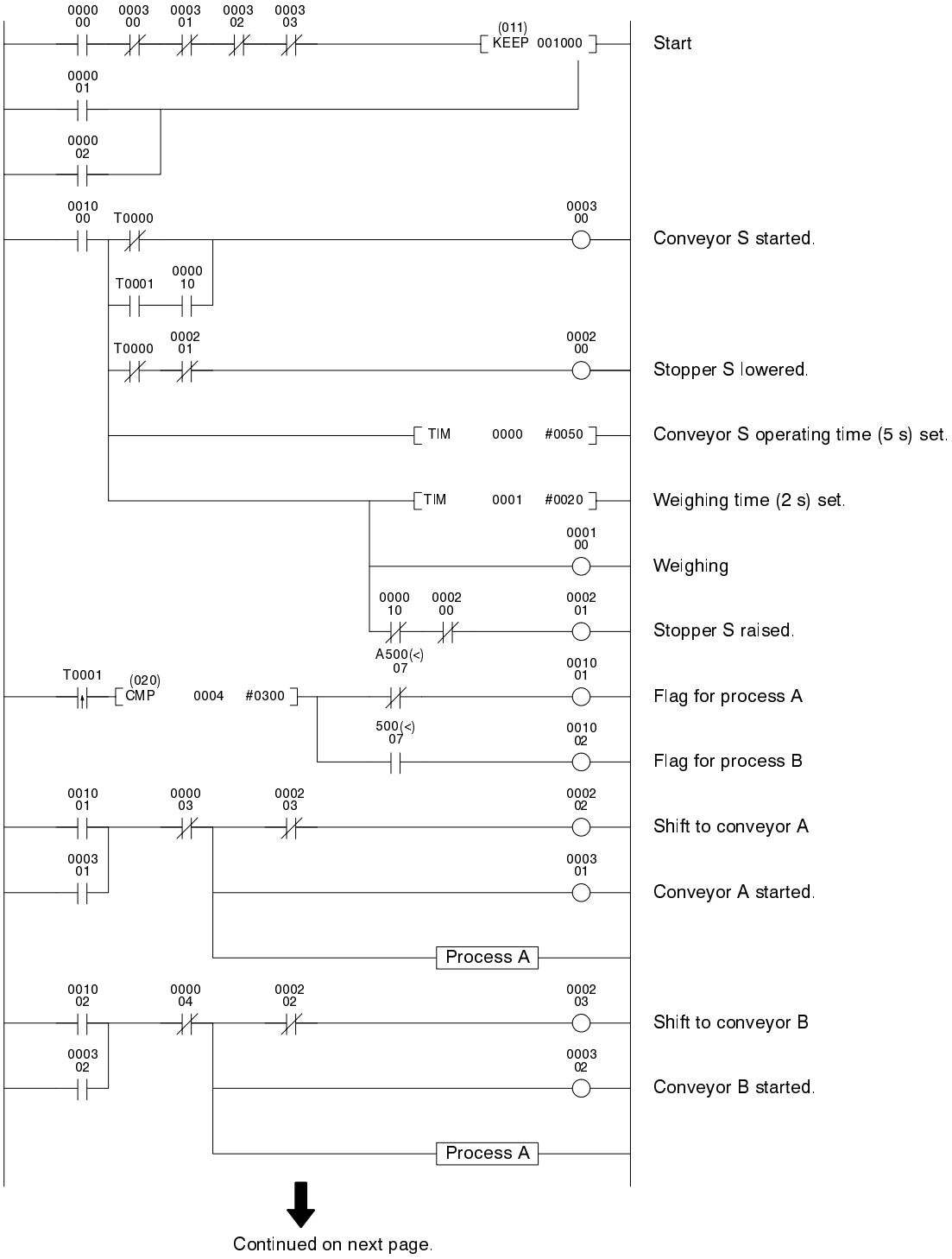
First of all, it is not easy to understand the sequence of processes in a program by looking at a ladder diagram. The larger a program gets, the more mutually exclusive conditions, such as interlocks, need to be programmed, and the more complicated programming becomes. In addition, when it comes to monitoring and maintenance, it is difficult to recognize the conditions for sequential processes merely by looking at a program list.

An SFC program, on the other hand, shows the flow of processes just as they are, making it quite simple to program sequential operations. Furthermore, each process in an SFC program stands independently, enabling programming without the need for setting complicated mutually exclusive conditions. And when it comes to monitoring and maintenance, you can follow the flow of processes simply by monitoring the active status of the steps. Debugging is also easy, because the program is broken down into discrete processes that have little influence on each other. It is thus a relatively simple matter to isolate and correct the problem area.

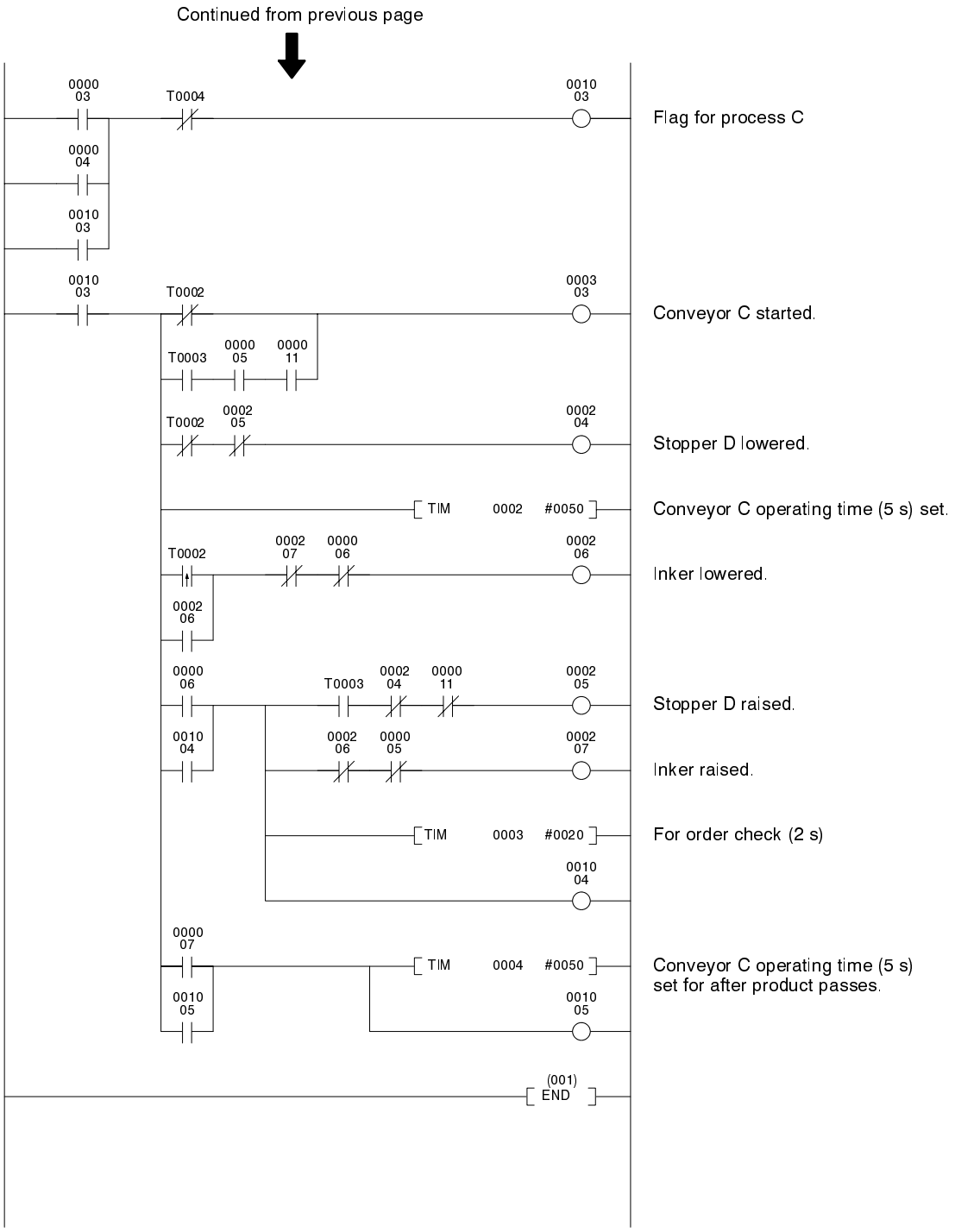
SFC programs are better suited to the overall control of processes than to the precise control of individual operations where, for example, conditions are

