Chapter 13 INDEXING – BATCHING APPLICATIONS

Introduction

This chapter discusses an important topic, moving of data and large data manipulation programs. This includes batching applications as well as the labs at the end of the chapter, Simon Says and Whack-a-Mole.

Modicon, a rival PLC manufacturer to A-B and Siemens, used three simple instructions with capabilities to move data from tables to a fixed location, from a fixed location to a table and from a location in a table to the same location in a second table. These instructions were called:

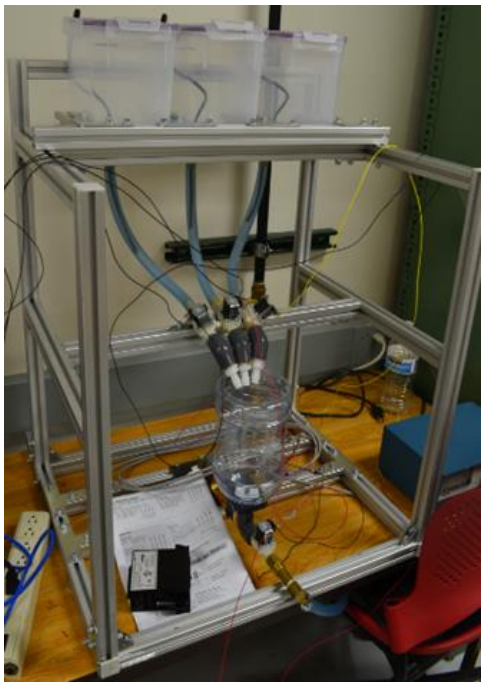
- Table to Register
- Register to Table
- Table to Table

All PLC manufacturers have instructions with capabilities similar to the Modicon instructions. A-B has a number of different instructions for moving data from a table. The SLC and PLC/5 instruction set used two different techniques for moving data. Siemens uses an indexer and matrices. The ControLogix language uses matrices and indexing as well. While RSLogix5000 looks very similar to RSLogix 500, the tag addressing scheme is very different as well as the instructions that perform indexes and moving data using indexing.

In this chapter, we will attempt to discuss the different indexing instructions presently employed by A-B and Siemens as well as look back at the older indexing instructions used by A-B with their SLC and PLC-5 processor lines. In addition, we will look at a specification from ISA called SP88 and some of its methodologies as to how to build a successful batching application.

SP88 will only be discussed here as to how it allows the PLC programmer to build an application that has the elements needed by the application but in a somewhat standardized way.

Batching systems vary in type and are pictured in the figures below. The first shows a simple batch system with various ingredients laid on the transfer belt at programmed rates. The system here does not include steps but only the process to mix a number of ingredients. More complex systems include steps after which various activities may occur such as agitation or heating. With the second figure, this is the case. Various ingredients can be added to a weigh mix tank a step at a time. This batch system requires the step architecture discussed in this chapter. Both require recipes. Different processes favor one over the other and some processes can use both. Obviously the transfer belt does not allow for wet ingredients adds directly on the belt. We will discuss the software needed to step through a recipe and the processes to program a batch with the PLC.



The three figures here show a variety of batch systems. The batch system at left was made by Capstone students at the University of Toledo. It contains three ingredient tanks and a mix tank. Below the mix tank is a weigh cell. Ingredients are added to the mix tank one at a time and then the combined mix is drained from the mix tank.
Lower left is a similar system from industry except the ingredients are solids. Each ingredient is added to the mix tank through an auger arrangement. Each ingredient tank is shown with vibration equipment that guarantee an even flow of solid into the auger.

Lower right is a mix system with ingredients simultaneously laying a weighed amount on the belt at a controlled rate. There is no mix tank in this system. The ingredients are layered onto the belt and then fed into a melter or other process.

Data Moving - an Example

We will be using an example of a simple batching system to discuss indexing and moving of data and construction of batching systems. The batch we discuss consists of:

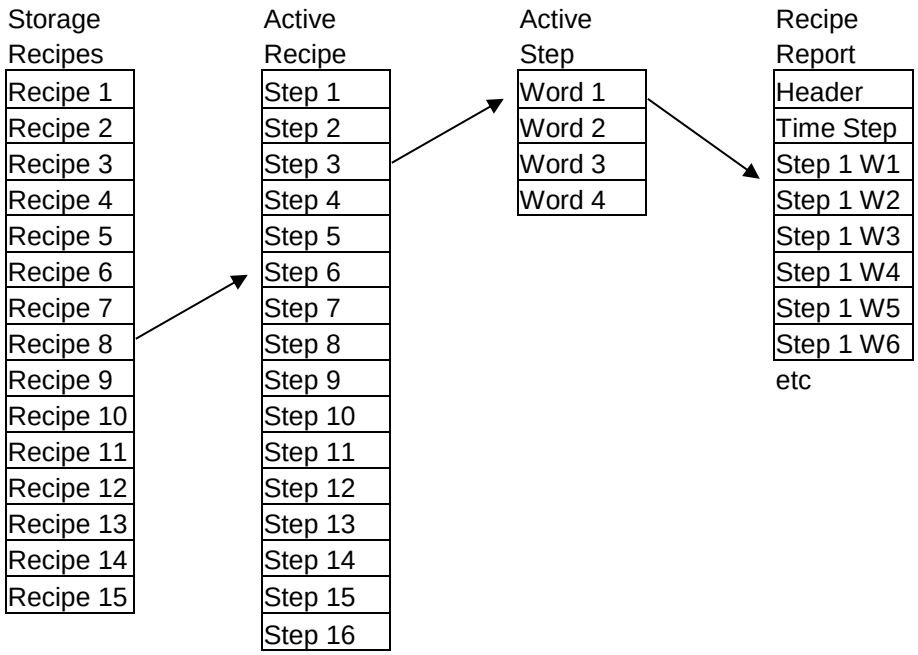
15 recipes for making, as an example, pancake batter:
- Each recipe consists of up to 16 steps
- Each step consists of 4 integer words
- Each recipe is given a 4-digit identifier number 0001-9999 that the operator may enter

into a given location prior to requesting that the batch be made

When the recipe is moved into a batch of batter, the recipe is moved to an active recipe area. With the start of the recipe, steps are moved one by one from the active recipe area to the active step area.

As a step is executed, information about the step is collected. For instance, the actual weight added is found and added to the information already found in the step.

As a step is made, the information about the step including the actual data is saved in a 6 integer word group with a time stamp in an output table. As a new recipe starts, the time stamp at the start of the recipe is saved with the recipe number. Then the step information is saved step by step.

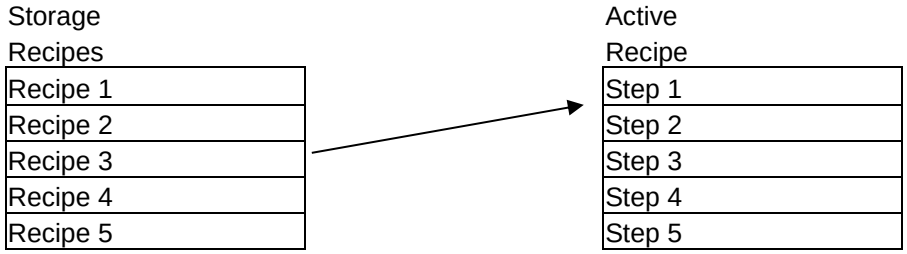


To selectively move a recipe from storage to the active area, with A-B processors, the COP command is selected. Moving of large portions of data at a time is referred to as a Copy move. Many times the computer supervising the process will store the many recipes in the computer memory and copying these recipes into the PLC. An alternative technique would provide the PLC with only the recipe presently being made and the next recipe to be made leaving only the two recipes in the PLC's memory. In general, it is advisable to use the database capabilities of the computer to control movement of recipes rather than storing and moving large portions of data in the PLC and occupying the PLCs memory with recipes that are rarely if ever used.

An Aside:

Data historically was stored in long strings of words, not necessarily in the format of an array. The user had to know that what was being programmed was an array and the data kept separate from other data even though there was no barrier between the array and other data. The programmer had to convey with a data table layout where the array was and its partitions. The user just had to be aware of where the array was in the data and not over-write data into it

unintentionally.
COP is used for large data table moves:



Contacts that are usually defined as one-shots are selecting large data blocks to be moved. In the example, B3:2/1, B3:2/2, B3:2/3 are selectively picking recipe 1, 2, or 3 to move to the active region to make one of the three recipes the active recipe. The active recipe storage area is in area N7 starting at word 10 and having length of 64 words.

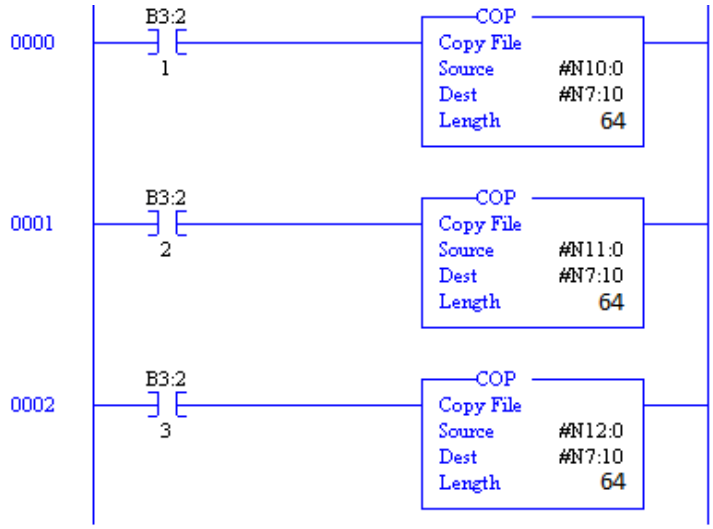
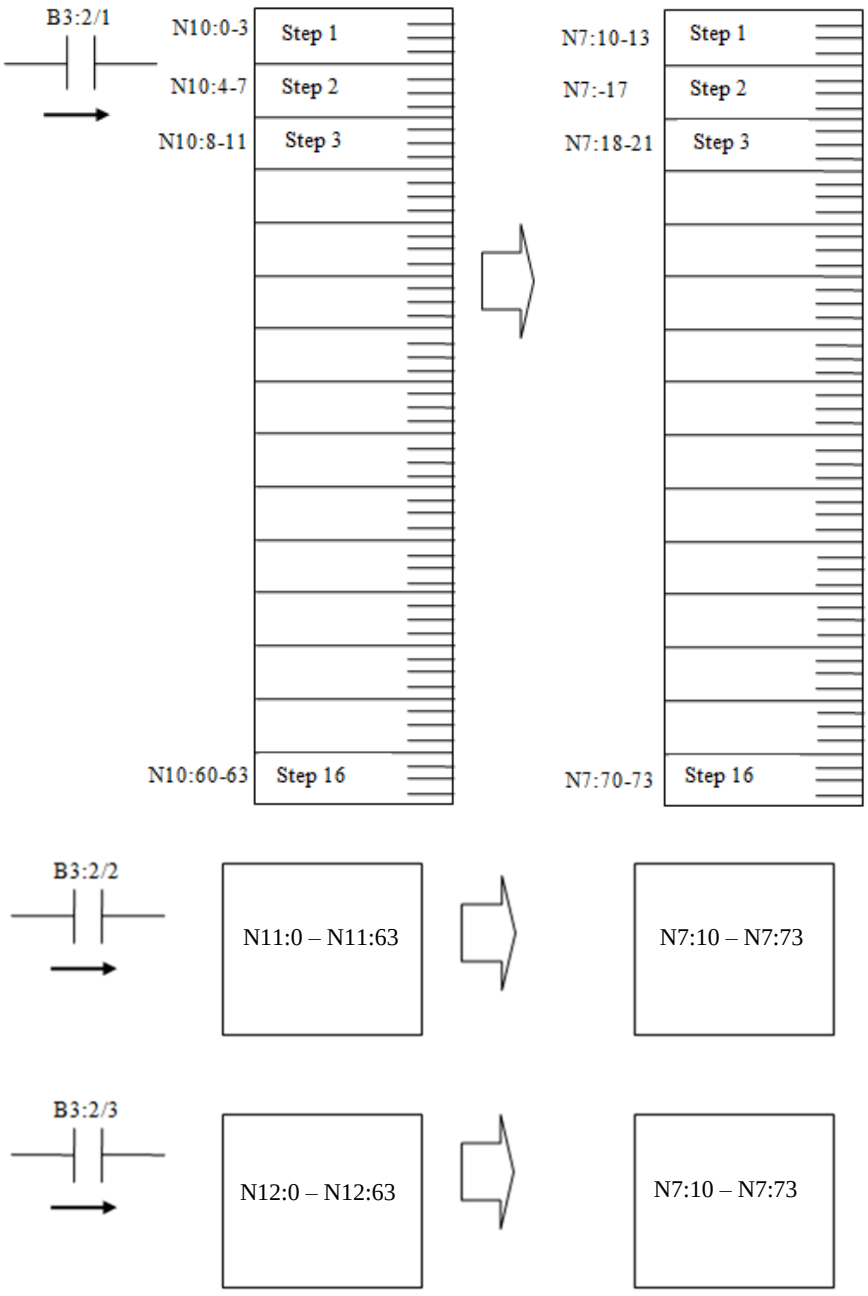


Fig. 13-1 A-B Copy Commands moving large amounts of data

Copy commands and other commands moving large amounts of data had the problem of moving the data in one scan or multiple scans. If the data was expected to be moved in one scan, care had to be taken to not over-burden the length of time needed to execute the instruction. When reading the instruction definition, look into the execution time of the command and what would happen if the time extends beyond a reasonable scan time.



For Siemens, moving large data blocks involves the following instructions:

MOVE_BLK: Move Block
Move_BLK_VARIANT: Move Block
UMOVE_BLK:Move block uninterruptible

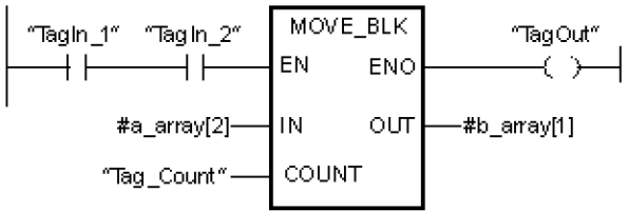
MOVE_BLK

The description for the MOVE_BLK instruction follows:

“You use the "Move block" instruction to move the content of a memory area (source range) to another memory area (target range). The number of elements to be moved to the target range is specified at input COUNT. The width of the elements to be moved is defined by the width of the element at input IN.”

Example

The following example shows how the instruction works:

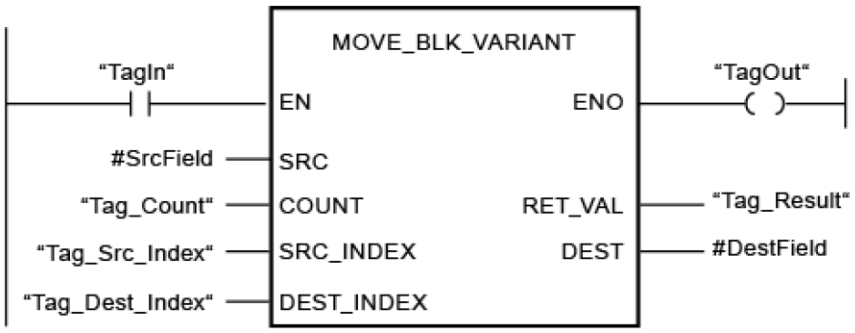


MOVE_BLK_VARIANT

“You use the "Move block" instruction to move the content of a memory area (source range) to another memory area (target range). You can copy a complete ARRAY or elements of an ARRAY to another ARRAY of the same data type. The size (number of elements) of source and destination ARRAY may be different. You can copy multiple or single elements within an ARRAY.”

Example

The following example shows how the instruction works:

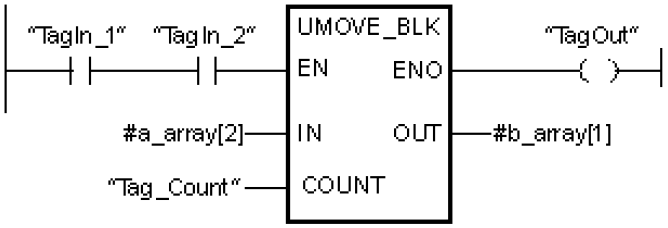


UMOVE_BLK: Move Block Uninterruptible

“You use the "Move block uninterruptible" instruction to move the content of a memory area (source range) to another memory area (target range). The instruction cannot be interrupted. The number of elements to be moved to the target range is specified with the COUNT parameter. The width of the elements to be moved is defined by the width of the element at input IN.”

Example

The following example shows how the instruction works:



These instructions introduce the new concepts of UDTs, arrays and variants. Earlier, the SLC examples treated arrays as only a string of numbers in successive data locations with the programmer having to keep track down the list. Now, the array or UDT indexes the variables by number. Building an array or UDT will be described later in the chapter.

Description of Variant Type – Siemens (from the Help Screen for MOVE_BLK_VARIANT)

"A parameter of VARIANT data type is a pointer or a reference It can point to tags of various data types. The VARIANT pointer cannot point to an instance and therefore cannot point to a multiple instance or ARRAY of multiple instances. The VARIANT pointer can be an object of an elementary data type, such as INT or REAL. It can also be a STRING, DTL, ARRAY of STRUCT, UDT, or ARRAY of UDT. The VARIANT pointer can recognize structures and point to individual structure components. An operand of data type VARIANT occupies no space in the instance data block or work memory. However, it will occupy memory space on the CPU.

