

MRS AVN COLLEGE POLYTECHNIC VISAKHAPATNAM
DEPARTMENT OF ELECTRICAL ELECTRONICS
ENGINEERING

PLC AUTOMATION SYSTEM
LABORATORY

Manual

Lab 1 : Getting familiar with Step 7- Micro/Win 32 and the S7-200 PLC

Objective

- To construct PLC programs in LAD using Siemens Step 7-Micro/Win32.
- To run and debug the programs on S7-200 PLC.

Introduction to PLC

Controllers may consist of logical components and connections among them. Depending on the current logical value of input, output is produced to change the status of the system. PLC may realize such controllers.

Today, the command and feedback control systems of industrial automation systems are realized by programmable logic controllers (PLCs). Siemens Simatic S7-200 is one of the PLC brands widely used in industry.

In order for PLCs to work as controllers, they must be able to realize some functions. These functions are basic and combinational logic operations such as AND, OR, AND-NOT, OR-NOT, timer and counter operations. In addition to these, PLCs may have the ability to realize several transfer, mathematical, and PID operations.

PLC consists of three main parts: CPU, memory and I/O units.

CPU is the brain of PLC. It reads the input values from inputs, runs the program existed in the program memory and writes the output values to the output register. Memory is used to store different types of information in the binary structure form. The memory range of S7-200 is composed of three main parts as program, parameter, and retentive data fields. I/O units provide communication between PLC control systems.

Constructing of PLC Program

There are mainly two methods for composing PLC programs: *Ladder Logic Diagram* (LAD) and *Statement List* (STL). LAD method is commonly used to implement the programs for process controls.

A network of LAD is a row of connected elements that form a complete circuit between the left and right power rail. The left power rail represents the energized conductor whereas the right power rail represents the return path conductor of the circuit. Power flows from the left rail, through the closed contacts to the coils or boxes connected to the right power rail. You can then use the power flow to activate the outputs according to your program.

Step 7-Micro/Win 32 is user-friendly development environment for S7-200. A screen shot of Step 7-Micro/Win 32 is shown in Figure 1.1.

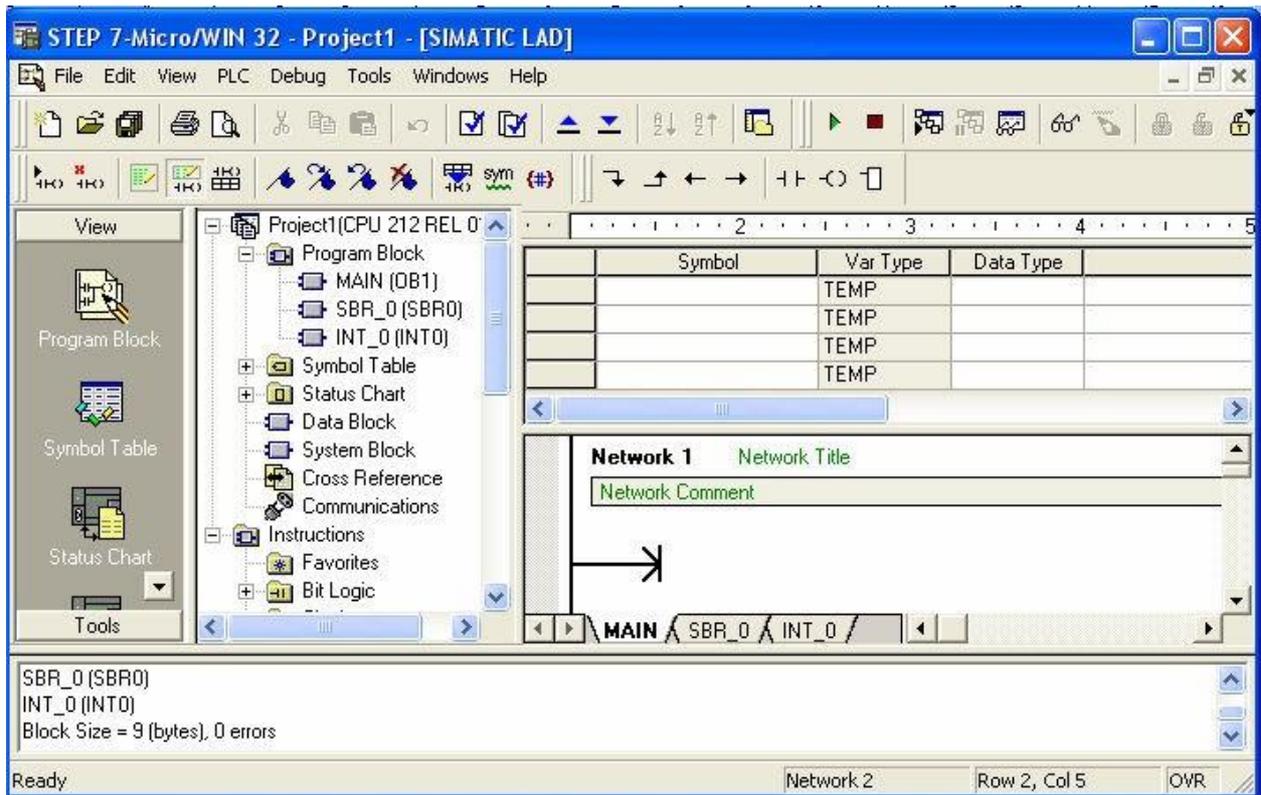


Figure 1.1 Step 7-Micro/Win 32

A simple LAD realizing some Boolean operations is given below:

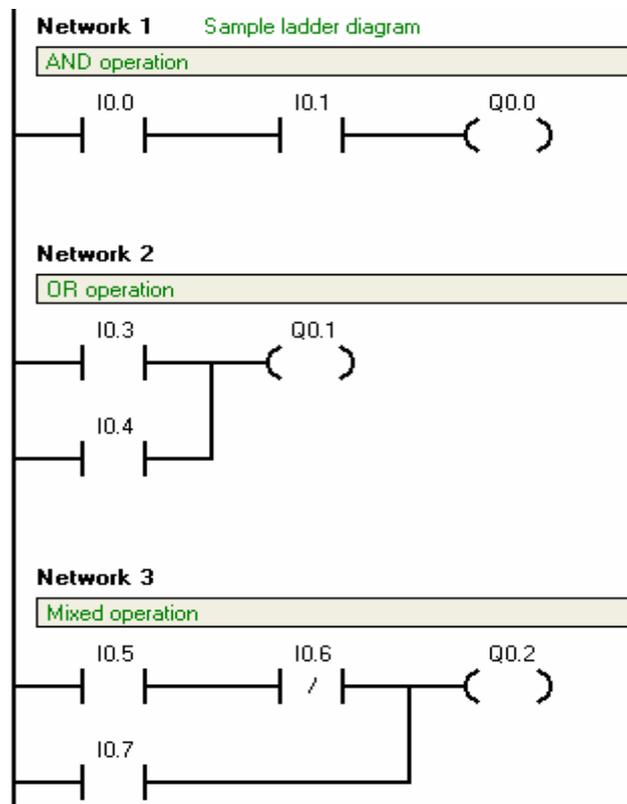


Figure 1.2 Sample LAD for Boolean operations.