applied to each input and output. Ladder diagrams, on the other hand, are based on relay circuits and are thus ideal for detailed control. To realize the benefits of both methods, you can program the overall framework of processes with an SFC program, and control the details within each process with ladder diagram programs.

Ladder Diagram Program

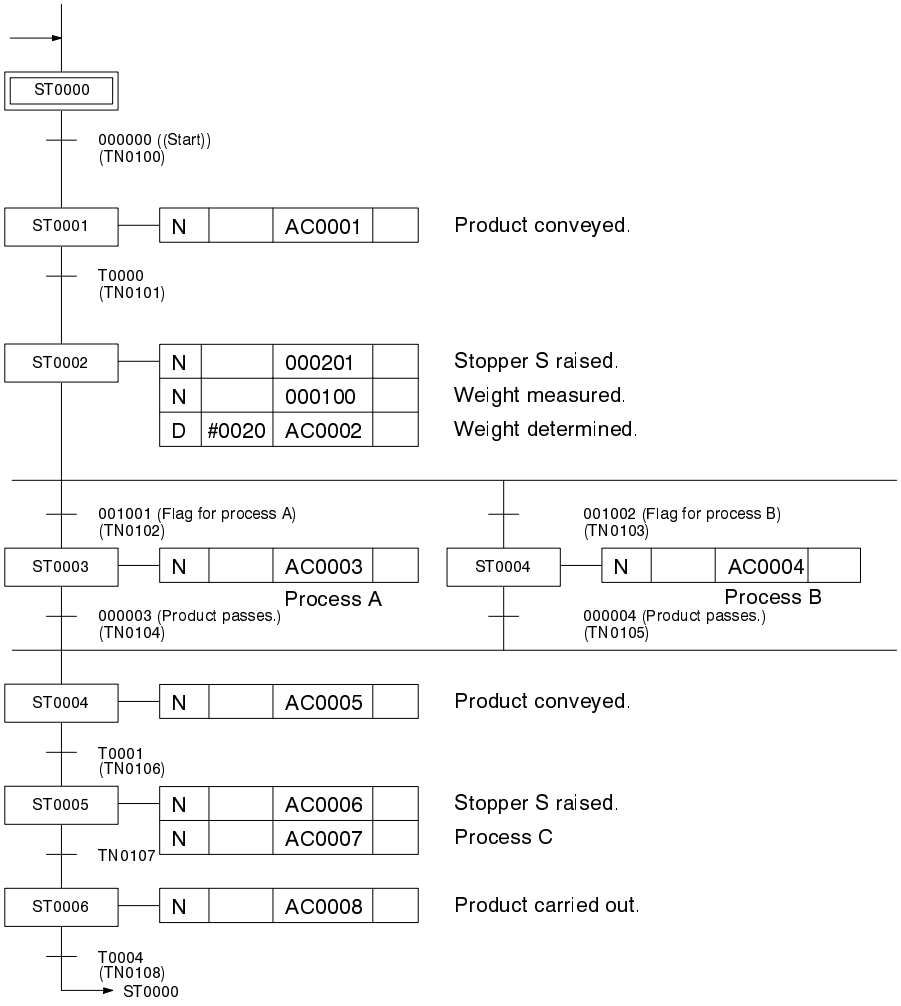


Continued on next page.


Continued from previous page
0000 03
T0004
0010 03
Flag for process C
0000 04
0010 03
0010 03
T0002
0003 03
Conveyor C started.
T0003
0000 05
0000 11
T0002
0002 05
0002 04
Stopper D lowered.
TIM 0002 #0050
Conveyor C operating time (5 s) set.
T0002
0002 07
0000 06
0002 06
Inker lowered.
0002 06
0000 06
T0003
0002 04
0000 11
0002 05
Stopper D raised.
0010 04
0002 06
0000 05
0002 07
Inker raised.
TIM 0003 #0020
For order check (2 s)
0010 04
0000 07
TIM 0004 #0050
Conveyor C operating time (5 s) set for after product passes.
0010 05
0010 05
(001) END

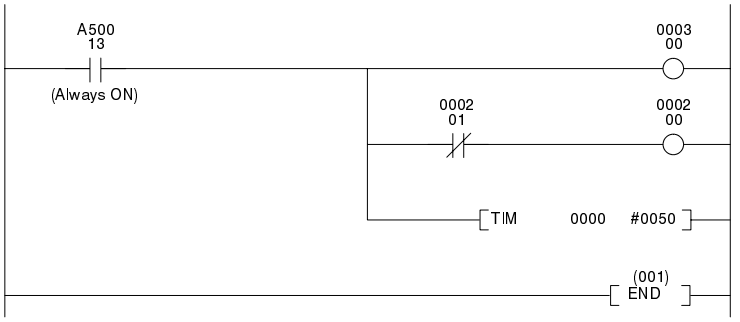
SFC Program

The same application is shown here as an SFC program, following the actual flow of processes.

