Commonly Used Ladder Logic Instructions

While there are hundreds of instructions available for use in Logix5000, there are some which are used routinely and frequently. These are the ones to be discussed at this point.

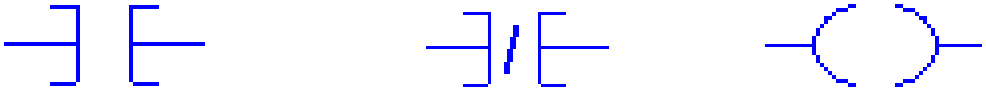
A **conditional bit input** instruction changes its true/false state to reflect the value of the bit to which it corresponds.

If you want a bit input instruction that . . .

Is **true** when the bit it is examining has a value of 1 (on). This is called **Examine if Closed** known as an **XIC.**

Is **true** when the bit it is examining has a value of 0 (off). This is called **Examine if Open** known as an **XIO.**

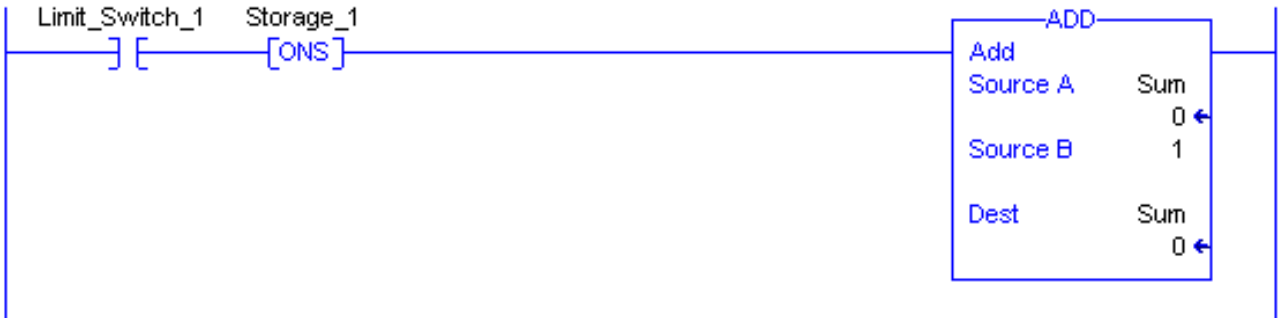
XIC **XIO** **OTE**



The **Output Energise** known as an **OTE will energise** when the inputs on the rung are true and switch off when the rung is false.

One Shot Instruction

The **One Shot Instruction** is a conditional instruction which when the condition in front of it transitions from **False to True,** the One Shot (**ONS),** will go true for **one scan only** to execute an output. This is used extensively to bring on a latched output or to execute a mathematical operation. An example is shown below.



