

Chapter Contents

3. STL Programming	3-1
3.1 What is STL, SFC And IEC1131 Part 3?	3-1
3.2 How STL Operates	3-2
3.2.1 Each step is a program	3-2
3.3 How To Start And End An STL Program	3-3
3.3.1 Embedded STL programs	3-3
3.3.2 Activating new states	3-3
3.3.3 Terminating an STL Program	3-4
3.4 Moving Between STL Steps	3-5
3.4.1 Using SET to drive an STL coil	3-5
3.4.2 Using OUT to drive an STL coil	3-6
3.5 Rules and Techniques For STL programs	3-7
3.5.1 Basic Notes On The Behavior Of STL programs	3-7
3.5.2 Single Signal Step Control	3-9
3.6 Restrictions Of Some Instructions When Used With STL	3-10
3.7 Using STL To Select The Most Appropriate Program	3-11
3.8 Using STL To Activate Multiple Flows Simultaneously	3-12
3.9 General Rules For Successful STL Branching	3-14
3.10 General Precautions When Using The FX-PCS/AT-EE Software	3-15
3.11 Programming Examples	3-16
3.11.1 A Simple STL Flow	3-16
3.11.2 A Selective Branch/ First State Merge Example Program	3-18
3.12 Advanced STL Use	3-20

3. STL Programming

FX1S

FX1N

FX2N

FX2NC

This chapter differs from the rest of the contents in this manual as it has been written with a training aspect in mind. STL/SFC programming, although having been available for many years, is still misunderstood and misrepresented. We at Mitsubishi would like to take this opportunity to try to correct this oversight as we see STL/SFC programming becoming as important as ladder style programming.

3.1 What is STL, SFC And IEC1131 Part 3?

The following explanation is very brief but is designed to quickly outline the differences and similarities between STL, SFC and IEC1131 part 3.

In recent years Sequential Function Chart (or SFC) style programming (including other similar styles such as Grafcet and Funktionplan) have become very popular through out Europe and have prompted the creation of IEC1131 part 3.

The IEC1131 SFC standard has been designed to become an interchangeable programming language. The idea being that a program written to IEC1131 SFC standards on one manufacturers PLC can be easily transferred (converted) for use on a second manufacturers PLC.

STL programming is one of the basic programming instructions included in all FX PLC family members. The abbreviation STL actually means S**T**ep Ladder programming.

STL programming is a very simple concept to understand yet can provide the user with one of the most powerful programming techniques possible. The key to STL lies in its ability to allow the programmer to create an operational program which 'flows' and works in almost exactly the same manner as SFC. This is not a coincidence as this programming technique has been developed deliberately to achieve an easy to program and monitor system.

One of the key differences to Mitsubishi's STL programming system is that it can be entered into a PLC in 3 formats. These are:

- I) Instruction - a word/mnemonic entry system
- II) Ladder - a graphical program construction method using a relay logic symbols
- III) SFC - a flow chart style of STL program entry (similar to SFC)

Examples of these programming methods can be seen on page 2-1.



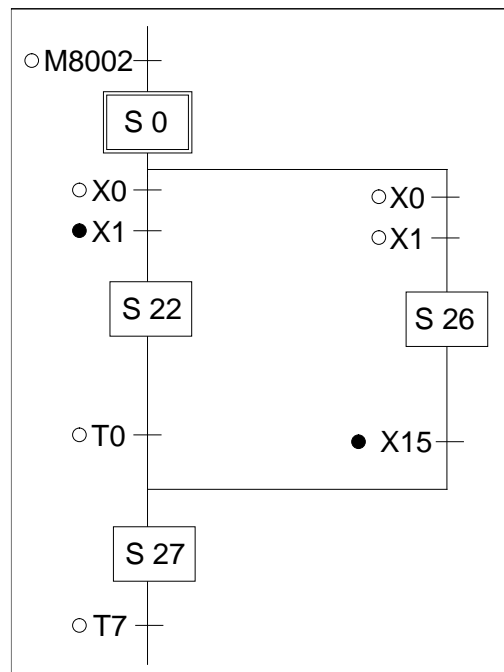
General note:

- IEC1131-3: 03.1993 Programmable controllers; part 3: programming languages.
The above standard is technically identical to the 'Euro-Norm'
EN61131-3: 07.1993

3.2 How STL Operates

As previously mentioned, STL is a system which allows the user to write a program which functions in much the same way as a flow chart, this can be seen in the diagram opposite.

STL derives its strength by organizing a larger program into smaller more manageable parts. Each of these parts can be referred to as either a state or a step. To help identify the states, each is given a unique identification number. These numbers are taken from the state relay devices (see page 4-6 for more details).



3.2.1 Each step is a program

Each state is completely isolated from all other states within the whole program. A good way to envisage this, is that each state is a separate program and the user puts each of those programs together in the order that they require to perform their task. Immediately this means that states can be reused many times and in different orders. This saves on programming time AND cuts down on the number of programming errors encountered.

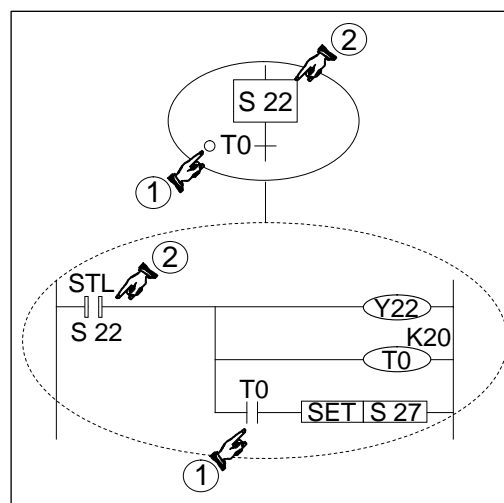
A Look Inside an STL

On initial inspection the STL program looks as if it is a rather basic flow diagram. But to find out what is really happening the STL state needs to be put 'under a microscope' so to speak. When a single state is examined in more detail, the sub-program can be viewed.

With the exception of the STL instruction, it will be immediately seen that the STL sub-program looks just like ordinary programming.

- ① The STL instruction is shown as a 'fat' normally open contact.
- All programming after an STL instruction is only active when the associated state coil is active.
- ② The transition condition is also written using standard programming.

This idea re-enforces the concept that STL is really a method of sequencing a series of events or as mentioned earlier 'of joining lots of smaller programs together'.



Combined SFC Ladder representation

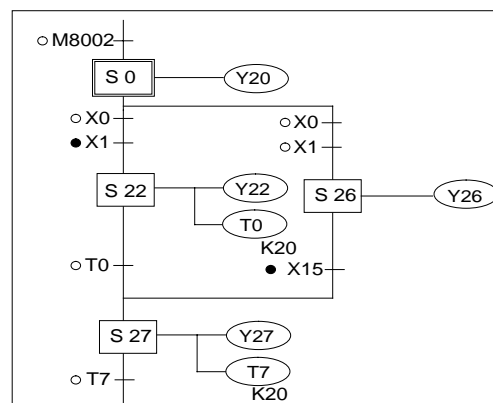
Sometimes STL programs will be written in hard copy as a combination of both flow diagram and internal sub-program. (example shown below).

Identification of contact states



- Please note the following convention is used:
 - Normally Open contact
 - Normally Closed contact

Common alternatives are 'a' and 'b' identifiers for Normally Open, Normally Closed states or often a line drawn over the top of the Normally Closed contact name is used, e.g. X000.

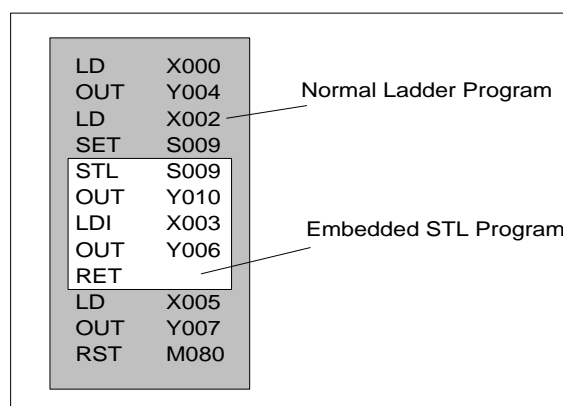


3.3 How To Start And End An STL Program

Before any complex programming can be undertaken the basics of how to start and more importantly how to finish an STL program need to be examined.