You can also switch between LAD and STL by selecting *Ladder* or *STL* from View menu. By this way, you can see how LAD and STL relate to each other.

Network 1		Sample ladder diagram
AND operation		
LD	I0.0	
A	I0.1	
=	Q0.0	
Network 2		
OR operation		
LD	I0.3	
O	I0.4	
=	Q0.1	
Network 3		
Mixed operation		
LD	I0.5	
AN	I0.6	
O	I0.7	
=	Q0.2	

Figure 1.3 STL codes for the LAD in Figure 1.2

In STEP 7-Micro/WIN 32, click on *New* option in the *Project* menu for new project. After composing program, click the *Compile* button on the taskbar. By clicking this button, software translates the program code block into machine language for execution by the CPU. A program can not be downloaded to the CPU until it is compiled. If there is an error in your program, your program will not be compiled and software will warn you about the errors in your program. After correcting errors, try to compile again.

✓ You can save your work by clicking *Save All* option in the *Project* menu. You can also load existed projects by clicking *Open* option in the *Project* menu

Running PLC Programs

When the program is compiled successfully, click on the *Download* button on the taskbar to transfer the compiled program to the PLC. During this operation, be sure that PLC is STOP or TERM mode. When the transfer is completed, switch the PLC to RUN mode or click RUN button in the toolbar. Now, PLC is running. Switch the inputs ON and OFF, and observe the change on outputs of the PLC module. Check if your program works correctly.

You can also transfer the existed program in PLC to PC. In a similar way, when the PLC is STOP mode, click on the *Upload* button on the taskbar. When the operation is completed, you can see the program code existed in PLC on computer screen. Now you can modify the program if it is necessary.

✓ You can change the mode of PLC to either RUN, STOP or TERM using the switch on the PLC module.

Experimental Work

1. Construct a LAD for the following Boolean functions operations using the STEP 7-Micro/WIN 32. Transfer and run your programs on the PLC and verify that the PLC operates according to the Boolean functions.

a. $\overline{A \oplus B}$

b. $(\overline{AB} + \overline{CD} + EF).G$

c. $A + BC + \overline{DEF}$

d. $A.(\overline{BC} + \overline{DE}).F$

2. A PLC motor controller has two START buttons and two STOP buttons. The motor is to run if two RUN buttons depressed simultaneously. The motor should run when the buttons are released. Motor stops by depressing any STOP button stops. Construct a LAD for this motor control task. Use the following symbols for the inputs and output:

START Button-1	START1
START Button-2	START2
STOP Button-1	STP1
STOP Button-2	STP2
MOTOR Starter	MOTOR

Lab 2 : Bit Logic Instructions

Objective

- To study the operation of bit logic instructions.
- To construct PLC program using the bit logic instructions.

Bit Logic Instructions

Contacts

Standard Contacts

The Normally Open contact instructions (LD, A, and O) and Normally Closed contact instructions (LDN, AN, ON) obtain the referenced value from the memory or from the process-image register. The standard contact instructions obtain the referenced value from the memory (or process-image register if the data type is I or Q). The Normally Open contact is closed (on) when the bit is equal to 1, and the Normally Closed contact is closed (on) when the bit is equal to 0. In STL, the Normally Open instructions Load, AND, or OR the bit value of the address bit to the top of the stack, and the Normally Closed instructions Load, AND, or OR the logical NOT of the bit value to the top of the stack.

Immediate Contacts

An immediate contact does not rely on the S7-200 scan cycle to update; it updates immediately. The Normally Open Immediate contact instructions (LDI, AI, and OI) and Normally Closed Immediate contact instructions (LDNI, ANI, and ONI) obtain the physical input value when the instruction is executed, but the process-image register is not updated. The Normally Open Immediate contact is closed (on) when the physical input point (bit) is 1, and the normally Closed Immediate contact is closed (on) when the physical input point (bit) is 0. The Normally Open instructions immediately load, AND, or OR the physical input value to the top of the stack, and the Normally Closed instructions immediately Load, AND, or OR the logical NOT of the value of the physical input point to the top of the stack.

NOT Instruction

The Not instruction (NOT) changes the state of power flow input (that is, it changes the value on the top of the stack from 0 to 1 or from 1 to 0).

Positive and Negative Transition Instructions

The Positive Transition contact instruction (EU) allows power to flow for one scan for each off-to-on transition. The Negative Transition contact instruction (ED) allows power to flow for one scan for each on-to-off transition. For the Positive Transition instruction, detection of a 0-to-1 transition in the value on the top of the stack sets the top of the stack value to 1; otherwise, it is set to 0. For a Negative Transition instruction, detection of a 1-to-0 transition in the value on the top of the stack sets the top of the stack value to 1; otherwise, it is set to 0. For run mode editing (when you edit your program in RUN mode), you must enter a parameter for the Positive Transition and Negative Transition instructions.

Table 2-1 Valid Operands for the Bit Logic Input Instructions

Inputs/Outputs	Data Type	Operands
Bit	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Bit (immediate)	BOOL	I

Coils

Output

The Output instruction (=) writes the new value for the output bit to the process-image register. When the Output instruction is executed, the S7-200 turns the output bit in the process-image register on or off. For LAD, the specified bit is set equal to power flow. For STL, the value on the top of the stack is copied to the specified bit.

Output Immediate

The Output Immediate instruction (=I) writes the new value to both the physical output and the corresponding process-image register location when the instruction is executed. When the Output Immediate instruction is executed, the physical output point (Bit) is immediately set equal to power flow. For STL, the instruction immediately copies the value on the top of the stack to the specified physical output bit (STL). The "I" indicates an immediate reference; the new value is written to both the physical output and the corresponding process-image register location when the instruction is executed. This differs from the non-immediate references, which write the new value to the process-image register only.

Set and Reset

The Set (S) and Reset (R) instructions set (turn on) or reset (turn off) the specified number of points (N), starting at the specified address (Bit). You can set or reset from 1 to 255 points.

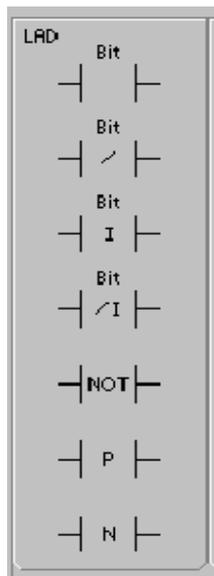
If the Reset instruction specifies either a timer bit (T) or counter bit (C), the instruction resets the timer or counter bit and clears the current value of the timer or counter.

Set Immediate and Reset Immediate

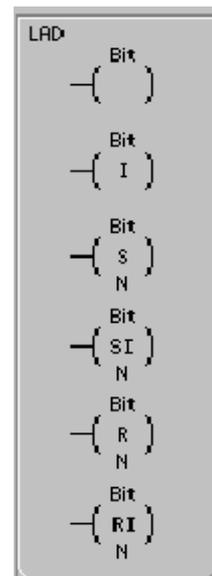
The Set Immediate and Reset Immediate instructions immediately set (turn on) or immediately reset (turn off) the number of points (N), starting at specified address (Bit). You can set or reset from 1 to 128 points immediately. The "I" indicates an immediate reference; when the instruction is executed, the new value is written to both the physical output point and the corresponding process-image register location. This differs from the non-immediate references, which write the new value to the process-image register only.