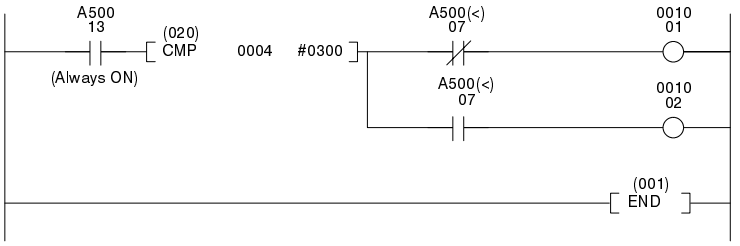


The number in the parentheses is the transition number for each bit used to define a transition.

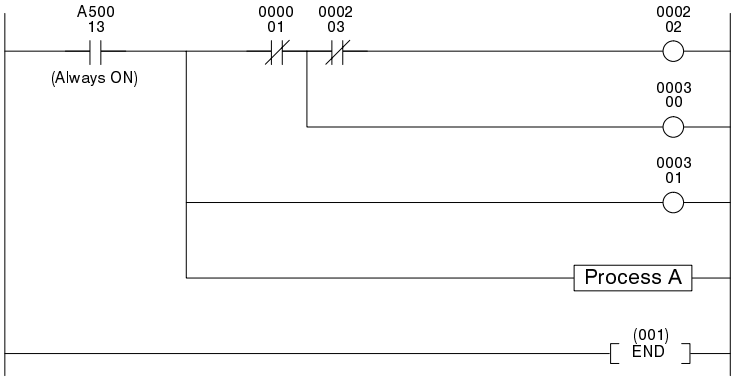
AC0001



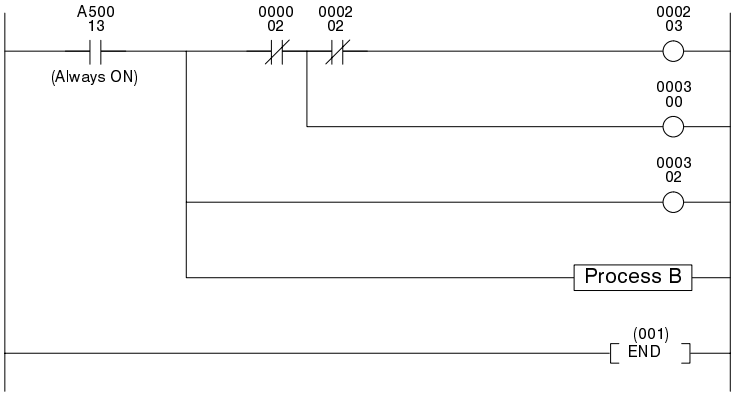
AC0002



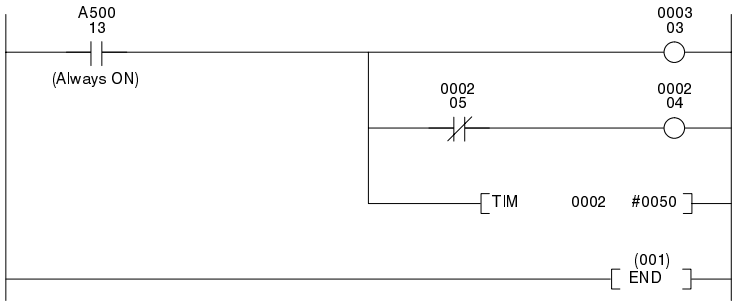
AC0003



AC0004



AC0005

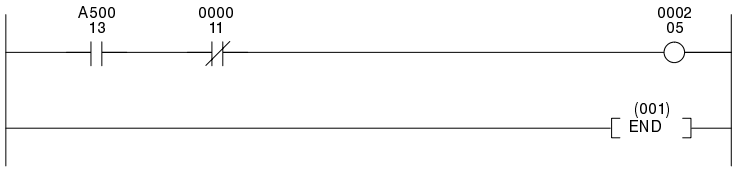


Conveyor C started.

Stopper D lowered.

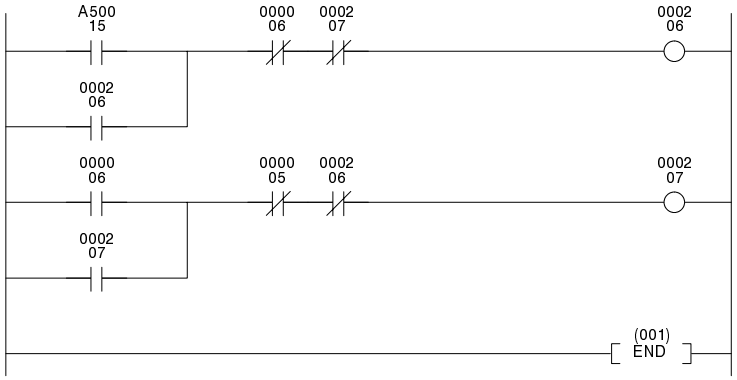
5 s counted.

AC0006



Stopper D raised.

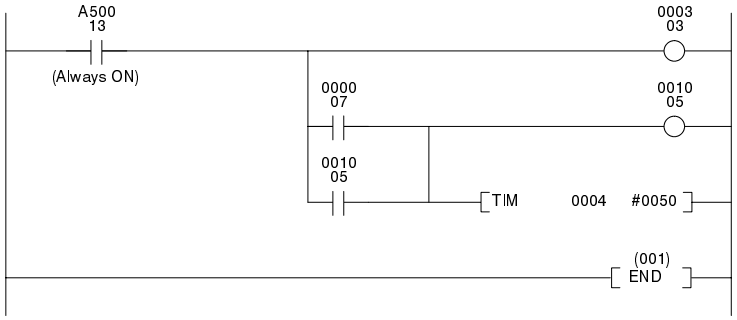
AC0007



Inker lowered.

Inker raised.

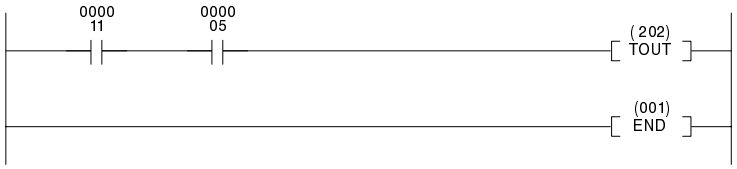
AC0008



Conveyor C started.

5 s counted after product passes.

TN0107



Glossary

address A number used to identify the location of data or programming instructions in memory or to identify the location of a network or a unit in a network.

advanced instruction An instruction input with a function code that handles data processing operations within ladder diagrams, as opposed to a basic instruction, which makes up the fundamental portion of a ladder diagram.