3.3.1 Embedded STL programs

An STL style program does not have to entirely replace a standard ladder logic program. In fact it might be very difficult to do so. Instead small or even large section of STL program can be entered at any point in a program. Once the STL task has been completed the program must go back to processing standard program instructions until the next STL program block. Therefore, identifying the start and end of an STL program is very important.

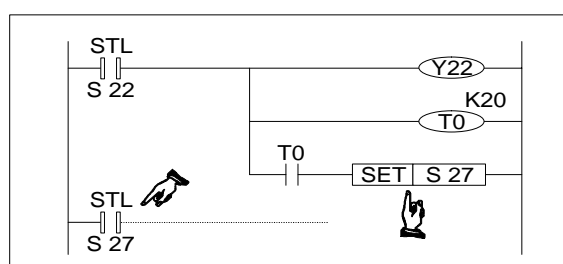


3.3.2 Activating new states

Once an STL step has been selected, how is it used and how is the program 'driven'?

This is not so difficult, if it is considered that for an STL step to be active its associated state coil must be ON. Hence, to start an STL sequence all that has to be done is to drive the relevant state ON.

There are many different methods to drive a state, for example the initial state coils could be pulsed, SET or just included in an OUT instruction. However, within Mitsubishi's STL programming language an STL coil which is SET has a different meaning than one that is included in an OUT instruction.

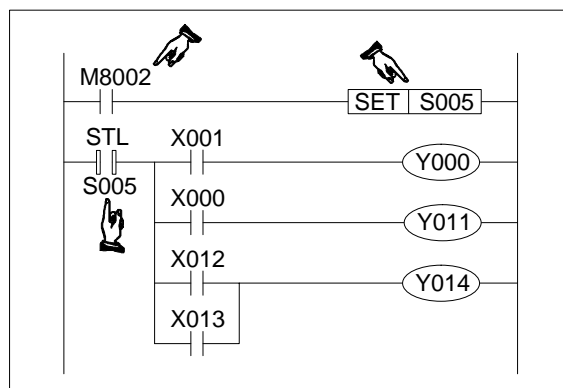


Note: For normal STL operation it is recommended that the states are selected using the SET instruction. To activate an STL step its state coil is SET ON.

Initial Steps

For an STL program which is to be activated on the initial power up of the PLC, a trigger similar to that shown opposite could be used, i.e. using M8002 to drive the setting of the initial state.

The STL step started in this manner is often referred to as the initial step. Similarly, the step activated first for any STL sequence is also called the initial step.



3.3.3 Terminating an STL Program

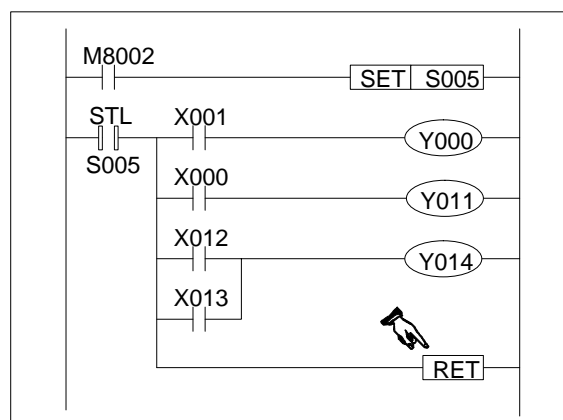
Once an STL program has been started the programmable controllers CPU will process all following instructions as being part of that STL program. This means that when a second program scan is started the normal instructions at the beginning of the program are considered to be within the STL program. This is obviously incorrect and the CPU will proceed to identify a programming error and disable the programmable controllers operation.

This scenario may seem a little strange but it does make sense when it is considered that the STL program must return control to the ladder program after STL operation is complete. This means the last step in an STL program needs to be identified in some way.

Returning to Standard Ladder

This is achieved by placing a RET or RETURN instruction as the last instruction in the last STL step of an STL program block.

This instruction then returns programming control to the ladder sequence.



Note: The RET instruction can be used to separate STL programs into sections, with standard ladder between each STL program. For display of STL in SFC style format the RET instruction is used to indicate the end of a complete STL program.

3.4 Moving Between STL Steps

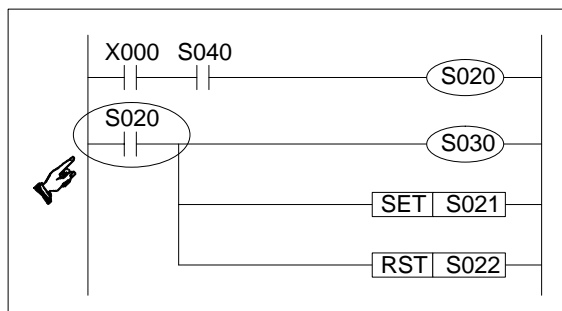
To activate an STL step the user must first drive the state coil. Setting the coil has already been identified as a way to start an STL program, i.e. drive an initial state. It was also noted that using an OUT statement to driving a state coil has a different meaning to the SET instruction. These difference will now be explained:

3.4.1 Using SET to drive an STL coil

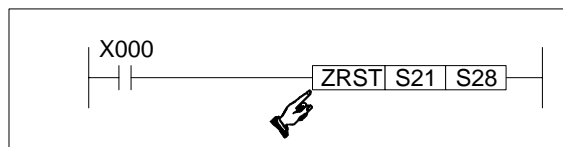
- SET is used to drive an STL state coil to make the step active. Once the current STL step activates a second following step, the source STL coil is reset. Hence, although SET is used to activate a state the resetting is automatic.

However, if an STL state is driven by a series of standard ladder logic instructions, i.e. not a preceding STL state, then standard programming rules apply.

In the example shown opposite S20 is not reset even after S30 or S21 have been driven. In addition, if S20 is turned OFF, S30 will also stop operating. This is because S20 has not been used as an STL state. The first instruction involving the status of S20 is a standard Load instruction and NOT an STL instruction.



Note: If a user wishes to forcibly reset an STL step, using the RST or ZRST (FNC 40) instructions would perform this task.



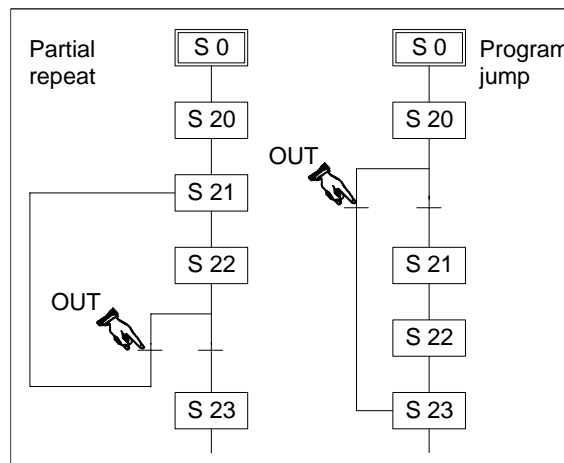
- SET is used to drive an immediately following STL step which typically will have a larger STL state number than the current step.
- SET is used to drive STL states which occur within the enclosed STL program flow, i.e. SET is not used to activate a state which appears in an unconnected, second STL flow diagram.

3.4.2 Using OUT to drive an STL coil

This has the same operational features as using SET. However, there is one major function which SET is not used. This is to make what is termed 'distant jumps'.

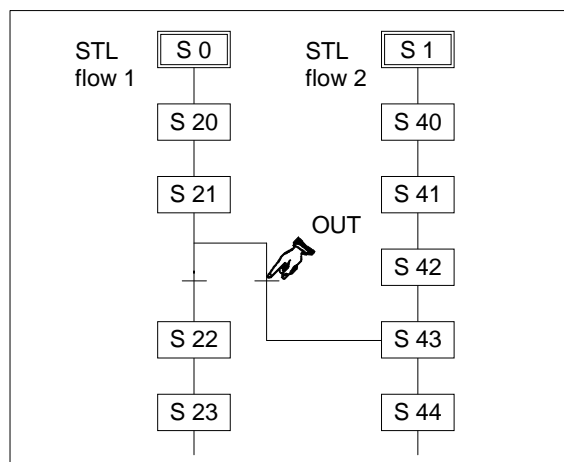
OUT is used for loops and jumps

If a user wishes to 'jump' back up a program, i.e. go back to a state which has already been processed, the OUT instruction would be used with the appropriate STL state number. Alternatively the user may wish to make a large 'jump' forwards skipping a whole section of STL programmed states.