Table 2-2 Valid Operands for the Bit Logic Output Instructions

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



Contact instructions



Coil instructions

Experimental Work

An automatic stamp system shown in Figure 2 works as follows: When start switch is turned on, system gets ready to run. When the operator puts a box at the beginning of the conveyor (on LS1) the motor runs and conveyor moves. Upon reaching the mid point of the conveyor (on LS2) the conveyor motor stops. Then the stamper comes down and puts the stamp on the box. When this process is finished, the stamper goes up and conveyor moves again to the other end of the conveyor. After box reaches to end of the conveyor (on LS3), the motor stops. The system waits for the box to get and the another box to be placed at the beginning of the conveyor. If start switch is turned off, the system can not run even if there is a box on conveyor. The light on the start box indicates that the system is active whereas *UP* and *Down* lights indicate that the stamper is *UP* and *DOWN* position respectively. Develop a LAD to control the stamp system.

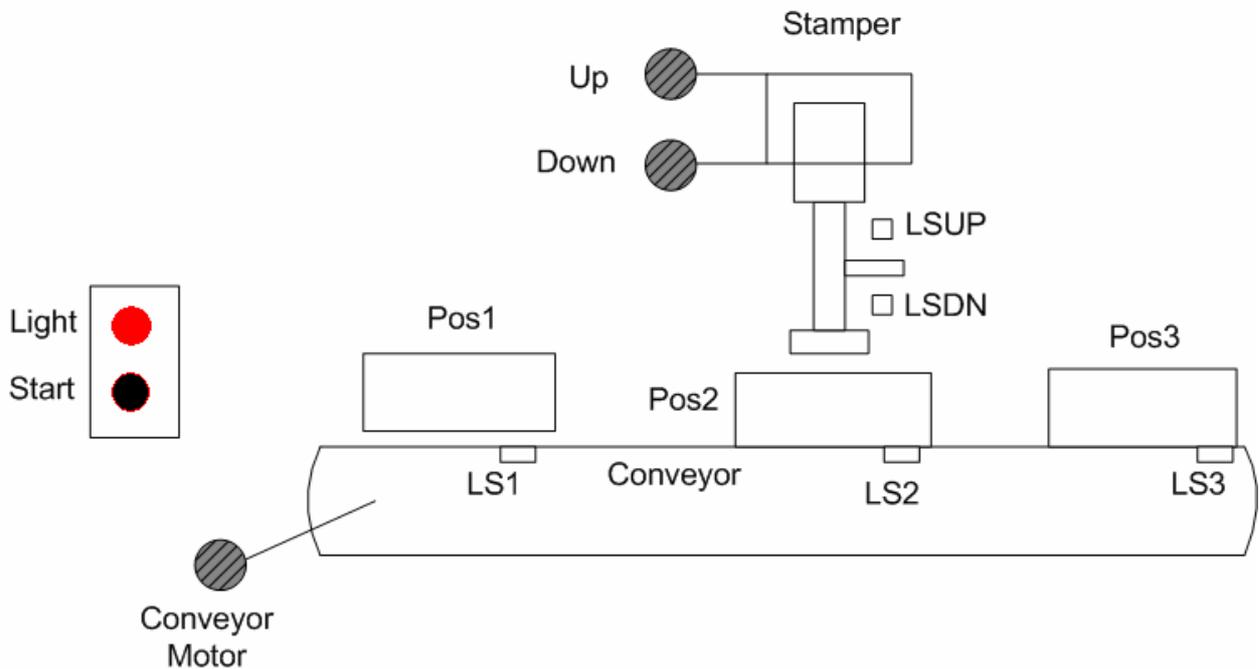


Figure 2. Automatic stamp machine

Lab 3 : Sequencer

Objective

- To construct sequencer using bit logic instructions only.

See the course notes.

Experimental Work

A vehicle shown in Figure 3 carries two different objects with different sizes. Initially, the vehicle waits at the mid-point of the system.

- When a large-sized object is placed on the vehicle, it moves backward up to the 1st drop point. After reaching to the point, vehicle waits until the object on is removed. Then, vehicle returns back to the mid-point.
- When a small-sized object is placed on the vehicle, it moves forward up to the 2nd drop point. After reaching to the point, vehicle waits until the object on is removed. Then, vehicle returns back to the mid-point.

During the movement, if an object comes in front of the vehicle stops with the help of a photo sensor, then a buzzer is turned on to indicate the situation. When the object is removed, the vehicle continues its movement.

Red LED is on when the object is detected in front of the vehicle. The green LED indicates that the vehicle reached 1st drop point, and yellow LED indicates that the vehicle reached to the 2nd drop point.

Construct a LAD for this system.

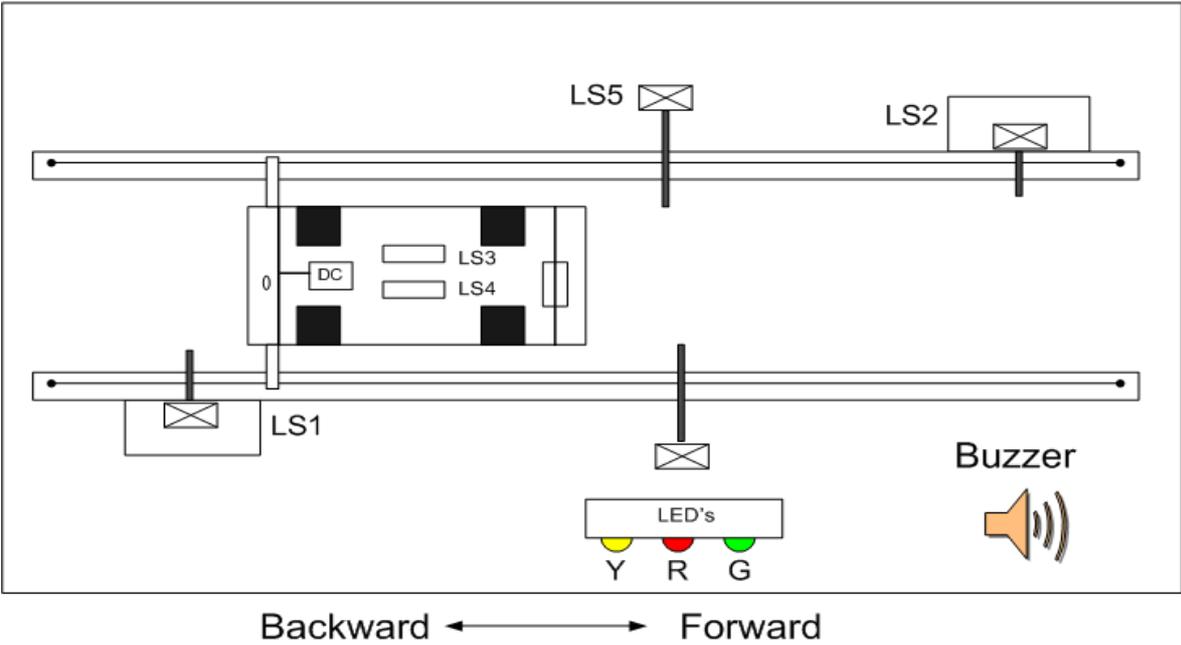


Figure 3. Transportation of the objects using vehicle

Lab 4 : Timers

Objective

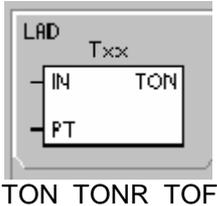
- To study the operation of different types of timers.
- To use the PLC timers in a process control.

