Module 3

Data Manipulation Instructions

Allen Bradley

CompactLogix / ControlLogix

Module 3:
Data Manipulation Instructions

Data Manipulation

The instructions discussed in this lesson are found in the Move/Logical tab

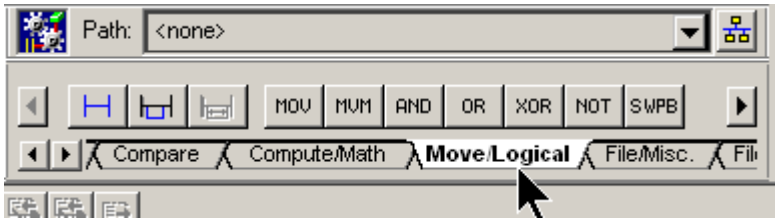


Figure 1-A Move Logical Tab

and in the File/Misc. tab.

Figure 2-A File/Misc. Tab

Use the tab scroll arrows to view tabs.

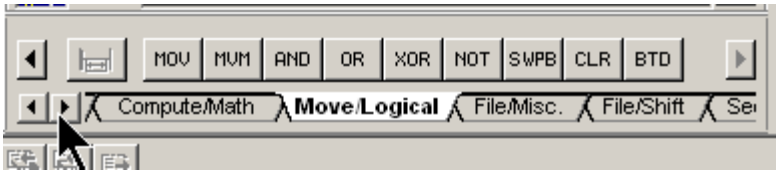


Figure 3-A Tab Scroll Arrows

Use the Instruction scroll arrows to view instruction icons not shown by default.

Figure 4-A Instruction Scroll Arrows

The following Data Manipulation Instructions will be covered in this lesson.

Word Type Instructions	File Type Instructions
MOV	COP / CPS
MVM	FLL
CLR	FAL

The MOV Instruction

The MOV (Move) instruction copies the data value from one tag memory location to another tag memory location in the ControlLogix / CompactLogix memory. The source data is not affected by the MOV. This is confusing because from the DOS days to Windows OS we learned that a Move actually moved the data from one location to another and a Copy did just that, it copied data without affecting the source. Well the MOV in Allen Bradley is a little backwards. It is called a MOV, but it acts like a copy. The source data is not changed.

A MOV instruction is an output on a ladder rung.

When the MOV instruction is energized (gets logic power flow) the source value is moved to the destination, as shown in the following illustration.

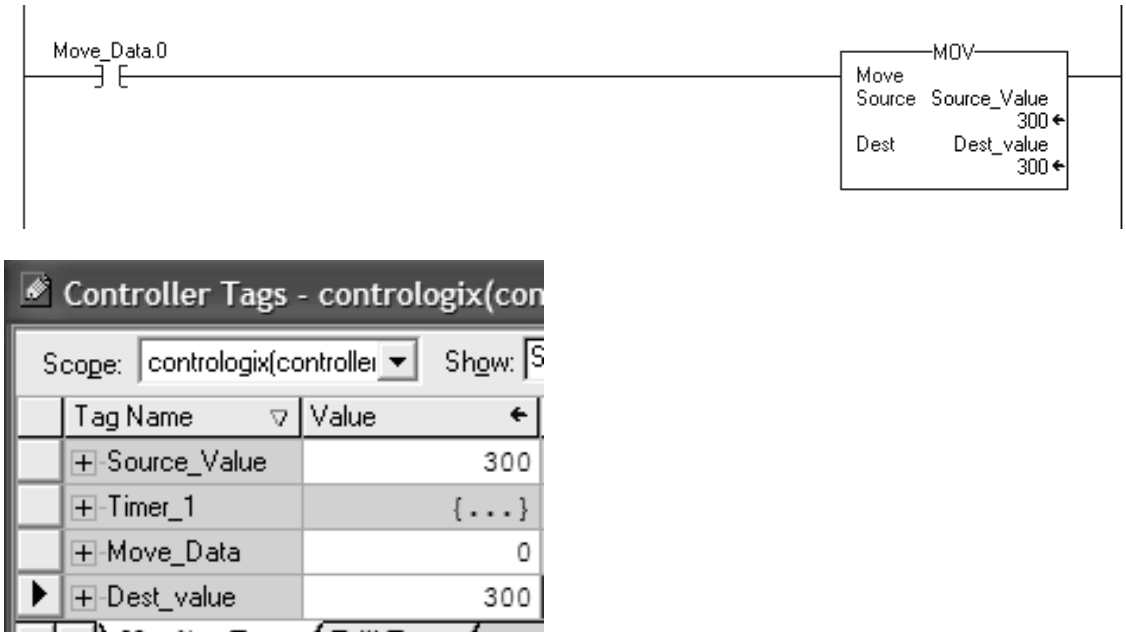


Figure 5-A. The MOV Instruction.