Out is used for distant jumps

If a step in one STL program flow was required to trigger a step in a second, separate STL program flow the OUT instruction would be used.



Note: Although it is possible to use SET for jumps and loops use of OUT is needed for display of STL in SFC like structured format.

3.5 Rules and Techniques For STL programs

It can be seen that there are a lot of advantages to using STL style programming but there are a few points a user must be aware of when writing the STL sub-programs. These are highlighted in this section.

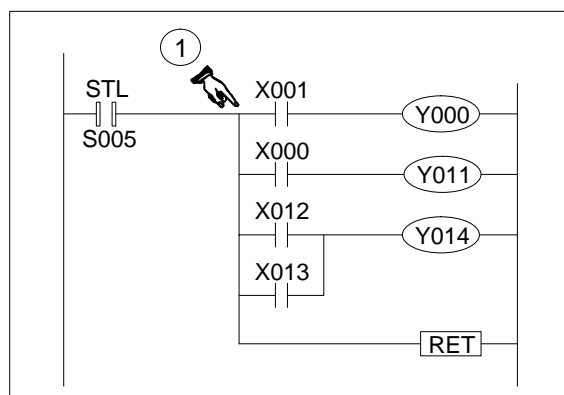
3.5.1 Basic Notes On The Behavior Of STL programs

- When an STL state becomes active its program is processed until the next step is triggered. The contents of the program can contain all of the programming items and features of a standard ladder program, i.e. Load, AND OR, OUT, ReSeT etc., as well as applied instructions.
- When writing the sub-program of an STL state, the first vertical 'bus bar' after the STL instruction can be considered in a similar manner as the left hand bus bar of a standard ladder program.

Each STL step makes its own bus bar. This means that a user, cannot use an MPS instruction directly after the STL instruction (see ①), i.e. There needs to be at least a single contact before the MPS instruction.



Note: Using out coils and even applied instructions immediately after an STL instruction is permitted.



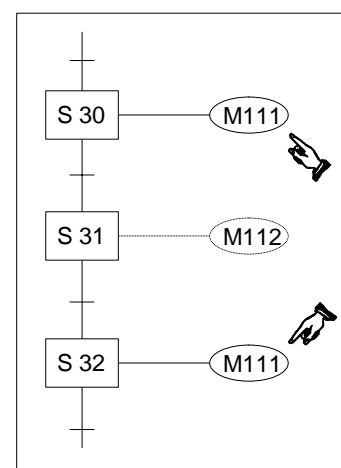
- In normal programming using dual coils is not an acceptable technique. However repetition of a coil in separate STL program blocks is allowed.

This is because the user can take advantage of the STL's unique feature of isolating all STL steps except the active STL steps.

This means in practice that there will be no conflict between dual coils. The example opposite shows M111 used twice in a single STL flow.

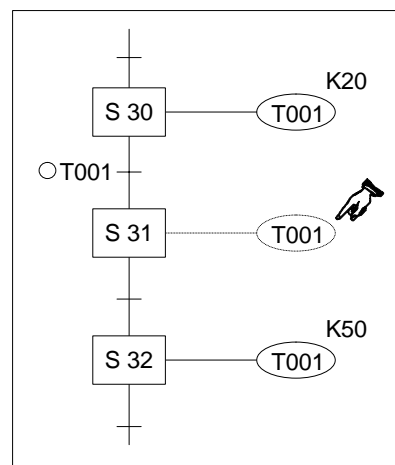


Caution: The same coil should NOT be programmed in steps that will be active at the same time as this will result in the same problem as other dual coils.





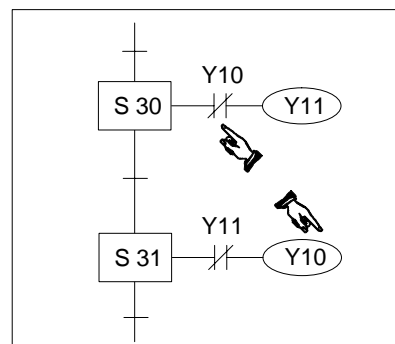
- When an STL step transfers control to the next STL step there is a period (one scan) while both steps are active. This can cause problems with dual coils; particularly timers.
If timers are dual coiled care must be taken to ensure that the timer operation is completed during the active STL step.
If the same timer is used in consecutive steps then it is possible that the timer coil is never deactivated and the contacts of the timer will not be reset leading to incorrect timer operation.
The example opposite identifies an unacceptable use of timer T001. When control passes from S30 to S31 T001 is not reset because its coil is still ON in the new step.



Note: As a step towards ensuring the correct operation of the dual timers they should not be used in consecutive STL steps.
Following this simple rule will ensure each timer will be reset correctly before its next operation.



- As already mentioned, during the transfer between steps, the current and the selected steps will be simultaneously active for one program scan. This could be thought of as a hand over or handshaking period.
This means that if a user has two outputs contained in consecutive steps which must NOT be active simultaneously they must be interlocked. A good example of this would be the drive signals to select a motors rotation direction. In the example Y11 and Y10 are shown interlocked with each other.



3.5.2 Single Signal Step Control

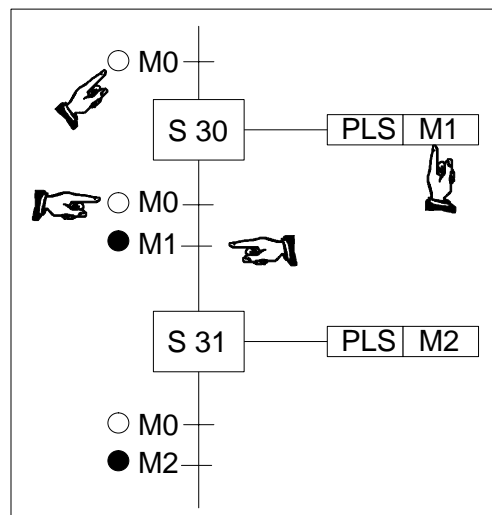
Transferring between active STL steps can be controlled by a single signal. There are two methods the user can program to achieve this result.

Method 1 - Using locking devices

FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

In this example it is necessary to program separate locking devices, and the controlling signal must only pulse ON. This is to prevent the STL programs from running through. The example shown below identifies the general program required for this method.

- S30 is activated when M0 is first pulsed ON.
- The operation of M1 prevents the sequence from continuing because although M0 is ON, the transfer requirements, need M0 to be ON and M1 to be OFF.
- After one scan the pulsed M0 and the 'lock' device M1 are reset.
- On the next pulse of M0 the STL step will transfer program control from S30 to the next step in a similar manner. This time using M2 as the 'lock' device because dual coils in successive steps is not allowed.
- The reason for the use of the 'lock' devices M1 and M2 is because of the handshaking period when both states involved in the transfer of program control are ON for 1 program scan. Without the 'locks' it would be possible to immediately skip through all of the STL states in one go!



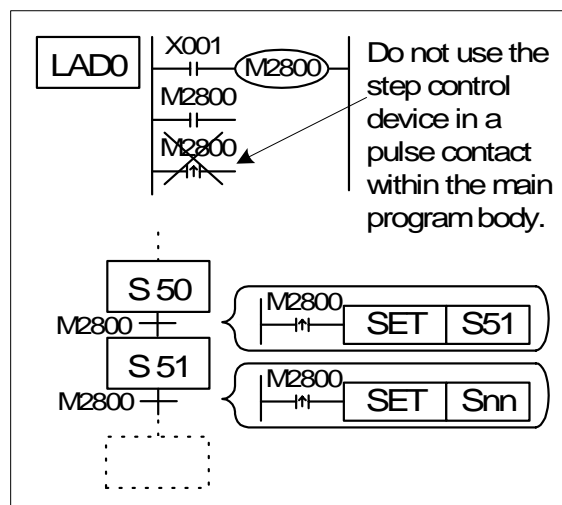
Method 2 - Special Single Pulse Flags

FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

Using the pulse contacts (LDP, LDF, ANP, etc.) and a special range of M devices (M2800 to M3071) the same result as method 1 can be achieved. The special feature of these devices prevents run through of the states, as only the first occurrence of the LDP instruction will activate.