Timer Instructions

On-Delay Timer

Retentive On-Delay Timer

The On-Delay Timer (TON) and Retentive On-Delay Timer (TONR) instructions count time when the enabling input is on. The timer number (Txx) determines the resolution of the timer, and the resolution is now shown in the instruction box.



Off-Delay Timer

The Off-Delay Timer (TOF) is used to delay turning an output off for a fixed period of time after the input turns off.

The timer number (Txx) determines the resolution of the timer, and the resolution is now shown in the instruction box.

Table 4-1 Valid Operands for the SIMATIC Timer Instructions

Inputs/Outputs	Data Types	Operands
Txx	WORD	Constant (T0 to T255)
IN	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
PT	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

- You can use a TON for timing a single interval.
- You can use a TONR for accumulating a number of timed intervals.
- You can use a TOF for extending time past an off (or false) condition, such as for cooling a motor after it is turned off.

Table 4-2 Operations of the Timer Instructions

Type	Current >= Preset	State of the Enabling Input (IN)	Power Cycle/First Scan
TON	Timer bit on Current continues counting to 32,767	ON: Current value counts time OFF: Timer bit off, current value = 0	Timer bit off Current value = 0
TONR	Timer bit on Current continues counting to 32,767	ON: Current value counts time OFF: Timer bit and current value maintain last state	Timer bit off Current value can be maintained ¹
TOF	Timer bit off Current = Preset, stops counting	ON: Timer bit on, current value = 0 OFF: Timer counts after on-to-off transition	Timer bit off Current value = 0

The TON and TONR instructions count time when the enabling input is on. When the current value is equal to or greater than the preset time, the timer bit is on.

- The current value of a TON timer is cleared when the enabling input is off, whereas the current value of the TONR timer is maintained when the input is off.
- You can use the TONR timer to accumulate time when the input turns on and off. Use the Reset instruction (R) to clear the current value of the TONR.
- Both the TON and the TONR timers continue counting after the preset is reached, and they stop counting at the maximum value of 32,767.

The TOF instruction is used to delay turning an output off for a fixed period of time after the input turns off. When the enabling input turns on, the timer bit turns on immediately, and the current value is set to 0. When the input turns off, the timer counts until the elapsed time reaches the preset time.

- When the preset is reached, the timer bit turns off and the current value stops incrementing; however, if the input turns on again before the TOF reaches the preset value, the timer bit remains on.
- The enabling input must make an on-to-off transition for the TOF to begin counting time intervals.
- If the TOF timer is inside an SCR region and the SCR region is inactive, then the current value is set to 0, the timer bit is turned off, and the current value does not increment.

You can reset a TONR only by using the Reset (R) instruction. You can also use the Reset instruction to reset any TON or TOF. The Reset instruction performs the following operations:

- Timer Bit = off
- Timer Current = 0

After a reset, TOF timers require the enabling input to make the transition from on to off in order for the timer to restart.

Determining the Resolution of the Timer

Timers count time intervals. The resolution (or time base) of the timer determines the amount of time in each interval. For example, a TON with a resolution of 10 ms counts the number of 10-ms intervals that elapse after the TON is enabled: a count of 50 on a 10-ms timer represents 500 ms. The SIMATIC timers are available in three resolutions: 1 ms, 10 ms, and 100 ms. As shown in Table 4-3, the timer number determines the resolution of the timer.

To guarantee a minimum time interval, increase the preset value (PV) by 1. For example: To ensure a minimum timed interval of at least 2100 ms for a 100-ms timer, set the PV to 22.

Table 4-3 Timer Numbers and Resolutions

Timer Type	Resolution	Maximum Value	Timer Number
TONR (retentive)	1 ms	32.767 s (0.546 min.)	T0, T64
	10 ms	327.67 s (5.46 min.)	T1 to T4, T65 to T68
	100 ms	3276.7 s (54.6 min.)	T5 to T31, T69 to T95
TON, TOF (non-retentive)	1 ms	32.767 s (0.546 min.)	T32, T96
	10 ms	327.67 s (5.46 min.)	T33 to T36, T97 to T100
	100 ms	3276.7 s (54.6 min.)	T37 to T63, T101 to T255

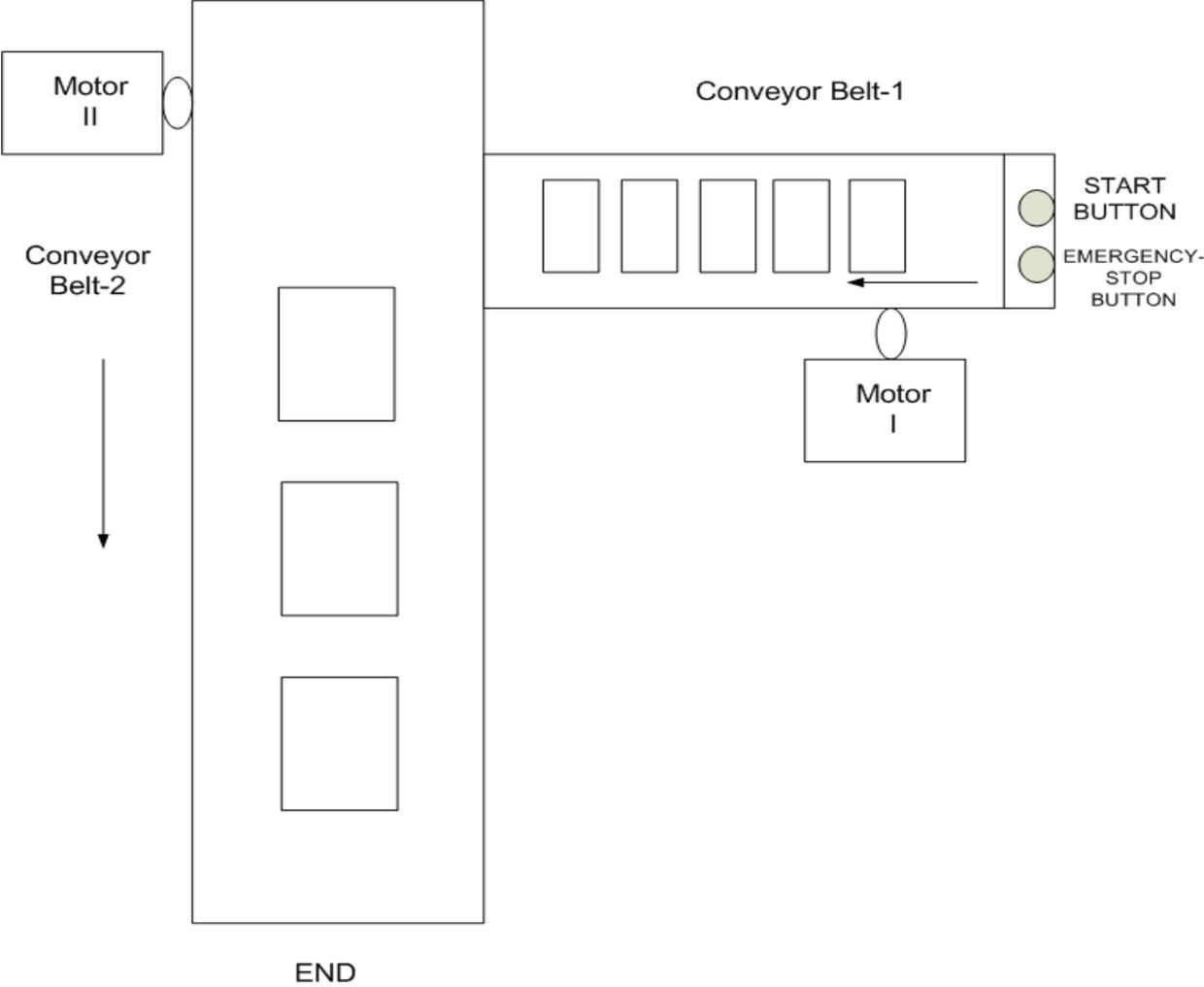
Understanding How Resolution Affects the Timer Action