AGF All-glass optical fiber cable; also known as crystal optical fiber cable.

allocation The process by which the PC assigns certain bits or words in memory for various functions. This includes pairing I/O bits to I/O points on Units.

analog Something that represents or can process a continuous range of values as opposed to values that can be represented in distinct increments. Something that represents or can process values represented in distinct increments is called digital.

Analog I/O Unit I/O Units that convert I/O between analog and digital values. An Analog Input Input converts an analog input to a digital value for processing by the PC. An Analog Output Unit converts a digital value to an analog output.

AND A logic operation whereby the result is true if and only if both premises are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.

APF An acronym for all-plastic optical fiber cable.

area See *data area* and *memory area*.

area prefix A one or two letter prefix used to identify a memory area in the PC. All memory areas except the CIO area require prefixes to identify addresses in them.

ASCII Short for American Standard Code for Information Interchange. ASCII is used to code characters for output to printers and other external devices.

asynchronous execution Execution of programs and servicing operations in which program execution and servicing are not synchronized with each other.

Auxiliary Area A PC data area allocated to flags and control bits.

auxiliary bit A bit in the Auxiliary Area.

Backplane A base to which Units are mounted to form a Rack. Backplanes provide a series of connectors for these Units along with buses to connect them to the CPU and other Units and wiring to connect them to the Power Supply Unit. Backplanes also provide connectors used to connect them to other Backplanes.

back-up A copy made of existing data to ensure that the data will not be lost even if the original data is corrupted or erased.

BASIC A common programming language. BASIC Units are programmed in BASIC.

basic instruction A fundamental instruction used in a ladder diagram. See *advanced instruction*.

BASIC Unit A CPU Bus Unit used to run programs in BASIC.

baud rate The data transmission speed between two devices in a system measured in bits per second.

BCD Short for binary-coded decimal.

binary A number system where all numbers are expressed in base 2, i.e., numbers are written using only 0's and 1's. Each group of four binary bits is equivalent to one hexadecimal digit. Binary data in memory is thus often expressed in hexadecimal for convenience.

binary-coded decimal A system used to represent numbers so that every four binary bits is numerically equivalent to one decimal digit.

bit The smallest piece of information that can be represented on a computer. A bit has the value of either zero or one, corresponding to the electrical signals ON and OFF. A bit represents one binary digit. Some bits at particular addresses are allocated to special purposes, such as holding the status of input from external devices, while other bits are available for general use in programming.

bit address The location in memory where a bit of data is stored. A bit address specifies the data area and word that is being addressed as well as the number of the bit within the word.

branch line A communications line leading from a Link Adapter to any Link Unit not designated as a terminator in a Link System. See *main line*.

buffer A temporary storage space for data in a computerized device.

building-block PC A PC that is constructed from individual components, or "building blocks." With building-block PCs, there is no one Unit that is independently identifiable as a PC. The PC is rather a functional assembly of Units.

bus A communications path used to pass data between any of the Units connected to it.

bus link A data link that passed data between two Units across a bus.

byte A unit of data equivalent to 8 bits, i.e., half a word.

central processing unit A device that is capable of storing programs and data, and executing the instructions contained in the programs. In a PC System, the central processing unit executes the program, processes I/O signals, communicates with external devices, etc.

channel See *word*.

character code A numeric (usually binary) code used to represent an alphanumeric character.

checksum A sum transmitted with a data pack in communications. The checksum can be recalculated from the received data to confirm that the data in the transmission has not been corrupted.

CIO Area A memory area used to control I/O and to store and manipulate data. CIO Area addresses do not require prefixes.

communications cable Cable used to transfer data between components of a control system and conforming to the RS-232C or RS-422 standards.

constant An input for an operand in which the actual numeric value is specified. Constants can be input for certain operands in place of memory area addresses. Some operands must be input as constants.

control bit A bit in a memory area that is set either through the program or via a Programming Device to achieve a specific purpose, e.g., a Restart Bit is turned ON and OFF to restart a Unit.

control signal A signal sent from the PC to effect the operation of the controlled system.

Control System All of the hardware and software components used to control other devices. A Control System includes the PC System, the PC programs, and all I/O devices that are used to control or obtain feedback from the controlled system.

controlled system The devices that are being controlled by a PC System.

Converting Link Adapter A Link Adapter used to convert between different types of optical fiber cable, different types of wire cable, or between optical fiber cable and wire cable. Such conversion is necessary to connect Units that use different forms of communication.

CPU See *central processing unit*.

CPU Backplane A Backplane used to create a CPU Rack.

CPU Bus Unit A special Unit used with CV-series PCs that mounts to the CPU bus. This connection to the CPU bus enables special data links, data transfers, and processing.

CPU Rack The main Rack in a building-block PC, the CPU Rack contains the CPU, a Power Supply, and other Units. The CPU Rack, along with the Expansion CPU Rack, provides both an I/O bus and a CPU bus.

crystal optical fiber cable See *AGF.*

C-series PC Any of the following PCs: C2000H, C1000H, C500, C200H, C40H, C28H, C20H, C60K, C60P, C40K, C40P, C28K, C28P, C20K, C20P, C120, or C20.

CV Support Software A programming package run on an IBM PC/AT or compatible to serve as a Programming Device for CV-series PCs.

CV-series PC Any of the following PCs: CV500, CV1000, CV2000, or CVM1

CVSS See *CV Support Software*.

cycle One unit of processing performed by the CPU, including SFC/ladder program execution, peripheral servicing, I/O refreshing, etc. The cycle is called the scan with C-series PCs.

cycle time The time required to complete one cycle of CPU processing.

data area An area in the PC's memory that is designed to hold a specific type of data.

data link An automatic data transmission operation that allows PCs or Units within PC to pass data back and forth via common data areas.

data register A storage location in memory used to hold data. In CV-series PCs, data registers are used with or without index registers to hold data used in indirect addressing.

data transfer Moving data from one memory location to another, either within the same device or between different devices connected via a communications line or network.

debug A process by which a draft program is corrected until it operates as intended. Debugging includes both the removal of syntax errors, as well as the fine-tuning of timing and coordination of control operations.

decimal A number system where numbers are expressed to the base 10. In a PC all data is ultimately stored in binary form, four binary bits are often used to represent one decimal digit, via a system called binary-coded decimal.

decrement Decreasing a numeric value, usually by 1.

default A value automatically set by the PC when the user does not specifically set another value. Many devices will assume such default conditions upon the application of power.

destination The location where an instruction places the data on which it is operating, as opposed to the location from which data is taken for use in the instruction. The location from which data is taken is called the source.

digit A unit of storage in memory that consists of four bits.

DIN track A rail designed to fit into grooves on various devices to allow the devices to be quickly and easily mounted to it.

DIP switch Dual in-line package switch, an array of pins in a signal package that is mounted to a circuit board and is used to set operating parameters.

distributed control A automation concept in which control of each portion of an automated system is located near the devices actually being controlled, i.e., control is decentralized and 'distributed' over the system. Distributed control is a concept basic to PC Systems.