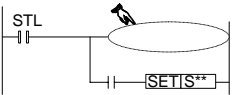
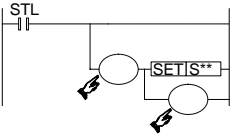
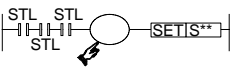
The example program below shows the necessary instructions.

- Assume S50 is already active.
- When X01 activates M2800, this in turn activates the LDP M2800 instruction in S50 and the flow moves on to step S51.
- The LDP M2800 instruction in the transition part of S51 does not execute because this is the second occurrence of M2800 in a pulse contact.
- When X01 next activates M2800, the LDP instruction in S51 is the first occurrence because S50 is now inactive. Thus, control passes to the next step in the same manner.



3.6 Restrictions Of Some Instructions When Used With STL

Although STL can operate with most basic and applied instructions there are a few exceptions. As a general rule STL and MC-MCR programming formats should not be combined. Other instruction restrictions are listed in the table below.

Operational State		Basic Instructions		
		LD, LDI, AND, ANI, OR, ORI, NOP, OUT, SET, RST, PLS, PLF	ANB, ORB, MPS, MRD, MPP	MC, MCR
Initial and general states		✓	✓	✗
Branching and merging states	Output processing 	✓	✓	✗
	Transfer processing 	✓	✗	✗

Restrictions on using applied instructions



- Most applied instructions can be used within STL programs. Attention must be paid to the way STL isolates each non-active step. It is recommended that when applied instructions are used their operation is completed before the active STL step transfers to the next step.

Other restrictions are as follows:

- FOR - NEXT structures can not contain STL program blocks.
- Subroutines and interrupts can not contain STL program blocks.
- STL program blocks can not be written after an FEND instruction.
- FOR - NEXT instructions are allowed within an STL program with a nesting of up to 4 levels.



For more details please see the operational compatibility listed in the two tables on pages 7-12, 7-13.



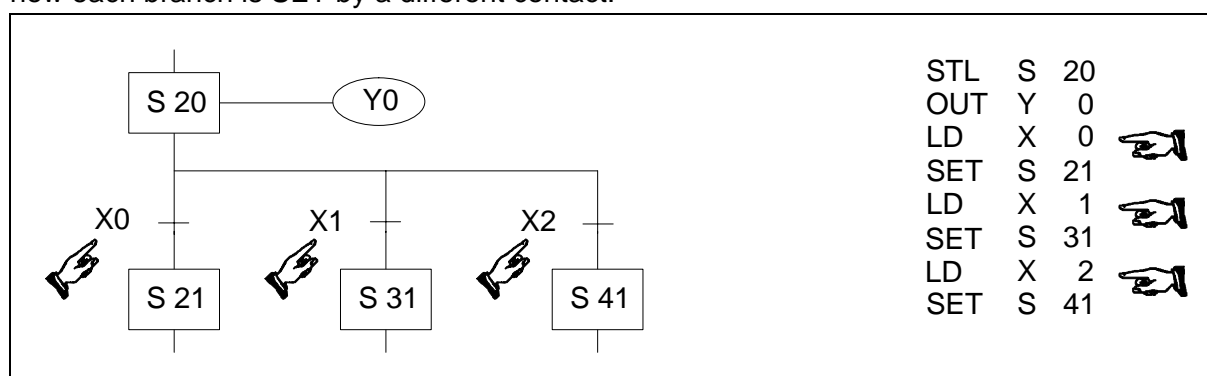
Using 'jump' operations with STL

- Although it is possible to use the program jump operations (CJ instruction) within STL program flows, this causes additional and often unnecessary program flow complications. To ensure easy maintenance and quick error finding it is recommended that users do not write jump instructions into their STL programs.

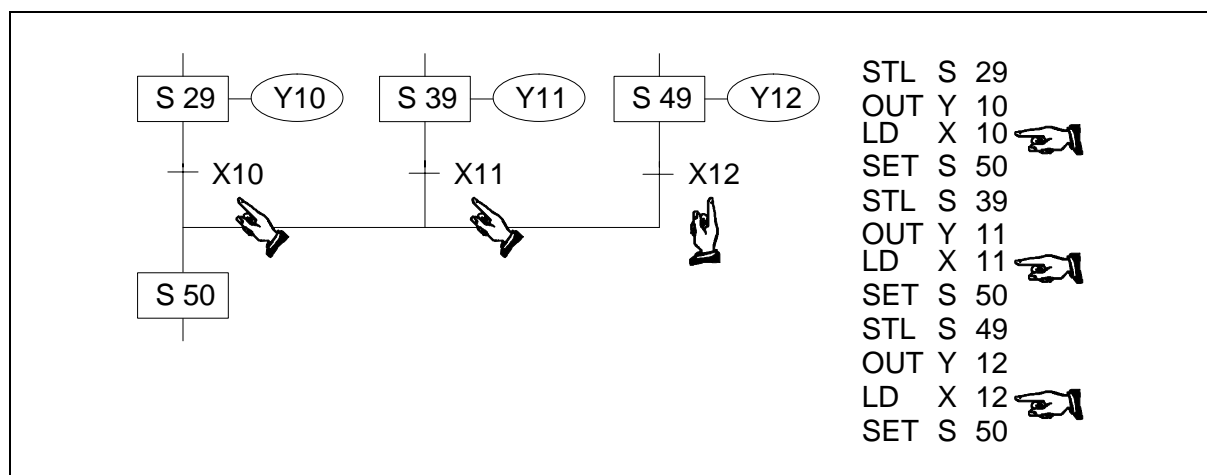
3.7 Using STL To Select The Most Appropriate Program

FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

So far STL has been considered as a simple flow charting programming language. One of STL's exceptional features is the ability to create programs which can have several operating modes. For example certain machines require a selection of 'manual' and 'automatic' modes, other machines may need the ability to select the operation or manufacturing processes required to produce products 'A', 'B', 'C', or 'D'. STL achieves this by allowing multiple program branches to originate from one STL state. Each branch is then programmed as an individual operating mode, and because each operating mode should act individually, i.e. there should be no other modes active; the selection of the program branch must be mutually exclusive. This type of program construction is called "Selective Branch Programming". An example instruction program can be seen below, (this is the sub-program for STL state S20 only) notice how each branch is SET by a different contact.



A programming construction to split the program flow between different branches is very useful but it would be more useful if it could be used with a method to rejoin a set of individual branches.



This type of STL program construction is called a "First State Merge" simply because the first state (in the example S29, S39 or S49) to complete its operation will cause the merging state (S50) to be activated. It should be noticed how each of the final STL states on the different program branches call the same "joining" STL state.



Limits on the number of branches

- Please see page 3-14 for general notes on programming STL branches.

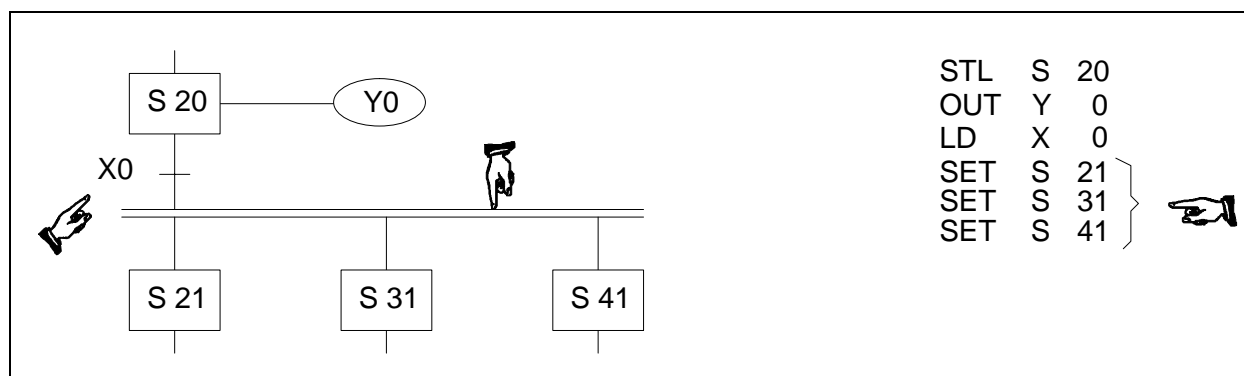
Notes on using the FX-PCS/AT-EE software

- Please see page 3-15 for precautions when using the FX-PCS-AT/EE software.

