19. STRUCTURED TEXT PROGRAMMING

Topics:
- Basic language structure and syntax
- Variables, functions, values
- Program flow commands and structures
- Function names
- Program Example

Objectives:
- To be able to write functions in Structured Text programs
- To understand the parallels between Ladder Logic and Structured Text
- To understand differences between Allen Bradley and the standard

19.1 INTRODUCTION

If you know how to program in any high level language, such as Basic or C, you will be comfortable with Structured Text (ST) programming. ST programming is part of the IEC 61131 standard. An example program is shown in Figure 261. The program is called *main* and is defined between the statements *PROGRAM* and *END_PROGRAM*. Every program begins with statements the define the variables. In this case the variable *i* is defined to be an integer. The program follows the variable declarations. This program counts from 0 to 10 with a loop. When the example program starts the value of integer memory *i* will be set to zero. The *REPEAT* and *END_REPEAT* statements define the loop. The *UNTIL* statement defines when the loop must end. A line is present to increment the value of *i* for each loop.

```
PROGRAM main
VAR
        i : INT;
END_VAR
i := 0;
REPEAT
        i := i + 1;
        UNTIL i >= 10;
END_REPEAT;
END_PROGRAM
```

Note: Allen Bradley does not implement the standard so that the programs can be written with text only. When programming in RSLogix, only the section indicated to the left would be entered. The variable 'i' would be defined as a tag, and the program would be defined as a task.

Figure 261 A Structured Text Example Program

One important difference between ST and traditional programming languages is the nature of program flow control. A ST program will be run from beginning to end many times each second. A traditional program should not reach the end until it is completely finished. In the previous example the loop could lead to a program that (with some modification) might go into an infinite loop. If this were to happen during a control application the controller would stop responding, the process might become dangerous, and the controller watchdog timer would force a fault.

ST has been designed to work with the other PLC programming languages. For example, a ladder logic program can call a structured text subroutine.

19.2 THE LANGUAGE

The language is composed of written statements separated by semicolons. The statements use predefined statements and program subroutines to change variables. The variables can be explicitly defined values, internally stored variables, or inputs and outputs. Spaces can be used to separate statements and variables, although they are not often necessary. Structured text is not case sensitive, but it can be useful to make variables lower case, and make statements upper case. Indenting and comments should also be used to increase readability and documents the program. Consider the example shown in Figure 262.

GOOD

```
FUNCTION sample
    INPUT_VAR
                start : BOOL;  (* a NO start input *)
                stop : BOOL;  (* a NC stop input *)
    END_VAR
    OUTPUT_VAR
                motor : BOOL;(* a motor control relay
*)
    END_VAR
    motor := (motor + start) * stop;(* get the motor output *)
END_FUNCTION
```

BAD

```
FUNCTION sample
INPUT_VAR
START:BOOL;STOP:BOOL;
END_VAR
OUTPUT_VAR
MOTOR:BOOL;
END_VAR
MOTOR:=(MOTOR+START)*STOP;END_FUNCTION
```

Figure 262 A Syntax and Structured Programming Example

19.2.1 Elements of the Language

ST programs allow named variables to be defined. This is similar to the use of symbols when programming in ladder logic. When selecting variable names they must begin with a letter, but after that they can include combinations of letters, numbers, and some symbols such as '_'. Variable names are not case sensitive and can include any combination of upper and lower case letters. Variable names must also be the same as other key words in the system as shown in Figure 263. In addition, these variable must not have the same name as predefined functions, or user defined functions.

Invalid variable names: START, DATA, PROJECT, SFC, SFC2, LADDER, I/O, ASCII, CAR, FORCE, PLC2, CONFIG, INC, ALL, YES, NO, STRUCTURED TEXT

Valid memory/variable name examples: TESTER, I, I:000, I:000/00, T4:0, T4:0/DN, T4:0.ACC

Figure 263 Acceptable Variable Names

When defining variables one of the declarations in Figure 264 can be used. These define the scope of the variables. The *VAR_INPUT*, *VAR_OUTPUT* and *VAR_IN_OUT* declarations are used for variables that are passed as arguments to the program or function. The *RETAIN* declaration is used to retain a variable value, even when the PLC power has been cycled. This is similar to a latch application. As mentioned before these are not used when writing Allen Bradley programs, but they are used when defining tags to be used by the structured programs.

Declaration	Description
VAR	the general variable declaration
VAR_INPUT	defines a variable list for a function
VAR_OUTPUT	defines output variables from a function
VAR_IN_OUT	defines variable that are both inputs and outputs from a function
VAR_EXTERNAL	
VAR_GLOBAL	a global variable
VAR_ACCESS	
RETAIN	a value will be retained when the power is cycled
CONSTANT	a value that cannot be changed
AT	can tie a variable to a specific location in memory (without this variable locations are chosen by the compiler
END_VAR	marks the end of a variable declaration

Figure 264 Variable Declarations

Examples of variable declarations are given in Figure 265.

Text Program Line	Description
VAR AT %B3:0 : WORD; END_VAR	a word in bit memory
VAR AT %N7:0 : INT; END_VAR	an integer in integer memory
VAR RETAIN AT %O:000 : WORD ; END_VAR	makes output bits retentive
VAR_GLOBAL A AT %I:000/00 : BOOL ; END_VAR	variable ‘A’ as input bit
VAR_GLOBAL A AT %N7:0 : INT ; END_VAR	variable ‘A’ as an integer
VAR A AT %F8:0 : ARRAY [0..14] OF REAL; END_VAR	an array ‘A’ of 15 real values
VAR A : BOOL; END_VAR	a boolean variable ‘A’
VAR A, B, C : INT ; END_VAR	integers variables ‘A’, ‘B’, ‘C’
VAR A : STRING[10] ; END_VAR	a string ‘A’ of length 10
VAR A : ARRAY[1..5,1..6,1..7] OF INT; END_VAR	a 5x6x7 array ‘A’ of integers
VAR RETAIN RTBT A : ARRAY[1..5,1..6] OF INT; END_VAR	a 5x6 array of integers, filled with zeros after power off
VAR A : B; END_VAR	‘A’ is data type ‘B’
VAR CONSTANT A : REAL := 5.12345 ; END_VAR	a constant value ‘A’
VAR A AT %N7:0 : INT := 55; END_VAR	‘A’ starts with 55
VAR A : ARRAY[1..5] OF INT := [5(3)]; END_VAR	‘A’ starts with 3 in all 5 spots
VAR A : STRING[10] := ‘test’; END_VAR	‘A’ contains ‘test’ initially
VAR A : ARRAY[0..2] OF BOOL := [1,0,1]; END_VAR	an array of bits
VAR A : ARRAY[0..1,1..5] OF INT := [5(1),5(2)]; END_VAR	an array of integers filled with 1 for [0,x] and 2 for [1,x]

Figure 265 Variable Declaration Examples

Basic numbers are shown in Figure 266. Note the underline ‘_’ can be ignored, it can be used to break up long numbers, ie. 10_000 = 10000. These are the literal values discussed for Ladder Logic.

number type	examples
integers	-100, 0, 100, 10_000
real numbers	-100.0, 0.0, 100.0, 10_000.0
real with exponents	-1.0E-2, -1.0e-2, 0.0e0, 1.0E2
binary numbers	2#111111111, 2#1111_1111, 2#1111_1101_0110_0101
octal numbers	8#123, 8#777, 8#14
hexadecimal numbers	16#FF, 16#ff, 16#9a, 16#01
boolean	0, FALSE, 1, TRUE

Figure 266 Literal Number Examples

Character strings defined as shown in Figure 267.

example	description
‘ ’	a zero length string
‘ ‘, ‘a’, ‘$”, ‘$$’	a single character, a space, or ‘a’, or a single quote, or a dollar sign $
‘RL’, ‘rl’,‘$0D$0A’	produces ASCII CR, LF combination - end of line characters
‘$P’, ‘$p’	form feed, will go to the top of the next page
‘$T’, ‘4t’	tab
‘this%Tis a test$R$L’	a string that results in ‘this<TAB>is a test<NEXT LINE>’

Figure 267 Character String Data

Basic time and date values are described in Figure 268 and Figure 269. Although it should be noted that for ControlLogix the GSV function is used to get the values.

Time Value	Examples
25ms	T#25ms, T#25.0ms, TIME#25.0ms, T#-25ms, t#25ms
5.5hours	TIME#5.3h, T#5.3h, T#5h_30m, T#5h30m
3days, 5hours, 6min, 36sec	TIME#3d5h6m36s, T#3d_5h_6m_36s

Figure 268 Time Duration Examples

description	examples
date values	DATE#1996-12-25, D#1996-12-25
time of day	TIME_OF_DAY#12:42:50.92, TOD#12:42:50.92
date and time	DATE_AND_TIME#1996-12-25-12:42:50.92, DT#1996-12-25-12:42:50.92

Figure 269 Time and Date Examples

The math functions available for structured text programs are listed in Figure 270. It is worth noting that these functions match the structure of those available for ladder logic. Other, more advanced, functions are also available - a general rule of thumb is if a function is available in one language, it is often available for others.

:=	assigns a value to a variable
+	addition
-	subtraction
/	division
*	multiplication
MOD(A,B)	modulo - this provides the remainder for an integer divide A/B
SQR(A)	square root of A
FRD(A)	from BCD to decimal
TOD(A)	to BCD from decimal
NEG(A)	reverse sign +/-
LN(A)	natural logarithm
LOG(A)	base 10 logarithm
DEG(A)	from radians to degrees
RAD(A)	to radians from degrees
SIN(A)	sine
COS(A)	cosine
TAN(A)	tangent
ASN(A)	arcsine, inverse sine
ACS(A)	arccosine - inverse cosine
ATN(A)	arctan - inverse tangent
XPY(A,B)	A to the power of B
A**B	A to the power of B

Figure 270 Math Functions

Functions for logical comparison are given in Figure 271. These will be used in expressions such as IF-THEN statements.

>	greater than
>=	greater than or equal
=	equal
<=	less than or equal
<	less than
<>	not equal

Figure 271 Comparisons

Boolean algebra functions are available, as shown in Figure 272. The can be applied to bits or integers.

AND(A,B)	logical and
OR(A,B)	logical or
XOR(A,B)	exclusive or
NOT(A)	logical not
!	logical not (note: not implemented on AB controllers)

Figure 272 Boolean Functions

The precedence of operations are listed in Figure 273 from highest to lowest. As normal expressions that are the most deeply nested between brackets will be solved first. (Note: when in doubt use brackets to ensure you get the sequence you expect.)

highest priority ↑

! - (Note: not available on AB controllers)
()
functions
XPY, **
negation
SQR, TOD, FRD, NOT, NEG, LN, LOG, DEG, RAD, SIN, COS, TAN, ASN, ACS, ATN
*, /, MOD
+, -
>, >=, =, <=, <, <>
AND (for word)
XOR (for word)
OR (for word)
AND (bit)
XOR (bit)
OR (bit)
ladder instructions

Figure 273 Operator Precedence

Common language structures include those listed in Figure 274.

IF-THEN-ELSIF-ELSE-END_IF;	normal if-then structure
CASE-value:-ELSE-END_CASE;	a case switching function
FOR-TO-BY-DO-END_FOR;	for-next loop
WHILE-DO-END_WHILE;	

Figure 274 Flow Control Functions

Special instructions include those shown in Figure 275.

```
RETAIN();        causes a bit to be retentive
IIN();           immediate input update
EXIT;            will quit a FOR or WHILE loop
EMPTY
```

Figure 275 Special Instructions

19.2.2 Putting Things Together in a Program

Consider the program in Figure 276 to find the average of five values in a real array 'f[]'. The FOR loop in the example will loop five times adding the array values. After that the sum is divided to get the average.

```
avg := 0;
FOR (i := 0 TO 4) DO
    avg := avg + f[i];
END_FOR;
avg := avg / 5;
```

Figure 276 A Program To Average Five Values In Memory With A For-Loop

The previous example is implemented with a WHILE loop in Figure 277. The main differences is that the initial value and update for 'i' must be done manually.

```
avg := 0;
i := 0;
WHILE (i < 5) DO
    avg := avg + f[i];
    i := i + 1;
END_WHILE;
avg := avg / 5;
```

Figure 277 A Program To Average Five Values In Memory With A While-Loop

The example in Figure 278 shows the use of an IF statement. The example begins with a timer. These are handled slightly differently in ST programs. In this case if 'b' is true the timer will be active, if it is false the timer will reset. The second instruction calls 'TONR' to update the timer. (Note: ST programs use the FBD_TIMER type, instead of the TIMER type.) The IF statement works as normal, only one of the three cases will occur with the ELSE defining the default if the other two fail.

```
t.TimerEnable := b;
TONR(t);
IF (a = 1) THEN
        x := 1;
ELSIF (b = 1 AND t.DN = 1) THEN
        y := 1;
        IF (I:000/02 = 0) THEN
            z := 1;
        END_IF;
ELSE
        x := 0;
        y := 0;
        z := 0;
END_IF;
```

Figure 278 Example With An If Statement

Figure 279 shows the use of a CASE statement to set bits 0 to 3 of ’a’ based upon the value of ’test’. In the event none of the values are matched, ’a’ will be set to zero, turning off all bits.

```
CASE test OF
        0:
           a.0 := 1;
        1:
           a.1 := 1;
        2:
           a.2 := 1;
        3:
           a.3 := 1;
ELSE
        a := 0;
END_CASE;
```

Figure 279 Use of a Case Statement

The example in Figure 280 accepts a BCD input from ’bcd_input’ and uses it to change the delay time for TON delay time. When the input ’test_input’ is true the time will count. When the timer is done ’set’ will become true.

```
FRD (bcd_input, delay_time);
t.PRE := delay_time;
IF (test_input) THEN
	t.EnableTimer := 1;
ELSE
	t.EnableTimer := 0;
END_IF;
TONR(t);
set := t.DN;
```

Figure 280 Function Data Conversions

Most of the IEC61131-3 defined functions with arguments are given in Figure 281. Some of the functions can be overloaded, for example ADD could have more than two values to add, and others have optional arguments. In most cases the optional arguments are things like preset values for timers. When arguments are left out they default to values, typically 0. ControlLogix uses many of the standard function names and arguments but does not support the overloading part of the standard.

Function	Description
ABS(A);	absolute value of A
ACOS(A);	the inverse cosine of A
ADD(A,B,...);	add A+B+...
AND(A,B,...);	logical and of inputs A,B,...
ASIN(A);	the inverse sine of A
ATAN(A);	the inverse tangent of A
BCD_TO_INT(A);	converts a BCD to an integer
CONCAT(A,B,...);	will return strings A,B,... joined together
COS(A);	finds the cosine of A
CTD(CD:=A,LD:=B,PV:=C);	down counter active <=0, A decreases, B loads preset
CTU(CU:=A,R:=B,PV:=C);	up counter active >=C, A decreases, B resets
CTUD(CU:=A,CD:=B,R:=C,LD:=D,PV:=E);	up/down counter combined functions of the up and down counters
DELETE(IN:=A,L:=B,P:=C);	will delete B characters at position C in string A
DIV(A,B);	A/B
EQ(A,B,C,...);	will compare A=B=C=...
EXP(A);	finds e**A where e is the natural number
EXPT(A,B);	A**B
FIND(IN1:=A,IN2:=B);	will find the start of string B in string A
F_TRIG(A);	a falling edge trigger
GE(A,B,C,...);	will compare A>=B, B>=C, C>=...
GT(A,B,C,...);	will compare A>B, B>C, C>...
INSERT(IN1:=A,IN2:=B,P:=C);	will insert string B into A at position C
INT_TO_BCD(A);	converts an integer to BCD
INT_TO_REAL(A);	converts A from integer to real
LE(A,B,C,...);	will compare A<=B, B<=C, C<=...
LEFT(IN:=A,L:=B);	will return the left B characters of string A
LEN(A);	will return the length of string A
LIMIT(MN:=A,IN:=B,MX:=C);	checks to see if B>=A and B<=C
LN(A);	natural log of A
LOG(A);	base 10 log of A
LT(A,B,C,...);	will compare A<B, B<C, C<...
MAX(A,B,...);	outputs the maximum of A,B,...
MID(IN:=A,L:=B,P:=C);	will return B characters starting at C of string A
MIN(A,B,...);	outputs the minimum of A,B,...
MOD(A,B);	the remainder or fractional part of A/B
MOVE(A);	outputs the input, the same as :=
MUL(A,B,...);	multiply values A*B*....
MUX(A,B,C,...);	the value of A will select output B,C,...
NE(A,B);	will compare A <> B
NOT(A);	logical not of A
OR(A,B,...);	logical or of inputs A,B,...

Function	Description
REAL_TO_INT(A);	converts A from real to integer
REPLACE(IN1:=A,IN2:=B,L:= C,P:=D);	will replace C characters at position D in string A with string B
RIGHT(IN:=A,L:=B);	will return the right A characters of string B
ROL(IN:=A,N:=B);	rolls left value A of length B bits
ROR(IN:=A,N:=B);	rolls right value A of length B bits
RS(A,B);	RS flip flop with input A and B
RTC(IN:=A,PDT:=B);	will set and/or return current system time
R_TRIG(A);	a rising edge trigger
SEL(A,B,C);	if a=0 output B if A=1 output C
SHL(IN:=A,N:=B);	shift left value A of length B bits
SHR(IN:=A,N:=B);	shift right value A of length B bits
SIN(A);	finds the sine of A
SQRT(A);	square root of A
SR(S1:=A,R:=B);	SR flipflop with inputs A and B
SUB(A,B);	A-B
TAN(A);	finds the tangent of A
TOF(IN:=A,PT:=B);	off delay timer
TON(IN:=A,PT:=B);	on delay timer
TP(IN:=A,PT:=B);	pulse timer - a rising edge fires a fixed period pulse
TRUNC(A);	converts a real to an integer, no rounding
XOR(A,B,...);	logical exclusive or of inputs A,B,...

Figure 281 Structured Text Functions

Control programs can become very large. When written in a single program these become confusing, and hard to write/debug. The best way to avoid the endless main program is to use subroutines to divide the main program. The IEC61131 standard allows the definition of subroutines/functions as shown in Figure 282. The function will accept up to three inputs and perform a simple calculation. It then returns one value. As mentioned before ControlLogix does not support overloading, so the function would not be able to have a variable size argument list.

```
....
D := TEST(1.3, 3.4); (* sample calling program, here C will default to 3.14 *)
E := TEST(1.3, 3.4, 6.28); (* here C will be given a new value *)
....

FUNCTION TEST : REAL
        VAR_INPUT A, B : REAL; C : REAL := 3.14159; END VAR
        TEST := (A + B) / C;
END_FUNCTION
```

Figure 282 Declaration of a Function

19.3 AN EXAMPLE

The example beginning in Figure 284 shows a subroutine implementing traffic lights in ST for the ControlLogix processor. The variable ’state’ is used to keep track of the current state of the lights. Timer enable bits are used to determine which transition should be checked. Finally the value of ’state’ is used to set the outputs. (Note: this is possible because ’=’ and ’:=’ are not the same.) This subroutine would be stored under a name such as ’TrafficLights’. It would then be called from the main program as shown in Figure 283.

Figure 283 The Main Traffic Light Program

```
SBR();
        IF S:FS THEN
                state := 0;
                green_EW.TimerEnable := 1;
                yellow_EW.TimerEnable := 0;
                green_NS.TimerEnable := 0;
                yellow_NS.TimerEnable := 0;
        END_IF;

        TONR(green_EW); TONR(yellow_EW);
        TONR(green_NS); TONR(yellow_NS);

        CASE state OF
        0:      IF green_EW.DN THEN
                        state :=1;
                        green_EW.TimerEnable := 0;
                        yellow_EW.TimerEnable := 1;
                END_IF
        1:      IF yellow_EW.DN THEN
                        state :=2;
                        yellow_EW.TimerEnable := 0;
                        green_NS.TimerEnable := 1;
                END_IF
        2:      IF green_NS.DN THEN
                        state :=3;
                        green_NS.TimerEnable := 0;
                        yellow_NS.TimerEnable := 1;
                END_IF
        3:      IF yellow_NS.DN THEN
                        state :=0;
                        yellow_NS.TimerEnable := 0;
                        green_EW.TimerEnable := 1;
                END_IF

light_EW_green := (state = 0);
light_EW_yellow := (state = 1);
light_EW_red := (state = 2) OR (state = 3);
light_NS_green := (state = 2);
light_NS_yellow := (state = 3);
light_NS_red := (state = 0) OR (state = 1);

RET();
```

Note: This example is for the AB ControlLogix platform, so it does not show the normal function and tag definitions. These are done separately in the tag editor.

```
state : DINT
green_EW : FBD_TIMER
yellow_EW : FBD_TIMER
green_NS : FBD_TIMER
yellow_NS : FBD_TIMER
light_EW_green : BOOL alias =
    rack:1:O.Data.0
light_EW_yellow : BOOL alias =
    rack:1:O.Data.1
light_EW_red : BOOL alias =
    rack:1:O.Data.2
light_NS_green : BOOL alias =
    rack:1:O.Data.3
light_NS_yellow : BOOL alias =
    rack:1:O.Data.4
light_NS_red : BOOL alias =
    rack:1:O.Data.5
```

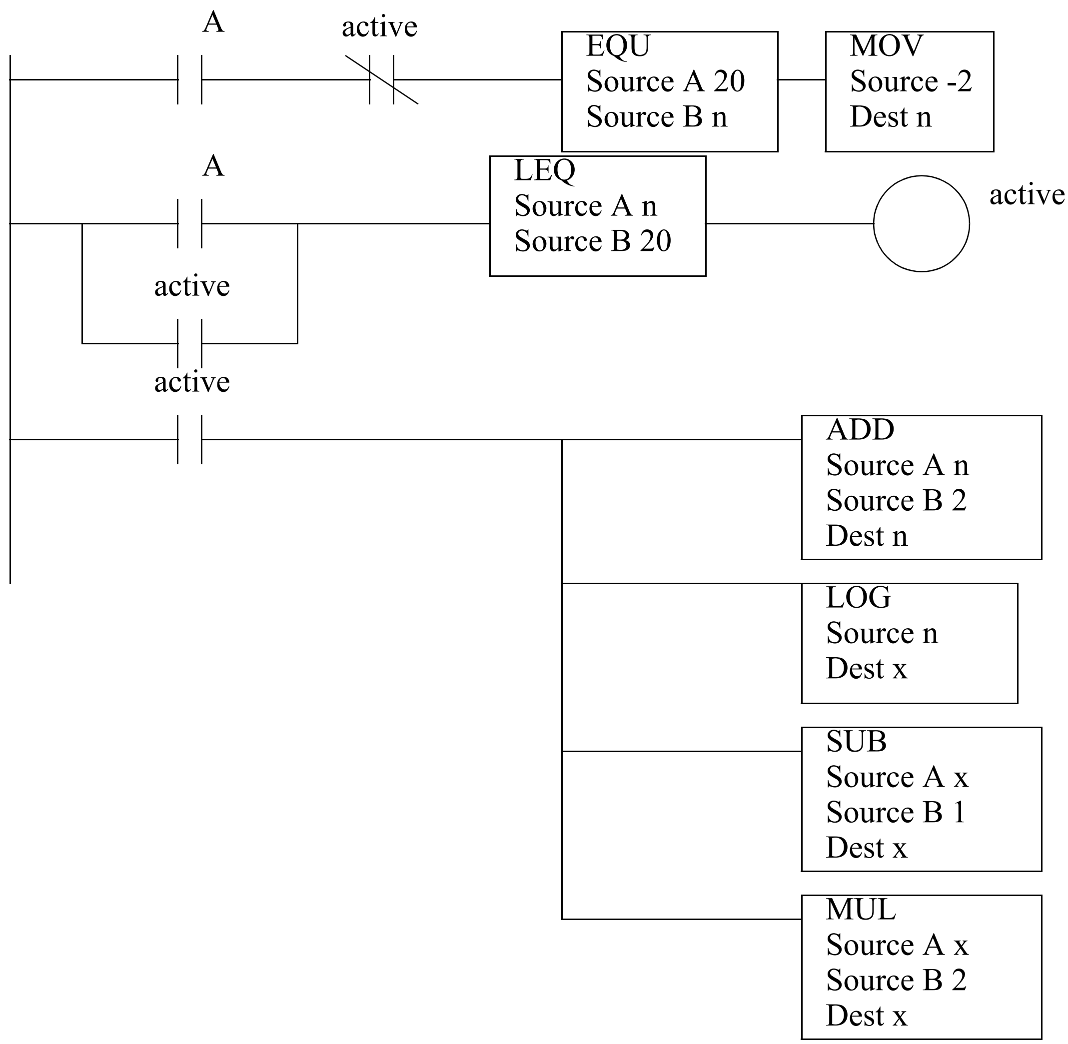
Figure 284 Traffic Light Subroutine

19.4 SUMMARY

- Structured text programming variables, functions, syntax were discussed.
- The differences between the standard and the Allen Bradley implementation were indicated as appropriate.
- A traffic light example was used to illustrate a ControlLogix application

19.5 PRACTICE PROBLEMS

1. Write a structured text program that will replace the following ladder logic.

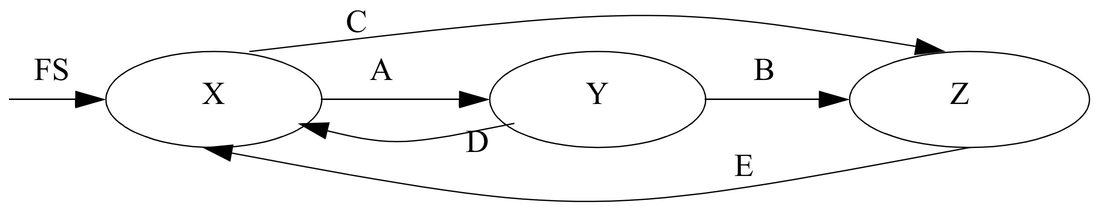


2. Implement the following Boolean equations in a Structured Text program. If the program was for a state machine what changes would be required to make it work?

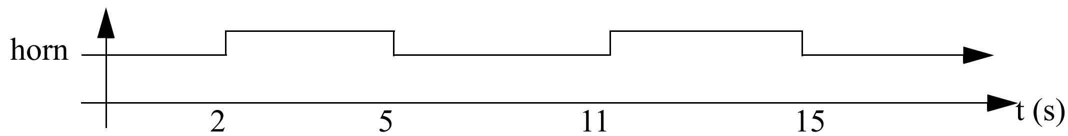
$$light = (light + dark \bullet switch) \bullet \overline{\overline{switch} \bullet light}$$

$$dark = (dark + light \bullet \overline{switch}) \bullet \overline{switch \bullet dark}$$

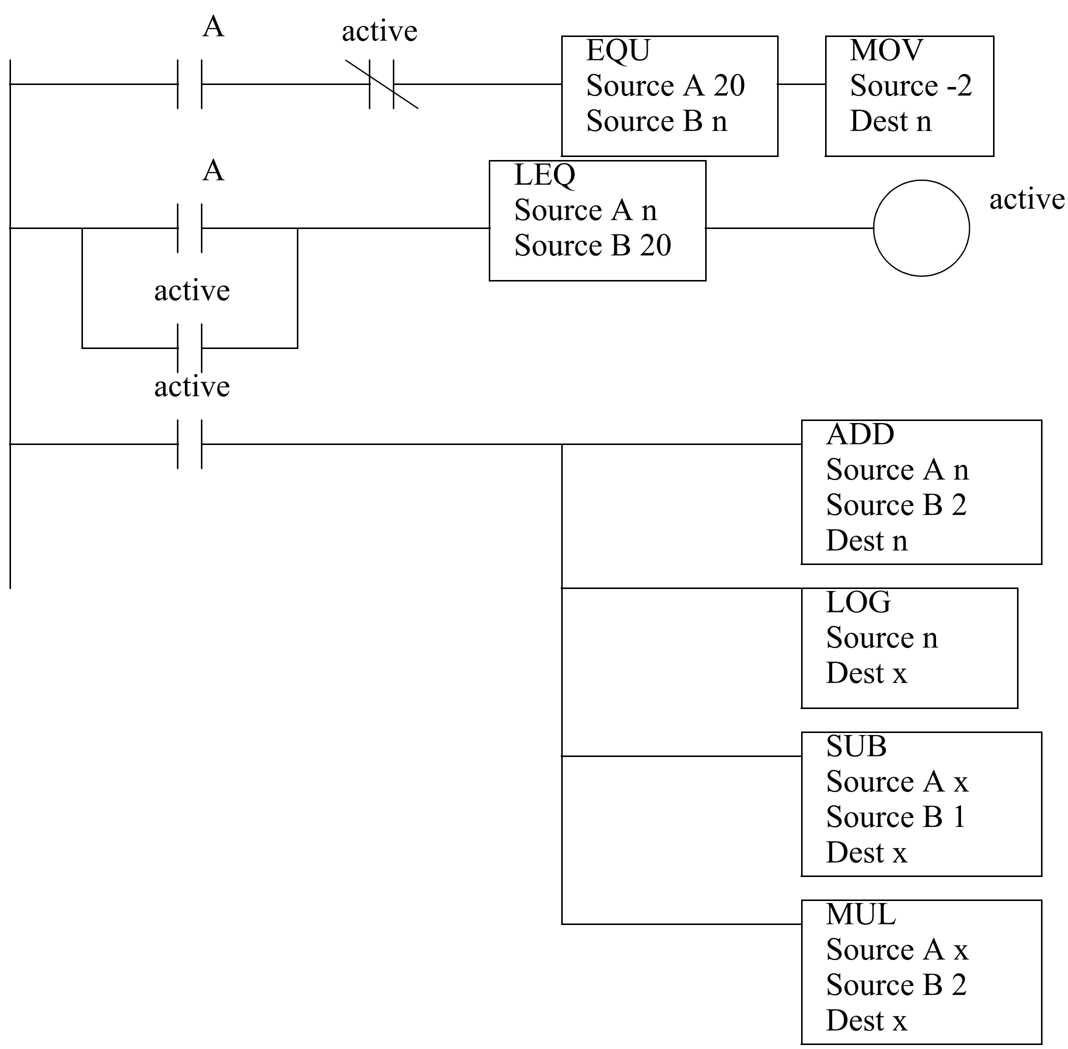
3. Convert the following state diagram to a Structured Text program.



4. A temperature value is stored in F8:0. When it rises above 40 the following sequence should occur once. Write a ladder logic program that implement this function with a Structured Text program.



5. Write a structured text program that will replace the following ladder logic.



19.6 ASSIGNMENT PROBLEMS

1. Write logic for a traffic light controller using structured text.

2. Write a structured text program to control a press that has an advance and retract with limit switches. The press is started and stopped with start and stop buttons.

3. Write a structured text program to sort a set of ten integer numbers and then find the median value.