STEP 7
Practical Help

1 Program conception

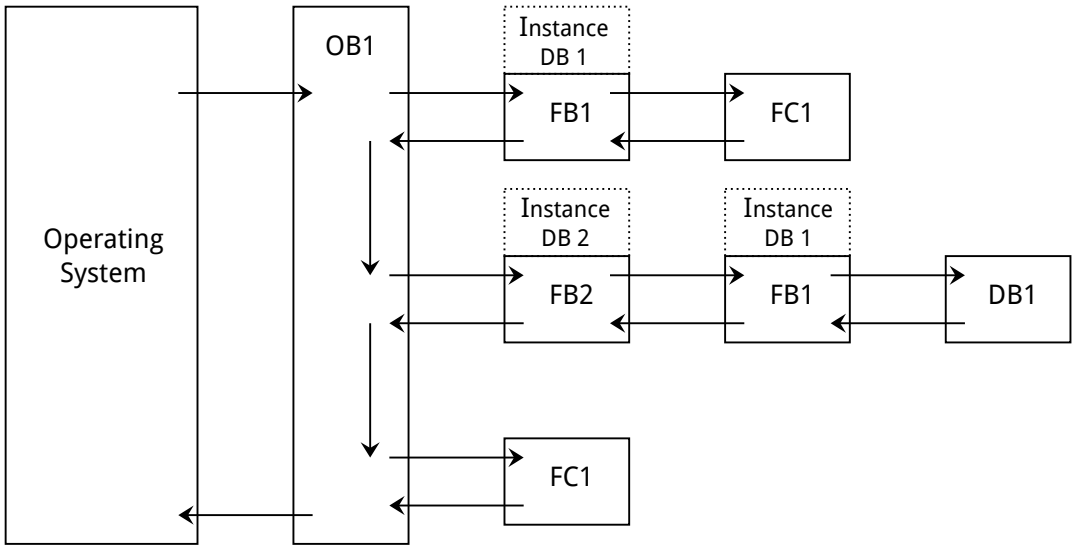
1.1 Generalities

The user program, loaded in the CPU, represents all the automation process. It is organized in BLOCS.

- Organization Blocs (OB) : define the user program structure
- Functional Blocs (FB) : associated to a Instance Data Bloc (DB)
- Function Blocs (FC) : contain all the frequently used function
- System Functional Blocs (SFB) : integrated to the CPU, they are for main system functions
- System Function Blocs (SFC) : integrated to the CPU
- Instance Data Blocs (DB) : associated to Functional and System Functional Blocs when those are called
- Global Data Blocs (DB) : contain user data shared with all the blocs

The number and the size of the bloc depend of the considered CPU.

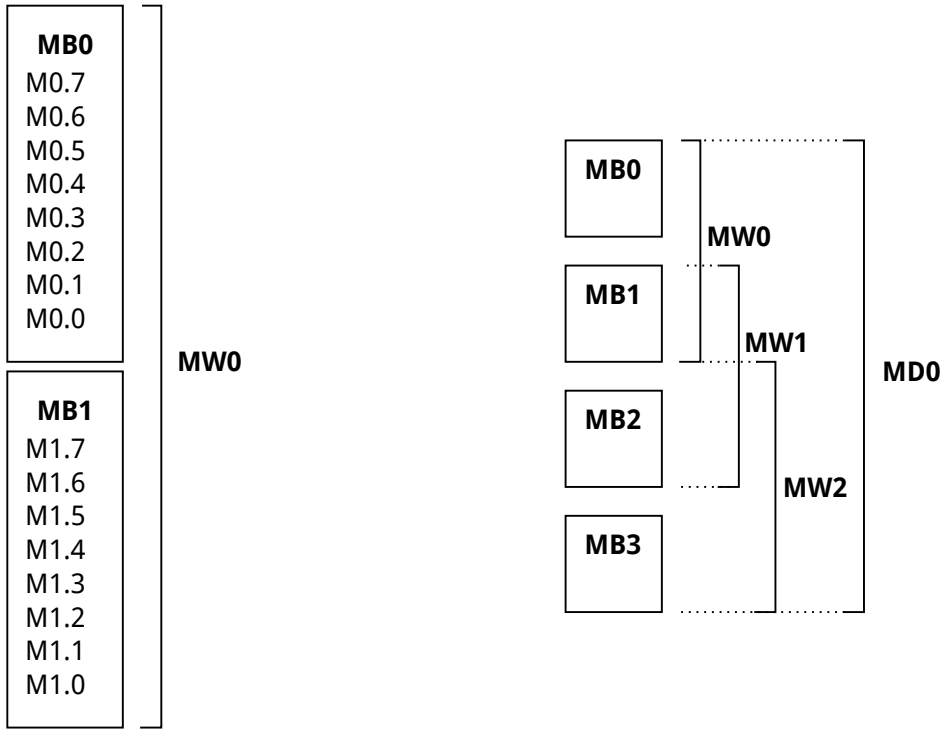
1.2 Example of a program structure



OB1 is the "conductor", it organizes the other blocs.
FB1 and FB2 have their own data (in the corresponding instance DB).
DB1 corresponds to a global data bloc (the data in this bloc can be used by the others).
Remark : a short program could be entirely implemented in OB1.

1.3 The PLC memory

STEP 7 considers bits (e.g. M0.0), bytes (e.g. MB3), 16 bit words (e.g. MW1) and double words (e.g. MD4). Note that all those types share the same memory space : the word MW0 is composed of the byte MB0 and the byte MB1, and the byte MB0 is composed of the 8 bits from M0.0 to M0.7.



During the programming it is needed to be careful not to use memory already used !

2 LIST language

2.1 Basic instructions

Operator AND

```
U       E1.0    //if the input E1.0 is true (equals "1")
U       E1.1    //and if the input E2.0 is true (equals "1")
=       A4.0    //then the output A4.0 is set to "1"
```

an other example :

```
U       E1.0    //if the input E1.0 is true (equals "1")
U       E1.1    //and if the input E2.0 is true (equals "1")
U       M3.0    //and if the bit M3.0 is true (equals "1")
U       M3.1    //and if the bit M3.1 is true (equals "1")
=       M3.2    //then the bit M3.2 is set to "1"
=       M3.3    //and the bit M3.3 is set to "1"
```

Operator AND NOT

```
U       E1.0    //if the input E1.0 is true (equals "1")
UN      E1.1    //and if the input E2.0 is false (equals "0")
=       A4.0    //then the output A4.0 is set to "1"
```

Operator OR

```
O       E1.0    //if the input E1.0 is true (equals "1")
O       E1.1    //or if the input E2.0 is true (equals "1")
=       A4.0    //then the output A4.0 is set to "1"
```

Operator OR NOT

```
O       E1.0    //if the input E1.0 is true (equals "1")
ON      E1.1    //or if the input E2.0 is false (equals "0")
=       A4.0    //then the output A4.0 is set to "1"
```

Operator EXCLUSIVE OR

```
X       E1.0    //if the input E1.0 is true and E1.1 is false
X       E1.1    //or if the input E1.0 is false and E1.1 is true
=       A4.0    //then the output A4.0 is set to "1"
```

More examples :

The boolean equation : **(E0.0 . M10.0) + (E0.2 . M0.3) + M10.1 = A4.0** can be written

```
U       E0.0
U       M10.0
O
U       E0.2
U       M0.3
O       M10.1
=       A4.0
```

The boolean equation : **(E0.0 + M10.0) . (E0.2 + M0.3) . M10.1 = A4.0** can be written

```
U(
O       E0.0
O       M10.0
)
U(
O       E0.2
O       M0.3
)
U       M10.1
=       A4.0
```

It is possible to use intermediate variables

```
O       E0.0
O       M10.0
=       M0.0
O       E0.2
O       M0.3
=       M0.1
```

```
U       M0.0
U       M0.1
U       M10.1
=       A4.0
```

Operators SET and RESET

```
U       E1.0    //if the input E1.0 is true
S       A4.0    //then set the output A4.0
U       E1.1    //if the input E1.1 is true
R       A4.0    //then reset the output A4.0
```

Other keywords

```
NOT     //inverse the value
SET     //set the value to true
CLR     //set the value to false
```

2.2 Delaying

It is possible to open several delaying at the same time (the number depends of the considered CPU, generally 256).

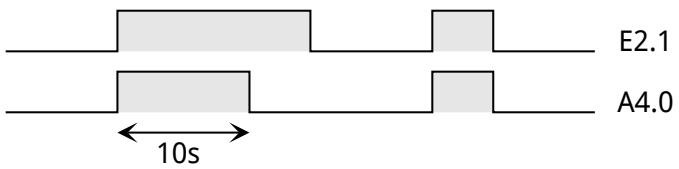
It exists 5 simple modes named SI, SV, SE, SS and SA.

At the beginning of each delaying, the time value must be stored in ACCU1 (an accumulator of the CPU). Here is the syntax to do so : **S5T#aH_bM_cS_dMS** with H hours, M minutes, S seconds and MS milliseconds.

Examples : S5T#10S stores 10 seconds and S5T#1M_35S_6MS stores 1 minute and 36.006 seconds.

Delaying SI

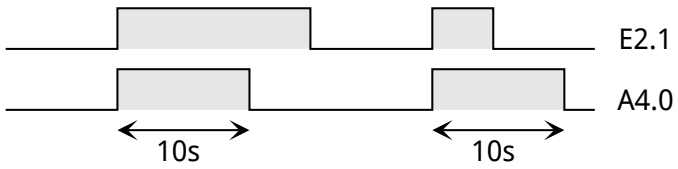
```
U       E2.1    //if the input E2.1 is true (equals "1")
L       S5T#10S //store the duration in ACCU1 (10 seconds here)
SI      T1      //activate the delaying T1
U       T1      //while T1 is true
=       A4.0    //the output A4.0 is true
```



Note that when E2.1 goes down, A4.0 goes down even if less than 10s occur.

Delaying SV

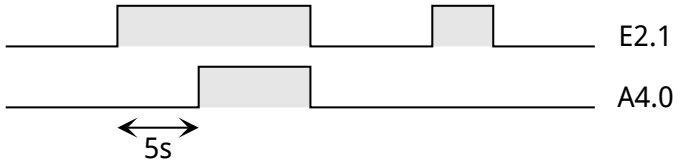
```
U       E2.1    //if the input E2.1 is true (equals "1")
L       S5T#10S //store the duration in ACCU1 (10 seconds here)
SV      T2      //activate the delaying T2
U       T2      //while T2 is true
=       A4.0    //the output A4.0 is true
```



Note that when E2.1 goes down, A4.0 remains activated if the 10s are not done

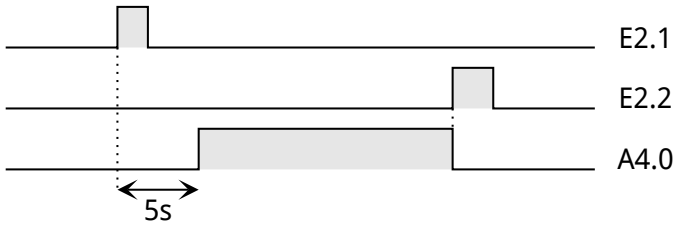
Delaying SE

```
U       E2.1    //if the input E2.1 is true (equals "1")
L       S5T#5S  //store the duration in ACCU1 (5 seconds here)
SE      T3      //activate the delaying T3
U       T3      //while T3 is true
=       A4.0    //the output A4.0 is true
```



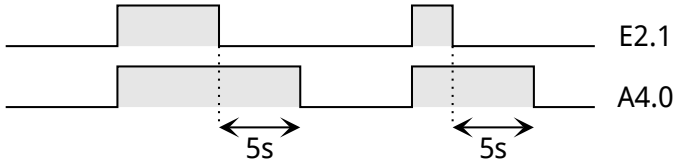
Delaying SS

```
U       E2.1    //if the input E2.1 is true (equals "1")
L       S5T#5S  //store the duration in ACCU1 (5 seconds here)
SS      T4      //activate the delaying T4
U       E2.2    //if the input E2.2 is true (equals "1")
R       T4      //stop the delaying T4
U       T4      //while T4 is true
=       A4.0    //the output A4.0 is true
```



Delaying SA

```
U       E2.1    //if the input E2.1 is true (equals "1")
L       S5T#5S  //store the duration in ACCU1 (5 seconds here)
SA      T5      //activate the delaying T5
U       T5      //while T5 is true
=       A4.0    //the output A4.0 is true
```



2.3 Counter

It is possible to open several counters at the same, the number of counters depends of the considered CPU (usually 256). They count from 0 to 999.

Example :

```
U       E0.0    //if the inpuy E0.0 is true (equals "1")
L       C#3     //the value 3 is stored in ACCU1
S       Z1      //the counter Z1 in initialize with the value 3
...
...
U       E0.1    //if E0.1 is true (equals "1")
ZV      Z1      //increments Z1 (considering a rising edge over E0.1)
...
...
U       E0.2    //if E0.2 is true (equals "1")
ZR      Z1      //decrements Z1 (considering a rising edge over E0.2)
...
...
R       Z1      //reset Z1 to 0 (as if it was a bit)
...
...
U       Z1      //equals "1" if Z1 is not equal to 0 (as if it was a bit)
...
...
UN      Z1      //equals "1" if Z1 is equal to 0 (as if it was a bit)
```

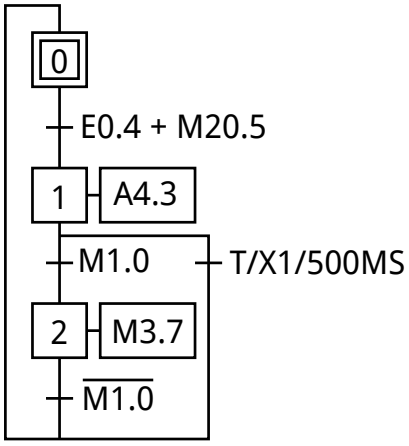
The test of a counter value can be done using the notation :

```
L       40      //store 40 in ACCU2
L       C1      //store the value of C1 in ACCU1
<=      I       //test if ACCU2<=ACCU1
=       A4.0    //if 40<=C1 then A4.0 is active

L       50      //store 50 in ACCU2
L       C2      //store the value of C2 in ACCU1
>       I       //test if ACCU2>ACCU1
=       A4.1    //if 50>C2 then A4.1 is active
```

3 Translation grafcet to LIST

Let's consider the following grafcet :



Here is its a translation in LIST language :

```
//INITIALISATION - FB1

S       M0.0    //set initial state
R       M0.1    //reset others
R       M0.2

//GRAFCET - FB3 (in [FB2;FB9])

U       M0.0    //if we are in initial state
U       (
        O       E0.4
        O       M20.5
)       //and we have E0.4 or M20.5
R       M0.0    //then reset the initial state
S       M0.1    //and set the next one

U       M0.1    //if we are in the second state
U       M1.0    //and M1.0 is true
R       M0.1    //then reset the second state
S       M0.2    //and set the third one

U       M0.1    //if we are in the second state
U       T1      //and T1 is true
R       M0.1    //then reset the second state
S       M0.0    //and set the first (initial) one

U       M0.2    //if we are in the third state
UN      M1.0    //and M1.0 is false
R       M0.2    //then reset the second state
S       M0.0    //and set the first (initial) one
```

```
//OUTPUTS - FB10

U       M0.1    //if we are in the second state (state 1)
=       A4.3    //then the output A4.3 is active

U       M0.2    //if we are in the third state (state 2)
=       M3.7    //then the memory bit M3.7 is true, else it is false

U       M0.1    //if we are in the second state (state 1)
L       S5T#500MS       //define the duration of the delay to 500ms
SE      T1      //and start the delay T1

//THE CALL OF THE FUNCTION BLOC - OB1
CALL    FB3,    DB3
```