3.8 Using STL To Activate Multiple Flows Simultaneously

FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

In the previous branching technique, it was seen how a single flow could be selected from a group. The following methods describe how a group of individual flows can be activated simultaneously. Applications could include vending machines which have to perform several tasks at once, e.g. boiling water, adding different taste ingredients (coffee, tea, milk, sugar) etc. In the example below when state S20 is active and X0 is then switched ON, states S21, S31 and S41 are ALL SET ON as the next states. Hence, three separate, individual, branch flows are 'set in motion' from a single branch point. This programming technique is often called a 'Parallel Branch'. To aid a quick visual distinction, parallel branches are marked with horizontal, parallel lines.

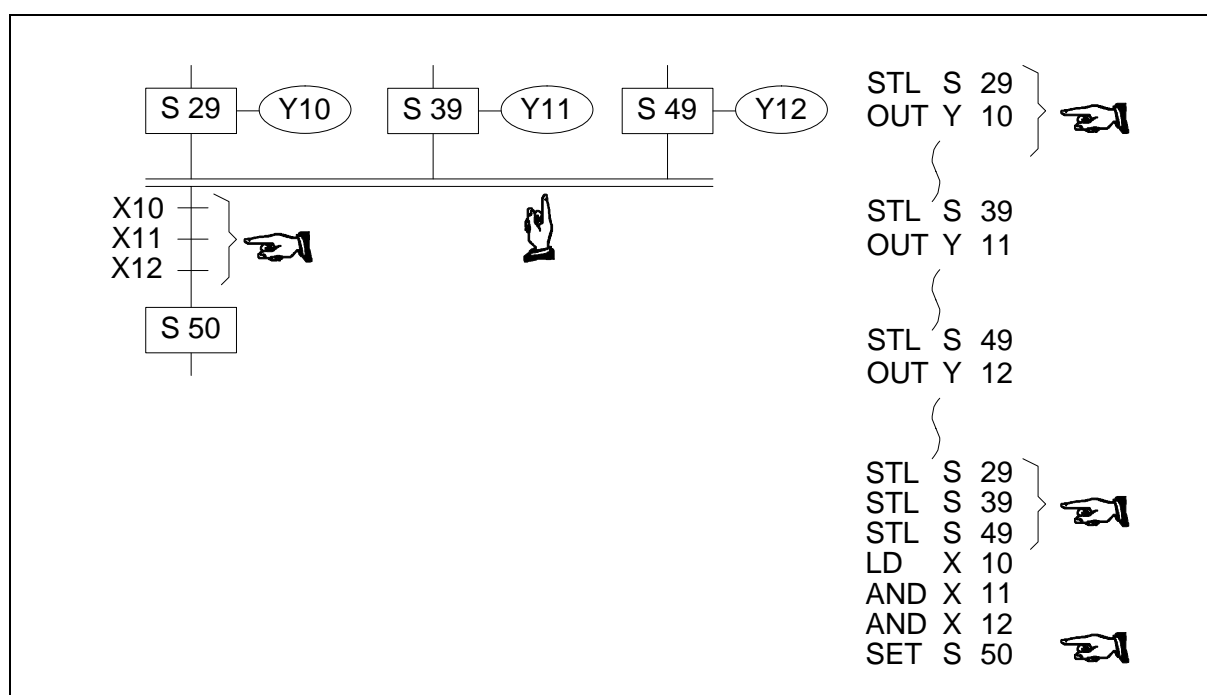


When a group of branch flows are activated, the user will often either;

- 'Race' each flow against its counter parts. The flow which completes fastest would then activate a joining function ("First State Merge" described in the previous section) OR
- The STL flow will not continue until ALL branch flows have completed there tasks. This is called a 'Multiple State Merge'.

An explanation of Multiple State Merge now follows below.

In the example below, states S29, S39 and S49 must all be active. If the instruction list is viewed it can be seen that each of the states has its own operating/processing instructions but that also additional STL instructions have been linked together (in a similar concept as the basic AND instruction). Before state S50 can be activated the trigger conditions must also be active, in this example these are X10, X11 and X12. Once all states and input conditions are made the merging or joining state can be SET ON. As is the general case, all of the states used in the setting procedure are reset automatically.



Because more than one state is being simultaneously joined with further states (some times described as a parallel merge), a set of horizontal parallel lines are used to aid a quick visual recognition.



Limits on the number of branches

- Please see page 3-14 for general notes on programming STL branches.

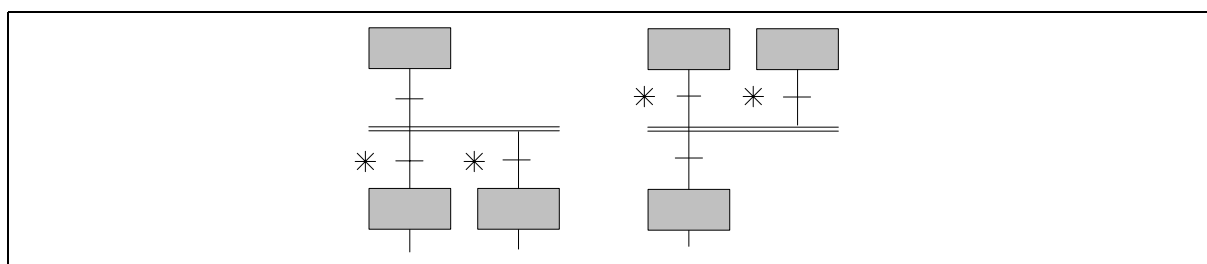
Notes on using the FX-PCS/AT-EE software

- Please see page 3-15 for precautions when using the FX-PCS-AT/EE software.

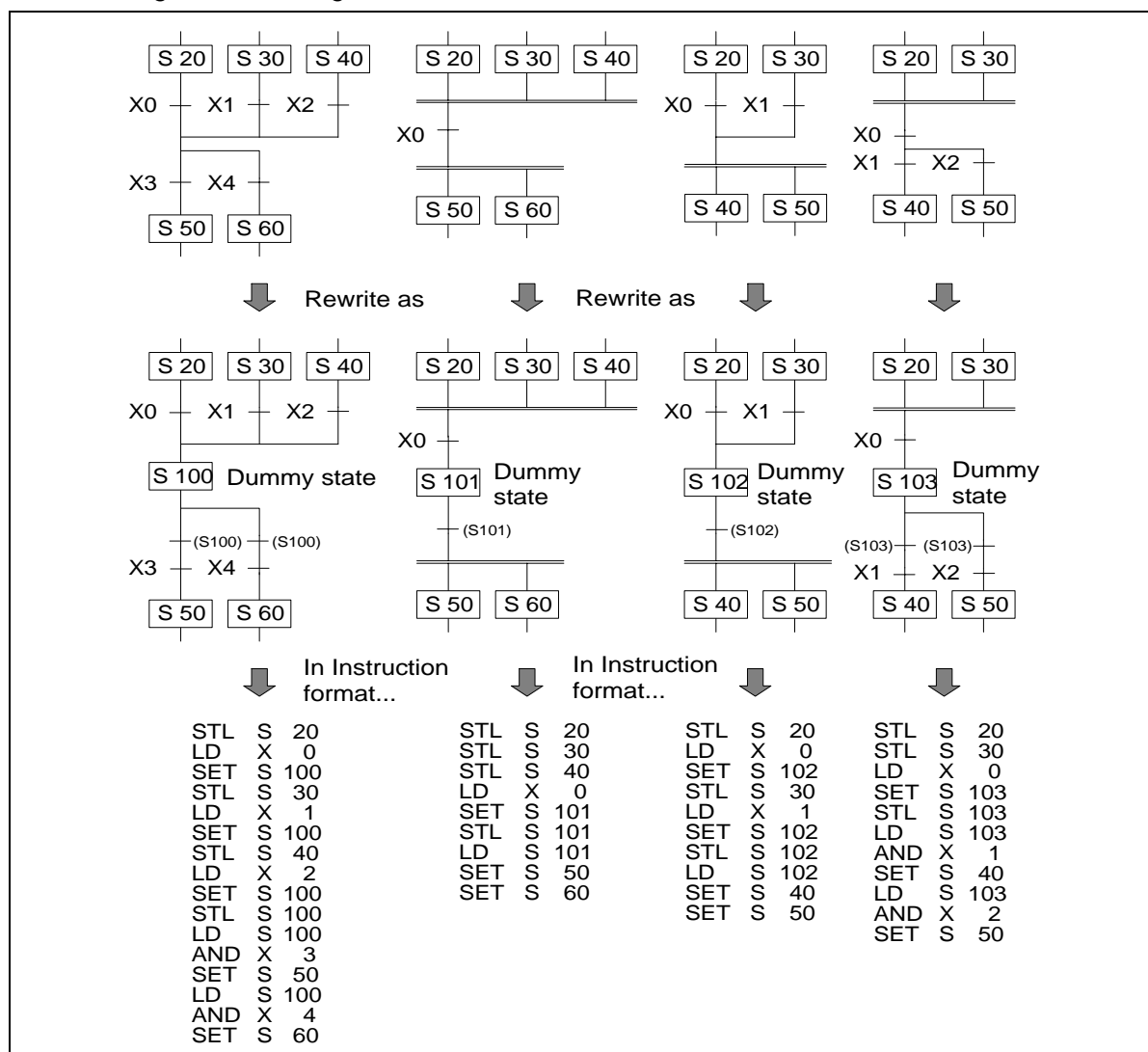
3.9 General Rules For Successful STL Branching