A tag of the VARIANT type is not an object but rather a reference to another object. Individual elements of the VARIANT type can only be declared on formal parameters within the block interface of a function in the VAR_IN, VAR_IN_OUT and VAR_TEMP sections. For this reason, it cannot be declared in a data block or in the static section of the block interface of a function block, for example, because its size is unknown. The size of the referenced objects can change.

Using the VARIANT data type, you can in particular create generic, standardized function blocks (FB) or functions (FC) for various data types. Various instructions in all programming languages are available to you for this purpose. During the creation of the program, you can specify which data types the block should be able to process. The VARIANT data type supports you here by allowing the interconnection of any tags. You can then react accordingly to their data types in the block. When a block is called, you can connect the parameters of the block to tags of any data type. When a block is called, the type information of the tag is transferred in addition to a pointer to the tag. The code of the block can then be executed according to its type in line with the tag transferred during runtime.

If, for example, a block parameter of a function has the VARIANT data type, then a tag of the integer data type can be transferred at one point in the program, and a tag of the PLC data type can be transferred at another point in the program. With the help of the VARIANT instructions, the function is then in a position to react to the situation without errors."

Moving Small Amounts of Data:

Indexed Addressing is used to selectively move a step from the active recipe to the active step. If B3:2/5 is energized (one-shot), the words from the active recipe are moved into locations for use in the active step. The first word selected moves to N7:30. The second word moves to N7:31. The third moves to N7:32 and the fourth to N7:33. Again, from the SLC – A/B.

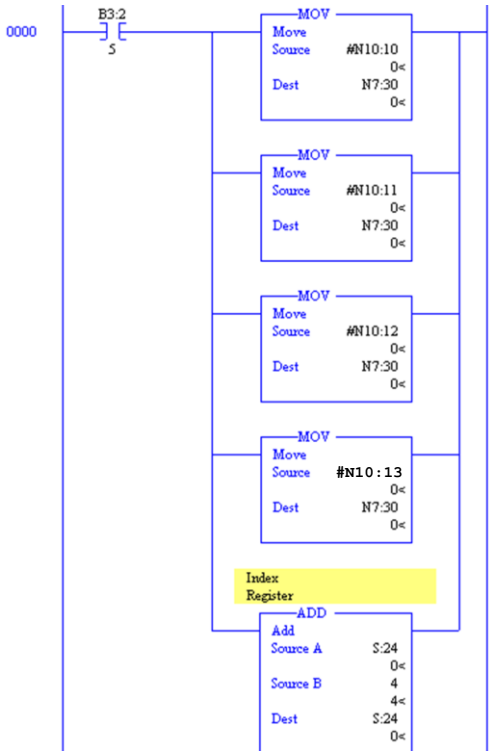


Fig. 13-2 A-B Move Commands moving small amounts of data

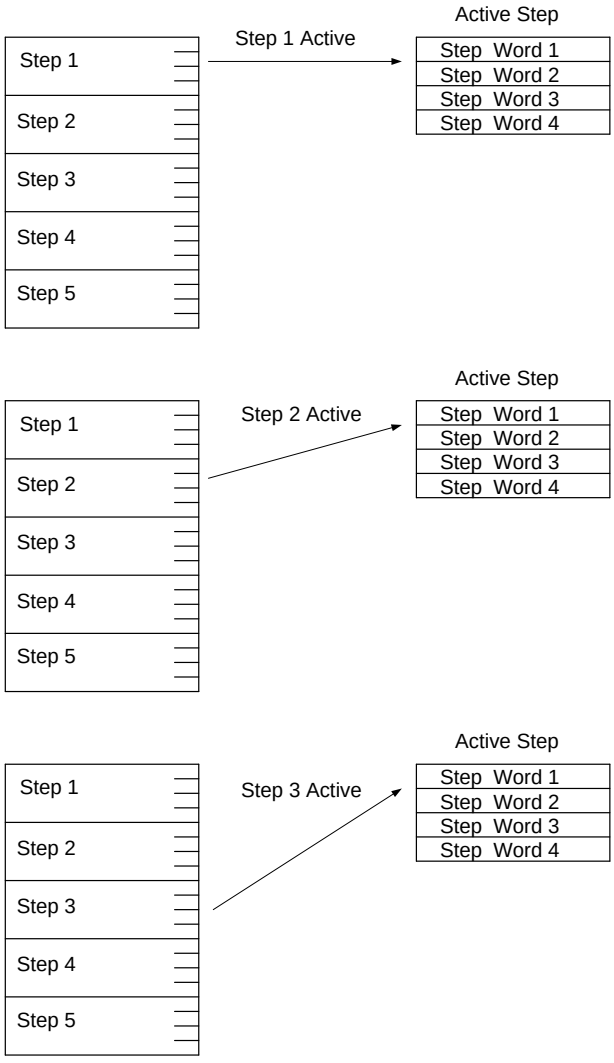
As the batch program moves through steps of the recipe, the indexing pointer moves starting at N7:10 and moves through the recipe.

MOV instructions are used for moving smaller amounts of data selectively using indexed or indirect addressing. Single word MOV statements are used to select individual word groups.

Indexing will be discussed later. For the SLC processors, the index is stored in location S:24. For Siemens and A-B Compact/Control Logix processors, the index is a word used in the move instruction. Matrices replace fixed locations for both Logix and Siemens processors.

The following shows words being moved to an active storage area from the steps in a recipe as the program moves through the steps. First the words of step are moved to the active area. This is also referred to as the work area or the register area. Next step two data is moved, then step three, etc. As each step is moved into the active area, a sequence of programming statements is executed allowing the program to execute the entire step and then move on. The sequence of the program can easily be displayed in a state diagram. The state diagram may have many parts or

be very simple. The state diagram may be as simple as found in Fig. 13-3.

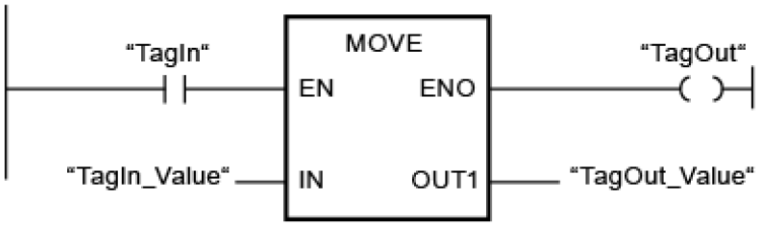


In a similar fashion, Siemens uses the MOVE command to move individual data content. The definition for the Siemens' MOVE command follows:

"You use the "Move value" instruction to transfer the content of the operand at the IN input to the operand at the OUT1 output. The transfer is always made in the direction of ascending address."

Example

The following example shows how the instruction works:



State Diagram of Step in Batch System

Execution at the step level is accomplished with a state diagram program. Movement through a recipe is accomplished with a set sequence of options that must be addressed for each step.

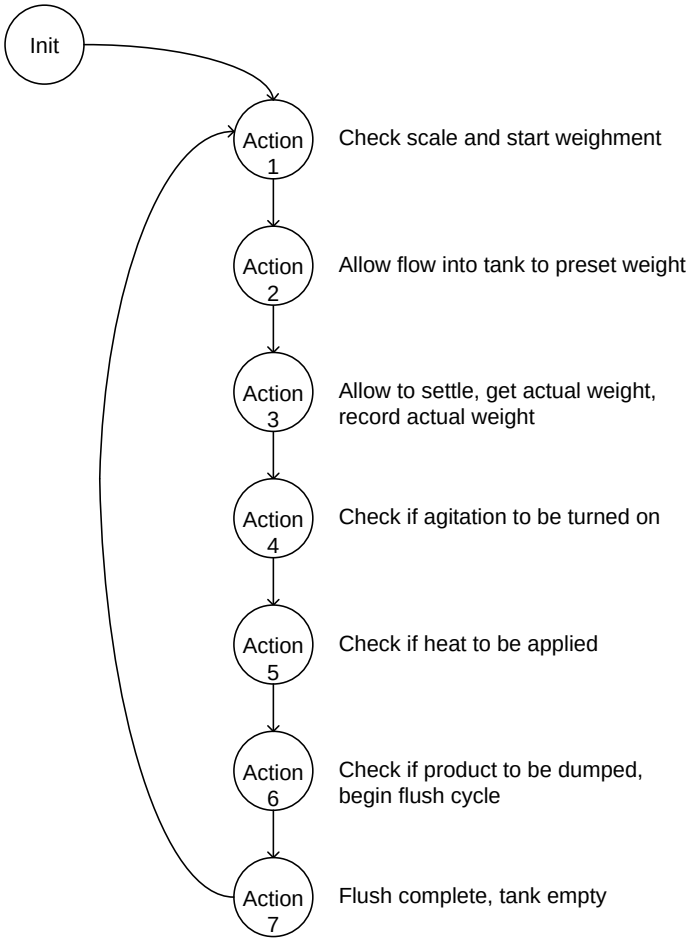
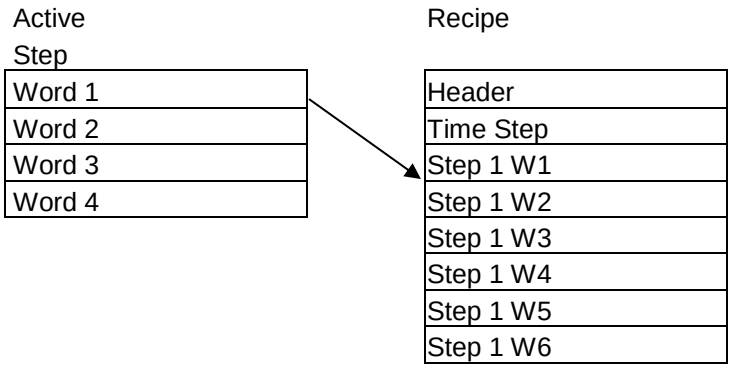
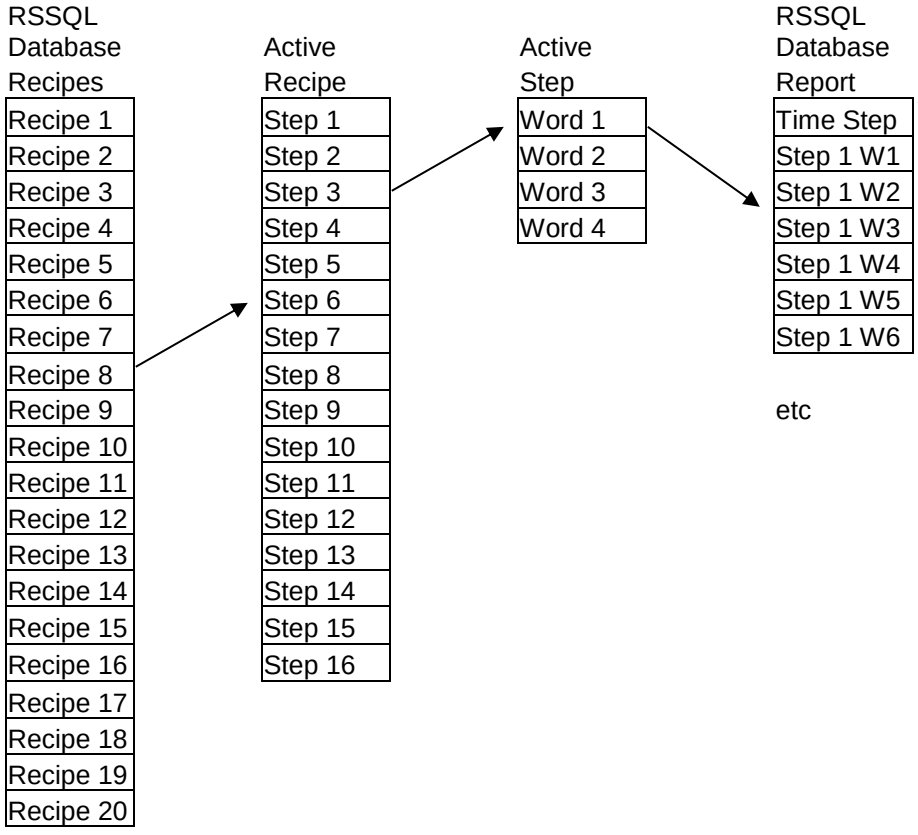


Fig. 13-3 Sample State Diagram for Step in Recipe

After a step is executed, data is collected about what actually happened. If a scale is used, the target weight is usually not the actual weight of the step. If the operator has an option to choose an action, the results of the choice are likewise remembered by the collected data. This data forms a step in the Recipe Report for a recipe. The word count may be equal to, more than or less than the active step recipe information.



A more ordered approach would be to allow the A-B database handler RSSQL handle the data from recipes. When requested, RSSQL would hand a recipe to the active recipe residing in storage in the PLC. After an active step is executed, RSSQL hands the data back to a second database in the computer that keeps track of what actually happened as the result of each step.



The older method from A-B was used in the above examples. The pointer in these examples is the number found in S:24. As this number is incremented, the pointer is moved down the table. The S:24 value is referred to as the indexer. The rules for indexing with this method are referred to as indexed addressing. The method was discontinued with the RSLogix 5000 instruction set.

With newer processors from A/B, indexing is accomplished with a matrix and the index is the number in the matrix [ ] location. Matrices are also standard with Siemens. With each method, the index value must be controlled or the program will have problems and probably cause the processor to fault. A pointer out of range is one of the first problems to look for with batching programs if the processor faults.

Creating Arrays

Creating arrays in Siemens and Allen-Bradley is not complex. For Siemens, follow the following two figures to create an array. Note that arrays are not available in the M Table.

To create arrays in Siemens PLCs, use the Data Block type for creating a storage array. First, choose the "Add new block" command and select Global DB. You will then be able to give information that defines the array desired. Length and data types are necessary. For most applications, the data type is INT or DINT. You may choose to have multiple arrays for multiple data types associated with a list of variables. You may also designate a UDT for this purpose.

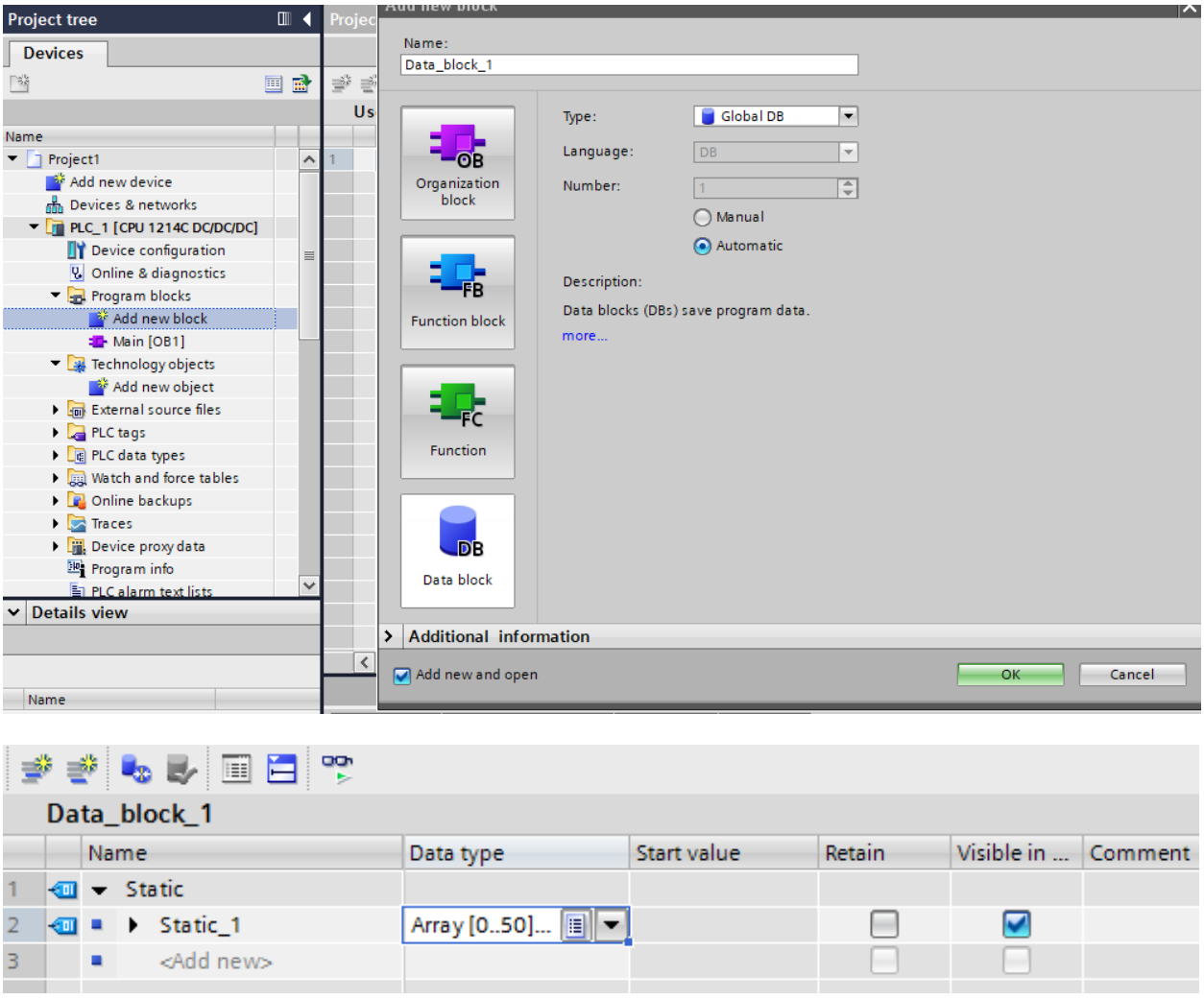
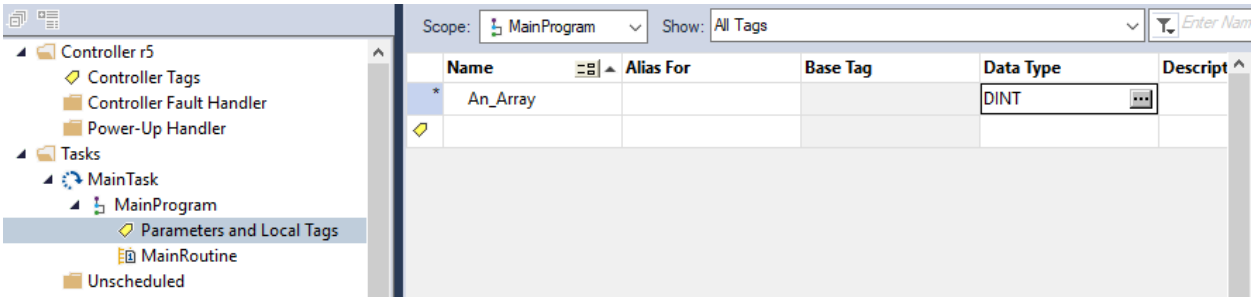


Fig. 13-4 Matrix defined in Data Block

The array created in the example above shows an array Array with 51 integer values.