For a timer with a resolution of 1 ms, the timer bit and the current value are updated asynchronous to the scan cycle. For scans greater than 1 ms, the timer bit and the current value are updated multiple times throughout the scan. For a timer with a resolution of 10 ms, the timer bit and the current value are updated at the beginning of each scan cycle. The timer bit and current value remain constant throughout the scan, and the time intervals that accumulate during the scan are added to the current value at the start of each scan. For a timer with a resolution of 100 ms, the timer bit and current value are updated when the instruction is executed; therefore, ensure that your program executes the instruction for a 100-ms timer only once per scan cycle in order for the timer to maintain the correct timing.

To guarantee that the output of a self-resetting timer is turned on for one scan each time the timer reaches the preset value, use a normally closed contact instead of the timer bit as the enabling input to the timer.

Experimental Work

The system to be controlled by PLC consists of two belts. If the Start button is pressed, Conveyor Belt-1 will begin to run. After 5 seconds Conveyor Belt-2 will be active. After the whole system runs for 15 seconds, Conveyor Belt-1 will stop. Then Conveyor Belt-2 continues to move for 5 seconds and then it will stop, too. Also the system can be reset by the emergency-stop button at any time. Construct a LAD for S7-200 PLC to control the system.



Lab 5 : Counters

Objective

- To study the operation of different types of counters.
- To use the PLC counters and timers in a process control.

Counter Instructions

Count Up Counter

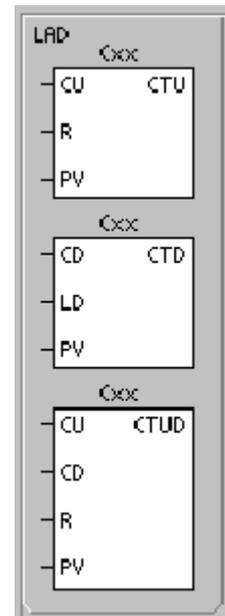
The Count Up instruction (CTU) counts up from the current value each time the count up (CU) input makes the transition from off to on. When the current value Cxx is greater than or equal to the preset value PV, the counter bit Cxx turns on. The counter is reset when the Reset (R) input turns on, or when the Reset instruction is executed. The counter stops counting when it reaches the maximum value (32,767).

STL operation :

- Reset input: Top of stack
- Count Up input: Value loaded in the second stack location

Count Down Counter

The Count Down instruction (CTD) counts down from the current value of that counter each time the count down (CD) input makes the transition from off to on. When the current value Cxx is equal to 0, the counter bit Cxx turns on. The counter resets the counter bit Cxx and loads the current value with the preset value PV when the load input LD turns on. The counter stops upon reaching zero, and the counter bit Cxx turns on.



Count Up/Down Counter

The Count Up/Down instruction (CTUD) counts up each time the count up (CU) input makes the transition from off to on, and counts down each time the count down (CD) input makes the transition from off to on. The current value Cxx of the counter maintains the current count. The preset value PV is compared to the current value each time the counter instruction is executed. Upon reaching maximum value (32,767), the next rising edge at the count up input causes the current count to wrap around to the minimum value (-32,768). On reaching the minimum value (-32,768), the next rising edge at the count down input causes the current count to wrap around to the maximum value (32,767).

When the current value Cxx is greater than or equal to the preset value PV, the counter bit Cxx turns on. Otherwise, the counter bit turns off. The counter is reset when the Reset (R) input turns on, or when the Reset instruction is executed. The CTUD counter stops counting when it reaches PV.

Table 5-1 Valid Operands for the SIMATIC Counter Instructions

Inputs/Outputs	Data Types	Operands
Cxx	WORD	Constant (C0 to C255)
CU, CD, LD, R	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
PV	INT	IW, QW, VW, MW, SMW, SW, LW, T, C, AC, AIW, *VD, *LD, *AC, Constant

Since there is one current value for each counter, do not assign the same number to more than one counter. (Up Counters, Up/Down Counters, and Down counters with the same number access the same current value.) When you reset a counter using the Reset instruction, the counter bit is reset and the counter current value is set to zero. Use the counter number to reference both the current value and the counter bit of that counter.

Table 5-2 Operations of the Counter Instructions

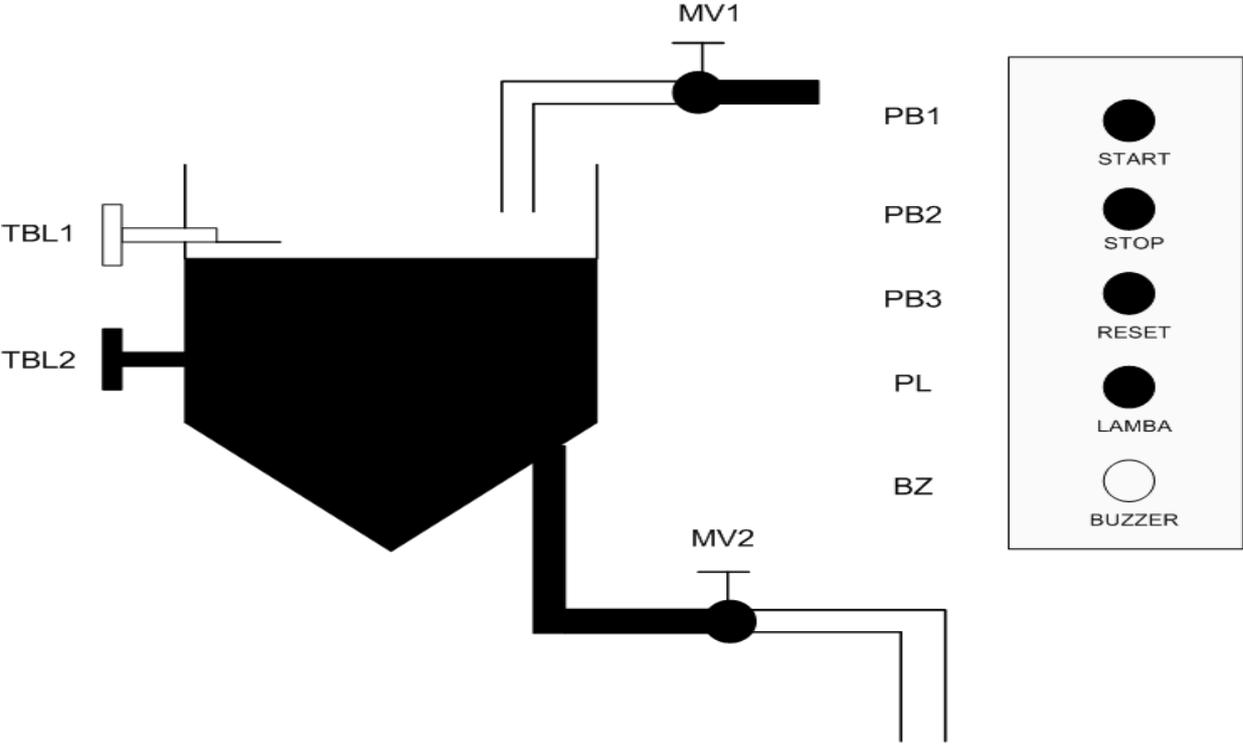
Type	Operation	Counter Bit	Power Cycle/First Scan
CTU	CU increments the current value. Current value continues to increment until it reaches 32,767.	The counter bit turns on when: Current value \geq Preset	Counter bit is off. Current value can be retained. ¹
CTUD	CU increments the current value. CD decrements the current value. Current value continues to increment or decrement until the counter is reset.	The counter bit turns on when: Current value \geq Preset	Counter bit is off. Current value can be retained. ¹
CTD	CD decrements the current value until the current value reaches 0.	The counter bit turns on when: Current value = 0	Counter bit is off. Current value can be retained. ¹