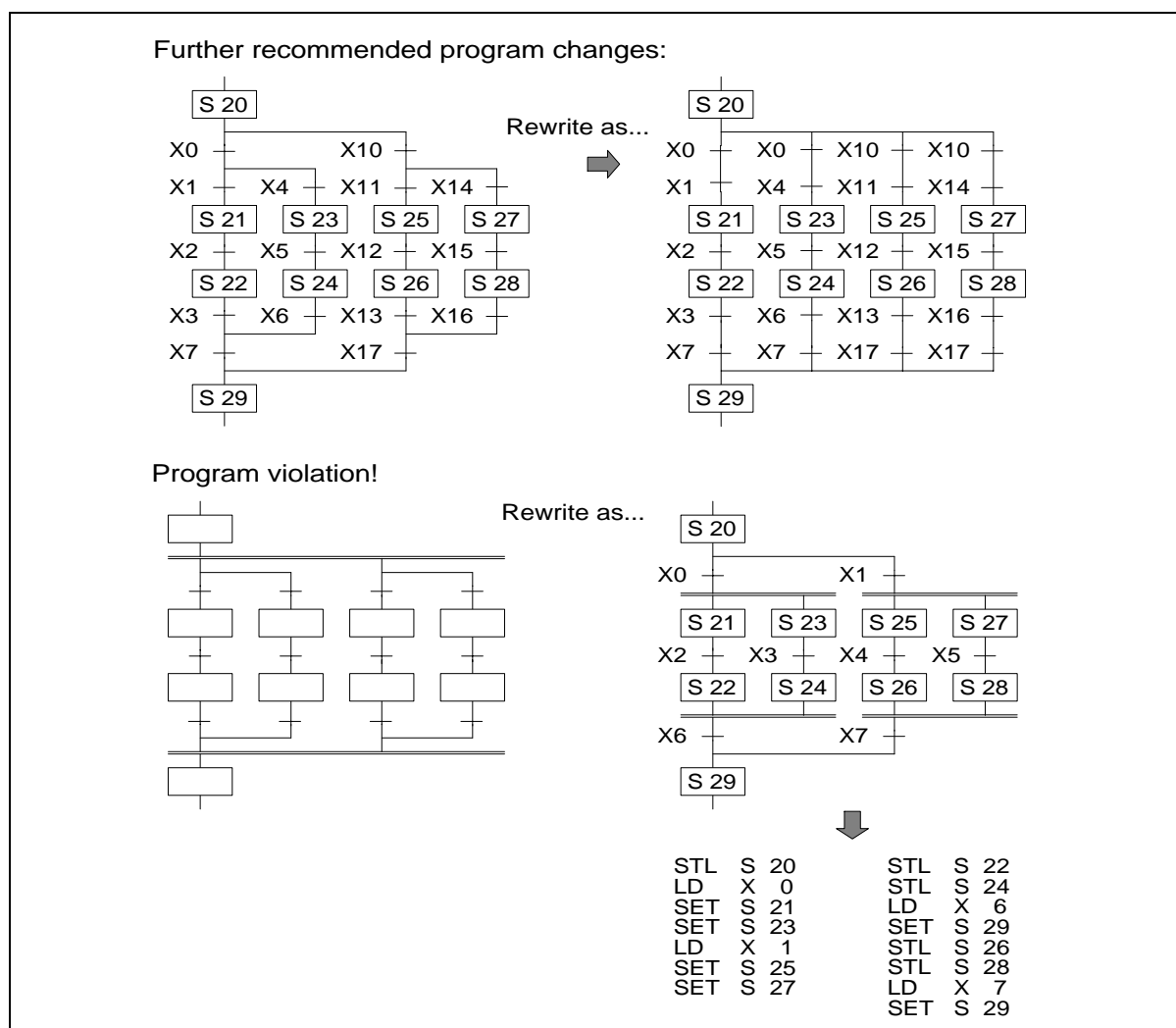
For each branch point 8 further branches may be programmed. There are no limits to the number of states contained in a single STL flow. Hence, the possibility exists for a single initial state to branch to 8 branch flows which in turn could each branch to a further 8 branch flows etc. If the programmable controllers program is read/written using instruction or ladder formats the above rules are acceptable. However, users of the FX-PCS/AT-EE programming package who are utilizing the STL programming feature are constrained by further restrictions to enable automatic STL program conversions (please see page 3-15 for more details).

When using branches, different types of branching /merging cannot be mixed at the same branch point. The item marked with a 'S' are transfer condition which are not permitted.



The following branch configurations/modifications are recommended:





3.10 General Precautions When Using The FX-PCS/AT-EE Software

FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

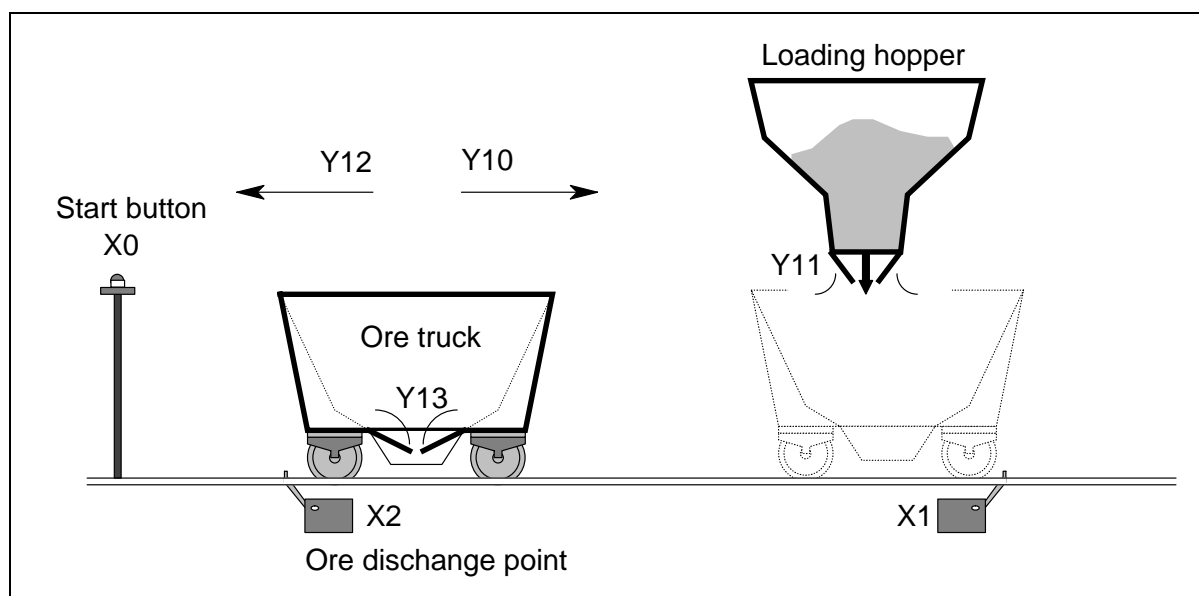
This software has the ability to program in SFC flow diagrams. As part of this ability it can read and convert existing STL programs back into SFC flows even if they were never originally programmed using the FX-PCS/AT-EE software. As an aid to allowing this automatic SFC flow generation the following rules and points should be noted:

- 1) When an STL flow is started it should be initialized with one of the state devices from the range S0 to S9.
- 2) Branch selection or merging should always be written sequentially moving from left to right. This was demonstrated on page 3-11, i.e. on the selective branch S21 was specified before S31 which was specified before S41. The merge states were programmed in a similar manner, S29 proceeded S39 which proceeded S49.
- 3) The total number of branches which can be programmed with the STL programming mode are limited to a maximum of 16 circuits for an STL flow. Each branch point is limited to a maximum of 8 branching flows. This means two branch points both of 8 branch flows would equal the restriction. These restrictions are to ensure that the user can always view the STL flow diagram on the computer running the FX-PCS-AT/ EE software and that when it is needed, the STL program flow can be printed out clearly.

3.11 Programming Examples

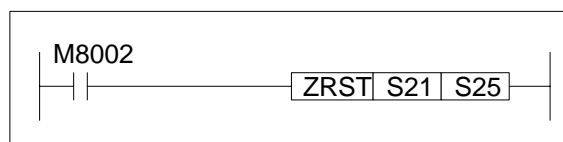
FX1S	FX1N	FX2N	FX2NC
------	------	------	-------

3.11.1 A Simple STL Flow



This simple example is an excerpt from a semi-automatic loading-unloading ore truck program. This example program has a built in, initialization routine which occurs only when the PLC is powered from OFF to ON. This is achieved by using the special auxiliary relay M8002.

This activates a Zone ReSeT (ZRST is applied instruction 40) instruction which ensures all of the operational STL states within the program are reset. The program example opposite shows an M8002/ZRST example.



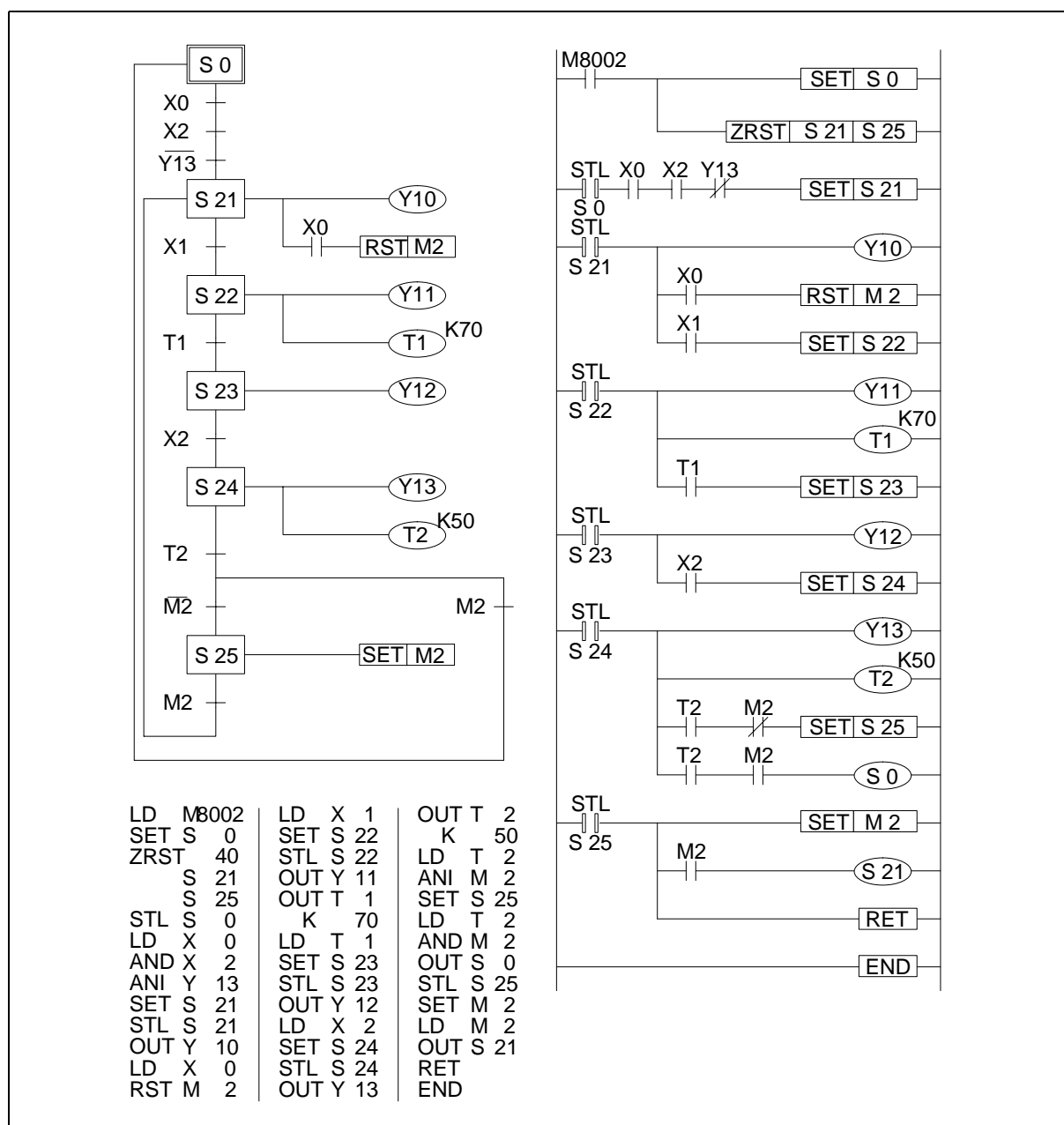
The push button X0 acts as a start button and a mode selection button. The STL state S0 is initialized with the ZRST instruction. The system waits until inputs X0 and X2 are given and Y13 is not active. In the scenario this means the ore truck is positioned at the ore discharge point, i.e. above the position sensor X2. The ore truck is not currently discharging its load, i.e. the signal to open the trucks unloading doors (Y13) is not active and the start button (X0) has been given. Once all of the points have been met the program steps on to state S21.

On this state the ore cart is moved (Y10) and positioned (X1) at the loading hopper. If the start button (X0) is pressed during this stage the ore cart will be set into a repeat mode (M2 is reset) where the ore truck is immediately returned to the loading hopper after discharging its current load. This repeat mode must be selected on every return to the loading station.

Once at the loading point the program steps onto state S22. This state opens the hoppers doors (Y11) and fills the truck with ore. After a timed duration, state S23 is activated and the truck returns (Y12) to the discharge point (X2).