If the instruction remains true, the MOV will transfer the data every scan.

The Source can be a constant or a memory location (tag).

The Destination must be a memory location (tag)..

The instruction going false has no effect on the Destination data, i.e. it does not zero the destination location.

The MVM Instruction

The MVM (Move with Mask – but usually called Masked Move) instruction is similar in operation to a MOV instruction that will mask out part of the data. Instead of moving all the bits of the Source tag, maybe only 12 bits of the Source tag is moved, and the remaining bits is masked out – not moved.

The Mask is many times referred to as the Filter. A “1” in the mask will pass the corresponding tag source data bit to the destination tag data bit. A “0” in the mask will not pass the corresponding source data bit to the destination data bit.

By default the Mask data is viewed in a decimal format (Style). If you want to enter the mask in another format (Style) you must precede the mask with one of the prefixes in the following table.

Prefix	Description
16#	Hexadecimal (ex. 16#00FF)
8#	Octal (ex. #8)
2#	Binary (ex. 2#0000000011111111)

Table 1-A

The following illustration shows the Masked Move. The data shown the Binary Style.

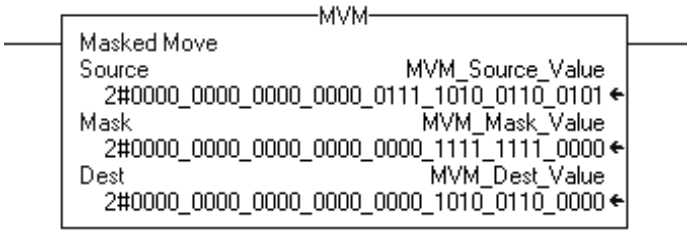


Figure 6-A. The MVM (Masked Move) Instruction.

Decimal values are 31333, 4080 and 2656

Basic Array Instructions

Tag Information

CompactLogix / ControlLogix stores data in tags, in contrast to PLC's and SLC's that use fixed data files that are numerically addressed.

Using tags allows you to document your application as you develop it and it also allows you to use tag names to organize your data to match your machinery.

While creating tags there are 3 properties that have to be assigned.

1. Scope

If the tag is going to be available to all the programs that you have created, select Controller Tag (Controller Scope). If the tag is only going to be used in one program select Program Tag (Program Scope).

2. Tag Name

This identifies the data within the Logix 5000 controller. Tags with different scopes can have the same name, although to avoid confusion it is not recommended.

3. Data Type

For the tag this defines the organization of the data (ex. Integer, Bit, floating-point number, Timer, etc.)

The following table displays the most common data types and their uses.

Data Type	Most Common Use
REAL	Analog device in floating-point mode
INT	Analog device in integer mode (for very fast sample rates)
string	ASCII characters
BOOL	Bit
COUNTER	Counter
BOOL	Digital I/O point
REAL	Floating-point number
DINT	Integer (whole number)
CONTROL	Sequencer
TIMER	Timer

Table 2-A

The following graphic shows the tag monitor / edit window.

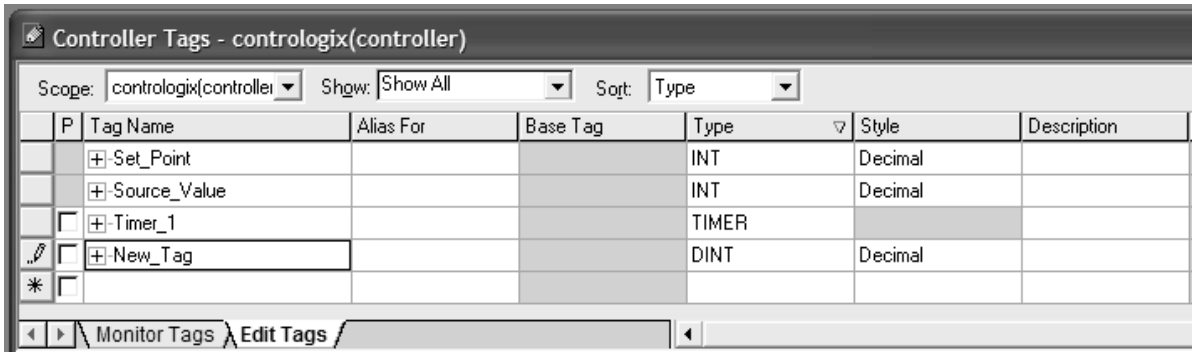


Figure 7-A. The Monitor / Edit Tag Window.