Experimental Work

A PLC-controlled system operates as follows. If the Start button is pressed, MV1 will be opened and dye starts to fill the tank. At the same time mixing-motor begins to run. As the level of the dye passes TBL2 and reaches to TBL1, MV1 will be closed and mixing-motor will stop. Then MV2 will be opened and dye begins to run out of the container. After the level of the dye reaches to the below of TBL2, MV2 will be closed. This process is repeated for two (2) times, then the system will stop.

We can observe the stopping of the system with the lamp and the buzzer.

After the system stops, buzzer will continue to run for 2 sec and then it stops. But the lamp continues to light until the reset button is pressed. When the reset button is pressed, the system will be ready to begin the same process again. Construct a LAD for this system.



Lab 6 : Program Control Instructions

Objective

- To use jump and subroutine in a process control.

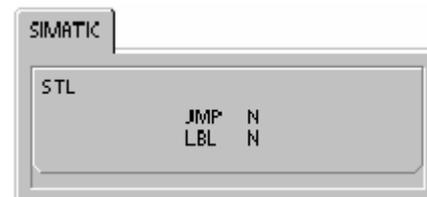
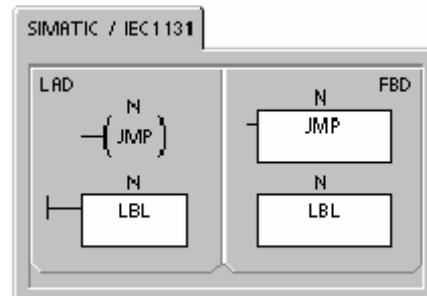
Jump Instructions

The Jump to Label instruction (JMP) performs a ranch to the specified label N within the program. The Label instruction (LBL) marks the location of the jump destination N.

You can use the Jump instruction in the main program, in subroutines, or in interrupt routines. The Jump and its corresponding Label instruction must always be located within the same segment of code (either the main program, a subroutine, or an interrupt routine).

You cannot jump from the main program to a label in either a subroutine or an interrupt routine. Likewise, you cannot jump from a subroutine or interrupt routine to a label outside that subroutine or interrupt routine.

You can use a Jump instruction within an SCR segment, but the corresponding Label instruction must be located within the same SCR segment.



Valid Operands for the Jump Instructions

Inputs/Outputs	Data Types	Operands
N	WORD	Constant (0 to 255)

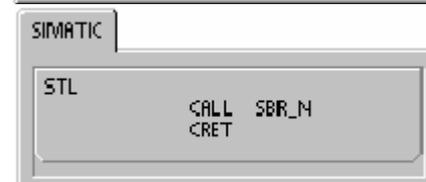
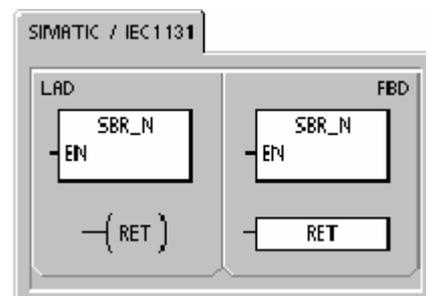
Subroutine Instructions

The Call Subroutine instruction (CALL) transfers control to the subroutine SBR_N. You can use a Call Subroutine instruction with or without parameters. After the subroutine completes its execution, control returns to the instruction that follows the Call Subroutine.

The Conditional Return from Subroutine instruction (CRET) terminates the subroutine based upon the preceding logic. To add a subroutine, select the Edit > Insert > Subroutine menu command.

From the main program, you can nest subroutines (place a subroutine call within a subroutine) to a depth of eight. From an interrupt routine, you cannot nest subroutines.

A subroutine call cannot be placed in any subroutine called from an interrupt routine. Recursion (a subroutine that calls



itself) is not prohibited, but you should use caution when using recursion with subroutines.

When a subroutine is called, the entire logic stack is saved, the top of stack is set to one, all other stack locations are set to zero, and control is transferred to the called subroutine. When this subroutine is completed, the stack is restored with the values saved at the point of call, and control is returned to the calling routine.

Accumulators are common to subroutines and the calling routine. No save or restore operation is performed on accumulators due to subroutine use. When a subroutine is called more than once in the same cycle, the edge/up, edge/down, timer and counter instructions should not be used.

Valid Operands for the Subroutine Instruction

Inputs/Outputs	Data Types	Operands
SBR_N	WORD	Constant <i>for CPU 221, CPU 222, CPU 224:</i> 0 to 63 <i>for CPU 224XP and CPU 226</i> 0 to 127
IN	BOOL BYTE WORD, INT DWORD, DINT STRING	V, I, Q, M, SM, S, T, C, L, Power Flow VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *LD, *AC ¹ , constant VW, T, C, IW, QW, MW, SMW, SW, LW, AC, AIW, *VD, *LD, *AC ¹ , constant VD, ID, QD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC ¹ , &VB, &IB, &QB, &MB, &T, &C, &SB, &AI, &AQ, &SMB, constant *VD, *LD, *AC, constant
IN/OUT	BOOL BYTE WORD, INT DWORD, DINT	V, I, Q, M, SM ² , S, T, C, L VB, IB, QB, MB, SMB ² , SB, LB, AC, *VD, *LD, *AC ¹ VW, T, C, IW, QW, MW, SMW ² , SW, LW, AC, *VD, *LD, *AC ¹ VD, ID, QD, MD, SMD ² , SD, LD, AC, *VD, *LD, *AC ¹
OUT	BOOL BYTE WORD, INT DWORD, DINT	V, I, Q, M, SM ² , S, T, C, L VB, IB, QB, MB, SMB ² , SB, LB, AC, *VD, *LD, *AC ¹ VW, T, C, IW, QW, MW, SMW ² , SW, LW, AC, AQW, *VD, *LD, *AC ¹ VD, ID, QD, MD, SMD ² , SD, LD, AC, *VD, *LD, *AC ¹