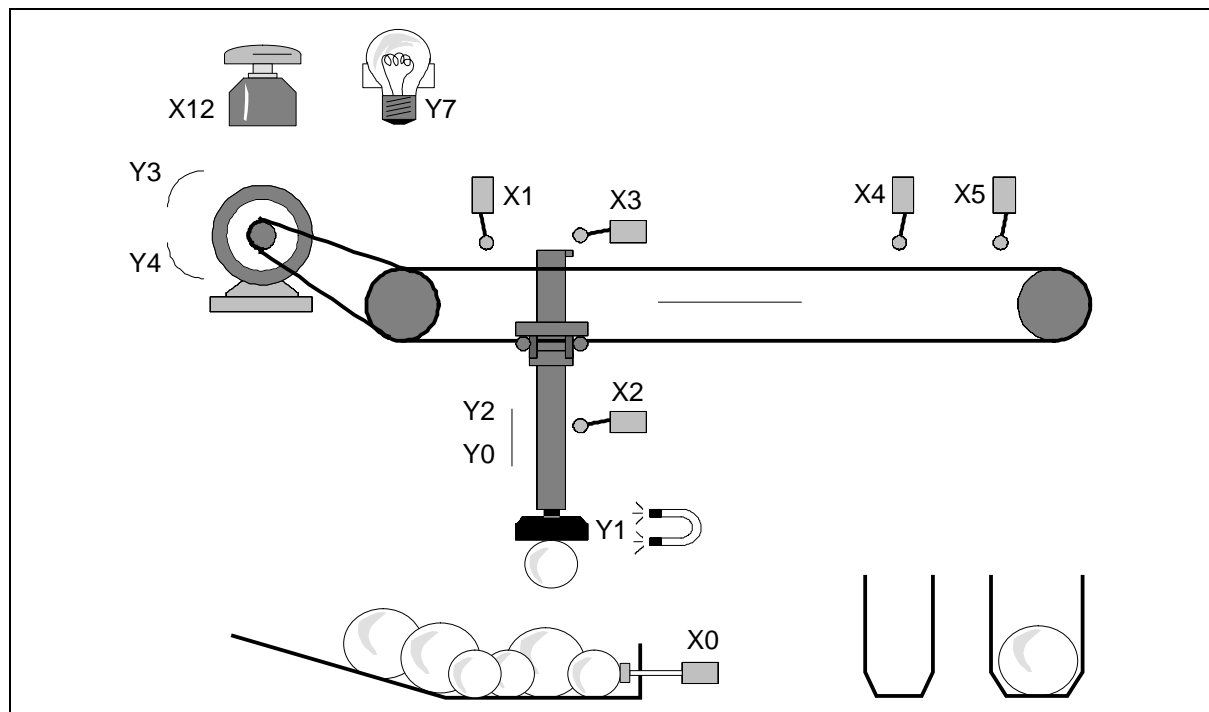
Once at the discharge point the truck opens its bottom doors (Y13). After a timed duration in which the truck empties its contents, the program checks to see if the repeat mode was selected on the last cycle, i.e. M2 is reset. If M2 was reset (in state S21) the program 'jumps' to step S21 and the ore truck is returned for immediate refilling. If M2 is not reset, i.e. it is active, the program cycles back to STL state S0 where the ore truck will wait until the start push button is given.

This is a simple program and is by no means complete but it identifies the way a series of tasks have been mapped to an STL flow.



3.11.2 A Selective Branch/ First State Merge Example Program

The following example depicts an automatic sorting robot. The robot sorts two sizes of ball bearings from a mixed 'source pool' into individual storage buckets containing only one type of ball bearing.



The sequence of physical events (from initial power On) are:

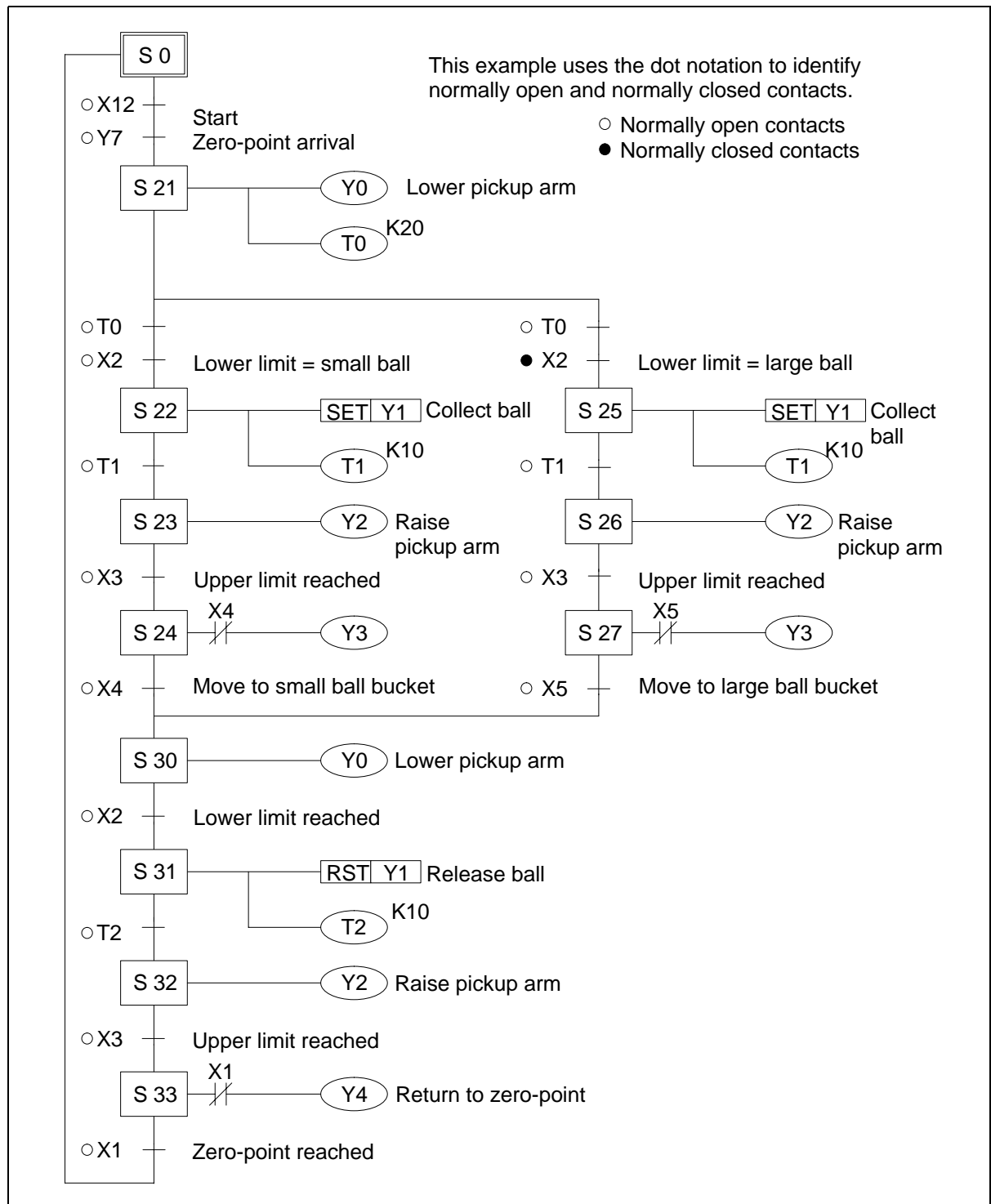
- 1) The pickup arm is moved to its zero-point when the start button (X12) is pressed. When the pickup arm reaches the zero-point the zero-point lamp (Y7) is lit.
- 2) The pickup arm is lowered (Y0) until a ball is collected (Y1). If the lower limit switch (X2) is made a small ball bearing has been collected; consequently no lower limit switch signal means a large ball bearing has been collected. Note, a proximity switch (X0) within the 'source pool' identifies the availability of ball bearings.
- 3) Depending on the collected ball, the pickup arm retracts (output Y2 is operated until X3 is received) and moves to the right (Y3) where it will stop at the limit switch (X4 or X5) indicating the container required for storage.
- 4) The program continues by lowering the pickup arm (Y0) until the lower limit switch (X2) is reached.
- 5) The collected ball being is released (Y1 is reset).
- 6) The pickup arm is retracted (Y2) once more.
- 7) The pickup arm is traversed back (Y4) to the zero-point (X1).



Points to note

- The Selective Branch is used to choose the delivery program for either small ball bearings or large ball bearings. Once the destination has been reached (i.e. step S24 or S27 has been executed) the two independent program flows are rejoined at step S30.
- The example program shown works on a single cycle, i.e. every time a ball is to be retrieved the start button (X12) must be pressed to initiate the cycle.

Full STL flow diagram/program.



3.12 Advanced STL Use

STL programming can be enhanced by using the Initial State Applied Instruction. This instruction has a mnemonic abbreviation of IST and a special function number of 60. When the IST instruction is used an automatic assignment of state relays, special auxiliary relays (M coils) is made. The IST instruction provides the user with a pre-formatted way of creating a multi-mode program. The modes available are:

- a) Automatic:
 - Single step
 - Single cycle
 - Continuous
- b) Manual:
 - Operator controlled
 - Zero return

More details on this instruction can be found on page 5-67.