DM Area A data area used to hold only word data. Words in the DM area cannot be accessed bit by bit.

DM word A word in the DM Area.

downloading The process of transferring a program or data from a higher-level or host computer to a lower-level or slave computer. If a Programming Device is involved, the Programming Device is considered the host computer.

Dummy I/O Unit An I/O Unit that has no functional capabilities but that can be mounted to a slot on a Rack so that words can be allocated to that slot. Dummy I/O Units can be used to avoid changing operand addresses in programs by reserving words for a slot for future use or by filling a slot vacated by a Unit to which words have already been allocated.

EEPROM Electrically erasable programmable read-only memory; a type of ROM in which stored data can be erased and reprogrammed. This is accomplished using a special control lead connected to the EEPROM chip and can be done without having to remove the EEPROM chip from the device in which it is mounted.

electrical noise Random variations of one or more electrical characteristics such as voltage, current, and data, which might interfere with the normal operation of a device.

EM Area Extended Data Memory Area; an area that can be optionally added to certain PCs to enable greater data storage. Functionally, the EM Area operates like the

	DM Area. Area addresses are prefixes with E and only words can be accessed. The EM Area is separated into multiple banks.
EM card	A card mounted inside certain PCs to added an EM Area.
EPROM	Erasable programmable read-only memory; a type of ROM in which stored data can be erased, by ultraviolet light or other means, and reprogrammed.
error code	A numeric code generated to indicate that an error exists, and something about the nature of the error. Some error codes are generated by the system; others are defined in the program by the operator.
event processing	Processing that is performed in response to an event, e.g., an interrupt signal.
Expansion CPU Backplane	A Backplane used to create an Expansion CPU Rack.
Expansion CPU Rack	A Rack connected to the CPU Rack to increase the virtual size of the CPU Rack. Units that may be mounted to the CPU Backplane may also be mounted to the Expansion CPU Backplane.
Expansion Data Memory Unit	A card mounted inside certain PCs to added an EM Area.
Expansion I/O Backplane	A Backplane used to create an Expansion I/O Rack.
Expansion I/O Rack	A Rack used to increase the I/O capacity of a PC. In CV-Series PC, either one Expansion I/O Rack can be connected directly to the CPU or Expansion CPU Rack or multiple Expansion I/O Racks can be connected by using an I/O Control and I/O Interface Units.
FA	Factory automation.
factory computer	A general-purpose computer, usually quite similar to a business computer, that is used in automated factory control.
fatal error	An error that stops PC operation and requires correction before operation can continue.
FINS	See *CV-mode*.
flag	A dedicated bit in memory that is set by the system to indicate some type of operating status. Some flags, such as the carry flag, can also be set by the operator or via the program.
force reset	The process of forcibly turning OFF a bit via a programming device. Bits are usually turned OFF as a result of program execution.
force set	The process of forcibly turning ON a bit via a programming device. Bits are usually turned ON as a result of program execution.
frame checksum	The results of exclusive ORing all data within a specified calculation range. The frame checksum can be calculated on both the sending and receiving end of a data transfer to confirm that data was transmitted correctly.
GPC	An acronym for Graphic Programming Console.
Graphic Programming Console	A programming device with advanced programming and debugging capabilities to facilitate PC operation. A Graphic Programming Console is provided with a

large display onto which ladder-diagram programs can be written directly in ladder-diagram symbols for input into the PC without conversion to mnemonic form.

hexadecimal A number system where all numbers are expressed to the base 16. In a PC all data is ultimately stored in binary form, however, displays and inputs on Programming Devices are often expressed in hexadecimal to simplify operation. Each group of four binary bits is numerically equivalent to one hexadecimal digit.

host interface An interface that allows communications with a host computer.

Host Link System A system with one or more host computers connected to one or more PCs via Host Link Units or host interfaces so that the host computer can be used to transfer data to and from the PC(s). Host Link Systems enable centralized management and control of PC Systems.

Host Link Unit An interface used to connect a C-series PC to a host computer in a Host Link System.

H-PCF cable An acronym for hard plastic-clad optical fiber cable.

I/O allocation The process by which the PC assigns certain bits in memory for various functions. This includes pairing I/O bits to I/O points on Units.

I/O Block Either an Input Block or an Output Block. I/O Blocks provide mounting positions for replaceable relays.

I/O Control Unit A Unit mounted to the CPU Rack to monitor and control I/O points on Expansion CPU Racks or Expansion I/O Racks.

I/O delay The delay in time from when a signal is sent to an output to when the status of the output is actually in effect or the delay in time from when the status of an input changes until the signal indicating the change in the status is received.

I/O device A device connected to the I/O terminals on I/O Units, Special I/O Units, etc. I/O devices may be either part of the Control System, if they function to help control other devices, or they may be part of the controlled system.

I/O Interface Unit A Unit mounted to an Expansion CPU Rack or Expansion I/O Rack to interface the Rack to the CPU Rack.

I/O point The place at which an input signal enters the PC System, or at which an output signal leaves the PC System. In physical terms, I/O points correspond to terminals or connector pins on a Unit; in terms of programming, an I/O points correspond to I/O bits in the IR area.

I/O refreshing The process of updating output status sent to external devices so that it agrees with the status of output bits held in memory and of updating input bits in memory so that they agree with the status of inputs from external devices.

I/O response time The time required for an output signal to be sent from the PC in response to an input signal received from an external device.

I/O Terminal A Remote I/O Unit connected in a Wired Remote I/O System to provide a limited number of I/O points at one location. There are several types of I/O Terminals.

I/O Unit The most basic type of Unit mounted to a Backplane. I/O Units include Input Units and Output Units, each of which is available in a range of specifications. I/O Units do not include Special I/O Units, Link Units, etc.

I/O verification error A error generated by a disagreement between the Units registered in the I/O table and the Units actually mounted to the PC.

I/O word A word in the CIO area that is allocated to a Unit in the PC System and is used to hold I/O status for that Unit.

IBM PC/AT or compatible A computer that has similar architecture to, that is logically compatible with, and that can run software designed for an IBM PC/AT computer.

initialize Part of the startup process whereby some memory areas are cleared, system setup is checked, and default values are set.

input The signal coming from an external device into the PC. The term input is often used abstractly or collectively to refer to incoming signals.

input bit A bit in the CIO area that is allocated to hold the status of an input.

Input Block A Unit used in combination with a Remote Interface to create an I/O Terminal. An Input Block provides mounting positions for replaceable relays. Each relay can be selected according to specific input requirements.

input device An external device that sends signals into the PC System.

input point The point at which an input enters the PC System. Input points correspond physically to terminals or connector pins.

input signal A change in the status of a connection entering the PC. Generally an input signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.