Allen-Bradley allows creation of an array with creation of a named tag and then choosing the data type and dimension. There is no M table with Allen-Bradley and the type and length is chosen below:



The array length is determined in the boxes below. Multiple array dimensions may be chosen as well.

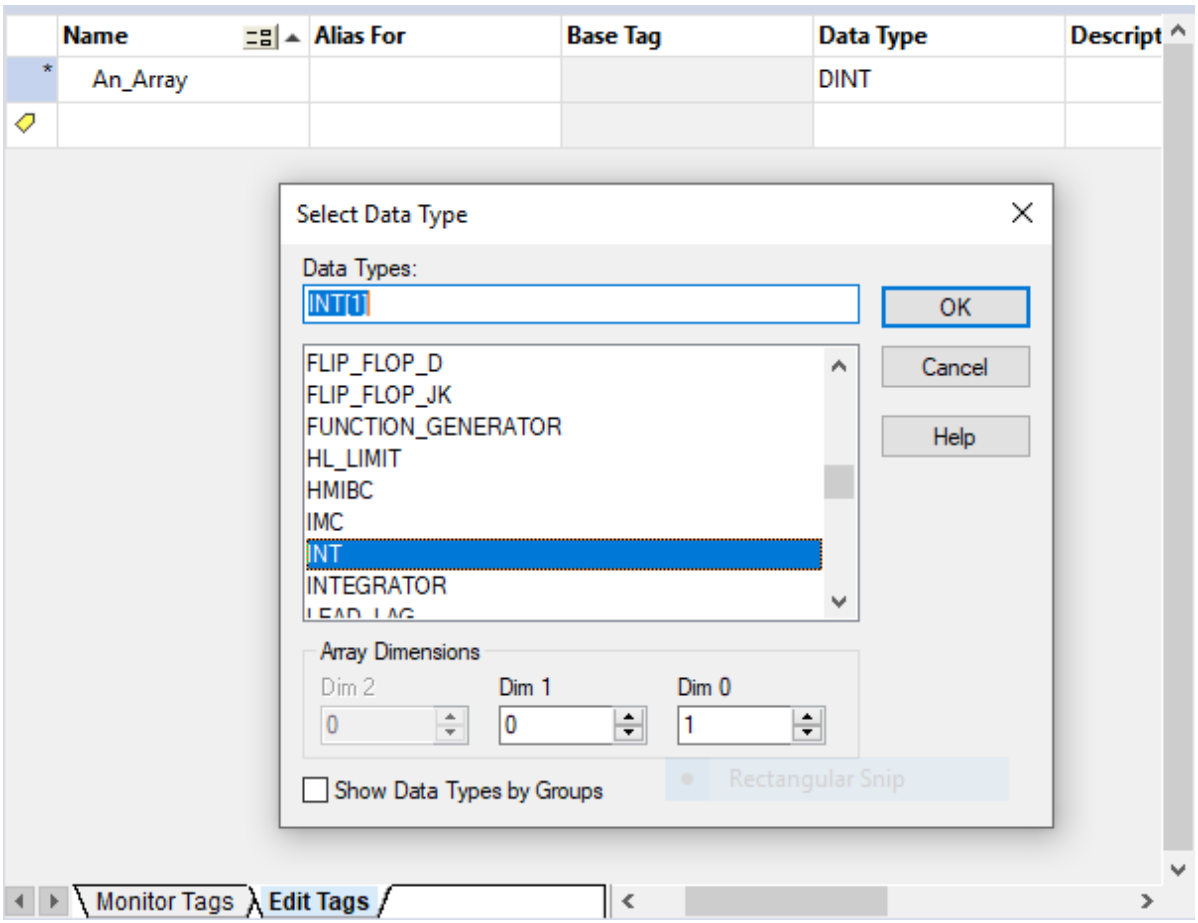


Fig. 13-5 Matrix defined in A-B's Tag Table

A-B Data Handling in ControLogix PLCs

Examples of the types of addressing available in the ControLogix processor will demonstrate addressing used in the newer A-B processors. In the following, rung 2 turning on a coil out1 is not as simple as it first may appear. A table table1 is entered in the database seen below. The table consists of 320 bits. Two separate pointers determine the status of a particular bit. Index1 may vary from 0 to 9 determining the word in the table to view. Index2 may vary between 0 and 31. The value of *index2* determines the bit in the word being viewed.

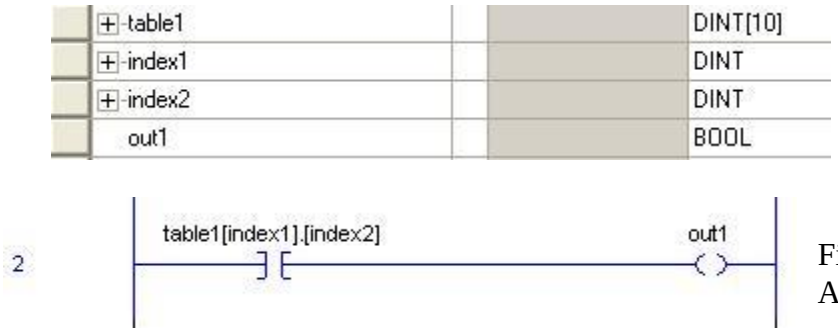
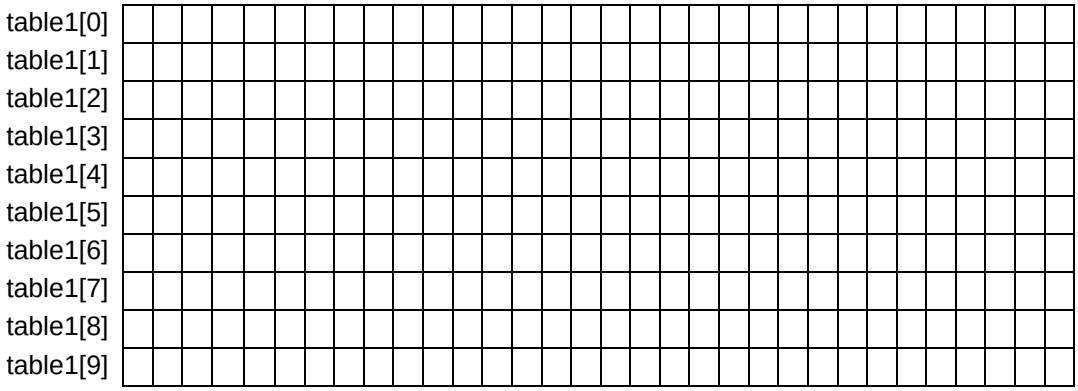


Fig. 13-8 A-B Array Bit Addressing



Consider the table expanded further as table2[index1,index2],[index3]. The index capabilities of arrays significantly enhance the power of the PLC to perform complicated operations in an orderly manner.

The following array in rung 3 uses a toggle bit to execute an ADD block. In this example, the same 10 word table is used to selectively add one to a second word in the table.

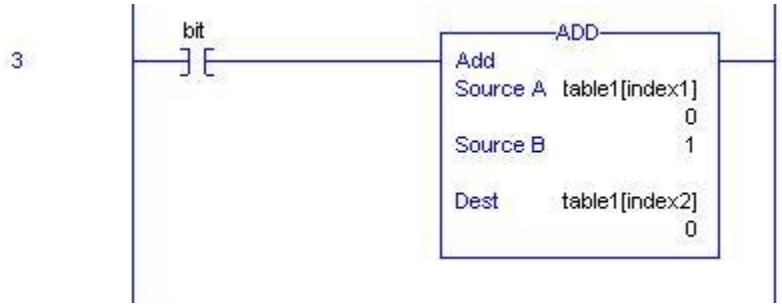


Fig. 13-9 A-B Array Word Addressing

table1[0]	
table1[1]	
table1[2]	
table1[3]	
table1[4]	
table1[5]	
table1[6]	
table1[7]	
table1[8]	
table1[9]	

For instance, if index1 = 1 and index2 = 1, the instruction would add 1 to table1[1]. If index1 = 3 and index2 = 4, the contents of table1[4] would be updated with the contents of table1[3] + 1. For-Next Loops are not traditionally included in ladder logic for the reason that a scan can appreciably be lengthened if any loop is executed. For-Next Loops provide a looping control mechanism that is very useful but is capable of lengthening the loop execution time. They have been included in the PLC-5 but not in the SLC ladder instruction set. They are also included in the RSLogix 5000 programming software for both the Control Logix processors as well as the Compact Logix processors. Excluded from the Control Logix and Compact Logix processors, however, are the two addressing modes: indexed and indirect.

The For-Next loop requires an indexer that increments each pass the program makes through a subroutine. The subroutine becomes the program used to execute the logic of the For-Next operation.

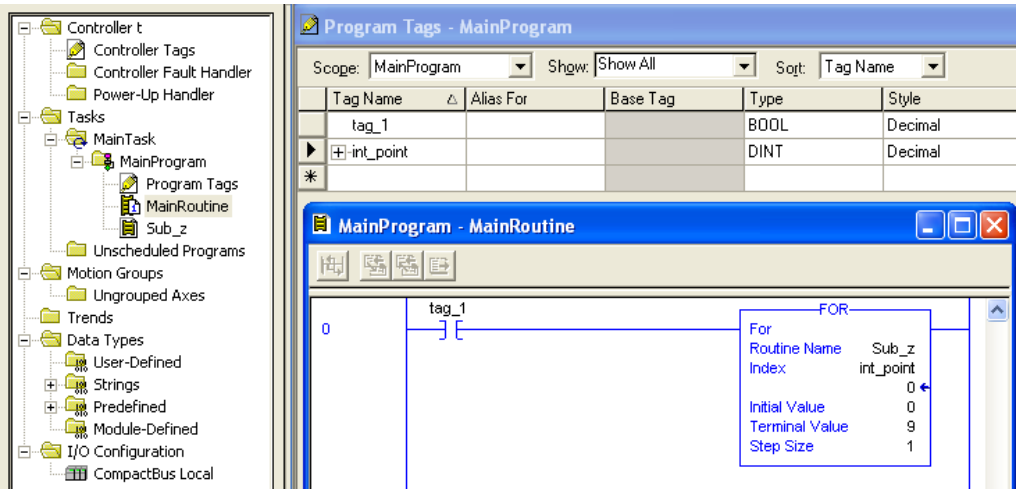


Fig. 13-10
Subroutine Loop
for CompactLogix

In the above example, the routine MainRoutine executes. Each scan that tag_1 energizes, the FOR loop Sub_z executes. The program Sub_z executes 10 times with the value of int_point incremented by 1 from 0 to 9.

Of course, Sub_z allowed to call a subroutine as well with the effect of providing a loop within a loop. The looping procedure is used for table manipulation similar to the MOV and COP commands described in this chapter. If the FOR command is not available, a method using a variable number of scans to execute a subroutine may be employed.

UDT's

Both A-B and Siemens use UDTs for defining variable arrays of data.

A user-defined structure can contain any base data type (e.g., SINT, INT, DINT, BOOL, REAL) or structure (either predefined or user-defined). In addition, a single-dimensional array can be included as a structure member.

To create a user-defined structure, right click on the User-Defined folder in the Controller Organizer, and choose New Data Type. The Data Type editor will appear, from which you can define your new data type.

Many control programs require the ability to store blocks of information in tables that can be traversed at runtime. RSLogix 5000 supports this requirement by providing the ability to create custom arrays with up to three separate dimensions (i.e., row, column, and depth). Individual cells within an array may contain any base data type (e.g., SINT, INT, DINT, BOOL, REAL) or structure (either predefined or user-defined).

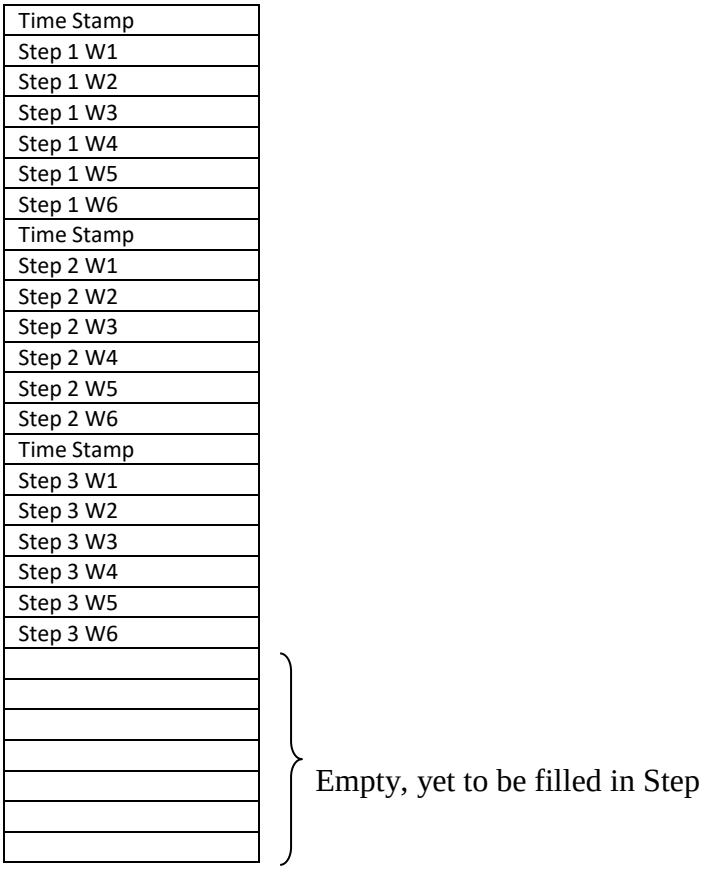
The example of a UDT is from the database example of a field. The field defines a set of data of different data types and allows entry into a table using these data types. UDTs are the same. An example of a database table 'employees' is as follows:

```
CREATE TABLE employees (
    emp_no      INT             NOT NULL,  -- UNSIGNED AUTO_INCREMENT??
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,  -- Enumeration of either 'M' or 'F'
    hire_date   DATE            NOT NULL,
```

Circular Table

One of the file types studied in a programming course that may be useful in the Recipe Report portion of the program is the circular table structure. The last step executed is presented in an area with its time stamp and the computer is assumed to pick up the actual step information and record it. If multiple steps are available for the polling computer, then the circular file structure becomes more important. For instance, the last few steps of information are kept in the file. The PLC program continues to increment through the steps working its way down the recipe and sending the actual step information to the circular table. When the end of the circular table is reached, the PLC starts again at the top of the table. This approach allows steps to be read multiple times by the computer. If the polling computer fails to read a record, the table will continue to fill till the end of the table is reached. Only when the table is full and wraps over old data will data be lost if the polling computer is still unable to read the data.

An example of a circular table:



While data may be stored in the PLC, most large databases are kept in a computer closely linked to the PLC. The circular table may be used to buffer data if the computer is not responding to the PLC. A database is useful in organizing large amounts of data. Using a database for applications such as this offloads the PLC from the burden of data handling, giving the PLC the flexibility to do what it is designed to do best - control machines.

From the Easy Book "

6.4 Easy to create data logs

Your control program can use the Data log instructions to store run-time data values in persistent log files. The data log files are stored in flash memory (CPU or memory card). Log file data is stored in standard CSV (Comma Separated Value) format. The data records are organized as a circular log file of a pre-determined size.

The Data log instructions are used in your program to create, open, write a record, and close the log files. You decide which program values will be logged by creating a data buffer that defines a single log record. Your data buffer is used as temporary storage for a new log record. New current values must be programmatically moved into the buffer during run-time. When all of the current data values are updated, you can execute the DataLogWrite instruction to transfer data from the buffer to a data log record.

You can open, edit, save, rename, and delete data log files from the File Browser page of the Web Server. You must have read privileges to view the file browser and you must have modify privileges to edit, delete, or rename data log files.