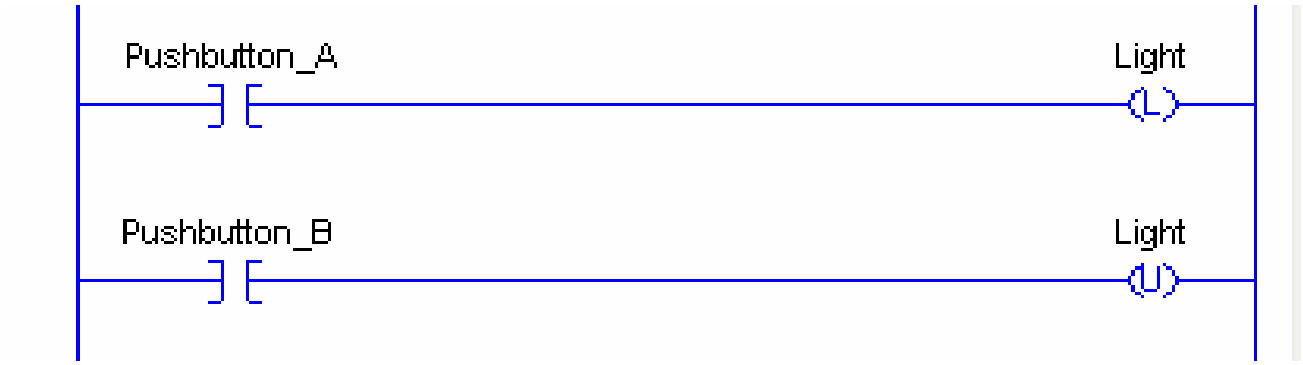
Latched Output Instructions

Output Latch **Output Unlatch**

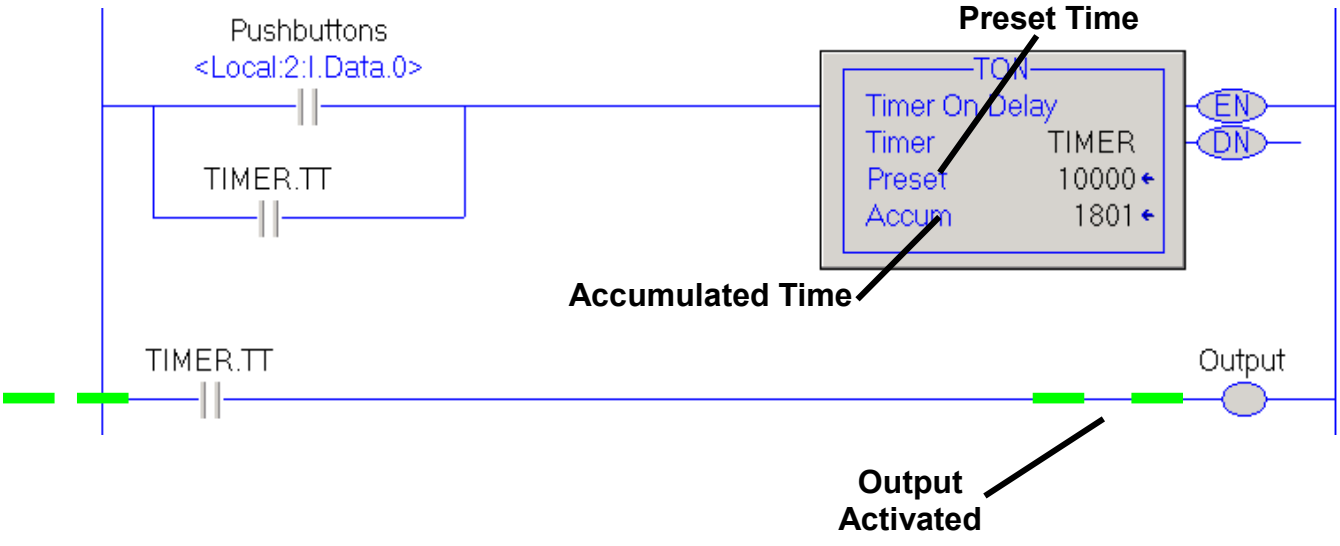
The above instructions work together and will reference the same tag.

The **Output Latch (OTL)** will switch on when the rung turns true, and will stay on when the rung goes false. To switch off the output, an **Output Unlatch (OTU)** must be used. An example is shown below.

Caution must be exercised when using these instructions. In the event of a power failure for instance, when power is restored, the latch instruction will retain the condition it was before the failure. This may prove hazardous in certain plant situations.



Seal-in logic is often used instead of the OTL and OTU instructions for applications in which it is undesirable to have latched bits following a power loss. In the example below, the Pushbutton will start the timer. The TT bit will come on. This will hold the timer until the preset is reached.