Tag Arrays

A tag array is a number of consecutive tag memory locations created in the Logix 5000 tag database. When you need to access data within one of these memory locations you need to use an index. An index is a value that is at the end of the tag name and identifies which one of the array locations (element) to use. The index can be a constant value or is another tag.

Example: The following tag is using a constant of '0' (index value) to identify the first element in the array - One_Dim_Array[0].

Note: When using arrays it is important to remember that array position number (index) start at 0. So a ten-element array will have valid index numbers of 0 through 9.)

The following window is accessed by clicking the ellipse button in the data type cell of the Tag Monitor / Edit window.

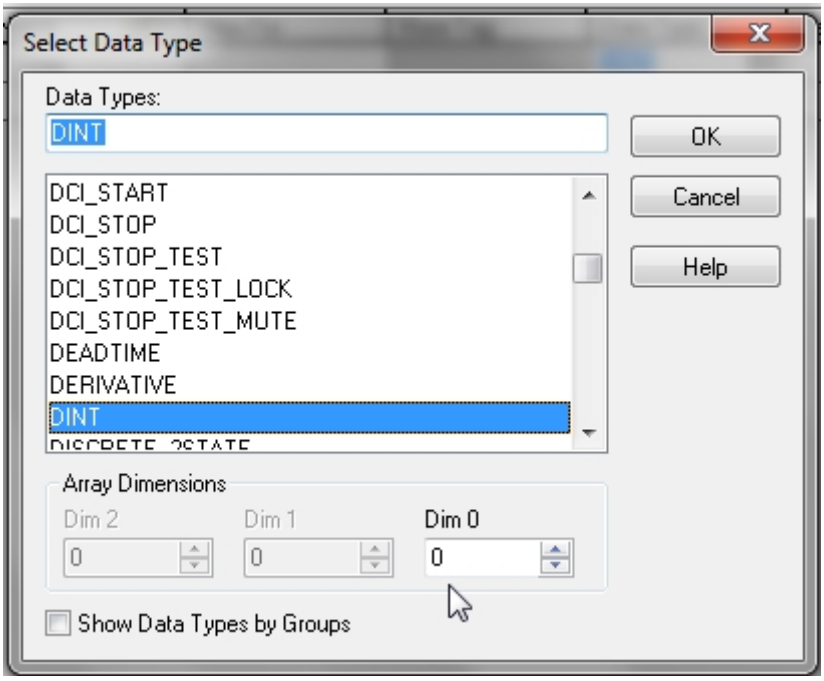


Figure 8-A. The Data Type Window.

When creating tag arrays notice in the above graphic that an array can have three dimensions.

A one-dimension (Dim 0) array can be thought of as one column of an Excel spreadsheet, when you put a number in the Dim 0 box you are increasing the number of rows that are contained in the column. For example if Dim 0 is 10 this means that 10 rows numbered 0 through 9 are available to store data.

A two-dimension array (Dim 0 and 1) can be thought of as one spreadsheet in an Excel workbook. When putting a number in Dim 0 you are increasing the number of rows in the spreadsheet. When you put a number in Dim 1 you are increasing the number of columns in the spreadsheet.

For example if Dim 0 is 10 and Dim 1 is 5 this means that 10 rows numbered 0 through 9 are available to store data and 5 columns numbered 0 through 4 are also available to store data creates 50 memory locations to store data. To access that memory location you will need to use two index numbers to identify an array element.

For example when using the tag 'Two_Dim_Array [3,4]' it is accessing the memory location in row 4, column 5 of the array. (Note: position numbers in arrays always start at 0.)