Calling a Subroutine With Parameters

Subroutines can contain passed parameters. The parameters are defined in the local variable table of the subroutine. The parameters must have a symbol name (maximum of 23 characters), a variable type, and a data type. Sixteen parameters can be passed to or from a subroutine. The variable type field in the local variable table defines whether the variable is passed into the subroutine (IN), passed into and out of the subroutine (IN_OUT), or passed out of the subroutine (OUT). Table 6-79 describes the parameter types for a subroutine. To add a parameter entry, place the cursor on the variable type field of the type (IN, IN_OUT, or OUT) that you want to add. Click the right mouse button to get a menu of options. Select the Insert option and then the Row Below option. Another parameter entry of the selected type appears below the current entry.

Parameter Types for a Subroutine

Parameter	Description
IN	Parameters are passed into the subroutine. If the parameter is a direct address (such as VB10), the value at the specified location is passed into the subroutine. If the parameter is an indirect address (such as *AC1), the value at the location pointed to is passed into the subroutine. If the parameter is a data constant (16#1234) or an address (&VB100), the constant or address value is passed into the subroutine.
IN_OUT	The value at the specified parameter location is passed into the subroutine, and the result value from the subroutine is returned to the same location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed for input/output parameters.
OUT	The result value from the subroutine is returned to the specified parameter location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed as output parameters. Since output parameters do not retain the value assigned by the last execution of the subroutine, you must assign values to outputs each time the subroutine is called. Note that the SET and RESET instructions only affect the value of the Boolean operand(s) when power flow is ON.
TEMP	Any local memory that is not used for passed parameters can be used for temporary storage within the subroutine.

The parameter types are listed below:

BOOL: This data type is used for single bit inputs and outputs. IN3 in the following example is a Boolean input.

BYTE, WORD, DWORD: These data types identify an unsigned input or output parameter of 1, 2, or 4 bytes, respectively.

INT, DINT: These data types identify signed input or output parameters of 2 or 4 bytes, respectively.

REAL: This data type identifies a single precision (4 byte) IEEE floating-point value.

STRING: This data type is used as a four-byte pointer to a string.

Power Flow: Boolean power flow is allowed only for bit (Boolean) inputs. This declaration tells STEP 7-Micro/WIN that this input parameter is the result of power flow based on a combination of bit logic instructions. Boolean power flow inputs must appear first in the local variable table before any other type input. Only input parameters are allowed to be used this way. The enable input (EN) and the IN1 inputs in the following example use Boolean logic.

Local variable table

Name	Var Type	Data Type	Comment	
EN	IN	BOOL		
LB0	FirstPare	IN	BOOL	First pare flag
LB1	Addr	IN	BYTE	Address of slave device
LW2	Date	IN	INT	Date to write to slave
LB4	Status	IN_OUT	BYTE	Status of write
LB0	Done	OUT	BOOL	Done flag
LW6	Error	OUT	WORD	Error number (f. anal)

Experimental Work

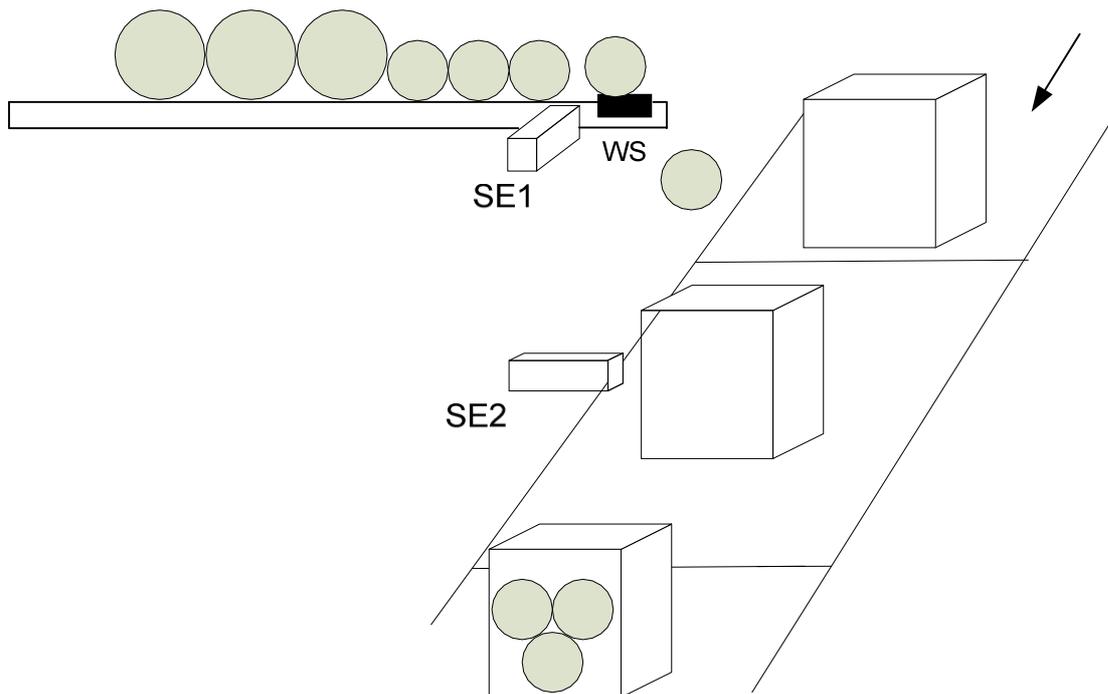
The system contains two conveyor belts. The conveyor carries balls and boxes. If the start button is pushed, box conveyor will begin to move. When the box triggers the sensor SE2, the box conveyor stops and the ball conveyor begins to move. The weight sensor WS measures the ball weights. If the ball weight is 500 g, the sensor WS is on state(logic 1).The ball conveyor carries two different size of the balls, 250 g and 500g. If weight of the first ball is 500 g, the box is filled with 3 balls. Otherwise the box filled with five balls with 250g. In the system, the sensor SE1 is used to count the balls. After the box is filled with the balls as specified above, the ball conveyor stops and box conveyor begins to move.

The system can be reset by a stop button any time.

Construct the LAD for this process using only a) jump instruction, and b) subroutine instruction.

Input	Output
Start and stop buttons Sensors SE1, SE2 and WS	Ball conveyor motor Box conveyor motor

Ball-Conveyor



Lab 7 : Math Instructions

Objective

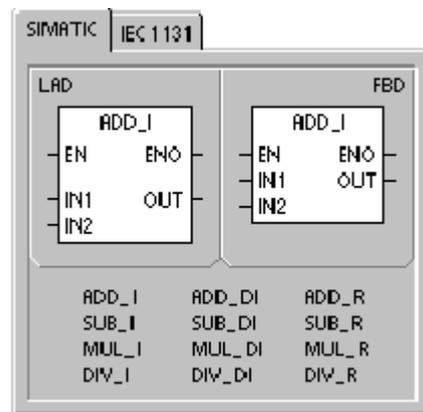
- To use some math instructions in a process control.