Timer Instructions

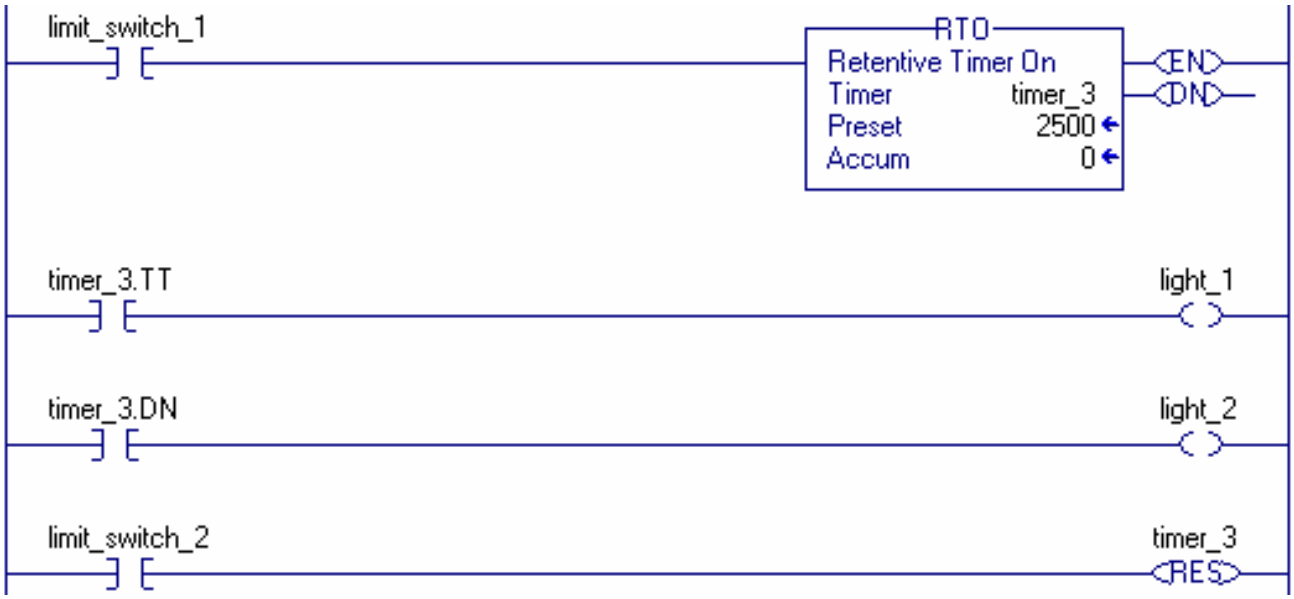
Timer Instructions are used to time an event or a period when something is supposed to execute. All timers work on a time base of **Milliseconds** so 10 secs will be visible as 10000.

There are 3 timer's in the logix5000 system

- **Timer On (TON)** This timer starts to time when the rung is true. It will continue until the **Preset** is reached. It will reset when the rung goes false.
- **Retentive Timer (RTO)** Works similar to the TON except when the rung goes false, it will stop at the accumulative value. It can only be reset using a **Reset (RES)** instruction.
- **Timer Off (TOF)** This timer works the opposite to the TON. When the rung is true, the timer is reset. When the rung goes false, it will start timing

All the timers have status bits which can be used in the logic.

- **DN bit (Timer Done).** This will become true when the **Accumulative** value equals the **Preset.**
- **EN bit (Enable)** The timer is enabled.
- **TT bit (Timer Timing)** When the timer is timing, this bit is **On.**



In the example above, an RTO is shown using a reset instruction. When the rung goes false, the accumulative does not reset. A reset instruction must be used.

Timer Status Bits

TON

Timer Condition When rung is……	EN	TT	DN	Operation
True ACC< PRE	1	1	0	Timer is timing
True ACC= PRE	1	0	1	The ACC has reached the PRE and the timing has stopped.
False	0	0	0	The timing has stopped and the ACC is reset to zero.

TOF

Timer Condition When rung is……	EN	TT	DN	Operation
False ACC< PRE	0	1	1	Timer is timing
True	1	0	1	The timer is reset and the ACC is reset to zero.
False and ACC=PRE	0	0	0	The timing has stopped and the ACC holds its value.

RTO

Timer Condition When rung is……	EN	TT	DN	Operation
True ACC< PRE	1	1	0	Timer is timing
True ACC= PRE	1	0	1	The ACC has reached the PRE and the timing has stopped.
False and ACC <= PRE	0	0	0	The timing has stopped and the ACC will hold its value.

The tables shown above, illustrate the state of the status bits under different timing conditions using the different timer types. The **RTO** can only be reset using the **RES** instruction. This must **Not** be used for the **TOF** as it sets all status bits to zero.