Use the DataLog instructions to programmatically store run-time process data in flash memory of the CPU. The data records are organized as a circular log file of a pre-determined size. New records are appended to the data log file. After the data log file has stored the maximum number of records, the next record written overwrites the oldest record. To prevent overwriting any data records, use the DataLogNewFile instruction. New data records are stored in the new data log file, while the old data log file remains in the CPU.

Table 6- 21 DataLogWrite instruction

LAD/FBD	SCL	Description

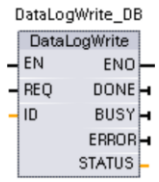
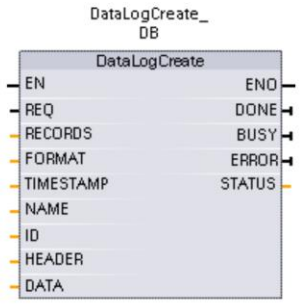
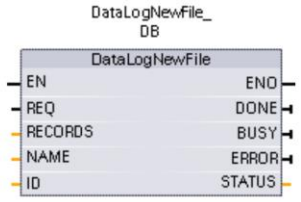
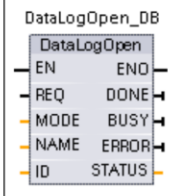
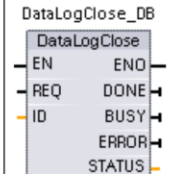
	`"DataLogWrite_DB"(`
	`req:=FALSE,`
	`done=>_bool_out_,`
	`busy=>_bool_out_,`
	`error=>_bool_out_,`
	`status=>_word_out_,`
	`ID:=_dword_inout_);`
		DataLogWrite writes a data record into the specified data log. The pre-existing target data log must be open.
		You must programmatically load the record buffer with current run-time data values and then execute the DataLogWrite instruction to move new record data from the buffer to the data log.
		If there is a power failure during an incomplete DataLogWrite operation, then the data record being transferred to the data log could be lost.

Table 6- 22 DataLogCreate and DataLogNewFile instructions

LAD/FBD	SCL	Description
	"DataLogCreate_DB"(req:=FALSE, records:=1, format:=1, timestamp:=1, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, name:=_variant_in_, ID:=_dword_inout_, header:=_variant_inout_, data:=_variant_inout_);	DataLogCreate[1] creates and initializes a data log file stored in the \DataLogs directory of the CPU. The data log file is created with a pre-determined fixed size.
	"DataLogNewFile_DB"(req:=FALSE, records:=1, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, name=:_variant_in_, ID:=_dword_inout_);	DataLogNewFile[1] allows your program to create a new data log file based upon an existing data log file. A new data log will be created and implicitly opened based with the specified NAME. The header record will be duplicated from the original data log along with the original data log properties. The original data log file will be implicitly closed.

[1] The DataLogCreate and DataLogNewFile operations extend over many program scan cycles. The actual time required for the log file creation depends on the record structure and number of records. Before the new data log can be used for other data log operations, your program logic must monitor the transition of the DONE bit to TRUE.

Table 6- 23 DataLogOpen and DataLogClose instructions

LAD/FBD	SCL	Description
	"DataLogOpen_DB"(req:=FALSE, mode:=0, name:=_variant_in_, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_, ID:=_dword_inout_);	The DataLogOpen instruction opens a pre-existing data log file. A data log must be opened before you can write new records to the log. Data logs can be opened and closed individually. Eight data logs can be open at the same time.
	"DataLogClose_DB"(req:=FALSE, done=>_bool_out_, busy=>_bool_out_, error=>_bool_out_, status=>_word_out_,	The DataLogClose instruction closes an open data log file. DataLogWrite operations to a closed data log result in an error. No write operations are allowed to this data log until another DataLogOpen operation is performed. A transition to STOP mode closes all open data log files.

”

Indirect Addressing for SLC and PLC/5 PLCs (Older Addressing Schemes)

This addressing format allows a storage location to specify the number in the file of the file, element, or bit in the direct logical address. Up to two address numbers in the direct address are allowed with indirect addresses.

For example: #F[N7:4]:0 identifies a floating point file whose number is found in the N7:4location. If this location contains 99, the 99th file would be accessed as element 0. This would be equal to F99:0.

Rules for using Indirect addressing:

1 Indirect addresses are indicated with brackets

2 Address must be direct logical address with N, T, C, or R types. Type N (Integer) is recommended but not required.

3 # is not to be used inside brackets

4 The element number must be within the file's length. A fault will occur if not.

5 When used to store file number, the file number must represent the same type as the prefix.

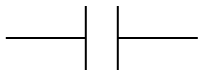
Examples:

B3/[ ], B[ ]/[ ], N7:[ ]/8, C[ ]:5.DN

Example:

The following contact in a program would have various values depending on the value in N7:0.

B3/[N7:0]



N7:0	B3/x Reference
0	B3/0
1	B3/1
10	B3/10
67	B3/67

Example:

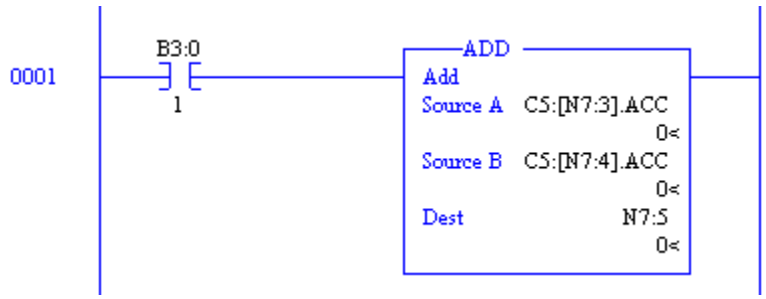


Fig. 13-11 A-B Indirect Addressing Example from SLC Instruction Set

The example makes the counter variables being added selectable from any of the counters in C5.

Note that addresses are not limited to whole word addresses. Bit addresses may be referenced as well.

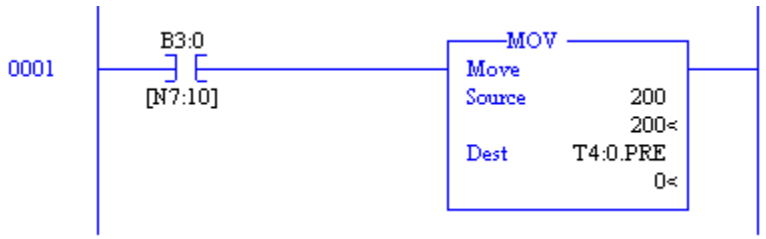


Fig. 13-12 A-B Indirect Addressing Example from SLC Instruction Set

The rung references B3/ and then a bit in the B3 table referenced from 0 using N7:10 as that reference. For example, if N7:10 contained the number 30, the bit reference would be B3/30 or B3:1/14.

An editorial comment on Indirect Addressing**:**

It is hard to debug or troubleshoot if you are not the initial programmer. So, don't use it unless you find no alternative means to program the task at hand.

It is believed that the use of this addressing method to obscure the logic behind the program led the designers to discontinue it with the RSLogix 5000 language. Both the indirect and indexed methods were discontinued with the newer languages.

If one becomes fluent with all the new languages, the procedural STL language may be the best to be used for indexing program development. Although we predominantly use LAD here, the STL language has many benefits and should be considered when planning a program with indexing in mind.

Indexed Addressing for SLC and PLC/5 PLCs

Indexed addressing allows an offset of an address by a number of words stored in location S:24. To identify indexed addressing, place the # symbol immediately before the file-type identifier in the address:

For example: #N7:0 refers to word 0 of the N7 file offset by the number stored in S:24.

When using Indexed Addressing:

- Use care to insure that the index value (positive or negative) does not exceed file bounds.
- With instructions using two or more indexed addresses in the same instruction, the offset will be the same for all addresses.
- Use care to reset the offset to its desired value before enabling an instruction having indexed addresses.

Example:

The MVM instruction uses indexed addressing as follows:

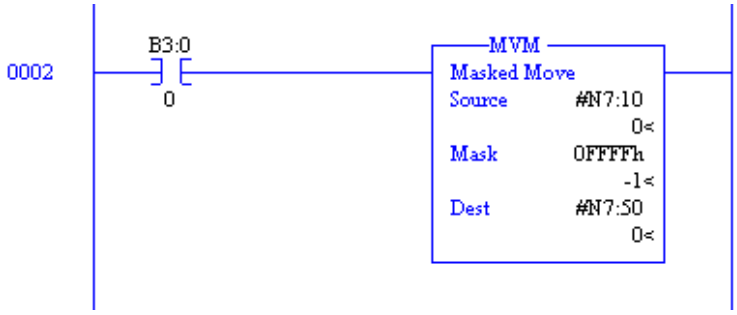
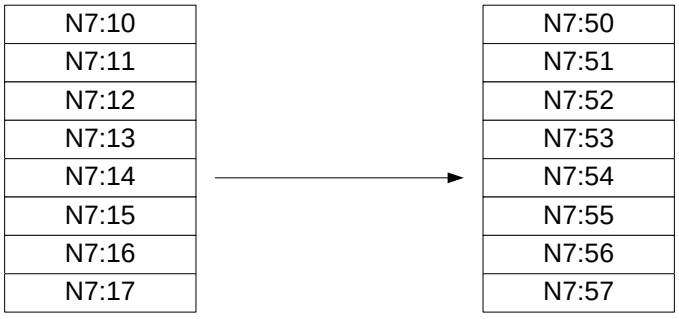


Fig. 13-13 A-B Indexed Addressing Example from SLC Instruction Set

Value	Base	Offset(Value in S:24)	Actual Address
Source	N7:10	4	N7:14
Destination	N7:50	4	N7:54



Using the MOV instruction as a reference, observe that the table of results uses the offset value found in S:24 to compute a new address for the MOV instruction. The example above moves the value in N7:14 to N7:54 when the offset in S:24 = 4. Only one word is moved when the rung is executed. Other program statements should be added to the rung to increment or decrement S:24 to move other locations.

Another example using one offset address and one fixed address:

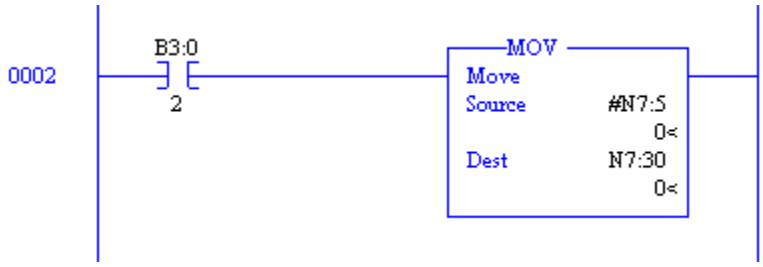
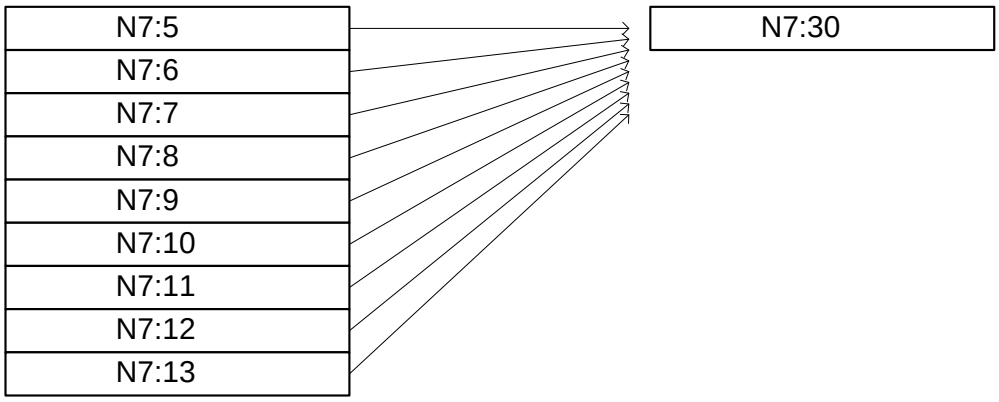


Fig. 13-14 A-B Indexed Addressing Example from SLC Instruction Set

Source	Offset (S:24)	Actual Source	Destination
N7:5	0	N7:5	N7:30
N7:5	1	N7:6	N7:30
	2		
	3		
	4		
	20		

This is commonly referred to as a Table to Fixed move. This type of rung is used to get a value sequentially or randomly from a table. It is used in programming recipe routines.



An example using a fixed first address and an indexed second address is included. It is used to get a value from a fixed location into a table. It is used many times to save the status of an event that happened in sequence. Picture a line of cars with numeric only license plates. The output table would save the sequential status these numbers starting with N7:50 and sequentially store the license plates in order as they passed by a point.

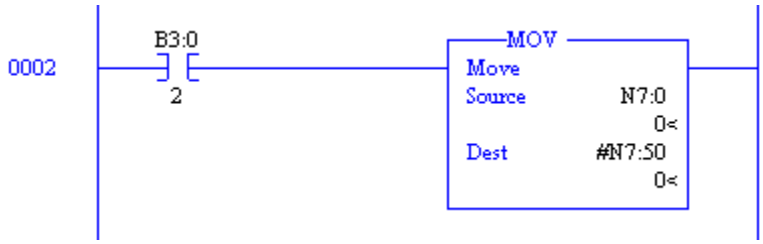
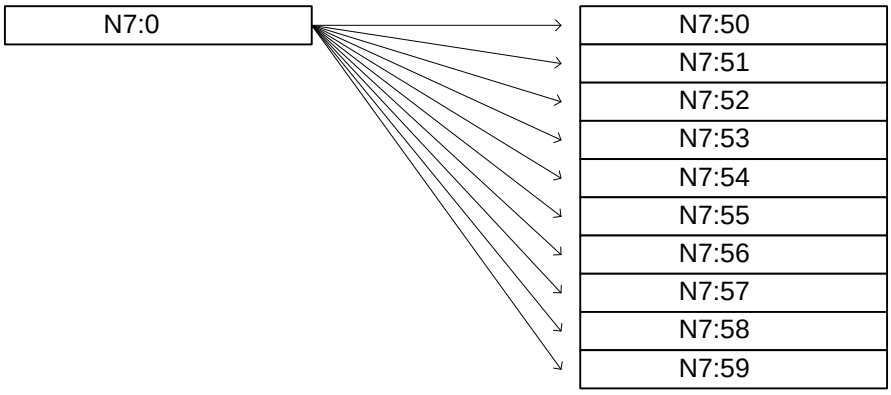


Fig. 13-15 A-B Indexed Addressing Example from SLC Instruction Set

Source	Destination	Offset(S:24)	Actual Dest
N7:0	N7:50	0	N7:50
N7:0	N7:50	1	N7:51
		2	
		3	
		4	
		20	



To enter a program using Indexed Addressing, build a table as follows and enter the program listed below:

Address	Value
N7:0	5
N7:1	10
N7:2	15
N7:3	20
N7:4	25
N7:5	30
N7:6	35
N7:7	40
N7:8	45

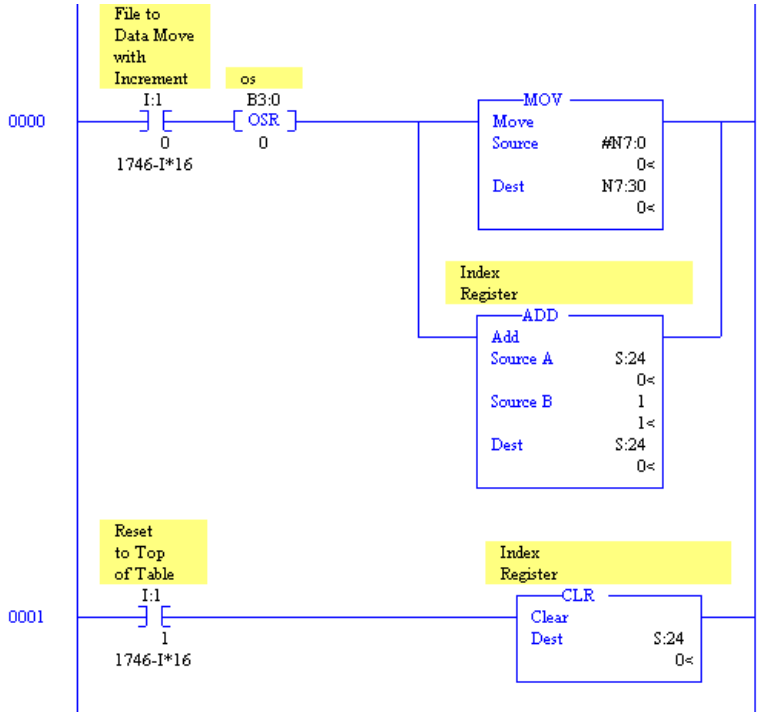


Fig. 13-16 A-B Indexed Addressing Example from SLC Instruction Set

Using the SLC processor to execute a For-Next loop employs a technique that allows a variable number of scans to access the subroutine code. For instance, the following code could accomplish the same function as that shown above if timing constraints were considered.