A three-dimension array (Dim 0, 1 and 2) can be thought of as an Excel workbook. When putting a number in Dim 0, increases the number of rows in the spreadsheet. Putting a number in Dim 1, increases the number of columns in the spreadsheet. Putting a number in Dim 2, increases the number of spreadsheets tabs at the bottom of the workbook. It's like stacking spreadsheets on top of each other.

For example when using the tag Three_Dim_Array [1,2,3] it is accessing the memory location in row 2, column 3, sheet 4 of the array. (Note: position numbers in arrays always start at '0'.)

Tags will be discussed in more detail in later Modules.

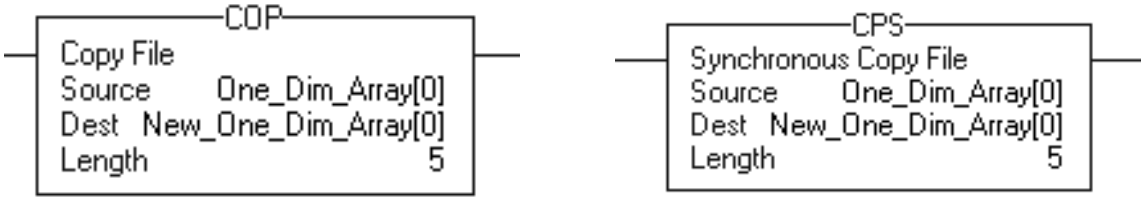
Note: One-dimension (Dim 0) arrays are most common and are required for some instructions

The COP and CPS Instruction

The COP (Copy) and CPS (Synchronous Copy) instructions are output type of tag array instructions that will copy tag array elements from one array to a second array. The source data values remain unchanged.

The difference between these two instructions is: The COP instruction will allow the data it is copying to be changed during the execution of the instruction. The CPS instruction will delay any task that attempts to update the data until the execution of the instruction has completed.

The following illustration shows a COP and a CPS instruction. When the COP goes true, the Source tag array data is copied into the Destination tag array locations.



One_Dim_Array	{...}
One_Dim_Array[0]	2#0000_0000_0111_1111
One_Dim_Array[1]	2#0000_0000_1111_1111
One_Dim_Array[2]	2#0000_0001_1111_1111
One_Dim_Array[3]	2#0000_0011_1111_1111
One_Dim_Array[4]	2#0000_0111_1111_1111

New_One_Dim_Array	{...}
New_One_Dim_Array[0]	2#0000_0000_0111_1111
New_One_Dim_Array[1]	2#0000_0000_1111_1111
New_One_Dim_Array[2]	2#0000_0001_1111_1111
New_One_Dim_Array[3]	2#0000_0011_1111_1111
New_One_Dim_Array[4]	2#0000_0111_1111_1111

Figure 9-A. The COP and CPS Instructions.

The COP and CPS instructions do not have status bits. As long as the instruction stays true, the data is re-copied every scan. The COP and CPS instructions operate on contiguous data memory and perform a straight byte-to-byte copy. When using these instructions it requires an understanding of the Logix 5000 memory layout.

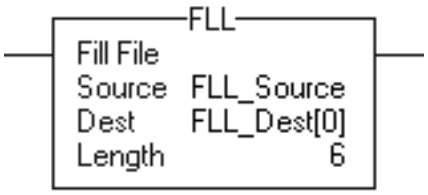
The COP and CPS instructions do not write past the end of an array. If the destination array has fewer memory locations the COP and CPS instructions will stop at the end of the array. This does not generate a Major Fault in the processor. That is why it is important to understand the memory layout.

The FLL Instruction – File Fill

The FLL is an output type of instruction that copies a value of a memory location – tag – (Source) to an array (Dest). The source data is not affected during the execution of the FLL. This instruction, like the COP, does not have status bits. For best results it is recommended that the same data types are used.

In the following illustration, the value in tag memory location FLL_Source is copied into 6 tag memory locations of the array FLL_Dest, starting at tag memory location FLL_Dest [0].

Note: The starting array location does not have to be zero (0).



⊞ FLL_Source	2#0000_0010_0110_0100
⊟ FLL_Dest	{...}
⊞ FLL_Dest[0]	2#0000_0010_0110_0100
⊞ FLL_Dest[1]	2#0000_0010_0110_0100
⊞ FLL_Dest[2]	2#0000_0010_0110_0100
⊞ FLL_Dest[3]	2#0000_0010_0110_0100
⊞ FLL_Dest[4]	2#0000_0010_0110_0100
⊞ FLL_Dest[5]	2#0000_0010_0110_0100

Figure 10-A. The FLL Instruction.