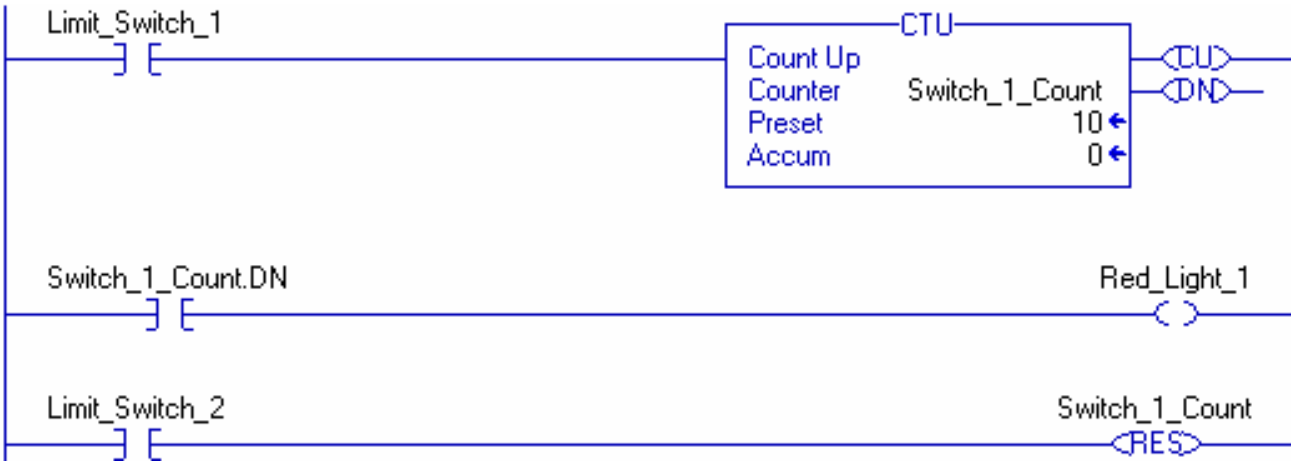
Counter Instructions

Counter Instructions are made up as a structure similar to timers. However, the execution is not based on time, it is based on a False to True rung condition. Each time the rung goes from False to true, there will be a count of 1 up or down depending on the counter being used. There are two types.

Count Up (CTU) This will increment upwards each time a rung goes true. There is no time relationship.

Count Down (CTD) This will increment downwards each time a rung goes true.

Both counters have to be reset using the **RES** instruction. This is the same instruction used for timers.



This illustration will count up each time Linit_Switch_1 goes from false to true. It will continue counting to the preset value at which time the **DN** bit will go true. Unlike a timer, the count can continue beyond the preset value. The DN bit will remain true. It can only be reset using the reset instruction.