Input Terminal An I/O Terminal that provides input points.

instruction A direction given in the program that tells the PC of the action to be carried out, and the data to be used in carrying out the action. Instructions can be used to simply turn a bit ON or OFF, or they can perform much more complex actions, such as converting and/or transferring large blocks of data.

interface An interface is the conceptual boundary between systems or devices and usually involves changes in the way the communicated data is represented. Interface devices such as NSBs perform operations like changing the coding, format, or speed of the data.

interrupt (signal) A signal that stops normal program execution and causes a subroutine to be run or other processing to take place.

Interrupt Input Unit A Rack-mounting Unit used to input external interrupts into a PC System.

IOIF An acronym for I/O Interface Unit.

IOM (Area) A collective memory area containing all of the memory areas that can be accessed by bit, including timer and counter Completion Flags. The IOM Area includes all memory area memory addresses between 0000 and 0FFF.

JIS An acronym for Japanese Industrial Standards.

jump A type of programming where execution moves directly from one point in a program to another, without sequentially executing any instructions in between.

Jumps in ladder diagrams are usually conditional on an execution condition; jumps in SFC programs are conditional on the step status and transition condition status before the jump.

least-significant (bit/word) See *rightmost (bit/word)*.

LED Acronym for light-emitting diode; a device used as for indicators or displays.

leftmost (bit/word) The highest numbered bits of a group of bits, generally of an entire word, or the highest numbered words of a group of words. These bits/words are often called most-significant bits/words.

link A hardware or software connection formed between two Units. “Link” can refer either to a part of the physical connection between two Units or a software connection created to data existing at another location (i.e., data links).

Link Adapter A Unit used to connect communications lines, either to branch the lines or to convert between different types of cable. There are two types of Link Adapter: Branching Link Adapters and Converting Link Adapters.

Link System A system used to connect remote I/O or to connect multiple PCs in a network. Link Systems include the following: SYSMAC BUS Remote I/O Systems, SYSMAC BUS/2 Remote I/O Systems, SYSMAC LINK Systems, Host Link Systems, and SYSMAC NET Link Systems.

Link Unit Any of the Units used to connect a PC to a Link System. These include Remote I/O Units, SYSMAC LINK Units, and SYSMAC NET Link Units.

linkable slot A slot on either a Backplane to which a Link Unit can be mounted. Backplanes differ in the slots to which Link Units can be mounted.

load The processes of copying data either from an external device or from a storage area to an active portion of the system such as a display buffer. Also, an output device connected to the PC is called a load.

main line In a Link System connected through Branching Link Adapters, the communications cable that runs from the Unit at each end of the System through the Link Adapters.

MCR Unit Magnetic Card Reader Unit.

megabyte A unit of storage equal to one million bytes.

memory area Any of the areas in the PC used to hold data or programs.

most-significant (bit/word) See *leftmost (bit/word).*

nesting Programming one loop within another loop, programming a call to a subroutine within another subroutine, or programming an IF–ELSE programming section within another IF–ELSE section.

Network Service Board A device with an interface to connect devices other than PCs to a SYSMAC NET Link System.

Network Service Unit A Unit that provides two interfaces to connect peripheral devices to a SYSMAC NET Link System.

noise interference Disturbances in signals caused by electrical noise.

nonfatal error A hardware or software error that produces a warning but does not stop the PC from operating.

NOT A logic operation which inverts the status of the operand. For example, AND NOT indicates an AND operation with the opposite of the actual status of the operand bit.

octal A number system where all numbers are expressed in base 8, i.e., numbers are written using only numerals 0 through 7.

OFF The status of an input or output when a signal is said not to be present. The OFF state is generally represented by a low voltage or by non-conductivity, but can be defined as the opposite of either.

OFF delay The delay between the time when a signal is switched OFF (e.g., by an input device or PC) and the time when the signal reaches a state readable as an OFF signal (i.e., as no signal) by a receiving party (e.g., output device or PC).

offset A positive or negative value added to a base value such as an address to specify a desired value.

ON The status of an input or output when a signal is said to be present. The ON state is generally represented by a high voltage or by conductivity, but can be defined as the opposite of either.

ON delay The delay between the time when an ON signal is initiated (e.g., by an input device or PC) and the time when the signal reaches a state readable as an ON signal by a receiving party (e.g., output device or PC).

on-line removal Removing a Rack-mounted Unit for replacement or maintenance during PC operation.

operand The values designated as the data to be used for an instruction. An operand can be input as a constant expressing the actual numeric value to be used or as an address to express the location in memory of the data to be used.

operating error An error that occurs during actual PC operation as opposed to an initialization error, which occurs before actual operations can begin.

optical connector A connector designed to be connected to an optical fiber cable.

optical fiber cable Cable made from light conducting filaments used to transmit signals.

OR A logic operation whereby the result is true if either of two premises is true, or if both are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.

output The signal sent from the PC to an external device. The term output is often used abstractly or collectively to refer to outgoing signals.

Output Block A Unit used in combination with a Remote Interface to create an I/O Terminal. An Output Block provides mounting positions for replaceable relays. Each relay can be selected according to specific output requirements.

output device An external device that receives signals from the PC System.

output point The point at which an output leaves the PC System. Output points correspond physically to terminals or connector pins.

output signal A signal being sent to an external device. Generally an output signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.

Output Terminal An I/O Terminal that provides output points.

overflow The state where the capacity of a data storage location has been exceeded.

overwrite Changing the content of a memory location so that the previous content is lost.

parity Adjustment of the number of ON bits in a word or other unit of data so that the total is always an even number or always an odd number. Parity is generally used to check the accuracy of data after being transmitted by confirming that the number of ON bits is still even or still odd.

parity check Checking parity to ensure that transmitted data has not been corrupted.

PC An acronym for Programmable Controller.

PC configuration The arrangement and interconnections of the Units that are put together to form a functional PC.

PC System With building-block PCs, all of the Racks and independent Units connected directly to them up to, but not including the I/O devices. The boundaries of a PC System are the PC and the program in its CPU at the upper end; and the I/O Units, Special I/O Units, Optical I/O Units, Remote Terminals, etc., at the lower end.

PCB An acronym for printed circuit board.

PCF An acronym for plastic-clad optical fiber cable.

PC Setup A group of operating parameters set in the PC from a Programming Device to control PC operation.

Peripheral Device Devices connected to a PC System to aid in system operation. Peripheral devices include printers, programming devices, external storage media, etc.

peripheral servicing Processing signals to and from peripheral devices, including refreshing, communications processing, interrupts, etc.

PID Unit A Unit designed for PID control.

port A connector on a PC or computer that serves as a connection to an external device.