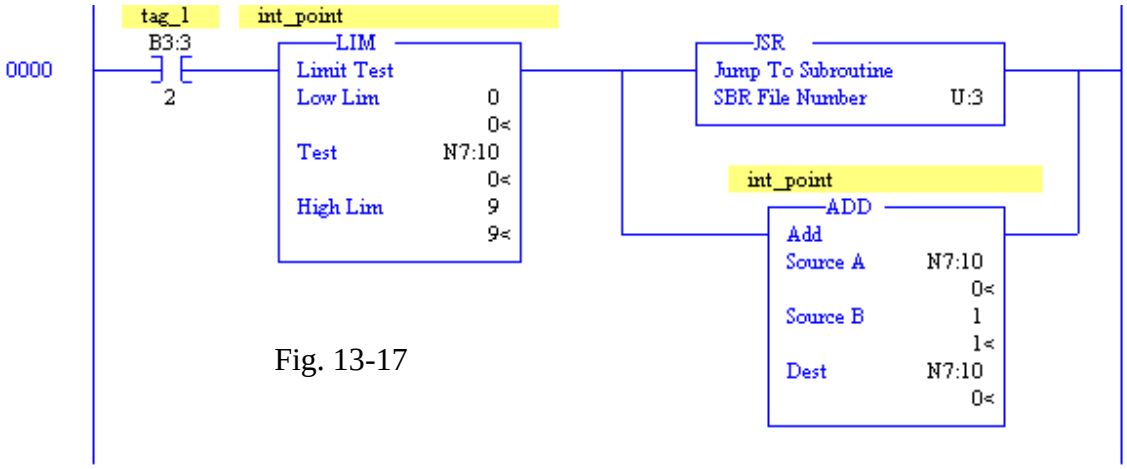


Fig. 13-17

The program called in U:3 may be used to respond in a manner similar to the For-Next program if the output of the code can wait the number of scans needed to loop through the entire range of numbers in int_point. Care must be taken to initialize N7:10 to 0 prior to execution of the code as well. When the final path through the loop is executed a bit usually is set to signal any user

program that data from the loop is available. This is more cumbersome than the program of the Compact Logix processor. **It also does not easily support loops within loops**. Extreme care must be taken when using this program technique. It is important to note that this type of program control is used by many programmers and the student should be aware of its implementation in existing automation programming.

Comparing Older AB Addressing Modes

As advances are made from the PLC/5 to SLC to RSLogix 5000 processors, addressing requirements have been enhanced as well. The RSLogix 5000 processors use indexed arrays to provide functional equivalent programming to the indexed and indirect methods of the PLC/5 and SLC processors. Modes may be mixed, causing a number of programming types which may or may not be substituted with the indexed array of RSLogix5000. The following list of comparisons shows some of the evolution from the PLC/5 to SLC to RSLogix5000 addressing.

Indexed:	`#N7:0` Supported by PLC/5, SLC Available in RSLogix5000 by using Indexed Arrays
Indirect Word:	`(N7:[N7:6])` Supported by PLC/5, SLC Available in RSLogix5000 by using Indexed Arrays
Indirect File:	`(N[N7:4]:0)` Supported by PLC/5, SLC Not available in RSLogix5000
Indexed + Indirect Word:	`(#N:[N7:2])` Supported by PLC/5, SLC Available in RSLogix5000 by using Indexed Arrays
Indexed + Indirect File:	`(#N[N7:8]:0)` Supported by PLC/5, SLC Not available in RSLogix5000
Indexed + Indirect File + Word:	`(#N[N7:3]:[N7:4])` Supported by PLC/5, SLC Not available in RSLogix5000

Nested Indirection: `(N7:[N7:[N7:2]])`

Supported by PLC/5, SLC
Not available in RSLogix5000

While PLC/5 and SLC processors use various combinations of Indexed and Indirect pointers to move data, the RSLogix5000 processors use indexed arrays to accomplish the same task.

FOR-NEXT looping is not used in the SLC processor family but is supported by the PLC/5 family. Use of multiple scans is necessary to provide the equivalent functionality to FOR-NEXT looping when using the SLC processors.

Comparison of MOVE Instructions (with Examples from Siemens and A-B)

The Table to Table Move

The MVM instruction uses indexed addressing as follows:

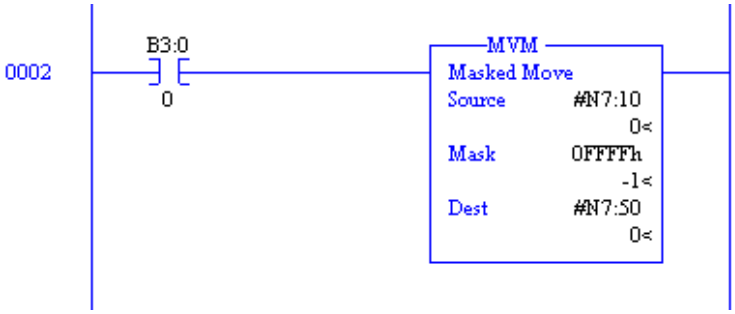
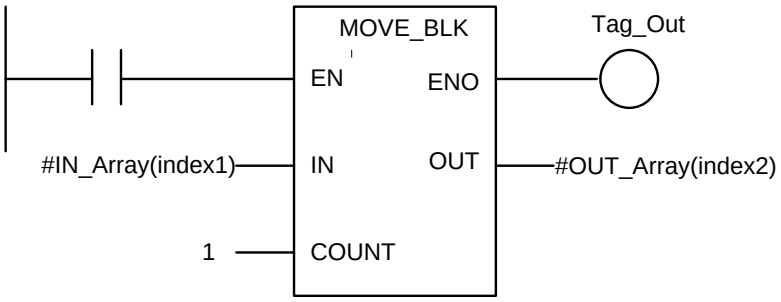


Fig. 13-18

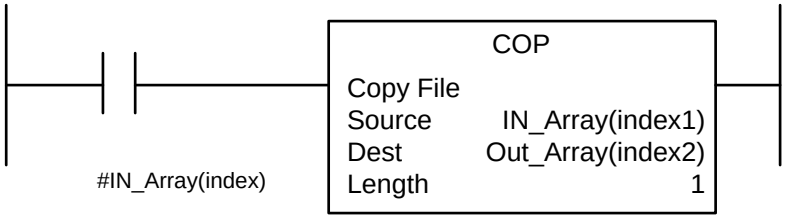
Value	Base	Offset(Value in S:24)	Actual Address
Source	N7:10	4	N7:14
Destination	N7:50	4	N7:54

N7:10	N7:50
N7:11	N7:51
N7:12	N7:52
N7:13	N7:53
N7:14 →	N7:54
N7:15	N7:55
N7:16	N7:56
N7:17	N7:57

Siemens' Move equivalent of the indexed address move above:



Allen-Bradley's Move equivalent of the indexed address move above:



Notice that both the Siemens and A-B newer move statements allow more than one element to be moved. Also notice that index values for the two arrays may be the same or different.

The Table to Register Move

Another example using one offset address and one fixed address:

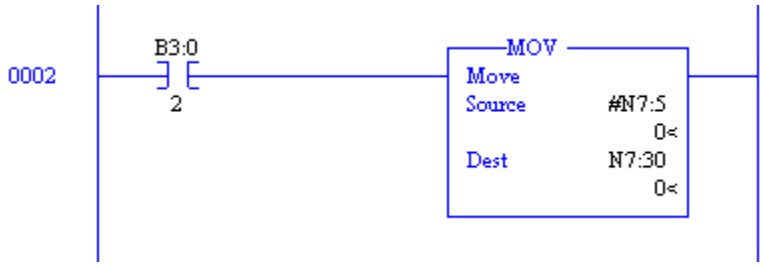
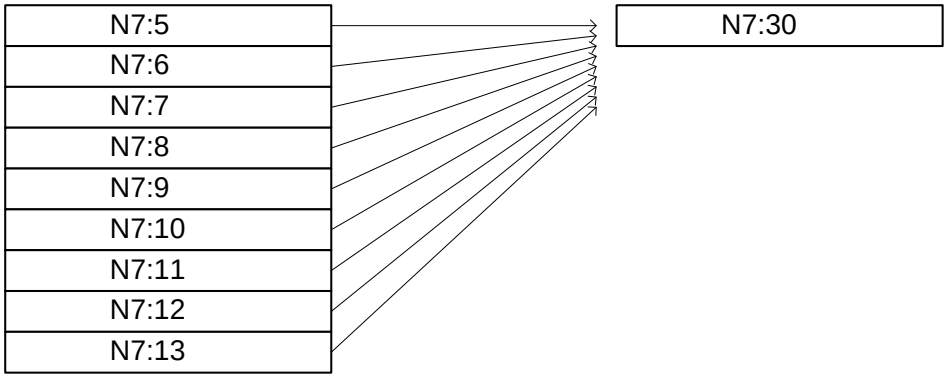


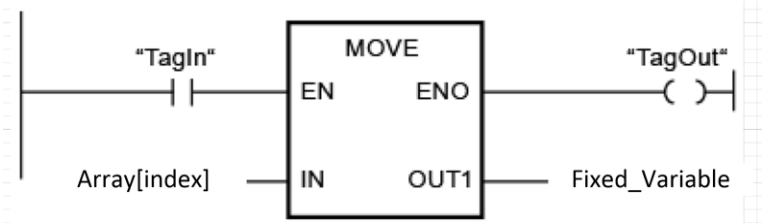
Fig. 13-19

Source	Offset (S:24)	Actual Source	Destination
N7:5	0	N7:5	N7:30
N7:5	1	N7:6	N7:30
	2		
	3		
	4		
	20		

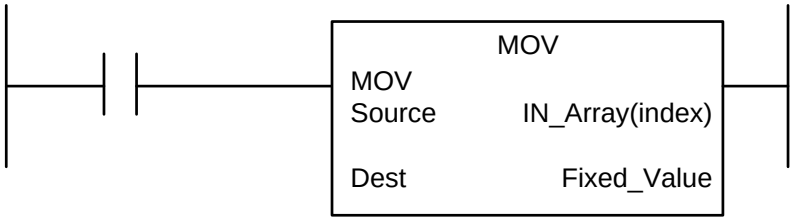
This is commonly referred to as a Table to Fixed move. This type of rung is used to get a value sequentially or randomly from a table. It is used in programming recipe routines.



Siemens' Move equivalent of the indexed address move above:



Allen-Bradley's Move equivalent of the indexed address move above:



The Register to Table Move

An example using a fixed first address and an indexed second address is included. It is used to get a value from a fixed location into a table

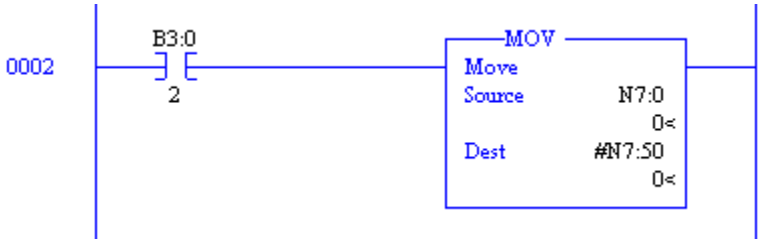
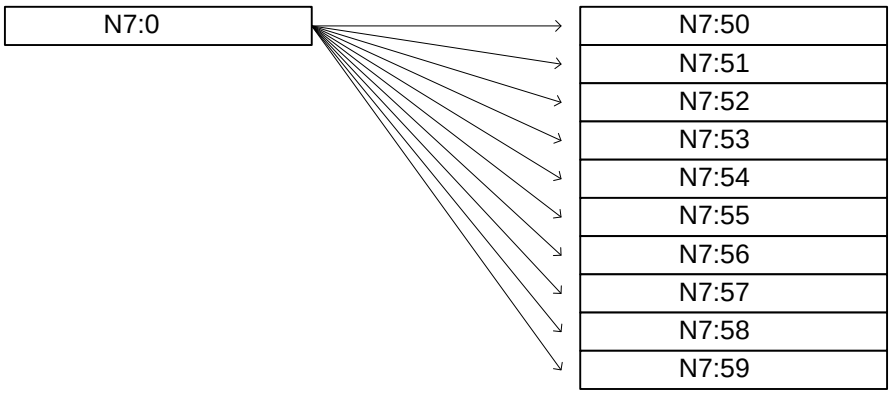


Fig. 13-20

Source	Destination	Offset(S:24)	Actual Dest
N7:0	N7:50	0	N7:50
N7:0	N7:50	1	N7:51
		2	
		3	
		4	
		20	



Siemens' Move equivalent of the indexed address move above:

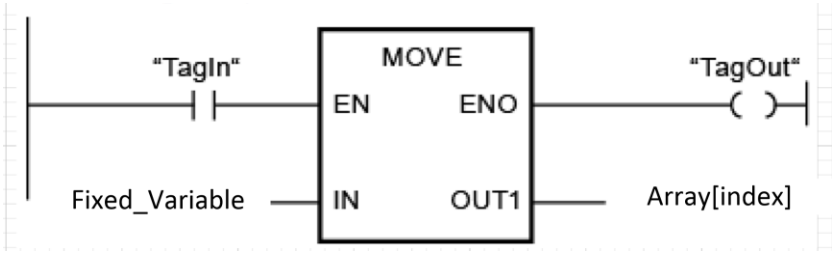


Fig. 13-21a

Allen-Bradley's Move equivalent of the indexed address move above:

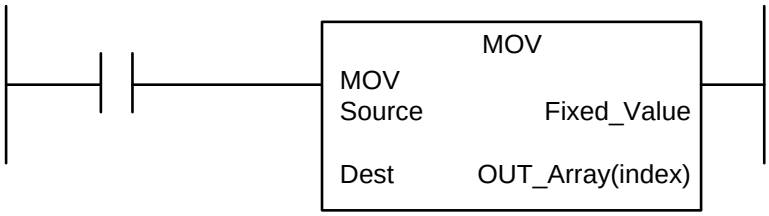


Fig. 13-21b

Scale Weighing

Scale Weighing Systems are an important part of a batch system. Siemens provides a load cell interface system for weighing applications called the SIWAREX WP231. It is pictured in Fig. 13-22 below.

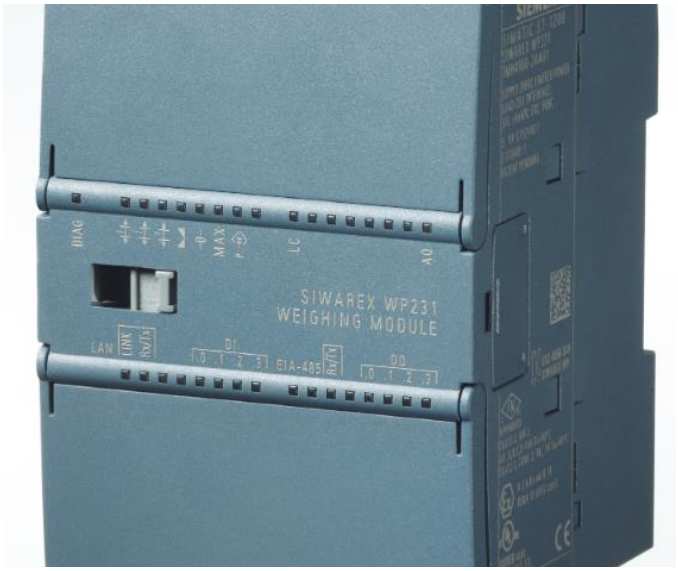


Fig. 13-22

The electronic weighing system has the following characteristics as listed by Siemens:
“

- Uniform design technology and consistent communication in SIMATIC S7-1200
- Parameter assignment by means of HMI panel or PC
- Uniform configuration option in the SIMATIC TIA Portal
- Measuring of weight with a resolution of up to 4 million divisions
- High accuracy, 3000 d, legal for trade according to OIML R76
- Legal-for-trade display with SIMATIC operator panel or PC
- High measuring rate of 100/120 Hz (effective interference frequency suppression)
- Limit monitoring
- Flexible adaptation to varying requirements
- Easy calibration of the scales using the SIWATOOL program
- Automatic calibration is possible without the need for calibration weights
- Module replacement is possible without recalibrating the scales
- Use in Ex Zone 2 / ATEX approval
- Intrinsically safe load cell supply for Ex Zone 1 (SIWAREX IS option)
- Diagnostics functions”

SIWATOOL overview

SIWATOOL does not only offer support when you set the scale but also when you analyze the diagnostic buffer that can be saved after being read out of the module together with the parameters. The display of the current scale status can be configured.