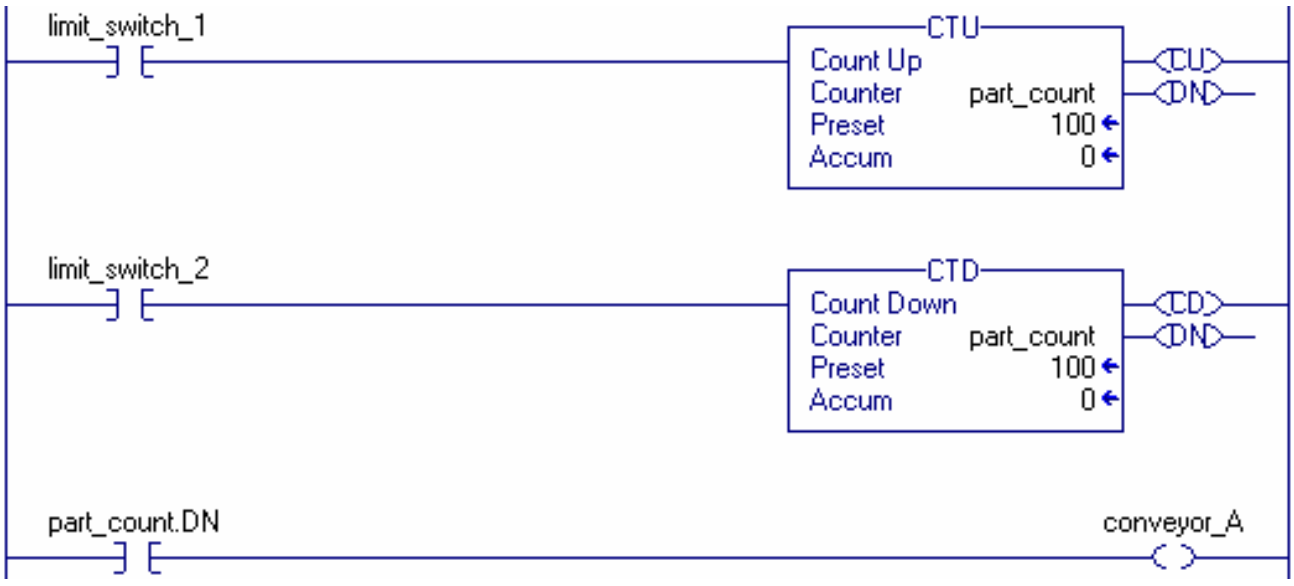
Counter Status Bits

Status Bits	This will Inducate
CU(Enable Count Up)	**The Count Up instruction is Enabled**
CD(Enable Count Down)	**The Count Down Instruction is Enabled**
DN(Done)	**The ACC >= PRE**
OV(Overflow)	**The value has exceeded the pos. decimal value of a Double Integer**
UN(Underflow)	**The value has exceeded the Neg value of a Double Integer**

The Status bits as shown in the illustration can be used in the program logic. The **OV bit** will become true if the decimal value exceeds **+2,147,483,647.**

The **UV bit** will become true should the value fall below **-2,147,481,648.**

It is unlikely that these values will ever be reached but should they be reached, the bits can be used in logic to notify the condition.



In the above illustration, it can be seen that a **CTU** and a **CTD** is being used referencing the same counter tag. This can be used if parts are going into and out of a buffer. The CTU will be counting parts leaving the buffer. The CTD will be counting parts remaining in the buffer.

Another example can be counting production and the CTD used to deduct the rejects.

Program Control Instructions

Within Logix 5000, when a project is created, there is one default routine called the **Main routine.** When the controller is placed in Run mode, this routine will execute. The main routine assignment can be changed if desired to a different routine.

There are 2 other types of routine which can be used,

Fault Routine. This is an option where logic can be written to prevent the controller going in to fault mode if certain events take place.

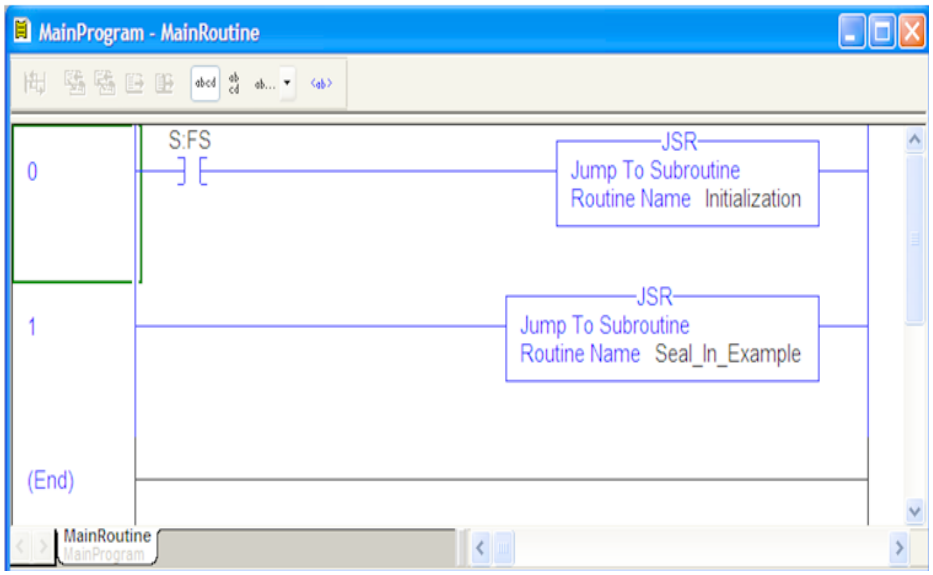
Sub Routine. Sub routines are created frequently to organize and separate certain sections of the project. To call a Sub Routine, a **Jump to Subroutine (JSR)** instruction must be used. Typically, they are normally configured within the Main Routine but can be used also in the Sub Routines to call other Sub Routines. The diagram below, shows a JSR both conditional and unconditional. The conditional rung is showing the **First Scan** bit which is a Logix5000 system bit. It will execute once only when the controller goes into Run mode. Any bit in the project can be used and attached to any tag in the program

The unconditional JSR will execute each time the rung is scanned.