Power Supply Unit A Unit that mounts to a Backplane in a Rack PC. It provides power at the voltage required by the other Units on the Rack.

present value The current value registered in a device at any instant during its operation. Present value is abbreviated as PV. The use of this term is generally restricted to timers and counters.

printed circuit board A board onto which electrical circuits are printed for mounting into a computer or electrical device.

Printer Interface Unit A Unit used to interface a printer so that ladder diagrams and other data can be printed out.

Programmable Controller A computerized device that can accept inputs from external devices and generate outputs to external devices according to a program held in memory. Programmable Controllers are used to automate control of external devices. Although single-unit Programmable Controllers are available, building-block Programmable Controllers are constructed from separate components. Such Programmable Controllers are formed only when enough of these separate components are assembled to form a functional assembly, i.e., there is no one individual Unit called a PC.

Programming Console The simplest form or programming device available for a PC. Programming Consoles are available both as hand-held models and as CPU-mounting models.

Programming Device A Peripheral Device used to input a program into a PC or to alter or monitor a program already held in the PC. There are dedicated programming devices, such as Programming Consoles, and there are non-dedicated devices, such as a host computer.

PROM Programmable read-only memory; a type of ROM into which the program or data may be written after manufacture, by a customer, but which is fixed from that time on.

PROM Writer A peripheral device used to write programs and other data into a ROM for permanent storage and application.

prompt A message or symbol that appears on a display to request input from the operator.

protocol The parameters and procedures that are standardized to enable two devices to communicate or to enable a programmer or operator to communicate with a device.

PV See *present value*.

Rack An assembly that forms a functional unit in a Rack PC System. A Rack consists of a Backplane and the Units mounted to it. These Units include the Power Supply, CPU, and I/O Units. Racks include CPU Racks, Expansion I/O Racks, and I/O Racks. The CPU Rack is the Rack with the CPU mounted to it. An Expansion I/O Rack is an additional Rack that holds extra I/O Units. An I/O Rack is used in the C2000H Duplex System, because there is no room for any I/O Units on the CPU Rack in this System.

rack number A number assigned to a Rack according to the order that it is connected to the CPU Rack, with the CPU Rack generally being rack number 0.

Rack PC A PC that is composed of Units mounted to one or more Racks. This configuration is the most flexible, and most large PCs are Rack PCs. A Rack PC is the opposite of a Package-type PC, which has all of the basic I/O, storage, and control functions built into a single package.

RAM Random access memory; a data storage media. RAM will not retain data when power is disconnected.

RAS An acronym for reliability, assurance, safety.

refresh The process of updating output status sent to external devices so that it agrees with the status of output bits held in memory and of updating input bits in memory so that they agree with the status of inputs from external devices.

relay-based control The forerunner of PCs. In relay-based control, groups of relays are interconnected to form control circuits. In a PC, these are replaced by programmable circuits.

reserved bit A bit that is not available for user application.

reserved word A word in memory that is reserved for a special purpose and cannot be accessed by the user.

reset The process of turning a bit or signal OFF or of changing the present value of a timer or counter to its set value or to zero.

Restart Bit A bit used to restart a Unit mounted to a PC.

restart continuation A process which allows memory and program execution status to be maintained so that PC operation can be restarted from the state it was in when operation was stopped by a power interruption.

retrieve The processes of copying data either from an external device or from a storage area to an active portion of the system such as a display buffer. Also, an output device connected to the PC is called a load.

retry The process whereby a device will re-transmit data which has resulted in an error message from the receiving device.

rightmost (bit/word) The lowest numbered bits of a group of bits, generally of an entire word, or the lowest numbered words of a group of words. These bits/words are often called least-significant bits/words.

rising edge The point where a signal actually changes from an OFF to an ON status.

ROM Read only memory; a type of digital storage that cannot be written to. A ROM chip is manufactured with its program or data already stored in it and can never be changed. However, the program or data can be read as many times as desired.

RS-232C interface An industry standard for serial communications.

RS-422 interface An industry standard for serial communications.

scan The process used to execute a ladder-diagram program. The program is examined sequentially from start to finish and each instruction is executed in turn based on execution conditions. The scan also includes peripheral processing, I/O refreshing, etc. The scan is called the cycle with CV-series PCs.

scan time The time required for a single scan of a ladder-diagram program.

self diagnosis A process whereby the system checks its own operation and generates a warning or error if an abnormality is discovered.

series A wiring method in which Units are wired consecutively in a string. In Link Systems wired through Link Adapters, the Units are still functionally wired in series, even though Units are placed on branch lines.

servicing The process whereby the PC provides data to or receives data from external devices or remote I/O Units, or otherwise handles data transactions for Link Systems.

set The process of turning a bit or signal ON.

set value The value from which a decrementing counter starts counting down or to which an incrementing counter counts up (i.e., the maximum count), or the time from which or for which a timer starts timing. Set value is abbreviated SV.

slot A position on a Rack (Backplane) to which a Unit can be mounted.

software error An error that originates in a software program.

software protect A means of protecting data from being changed that uses software as opposed to a physical switch or other hardware setting.

software switch See *memory switch*.

Special I/O Unit A Unit that is designed for a specific purpose. Special I/O Units include Position Control Units, High-speed Counter Units, Analog I/O Units, etc.

SRAM Static random access memory; a data storage media.

subroutine A group of instructions placed separate from the main program and executed only when called from the main program or activated by an interrupt.

SV Abbreviation for set value.

switching capacity The maximum voltage/current that a relay can safely switch on and off.

synchronous execution Execution of programs and servicing operations in which program execution and servicing are synchronized so that all servicing operations are executed each time the programs are executed.

syntax The form of a program statement (as opposed to its meaning). For example, the two statements, `LET A=B+B` and `LET A=B*2` use different syntaxes, but have the same meaning.

syntax error An error in the way in which a program is written. Syntax errors can include 'spelling' mistakes (i.e., a function code that does not exist), mistakes in specifying operands within acceptable parameters (e.g., specifying read-only bits as a destination), and mistakes in actual application of instructions (e.g., a call to a subroutine that does not exist).

system configuration The arrangement in which Units in a System are connected. This term refers to the conceptual arrangement and wiring together of all the devices needed to comprise the System. In OMRON terminology, system configuration is used to describe the arrangement and connection of the Units comprising a Control System that includes one or more PCs.

system error An error generated by the system, as opposed to one resulting from execution of an instruction designed to generate an error.

system error message An error message generated by the system, as opposed to one resulting from execution of an instruction designed to generate a message.

terminator The code comprising an asterisk and a carriage return (* CR) which indicates the end of a block of data in communications between devices. Frames within a multi-frame block are separated by delimiters. Also a Unit in a Link System designated as the last Unit on the communications line.

timer A location in memory accessed through a TC bit and used to time down from the timer's set value. Timers are turned ON and reset according to their execution conditions.