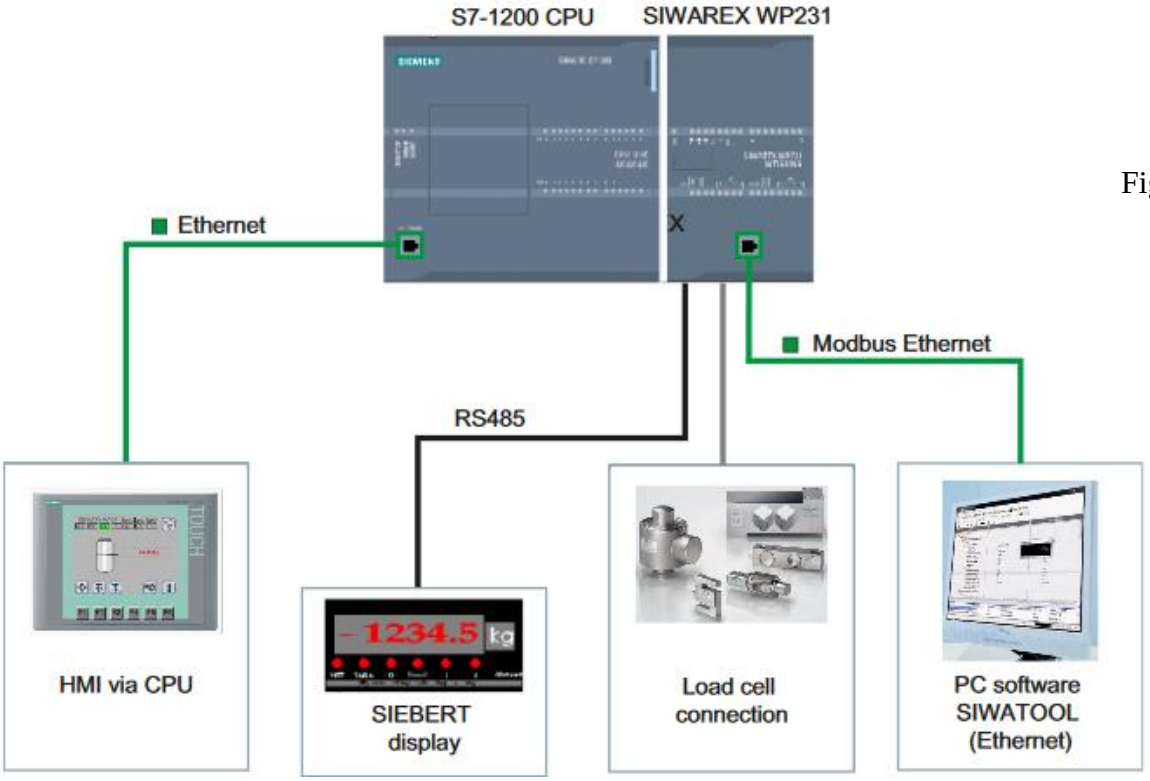


Fig. 13-23

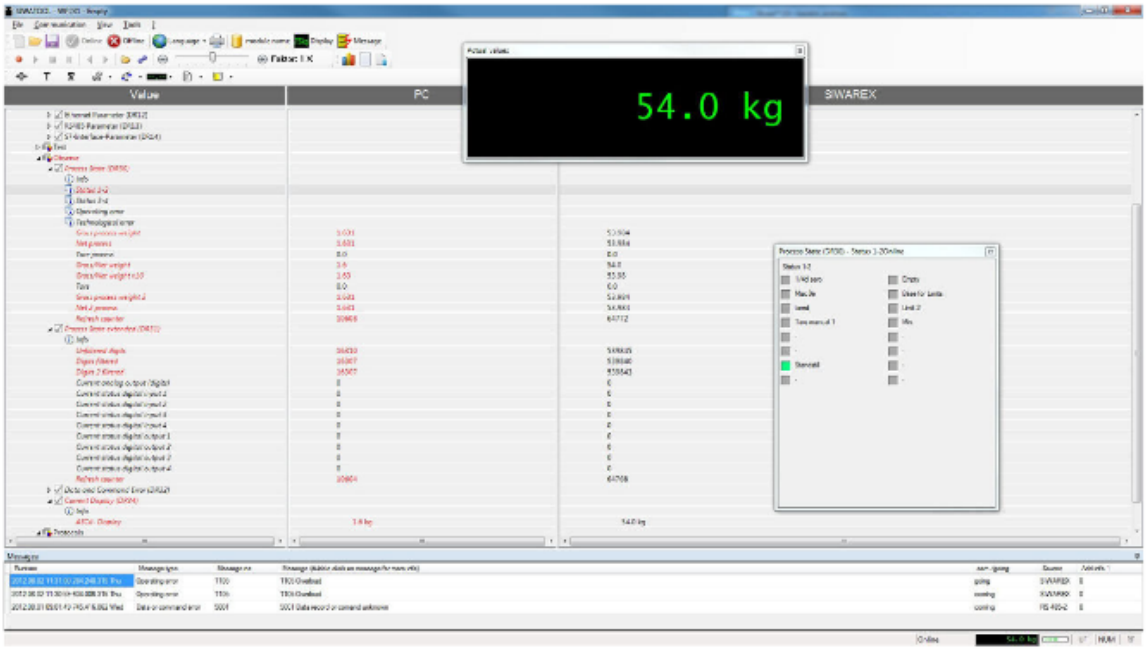


Fig. 13-24

A load cell is pictured in the figure below:

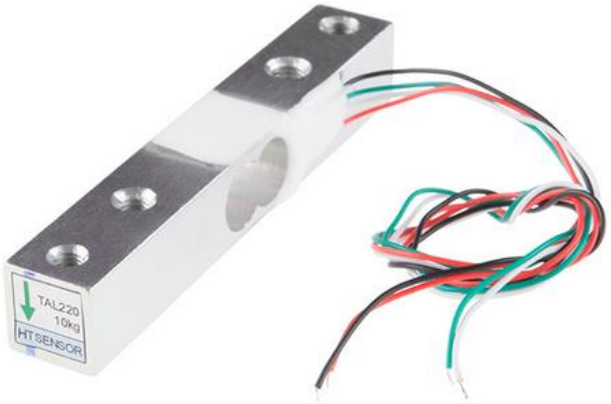


Fig. 13-25

A weigh vessel with load cells usually includes four load cells. Some vessels may only include three load cells, however.

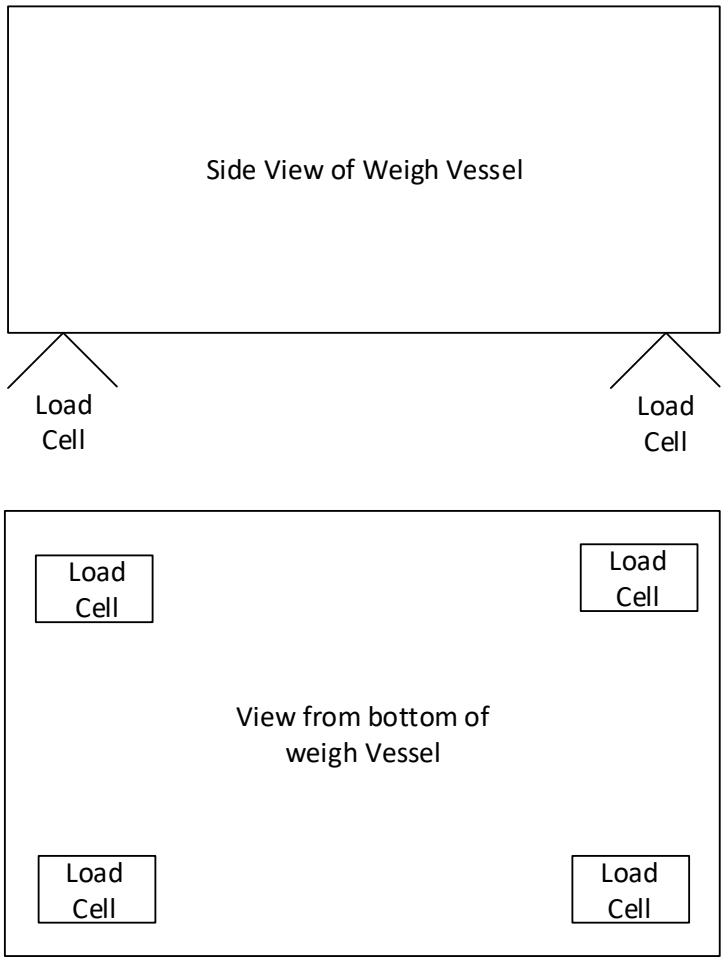


Fig. 13-26

There are also load cells located under traveling weigh belts. Systems that weigh moving material include averaging algorithms that weigh the belt and material but tare the weight of the belt out of the weight.

Fig. 13-27

The following weigh vessels show the location of load cells with the vessel suspended in space. These weigh vessels show the traditional locating of load cells at four corners of the tank.

Fig. 13-28

The Siemens system is described in the manual:

Weighing systems Electronic weighing system SIWAREX WP231.

Also, the load cell can be terminated in a converter box similar to the Red Lion Strain/Load Cell Panel Meter pictured below. Since it can be used to interface to a PLC using an optional analog output, the Red Lion is used simply to pass through a signal from the scale to the PLC after linearization has occurred with the scale signal.

Red Lion PAXS0000 Strain/Load Cell Panel Meter

Red Lion – Mfr # PAXS0000 – Item # EW-68486-20

No Reviews Write the First Review

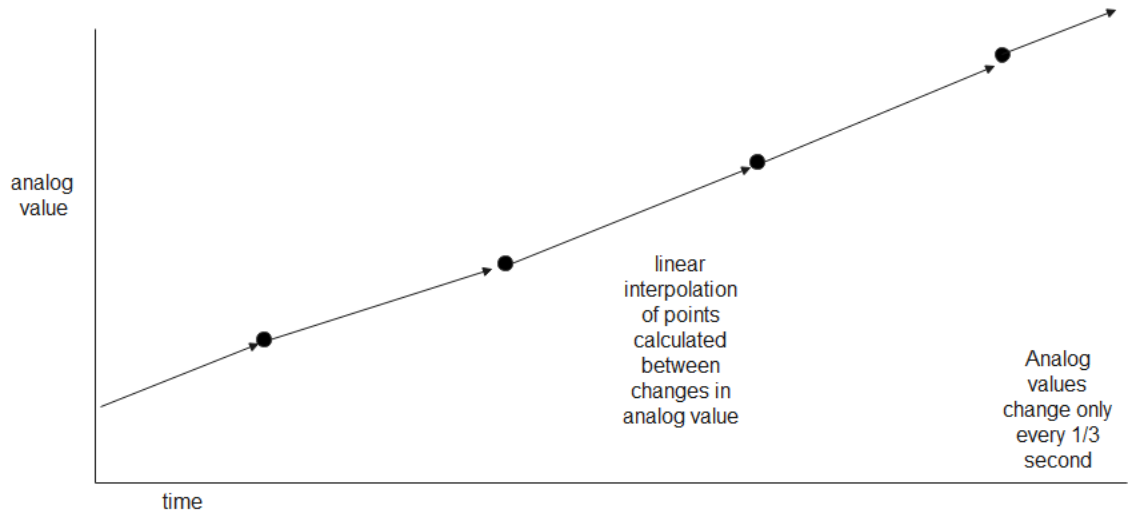
Upgradeable for customized solutions – Choose from temperature, process, voltage, and current inputs

- Process, Voltage, Current, Temperature, And Strain Gauge Inputs
- 6-Digit 0.56" Red Sunlight Readable Display
- Variable-intensity display for viewing in all lighting conditions
- 16 Point Scaling For Non-Linear Processprogrammable Function Keys/User Inputs
- 9 Digit Totalizer (Integrator) With Batching

Fig. 13-29

Since this device only updates every 1/3 second, there must be programs written to articulate between data points and anticipate when the output should turn off based on the rate of change of the scale.

From the mathematical calculations of rate of change, a predicted value should allow the program to turn off a valve very close to the actual turn-off point. This is left as an exercise.



Check List for Writing PLC Batch Application

From the ISA website, the following gives an insight into considerations for the batching application:

"The ISA88 committee has published a series of standards on batch control in industrial automation systems."

They are:

ANSI/ISA-88.00.01-2010, *Batch Control Part 1: Models and Terminology*

This standard "provides standard models and terminology for defining the control requirements for batch manufacturing plants"

ANSI/ISA-88.00.02-2001, *Batch Control Part 2: Data Structures and Guidelines for Language*

This standard "defines data models that describe batch control as applied in industrial automation systems, data structures for facilitating communications within and between batch control implementations, and language guidelines for representing recipes"

ANSI/ISA-88.00.03-2003, *Batch Control Part 3: General and Site Recipe Models and Representation*

This standard "defines a model for general and site recipes; the activities that describe the use of general and site recipes within a company and across companies; a representation of general and site recipes; and a data model of general and site recipes"

ANSI/ISA-88.00.04-2006, *Batch Control Part 4: Batch Production Records*

This standard "provides a detailed definition for batch production records, establishing a reference model for developing applications for the storage and/or exchange of batch production records. Implementations based upon the standard will allow retrieval, analysis, and reporting of selected batch production record data"

A fifth standard is under development:

Implementation Models & Terminology for Modular Equipment Control

The purpose of the SP88-ISA-88 specification is to provide a common strategy for all batch applications programmed. These include PLC programs. To be included are the physical and functional implementations. The functional model includes "the relationships between the five types of control recipe management, scheduling, sequential control, regulatory control, and safety interlock systems."

Examples are the definitions for modes, states and alarms for a typical batch system. They are

defined in the specification as:

- Auto
- SemiAuto
- Manual
- Bypassed
- Controlled
- Reset
- AlarmPresent
- AlarmAcknowledged

As discussed previously, the reporting function is important in that values are required by the higher level computer. Such values as actual weights by the scale, errors, status of the batch should be saved and reported for each step of the batch.

While the chemical engineer or mechanical engineer is usually in charge of determining the equipment design and feeds for the process, the electrical engineer/program designer is responsible for programming the process. Thus there is a concern and desire to have input in the process from the beginning. The PLC program should be knowledgeable of the feed rates, the ingredients to fill, the coordination of when and how to mix and dump the material, when to apply heat, etc.

With the batching system, scales and other material feed devices are required to monitor and control the feed rates of the various materials. SP-88 defines the following material feed types:

- **FILLING**
  "The SINGLE transfer (movement) of a specified amount of product from one single location to another location"

- **DOSING**
  "The SINGLE transfer (movement) of a specified amount of product from one location into a continuous process"
- **FORMULATION**
  "MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location"

- **BLENDING**
  "MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location plus a single ADDITIONAL process phase – mixing"

- **BATCHING**

  "MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location plus multiple ADDITIONAL process phases – heating, cooling, wait, mix, agitate, dump etc."

The different methods of adding material to a batch are discussed. Most involve scales but there are those that involve a meter or level sensor. Methods for measuring material include:

Material Feed Types

- Gain In Weight Feeders
- Loss In Weight Feeders
- Flow Meter Feed or Metered Feed
- Dump To Empty
- Hand Add

To add material using any of the above methods, a scale or load cell system should probably be used to accurately weigh the product being added. The scale can be either on the container being added to or fed from. The steps in weighing are critical in that consistency needs to be maintained and proper records kept of all transactions. Decisions need to be made as well whether the weight added is acceptable or not. The SP-88 document divides the weighing cycle into six states as follows:

- Pre Feed
- Feed Start
- Feed
- Feed Stop
- Feed Finish
- Post Feed

It may be important to the programmer to identify every step and build their program around these steps. Certainly the steps add order to the program and aid in troubleshooting the system. The state diagram shown below also helps in identifying areas of concern and giving aid in programming the feed.

State Diagram of Material Add System

A state diagram is included in the SP88 diagram for each step of adding material and used as a review of the steps outlined above:

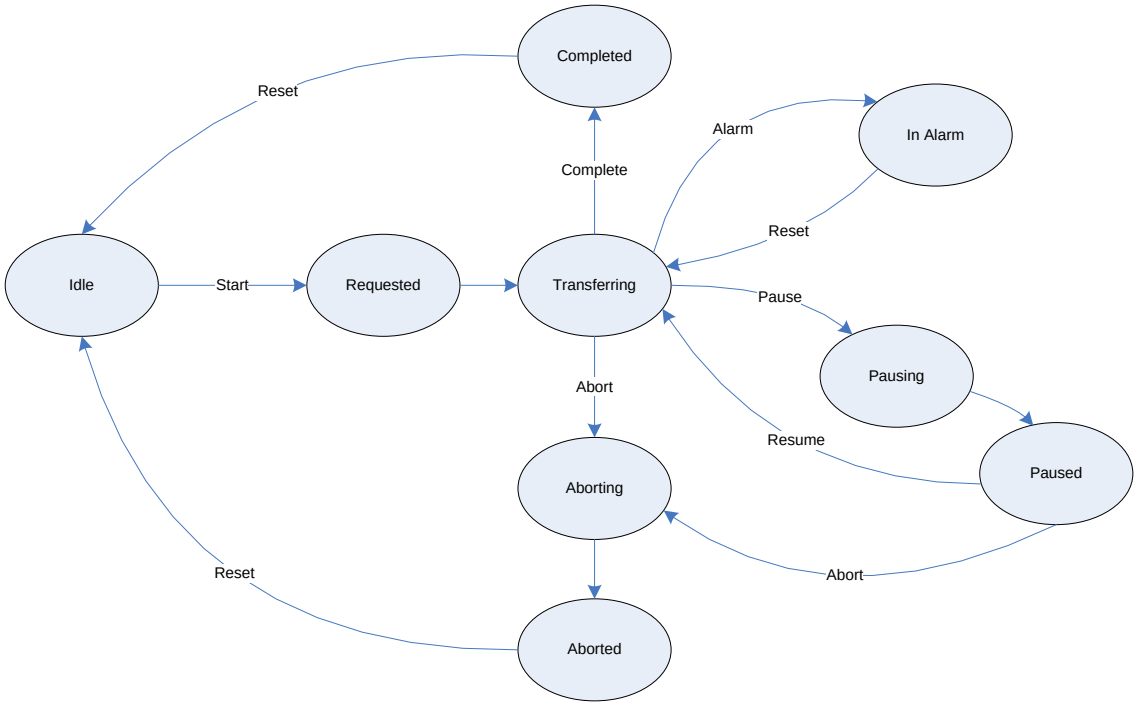


Fig 13-30

Alarms are likewise included the presence of an alarm, the setpoint over or under-run of the add and whether or not the actual weight was acceptable to the operation before proceeding.

REPORTS

The following reports should be considered when weight is added to a mix:

- ActualFeedWeight
- Error
- ExistStatus

Material Transfer Terminology

Some of the terms used in the addition of material using scales or rate feeders include:

Weighing Terminology

Gross
Net
Units
Zero
Center of Zero
Under Zero
Tare
Clear (Tare)
Over Capacity
Motion
Print

Material Transfer Terminology (definitions from SP-88)

"Target (= Setpoint + Spill)
Set Point (= Target – Spill)
Spill (=Target –Set Point) (preact, in-flight, offset, bias)

- Fixed Spill
- Adaptive Spill
- Predictive Adaptive Spill

Fast Feed (coarse feed)
Feed (fine feed, slow feed, dribble feed)

Control Methods

- BASIC Control – single speed transfer control

Dribble Setpoint or Dribble Target or Slow Feed Target(slow-feed setpoint, dribble setpoint)

Flow Rate

Tolerance
Jog (re-dispensing)

- Manual Jog – operator starts jog feed and operator ends jog feed
- Semi-Automatic Jog – operator acks out of tol, starts jog feed and controller ends jog feed
- Automatic Jog – operator acks selects auto accept out tol, controller starts jog feed and ends jog feed
- Setpoint Error, Target Error, Feed Error"

The inclusion of this kind of information regarding the addition of a material to a batch may seem too much to digest but when faced with the task of programming the entire batch system, many times from no prior program, it is important to ask the right questions and be able to make the system perform to specification. Many people are depending on the accuracy of the batching system since the product being made must guarantee accuracy within specifications on the label of the product in the store. For pancake mix, this is very important, especially for those of us who like good pancakes.

For a simple batching system with one main mix tank

Now that we have somewhat of an idea about a batching system structure, we could picture the instructions used by the various PLC's to provide the step move to an active step. The steps are activated sequentially starting with step 1, step 2, …

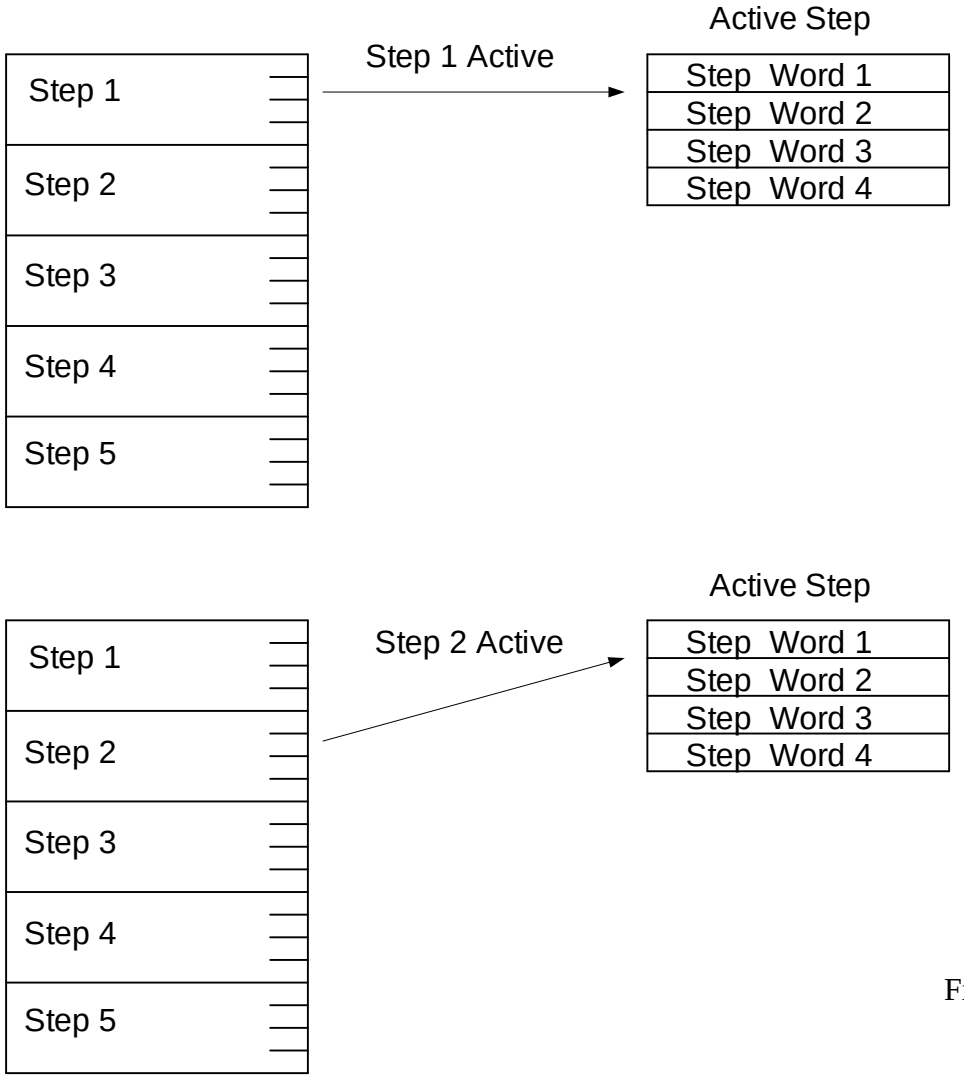


Fig. 13-31

How Would You Program This?

The following picture is of a liquid batching system capable of simultaneously mixing batches in three main mix tanks. In addition, there are six pre-mix tanks – any of which can be mixing a pre-mix for any one of the main mix batches. Each tank measures addition of weight via scales.

Multiple pre-mixes may be required for each main mix and they mix concurrently in various pre-mix tanks. They then add to the main mix at a prescribed time.

Begin to plan the programming of how a recipe is to be constructed to handle such a system as this.

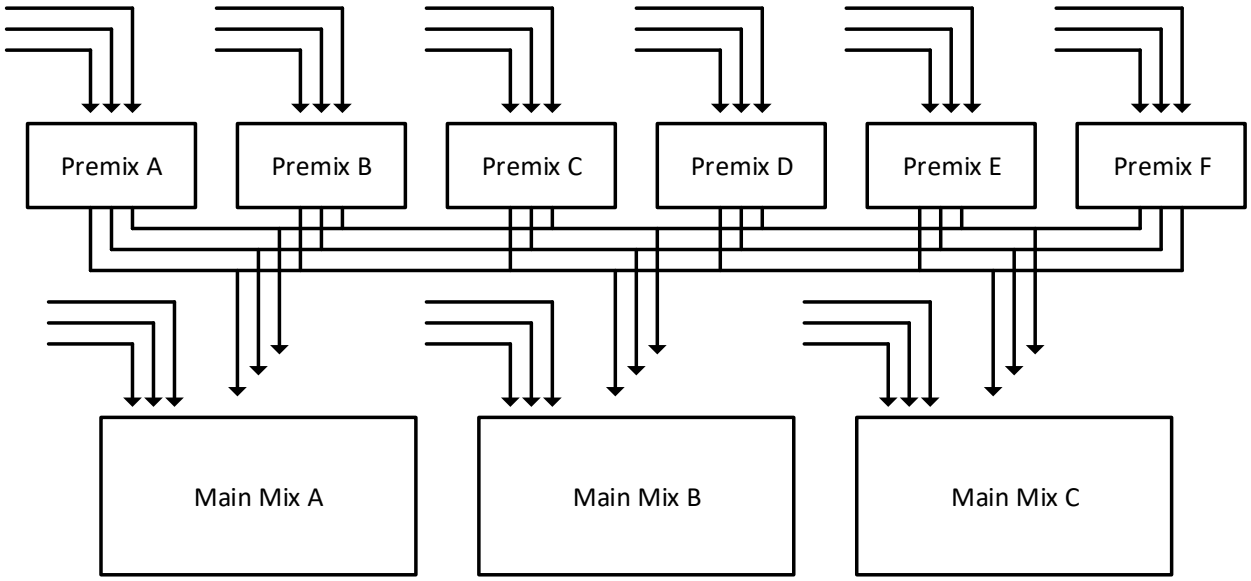


Fig. 13-32 Liquid Batch System

This is left as a problem for discussion.

In RealPars Video Series, the author discusses 'What is a Functional Design Specification (FDS)'

https://www.youtube.com/watch?app=desktop&v=hV9--cPj_zA&list=PLln3BHg93SQ9SEN8jXvhycxAeFcmkjTNs&index=3

This video explains more in detail what a Functional Design Specification such as SP88 may contain and how it can be useful in an Engineering Project.

Other Types of Recipes

There are other recipe types not necessarily using scales and mx tanks. The following two examples give some insight into the diversity of types of recipes.

Valve Nest:

This example requires a number of paths through a number of valves in a valve nest. Each valve has two limit switches, one closed with a closed valve and one closed with an open valve. Each valve in the transfer path must be set appropriately and its limit switches closed or opened in a recipe in order to allow a transfer to occur. With a number of different transfer paths, these paths may be considered recipes.

Steel Heat/Soak Cycle:

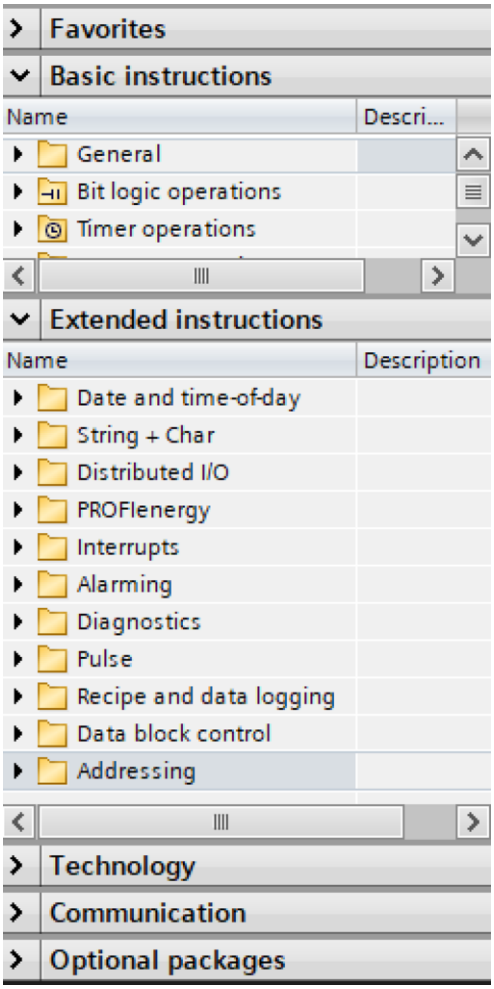
The heating of steel in the hardening process requires a recipe of time and heat setpoint. The hardness of a piece of steel is determined with a recipe.

Several pieces can be inserted in the oven at the same time if they all support the same setpoints and soak times. If various types are to be introduced at the same time, then a check must be performed to determine whether the different recipes can be used at the same time.

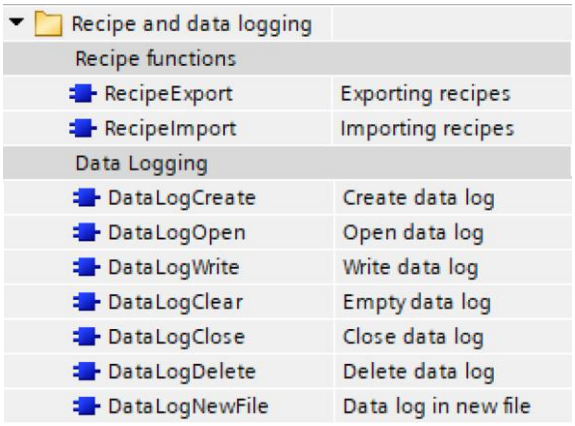
This is similar to baking a cake. If one is determined to use the oven to bake a large number of cakes, if one cake is ready to take out, then another cake can be inserted in the oven even if there is a longer baking cake still in the oven.

The rules in the steel example are used to determine whether multiple pieces can be inserted at the same time. These rules can be intricate.

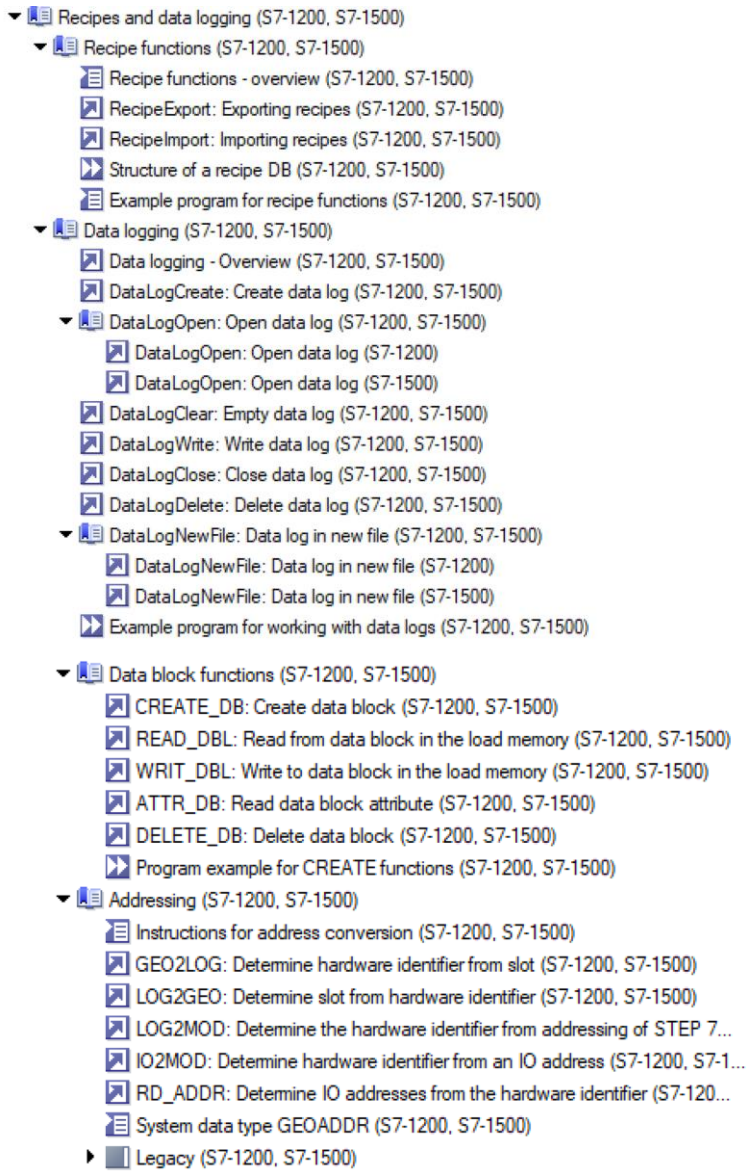
To access the instructions for Recipe and Data logging for Siemens, the following basic instructions are selected:



From the instruction bar at right on the TIA portal one can find in Extended Instructions the entry for Recipe and data logging, the focus of this chapter. Expansion of the entry give the instructions below:



The instructions for Recipe and data logging are expanded here:

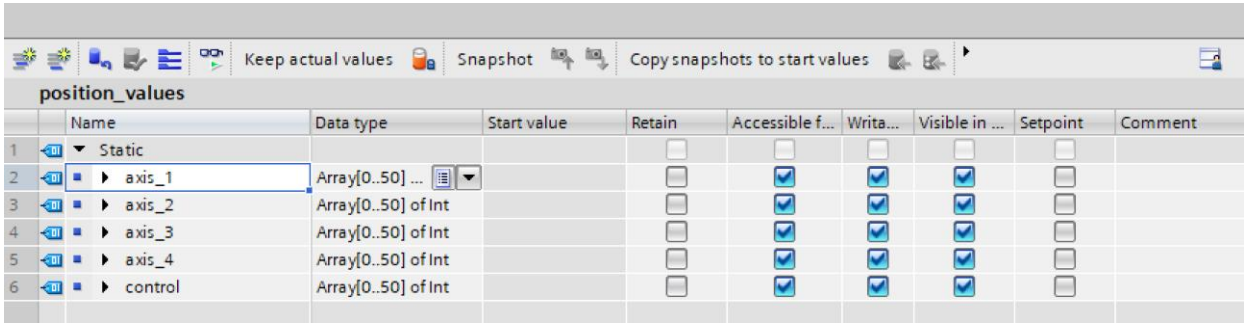


Using the help box for Recipes gives the following:

This list gives a more extensive listing of what can be learned under the Recipes and data logging operations.

What do all these do? I don't know. I would have to check each out individually using the help menu. Someday...

How is data saved for future use if the PLC shuts down and has to re-start? The following shows the method of saving data for this purpose:



How to save data once its been entered into an array:

In online mode, toggle 'Shapshot' above. Then copy the shapshots to start values prior to uploading and saving. This saves your data recently updated. A nice technique to know.