There are two other instructions which can be used. They are the **SBR and RET.** These can be used if parameter values are being passed from one routine to another. The SBR will be configured on the first rung of the sub routine and the return at a point in the subroutine where it will return to the point below the JSR which called it and continue execution.

Conditional

Unconditional

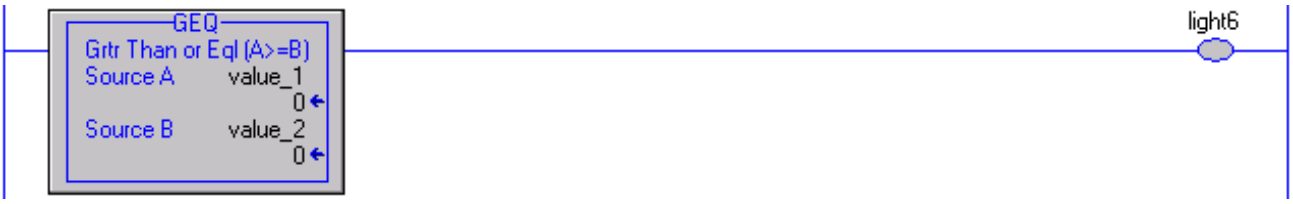


Comparison Instructions

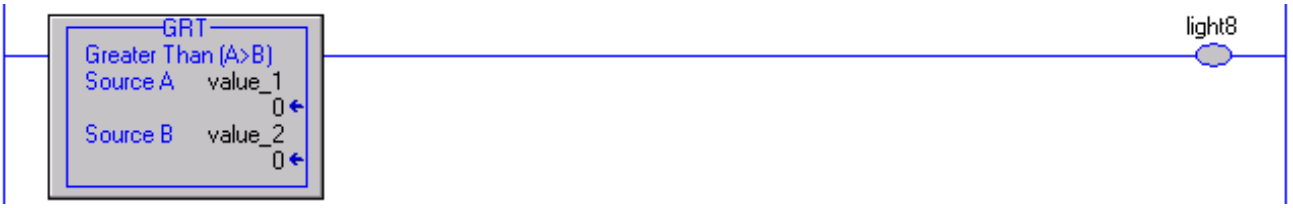
Comparison instructions are Input instructions. They compare 2 values and if the comparison is true, then it will activate an output or outputs.

Examples shown below.

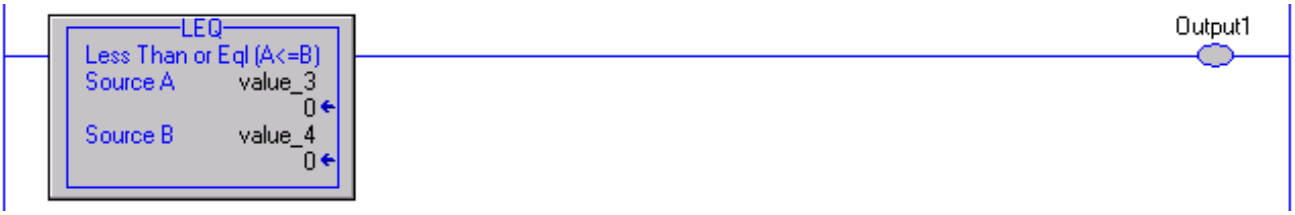
Greater than or Equal(GEQ). If A is greater or equal to B, the output will switch On.



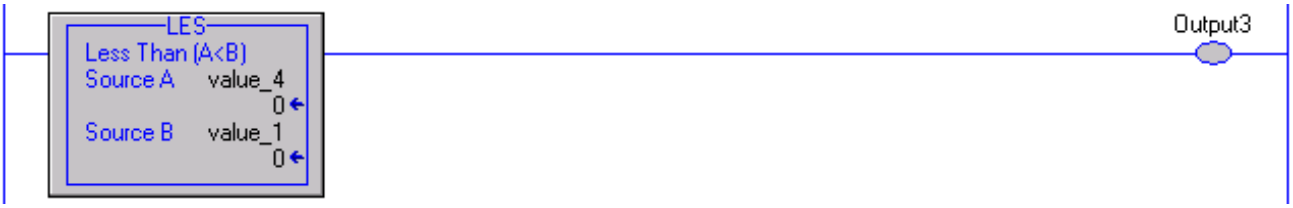
Greater than(GRT). If A is greater than B, the output will switch On.