TR Area A data area used to store execution conditions so that they can be reloaded later for use with other instructions.

TR bit A bit in the TR Area.

transfer The process of moving data from one location to another within the PC, or between the PC and external devices. When data is transferred, generally a copy of the data is sent to the destination, i.e., the content of the source of the transfer is not changed.

transmission distance The distance that a signal can be transmitted.

UM area The memory area used to hold the active program, i.e., the program that is being currently executed.

Unit In OMRON PC terminology, the word Unit is capitalized to indicate any product sold for a PC System. Though most of the names of these products end with the word Unit, not all do, e.g., a Remote Terminal is referred to in a collective sense as a Unit. Context generally makes any limitations of this word clear.

unit address A number used to control network communications. Unit addresses are computed for Units in various ways, e.g., 10 hex is added to the unit number to determine the unit address for a CPU Bus Unit.

unit number A number assigned to some Link Units, Special I/O Units, and CPU Bus Units to facilitate identification when assigning words or other operating parameters.

uploading The process of transferring a program or data from a lower-level or slave computer to a higher-level or host computer. If a Programming Devices is involved, the Programming Device is considered the host computer.

watchdog timer A timer within the system that ensures that the scan time stays within specified limits. When limits are reached, either warnings are given or PC operation is stopped depending on the particular limit that is reached.

WDT See *watchdog timer*.

wire communications A communications method in which signals are sent over wire cable. Although noise resistance and transmission distance can sometimes be a problem with wire communications, they are still the cheapest and the most common, and perfectly adequate for many applications.

word A unit of data storage in memory that consists of 16 bits. All data areas consists of words. Some data areas can be accessed only by words; others, by either words or bits.

word address The location in memory where a word of data is stored. A word address must specify (sometimes by default) the data area and the number of the word that is being addressed.

word allocation The process of assigning I/O words and bits in memory to I/O Units and terminals in a PC System to create an I/O Table.

work area A part of memory containing work words/bits.

work bit A bit in a work word.

work word A word that can be used for data calculation or other manipulation in programming, i.e., a 'work space' in memory. A large portion of the IR area is always reserved for work words. Parts of other areas not required for special purposes may also be used as work words.

write protect switch A switch used to write-protect the contents of a storage device, e.g., a floppy disk. If the hole on the upper left of a floppy disk is open, the information on this floppy disk cannot be altered.

write-protect A state in which the contents of a storage device can be read but cannot be altered.

Index

A

action block, 9, 20, **28**
  action execution timing, 33
  action number, 9
  action qualifier. *See* AQ
  action reset timing, 34
  actions, 29
  feedback variable, 29
  ladder diagram program, 29
  set value, 29

action qualifier. *See* AQ

AQ, 12, 28, **29**
  evaluation, 68
  example, 30, 84
  execution cycle, 35
  hold option, 31
  operation (table), 29

B

bit, transition, 27

block, action. *See* action block

branch
  conditional, 9, 11, 12, 20, **36**
    common errors, 38
    example, 81
    number of branches, 13
  parallel, 10, 11, 14, 21, **38**
    common errors, 40
    example, 88
    number of branches, 14

C

concurrent execution. *See* execution cycle, multiple active steps

conditional execution. *See* branch, conditional

control instruction, 11, 54
  ladder symbol, 54
  operand data area, 54
  table, 49

CVSS, 8

cycle time, 2, 74

D

debugging, 2, 29

dummy step, 21, 28
  See also subchart dummy step

E

error codes
  fatal, 76
  non-fatal, 76

execution, parallel. *See* branch, parallel

execution cycle, **68**
  AQ, 35
  multiple active steps, 68
  power−up, 71
  restart continuation, 71
  subcharts, 69

F

feedback variable, 29

First Cycle flag, 26

flag
  Execution Error(A40211), 76
  Fatal Error(A40107), 76
  First Cycle, 26
  transition, 26

FV. *See* feedback variable

H

hold option (of AQ), 31

I

IF−THEN. *See* branch, conditional

interrupt
  I/O, 22
  power−off, 22
  power−on, 22
  precautions, 45
  program, **45**
    and SFC control instructions, 52
    common errors, 46
  scheduled, 22

J

join
  conditional, 10, 12, 21, **36**
    common errors, 38
    example, 81
  parallel, 10, 14, 21, **39**
    common errors, 40
    example, 88
    maximum number of branches, 15

jump, 11, 22, **43**
  common errors, 44

jump entry, 22

L

ladder diagram
  action block program, 29
  compared to SFC program, 3
  control instruction symbol, 54
  converting to SFC, 91
  transition program, 27

M

memory areas, 17

multiple active steps, 68

O

operand data area, control instruction, 54

P

parallel execution, 68

power−on execution, 71

program
  action block, 29
  interrupt. *See* interrupt program
  transition, 27

programming devices, 17

Q

qualifier, action. *See* AQ

R

restart continuation, 71, 73

S

SA(210), 49, **55**

SE(214), 49, **59**

sequential control, example, 80

Sequential Function Chart. *See* SFC

SF(213), 49, **58**

SFC, 2, **8**
  monitoring, 17
  programming, 17

SFC program
  compared to ladder diagram program, 3
  creating, 18
  method of input, 8

sheet, 8

simultaneous execution. *See* branch, parallel; multiple active steps

SOFF(215), 49, **60**

SP(211), 49, **56**

SR(212), 49, **57**

status, 8, 23
  execute, 48
  halt, 48
  inactive, 48
  pause, 48
  step. *See* step status
  transfer, 8, **15**, 68

step, 2, 8, 20, **23**
  controlling, 48
  dummy, 21, 23
  initial, 8, 20, **24**
  non−existent, 16
  number, 8
  sequential, 11, 12
  status, 8, 48
  synchronous, 11

subchart, 11, **40**, 51
  common errors, 43
  control instructions, 51
  dummy step, 21
  entry terminal, 21
  nesting, 40
  precautions, 42
  return terminal, 21

subroutine. *See* subchart

T

TCNT(123), 49, **62**

time
  cycle. *See* cycle time
  user processing, 74

timing, action execution, 33

TOUT(202), 49, **61**

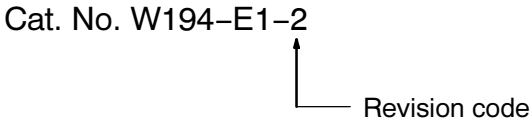
transition, 8, 20, **26**
  bit, 27
  condition, 8
  evaluation, 68
  flag, 26
  number, 9, 26
  program, 27

TSR(124), 49, **63**

TSW(125), 49, **64**

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.



The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	February 1991	Original production
2	May 1993	CV2000 added. **Page 26:** Note added on using transition programs.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP