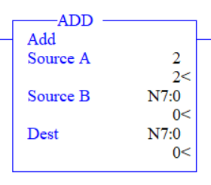
Title: **Math Functions and One Shots** Handout: 6

Course: Introduction to Automation Unit: Introduction of PLC CLO: 4

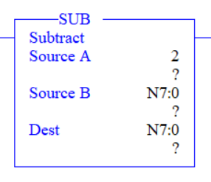
**Objectives**

1. Student shall develop a knowledge of the ADD, SUB, MUL, DIV instructions.
2. Student shall develop a knowledge of the One-Shot (OSR/OSF) input instructions.

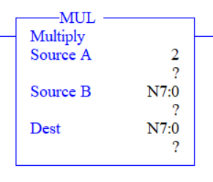
**Theory**

Addition (ADD)

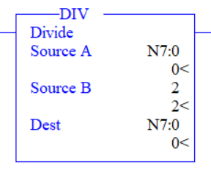
When the instruction gains rung continuity, this output instruction adds Source A to Source B and stores the sum in the destination address. See Notes A and B below.

Subtraction (SUB)

When the instruction gains rung continuity, this output instruction subtracts Source B from Source A and stores the difference in the destination address. See Notes A and B below.

Multiply (MUL)

When the instruction gains rung continuity, this output instruction multiplies Source A to Source B and places the product in the destination address. See Notes A and B below.

Divide (DIV)

When the instruction gains rung continuity, this output instruction divides Source A by Source B and places the quotient in the destination address. The value stored in the destination is rounded. The value stored in the math register consists of the unrounded quotient placed in the most significant word (S:13) and the remainder placed in the least significant word (S14). See Notes A and B below.

**Note A:** Source A and Source B can either be constant values or addresses that contain values, however Source A and Source B cannot both be constants.

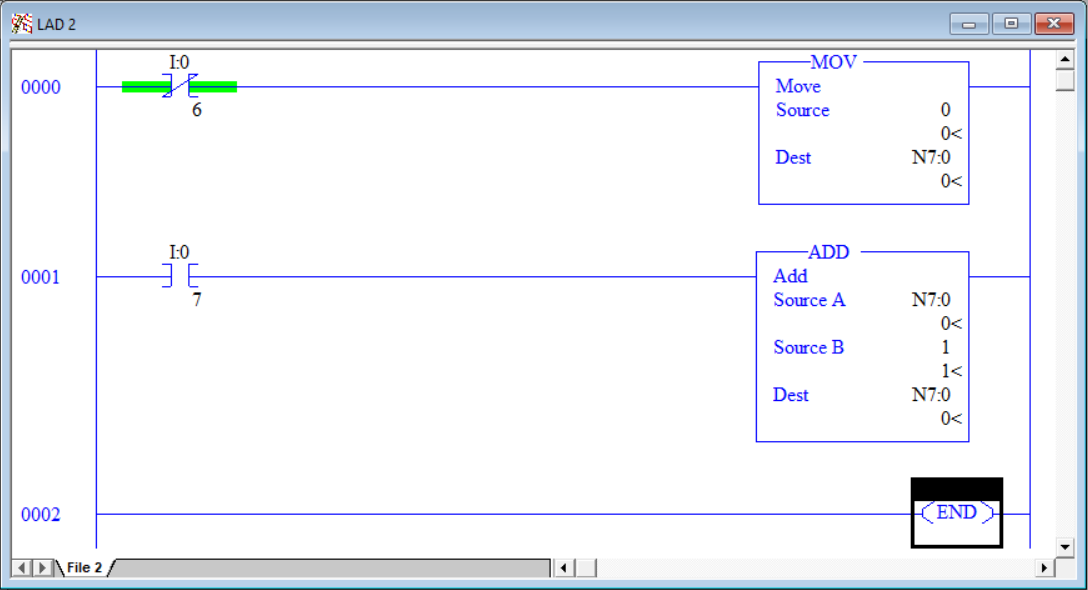
**Note B:** The MicroLogix processor is capable of 32-bit addition and subtraction meaning results can be as low as -32,768 or as great as +32,767. Exceeding either limit will set the S:0/1 (overflow bit) and S:5/0 (overflow trap bit). Monitor bit S:5/0 in your program to avoid this potentially dangerous situations.

**Devices**

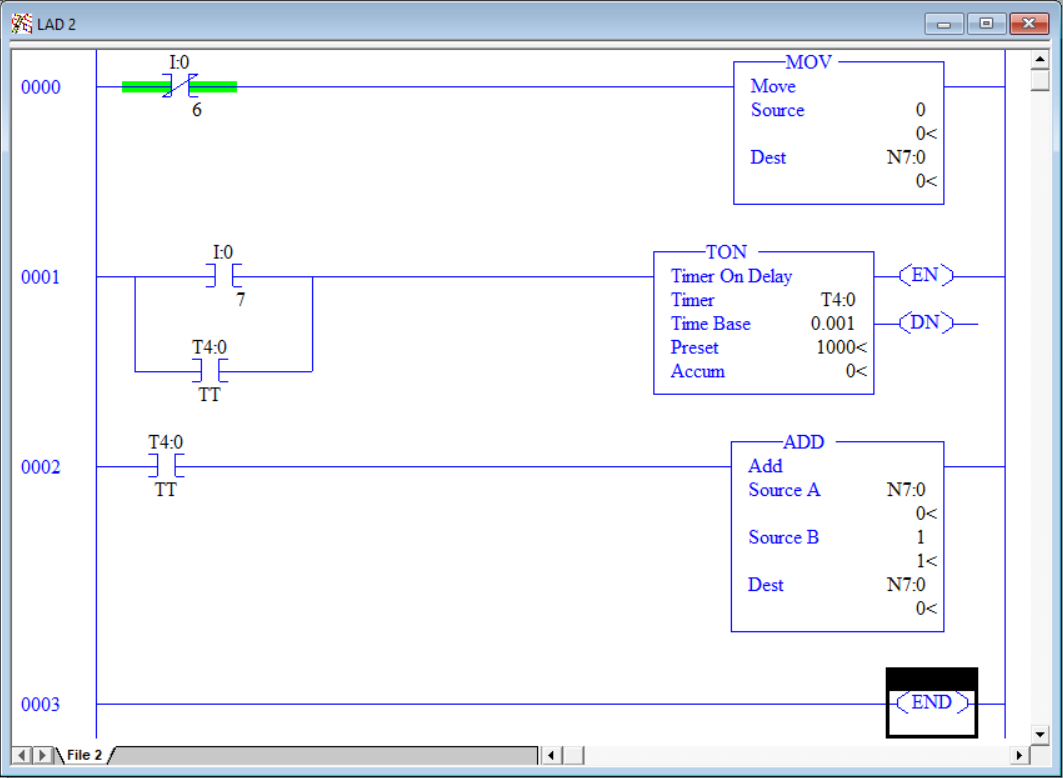
|  |  |  |
| --- | --- | --- |
| Inputs | | |
| *Device* | *Description* | *Symbol* |
| Two-position Selector Switch | Select Add/Subtract or Multiply/Divide | SEL\_MTH |
| NC Pushbutton (PB1) | Reset | RST |
| NO Pushbutton (PB2) | Add or Multiply | ADD\_MUL |
| NO Pushbutton (PB3) | Subtract or Divide | SUB\_DIV |
| Outputs | | |
| *Device* | *Description* | *Symbol* |
|  |  |  |

**Instructions**

Program the follow logic.

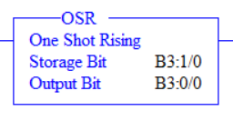


1. Download the program but **do not** press any buttons.
2. Rung 0000 resets the integer N7:0 back to 0 when PB1 is pressed.
3. Rung 0001 is constructed to add 1 to the integer N7:0 when PB2 is pressed.
4. Do you think the logic will work? Press PB2 and observe what happens. Did N7:0 be incremented by 1? The issue is that 1 is being added the N7:0 **every scan** that the rung has continuity. Since the PLC scans very fast, as long as you are holding the button down, each scan will add 1 to N7:0. Let’s see how fast our PLC is scanning. Modify the program to represent the logic below.

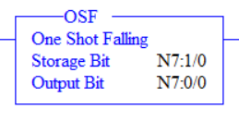


1. This new program will start a 1 second timer . While the timer is timing, the add function in going to add 1 to the integer N7:0 every scan. After one second, the sum in N7:0 will equal the number of scans that occurred during that second.
2. Press and release PB2 quickly. (If you hold the button down for more than a second, the results will be inaccurate.)
3. Observe the sum in N7:0. That number represents the number of scans that occurred during that 1 second. It’s alarming how fast the PLC operates. Scan cycles are completely dependent on the size of the program and complexity of the functions that it needs to perform. Since this program is only three rungs, the processor is not taxed and can scan very quickly.
4. Press PB1 to reset the counter then press and release PB2 quickly to re-run the test. Was the result the exact same number? Repeat this sequence a few times and notice that the result can vary by several scans/second.
5. So now that we know we can’t use the first program to increment the integer stored in N7:0 and have an understanding of just how fast the PLC operates, how can be accomplish capturing an action for only one scan? The answer is one-shots.

**Theory**

One Shot Rising (OSR)

The one shot rising is a conditional input instruction that triggers an event to occur one time. Use this instruction when an event must activate based on the change of rung state from false to true, as triggered by a pushbutton, limit switch or other descrete device. See Note A for address assignments. The output bit shall be set to true for one scan when the rung *gains* logical continuity. Subsequent true rung scans have no affect on the output bit. The rung must lose continuity then regain continuity for the output bit to be set true again.

One Shot Falling (OSF)

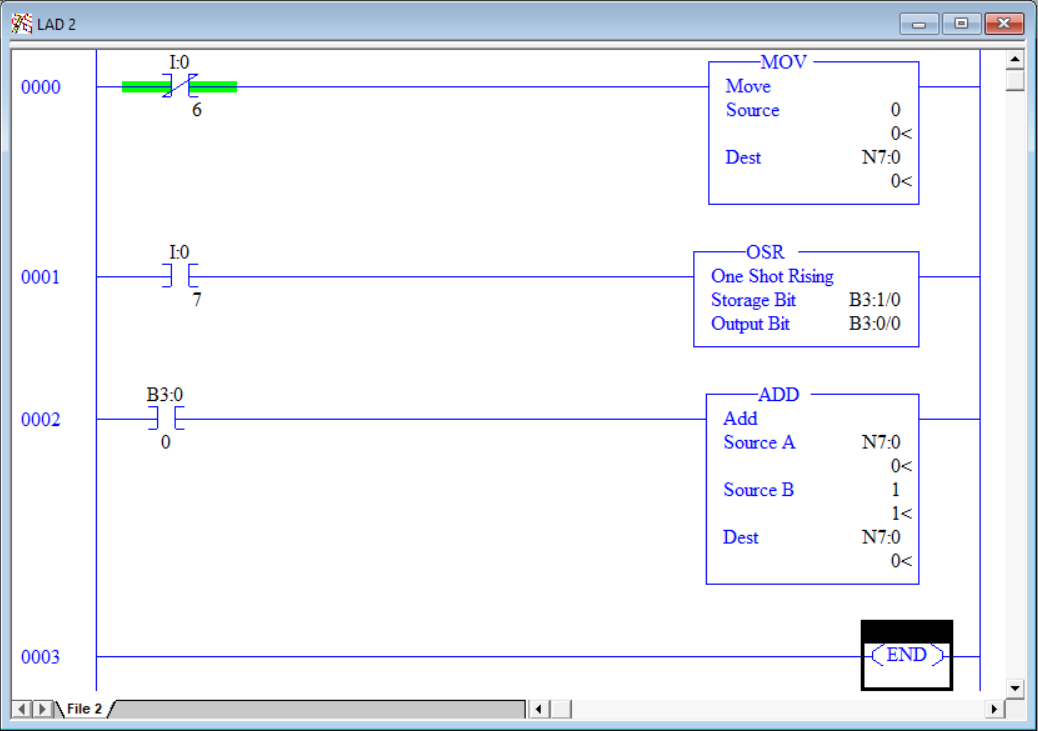
The one shot falling is a conditional input instruction that triggers an event to occur one time. Use this instruction when an event must activate based on the change of rung state from true to false, as triggered by a pushbutton, limit switch or other descrete device. See Note A for address assignments. The output bit shall be set to true for one scan when the rung *loses* logical continuity. Subsequent false rung scans have no affect on the output bit. The rung must gain continuity then lose continuity for the output bit to be set true again.

**Note A:** You must enter bit level addresses for the OSR. Use either a binary file bit (B3:0/0) or an integer file address specified to the bit level (N7:0/0). There are two bits associated with this instruction; the storage bit and the output bit. Each bit address used must be unique. Do not use either address elsewhere in the program. The storage bit is used the remember that state of the output bit during the last scan.

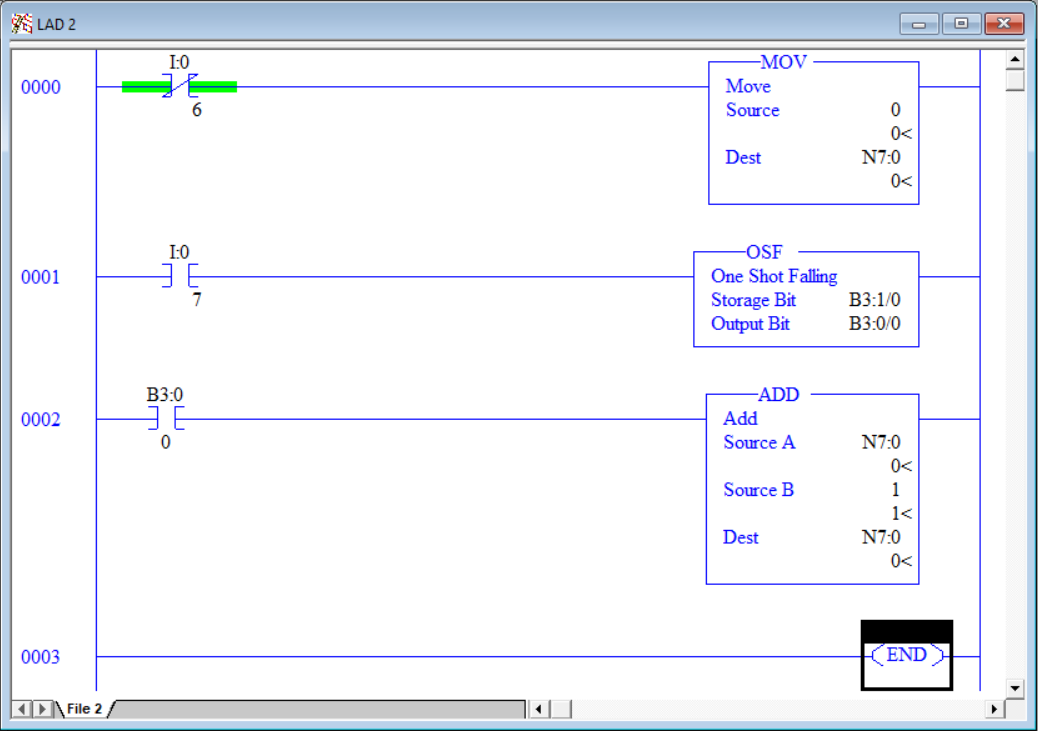
**Graphic**

|  |  |
| --- | --- |
| OSR | OSF |
|  |  |

1. Program the following logic.



1. Press PB2 and hold it down. Notice that the addition was only executed once. Holding down the button does for result in 1 continually being added the N7:0 as in the previous example.
2. Release PB2.
3. Press PB2 again. Notice that again, 1 was added to N7:0 only once. Using a one shot is a great way to ensure that the actual desired function only executes once.
4. Modify the program to use the OSF in place of the OSR as seen below.



1. Press PB2 and hold it down. Notice that the addition was not executed. Holding down the button creates a rung true condition which the OSF instruction is waiting for, but for this instruction to take action the rung needs to go from true to false.
2. Release PB2. Notice that on the release of the button, the addition function was performed and only once.
3. Experiment with the program changing the ADD function with SUB, MUL and DIV.