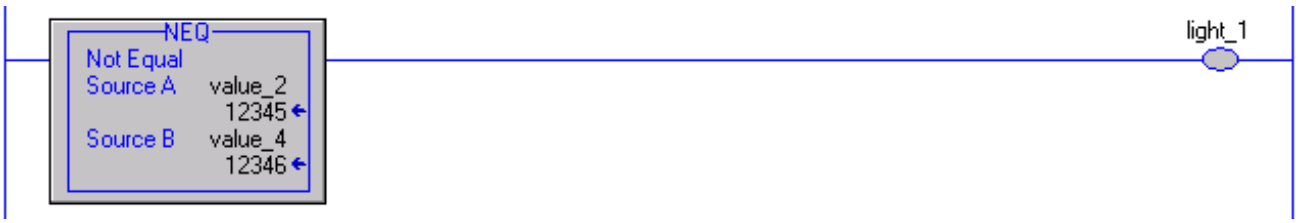
Less than or Equal(LEQ). If A is less than or equal to B, the output will switch On.



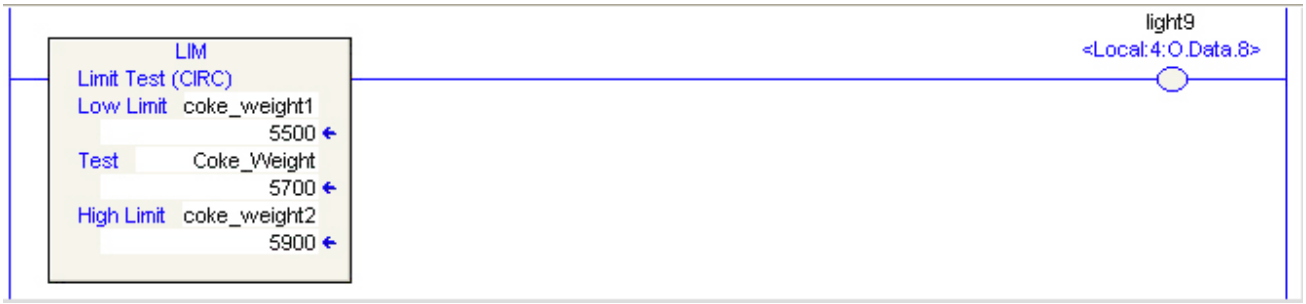
Less than(LES). If A is less than B, the output will switch On.



Not Equal(NEQ). If A is not equal to B. the output will switch On.

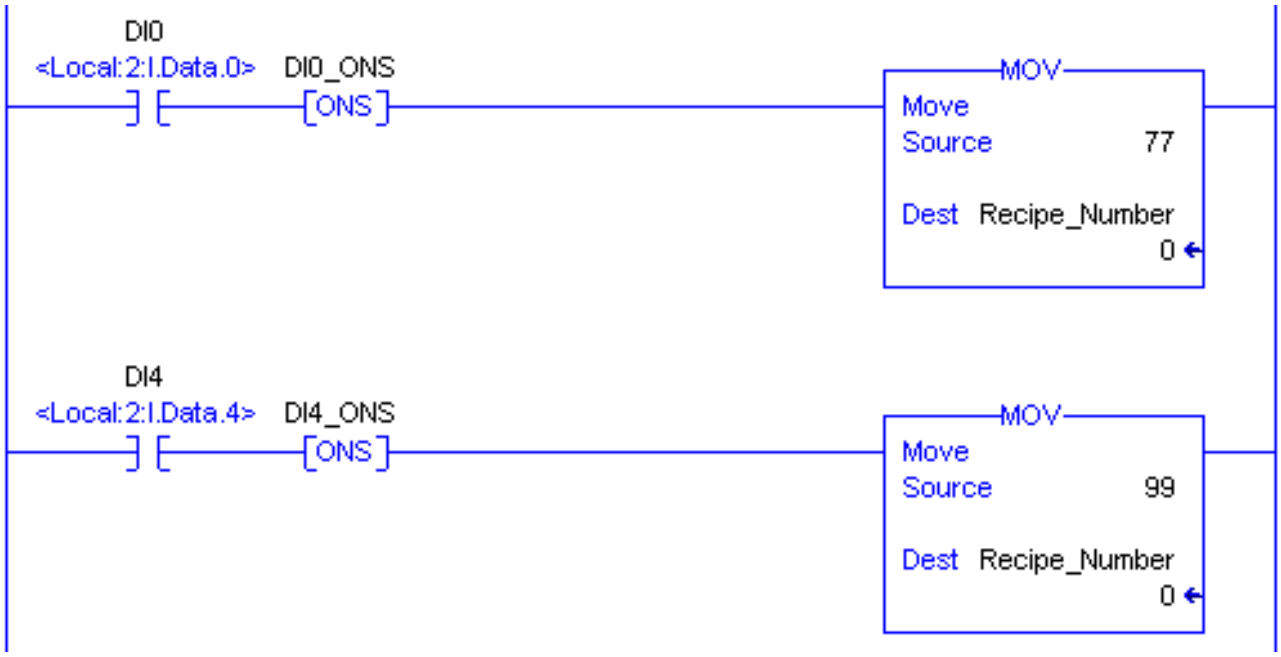


Limit Instruction(LIM). If the test value is between the Low limit and the High limit, the output will switch On. The comparison values can be constants of INT or Real values with the Test value being a tag.



Move Instruction(MOV)

A Move instruction moves a value from a source to a destination. Bothe the source and destination can be a mix of tags and fixed values. Very common instruction and used extensively. An illustration is shown below moving a value into a tag using a ONS.



Maths Instructions

Maths Instructions are used to perform calculations between 2 values or tags. They are output instructions. The following instructions are available

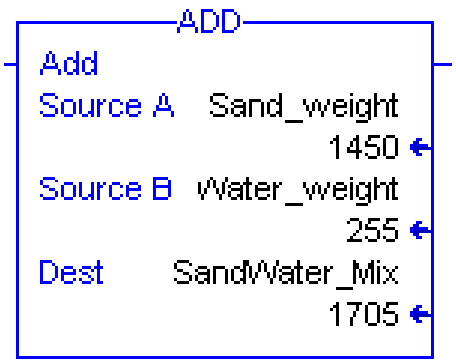
ADD. This will add 2 values and place the result in a destination tag

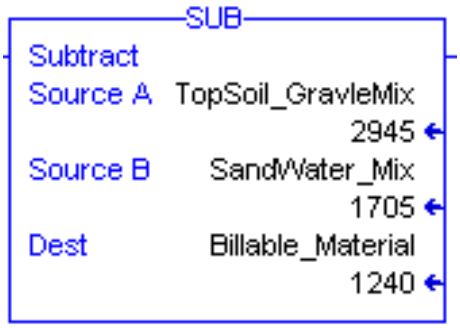
MUL. This will multiply 2 values together and place the result in a destination tag

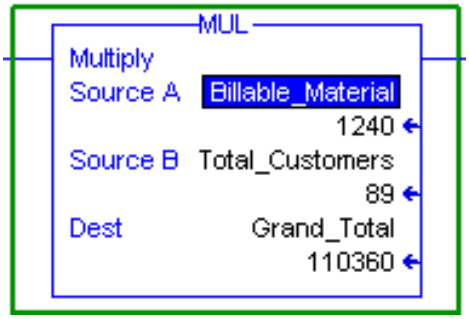
SUB. This will subtract Source B from source A and places the result in a destination tag.

DIV. This will divide Source A by Source B and place the result in a destination tag.

Illustrated below are the maths instructions commonly used.







Tags and values can be Real or Integer types. You can mix data types but accuracy may be lost due to loss of decimal point values when using Integer. If the result is an Integer from a Real, the value will be rounded up or down so the decimal value is lost.