The FAL Instruction

The FAL (File Arithmetic/Logic) is an output instruction that can perform a copy, do arithmetic with tag data values, and perform function operations on data stored in tag arrays. All defined by the expression in the FAL instruction.

The FAL instruction is shown in the following illustration.

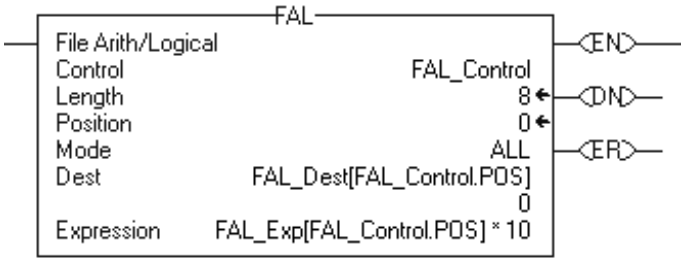


Figure 11-A. The FAL Instruction.

The following is a basic explanation of the parameters in a FAL:

Control	This is the tag memory location for this particular FAL. The tag FAL_Control should not be used for any other instruction. The Control tag identifies other elements of the instruction. Example: FAL_Control.LEN is the number of tag array memory locations that will be modified. FAL_Control.POS is the position of the tag memory location pointer in the tag array.
Length	This is the number of elements in the tag array that will be manipulated.
Position	This is the current data element position in the tag array. Typically the initial value of this is 0.

Mode	This is the number of positions or elements that are acted on per scan. If you are moving a large file, you can distribute this move over multiple scans, thus not slowing down the program scan time too much. The 3 mode options are: *All* – All tag array memory locations are moved in one scan. *Numeric Value* – You can set the number of tag array elements to be moved per scan (ie. 50 tags per scan). *Incremental* – One tag array element is moved every time the FAL instruction is toggled true. (Leading edge of a pulse)
Destination	The tag array or tag location that stores the result of the FAL.
Expression	This is the expression of the FAL that stores the tag locations, operators and constants needed for the FAL. *FAL_Exp [0]* – would specify just a tag array element for an array to array or array to tag transfer. *FAL_Exp + 100* – would add a value of 100 to each element of the array.

The Basic Status Bits for the FAL is listed below:

The Control tag for the FAL instruction will be the prefix for the tags.

.EN	Enable Bit	This bit is True when the FAL is powered - True (FAL_Control.EN)
.DN	Done Bit	This bit is True when the FAL has completed its operation - (FAL_Control.DN).
.ER	Error Bit	This bit is True when the FAL operation creates an error (FAL_Control.ER) and the instruction stops executing until the program clears the .ER bit. The .POS value contains the position of the element that caused the fault.
.LEN	Length (DINT)	This specifies the number of elements within the array that the FAL instruction operates.
.POS	Position (DINT)	This contains the position of the current element that the instruction is accessing.

Review Questions

1. T F A MOV will allow you to move multiple tag memory locations at one time.

2. The instruction used to copy multiple tag memory locations (Array) at one time:

a) COP

b) MOV

c) BTD

d) FAL

3. To move only the lower 8 bits of a word from the Source to the Destination in a MVM, the mask data in Hex Style must be?

a) FFFF

b) 0FF0

c) FF00

d) 00FF

4. The mode parameter for the FAL instruction that will move one tag memory location of the array every time the FAL instruction is toggled true is:

a) All

b) Numeric

c) Incremental

d) Toggle

5. Which of the following file instructions has status bits available?

a) FAL

b) COP

c) FLL

d) MVM

6. The mnemonic for the instruction that when true will copy the data from one tag into a 10 tag array is:

a) FLL

b) MVM

c) FVM

d) COP

7. T F The MOV instruction when true will copy the data from the Source to the Destination and delete the Source value.

8. T F The Destination value in a MOV instruction cannot be a constant value, only a valid tag location

9. The tag that identities the FAL instruction's Length and Position tags is:

a) Mode

b) Expression

c) DEST

d) Control

10. T F The COP instruction must start with element 0 in an array

Review Question Answers

1. **F**
2. **a**
3. **d**
4. **c**
5. **a**
6. **a**
7. **F**
8. **T**
9. **d**
10. **T**