Multiply Integer to Double Integer and Divide Integer with Remainder

Multiply Integer to Double Integer

IN1 * IN2 = OUT LAD and FBD
 IN1 * OUT = OUT STL

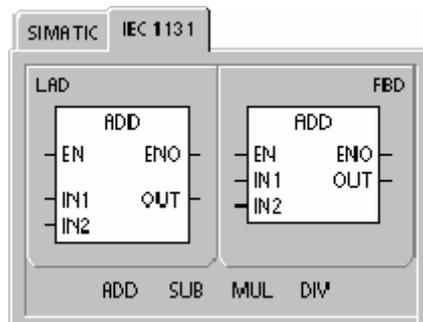
The Multiply Integer to Double Integer instruction (MUL) multiplies two 16-bit integers and produces a 32-bit product. In the STL MUL instruction, the least-significant word (16 bits) of the 32-bit OUT is used as one of the factors.



Divide Integer with Remainder

IN1 / IN2 = OUT LAD and FBD
 OUT / IN1 = OUT STL

The Divide Integer with Remainder instruction (DIV) divides two 16-bit integers and produces a 32-bit result consisting of a 16-bit remainder (the most-significant word) and a 16-bit quotient (the least-significant word). In STL, the least-significant word (16 bits) of the 32-bit OUT is used as the dividend.

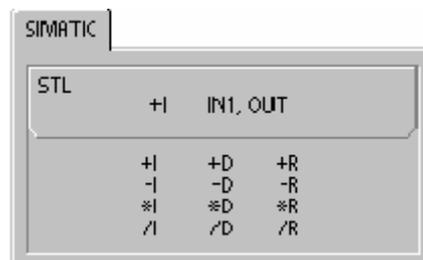


SM Bits and ENO

For both of the instructions on this page, Special memory (SM) bits indicate errors and illegal values. If SM1.3 (divide by zero) is set during a divide operation, then the other math status bits are left unchanged. Otherwise, all supported math status bits contain valid status upon completion of the math operation.

Special Memory bits affected

- SM1.0 (zero)
- SM1.1 (overflow)
- SM1.2 (negative)
- SM1.3 (divide by zero)



Valid Operands for Add, Subtract, Multiply, and Divide Instructions

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

Multiply Integer to Double Integer and Divide Integer with Remainder

Multiply Integer to Double Integer

IN1 * IN2 = OUT LAD

IN1 * OUT = OUT STL

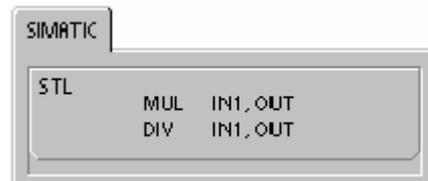
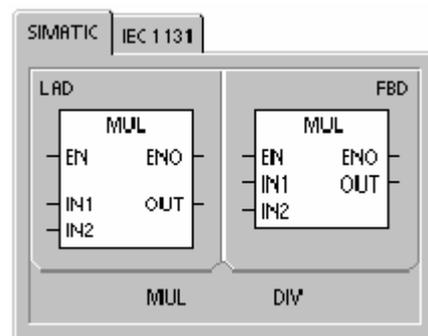
The Multiply Integer to Double Integer instruction (MUL) multiplies two 16-bit integers and produces a 32-bit product. In the STL MUL instruction, the least-significant word (16 bits) of the 32-bit OUT is used as one of the factors.

Divide Integer with Remainder

IN1 / IN2 = OUT LAD

OUT / IN1 = OUT STL

The Divide Integer with Remainder instruction (DIV) divides two 16-bit integers and produces a 32-bit result consisting of a 16-bit remainder (the most-significant word) and a 16-bit quotient (the least-significant word). In STL, the least-significant word (16 bits) of the 32-bit OUT is used as the dividend.



SM Bits and ENO

For both of the instructions on this page, Special Memory (SM) bits indicate errors and illegal values. If SM1.3 (divide by zero) is set during a divide operation, then the other math status bits are left unchanged. Otherwise, all supported math status bits contain valid status upon completion of the math operation.

Special Memory bits affected

- SM1.0 (zero)
- SM1.1 (overflow)
- SM1.2 (negative)
- SM1.3 (divide by zero)

Valid Operands for Multiply Integer to Double Integer and Divide Integer with Remainder

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

Increment and Decrement Instructions

Increment

IN + 1 = OUT *LAD*
 OUT + 1 = OUT *STL*

Decrement

IN -- 1 = OUT *LAD*
 OUT -- 1 = OUT *STL*

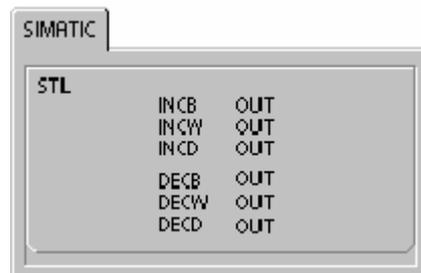
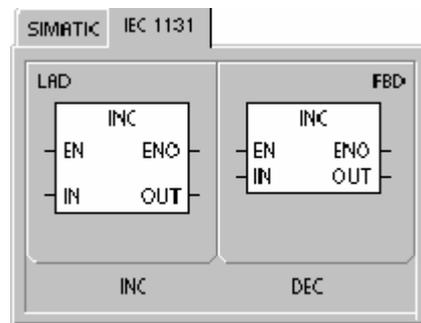
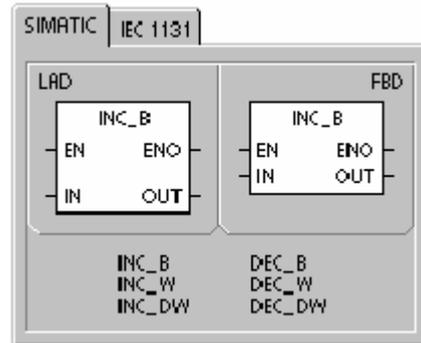
The Increment and Decrement instructions add or subtract 1 to or from the input IN and place the result into the variable OUT.

Increment Byte (INCB) and Decrement Byte (DECB) operations are unsigned. Increment Word (INCW) and Decrement Word (DECW) operations are signed.

Increment Double Word (INCD) and Decrement Double Word (DECD) operations are signed.

Special Memory bits affected:

- SM1.0 (zero)
- SM1.1 (overflow)
- SM1.2 (negative) for Word and Double Word operations



Valid Operands for the Increment and Decrement Instructions

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

Experimental Work

The parking lot which has a capacity of 100 cars is to be controlled by a PLC system. The sensor S1 and S2 are used to count the car at the entrance and exit. If the number of the cars reaches to 100, red light is lit and the gate arm is closed. The arm stays closed until one or more parking space is available in the lot. The gate arm is controlled by activating/deactivating the gate solenoid (GS).

Construct LAD for the parking lot controller by using math instructions. Do not use counter instructions.

Input	Output
Start and stop buttons Sensors S1 and S2	Gate Solenoid Light