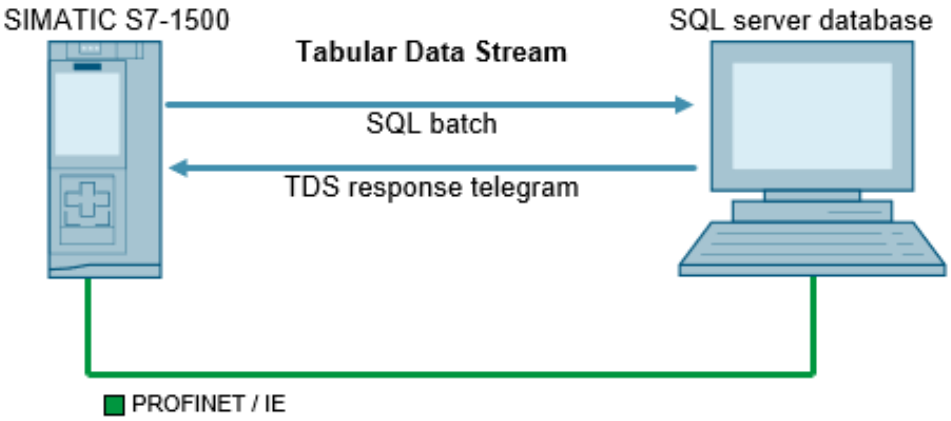
How do we send data to a computer? A number of methods exist for this purpose but the one most recommended is the following:

Connecting a S7-1200 PLC / S7-1500 PLC to a SQL Database

- Entry
- Associated product(s)

The Tabular Data Stream protocol (TDS) gives you the ability to establish a direct connection with a Microsoft SQL server. Using TDS, you can log in to an SQL server database and transmit SQL instructions. In this way it is possible to read data from the database, or send them to the database for storage.

Based on the "Open User Communication blocks" (TCON, TSEND, TRCV and TDISCON), S7-1500 PLCs and S7-1200 PLCs can emulate the TDS protocol and establish a connection to a Microsoft SQL server. By using the SQL statements "insert into", "update" and "select" you can store data in the database, update and read data from the database.



Connecting an
S7-1500 / S7-1200 to
a SQL database
TIA Portal V17 / S7-1500 / S7-1200 Microsoft SQL /
Tabular Data Stream (SQL)
https://support.industry.siemens.com/cs/ww/en/view/109779336

When beginning a batch process, also consider the vendor. Allen-Bradley offers a toolkit, the Batch Application Toolkit. Likewise, Siemens offers a Batch Process Control System:

SIMATIC

Process control system PCS 7
SIMATIC BATCH V9.0 Getting Started

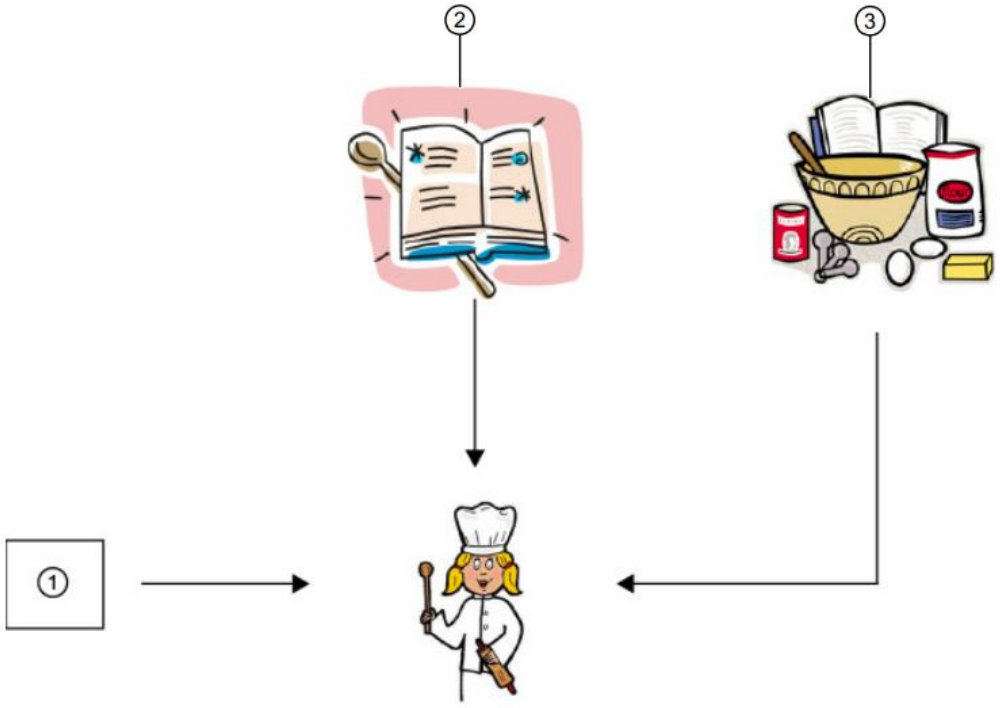
Getting Started

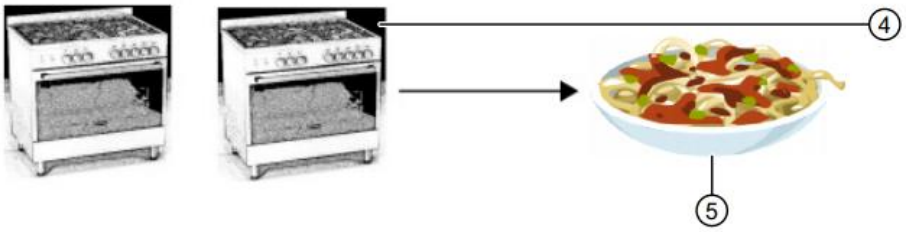
The Siemens approach to batching resembles a kitchen and the baking of a cake or other product, in this case, spaghetti:

“

Introduction to batch processes

3.6 The Chef - Working Environment and Working Procedures





(1) Order

(2) Recipe

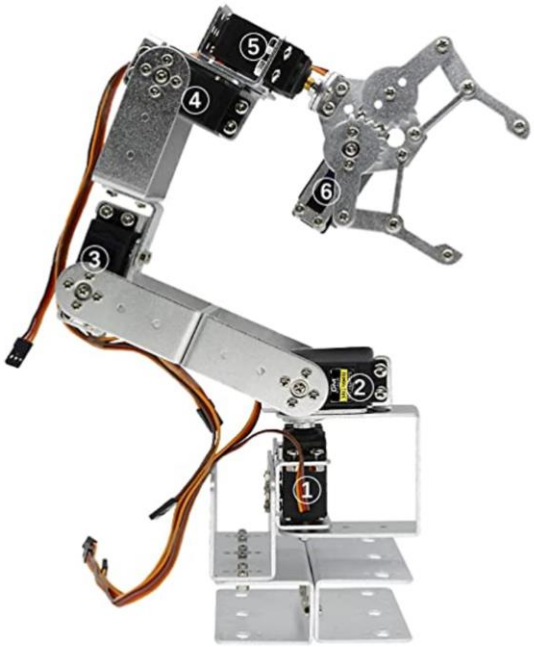
(3) Ingredients for cooking pasta

(4) Cooking pasta

(5) Spaghetti Bolognese served

”

Chapter 31 of the Lab Text Book describes a robotic arm that was programmed to pick up and place a checker in a box. The robotic arm consists of 4 axes and a suction attachment (not shown). The entire program was written in a week and used for a capstone group's display. The program was written entirely with array storage of the positions of the robot arm. The program is given in its entirety in Ch. 31 of that text.



Due to the weight of the end effector, the 5th and 6th servos are removed. The end of arm tool is replaced with the suction device shown below. This is attached to servo 4 shown at right.

Another use of matrices is for the solution of the Rubiks Cube program. This program can be solved through the use of various algorithms stored in matrices. This approach promises to minimize computer space and allow this program to be stored in the PLC.

Summary

This chapter explains a number of data handling instructions and provides applications for their use in factory automation. Other interesting instructions in data handling include the bit-shift instructions. These instructions are used for shift register part tracking. Instructions used in data manipulation were shown with examples included for each type of instruction. An example of a batching application demonstrated the handling of large amounts of data. The chapter concluded with a discussion of For-Next loops and their inclusion in possible data-handling operations.

Included at the end of the chapter are two important labs using file manipulation to control simple games. The games Simon Says and Whack-a-Mole are used to provide an experience in the use of table-to-working register functions. In the MicroLogix 1000 processor, the only available mode to consider table-to-register moves is the indexed mode. Whether using the MicroLogix 1000 or other processor, these labs provide valuable experience programming simple batching applications such as these. The example program was written for the MicroLogix 1000 processor.

You may be curious how two games, Simon Says and Whack-a-Mole have anything to do with batch applications. If run as sequences of numbers, the same operations used in these games are used for batch applications. The games are to be programmed in this way. They are not to be programmed using random number generation but rather as storage of information in tables and then retrieving the information as the game is being played. The Whack-a-Mole game has the added similarity to a batching program in that one of the options asks for a report back as to how soon after the mole popped up that the button was pushed. This is similar to the report of actual weight added for the batch report. Hopefully these labs will give some insight into actual batch programs.

Lab 13.1 Simon Says

Use four illuminated push buttons to build the game Simon Says.
Use a fifth button to start the game.
Use a sixth light to signal the game is done good.
Use a seventh light to signal the game is done bad.

Simon Says is a sequential game that plays a four-note tune. This lab has no sound so lights will have to do. After the start button is pushed, one light at a time is turned on. The player must mimic or push the button attached to the light. First one light is lit, then two, then three, etc., until a final number is attained. Assume 15 is the final number of steps in the game. The steps are repeated each time the game is played. The values are to be **stored in a table and re-used.**

Lab 13.1A Add a three-position switch to allow for easy, middle, hard ranges varying the number of steps from 10 to 15 to 20.

Lab 13.1B Automatically rotate through 3 different sets of conditions for each range – easy, middle, and hard.

Lab 13.1C Vary the speed that the lights are display as the player gets closer to the last step.

Lab 13.1D Create a teach mode using a separate button to teach the game a sequence of steps.

Use the program starting on the next page as a suggested beginning point for your logic:

Fig. 13-33 Simon Says Game

The program below is programmed for three, not four buttons and is programmed using indexed addressing from the SLC architecture PLCs from A-B.

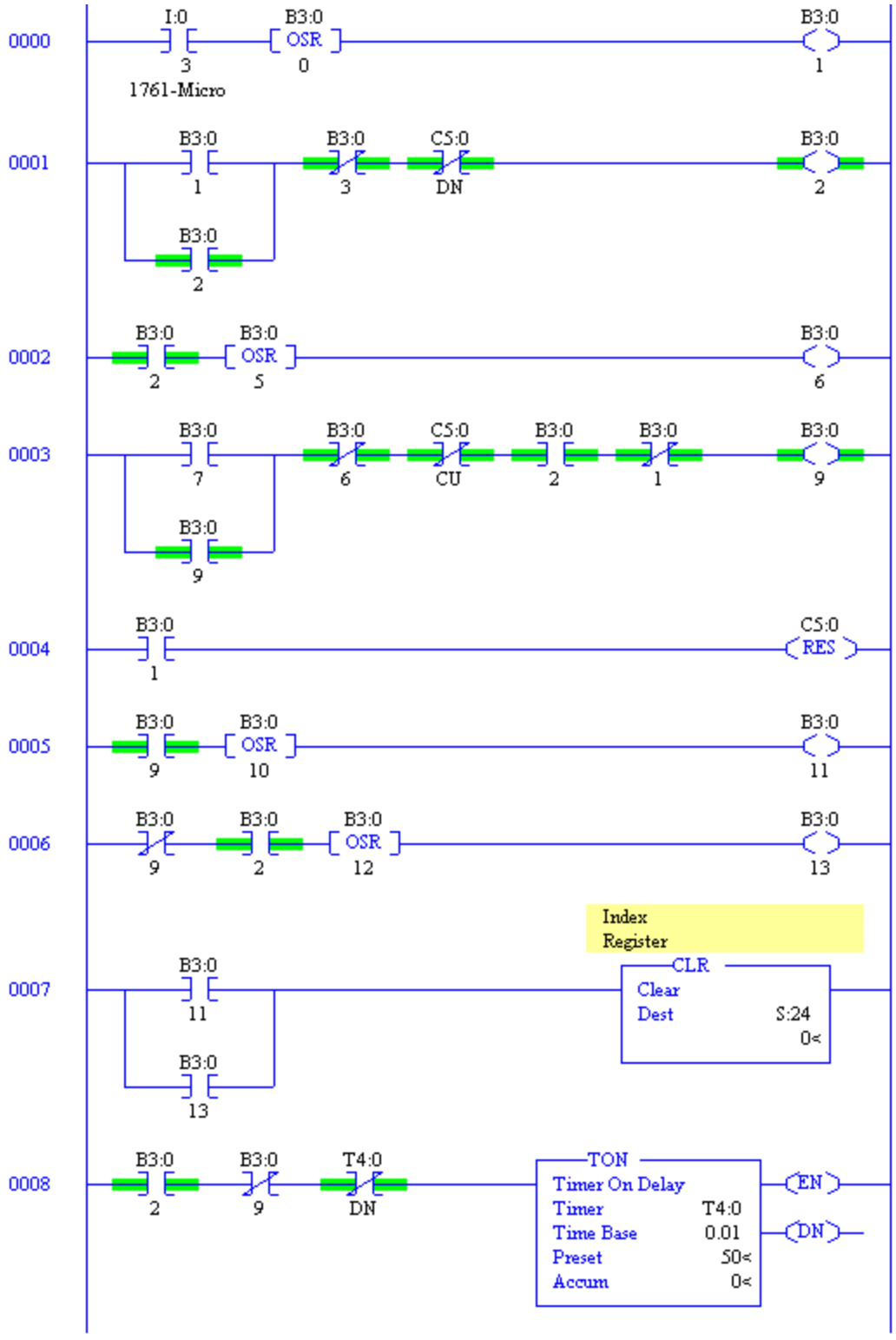


Fig. 13-34a Simon Says Game Program in SLC

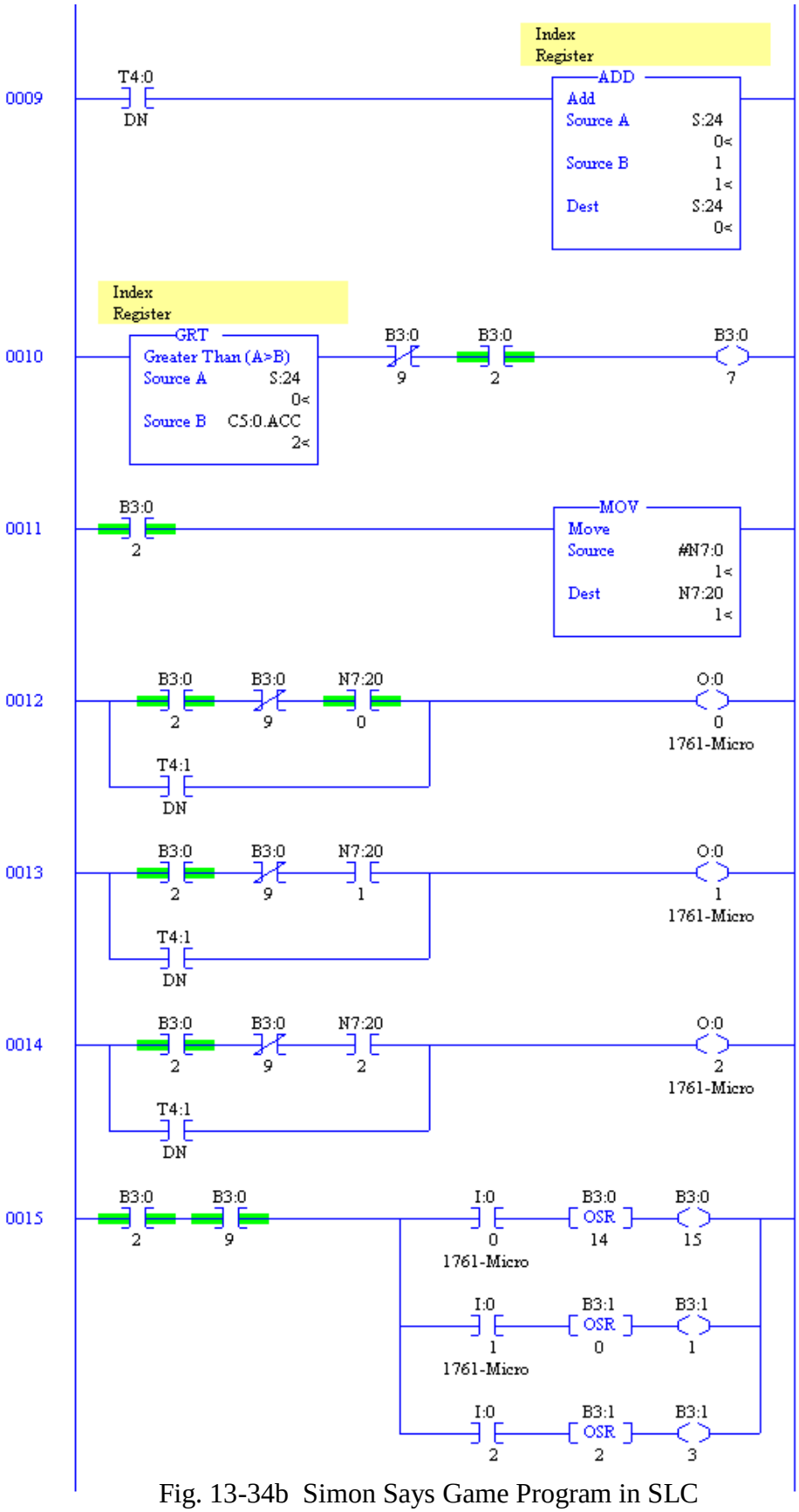


Fig. 13-34b Simon Says Game Program in SLC

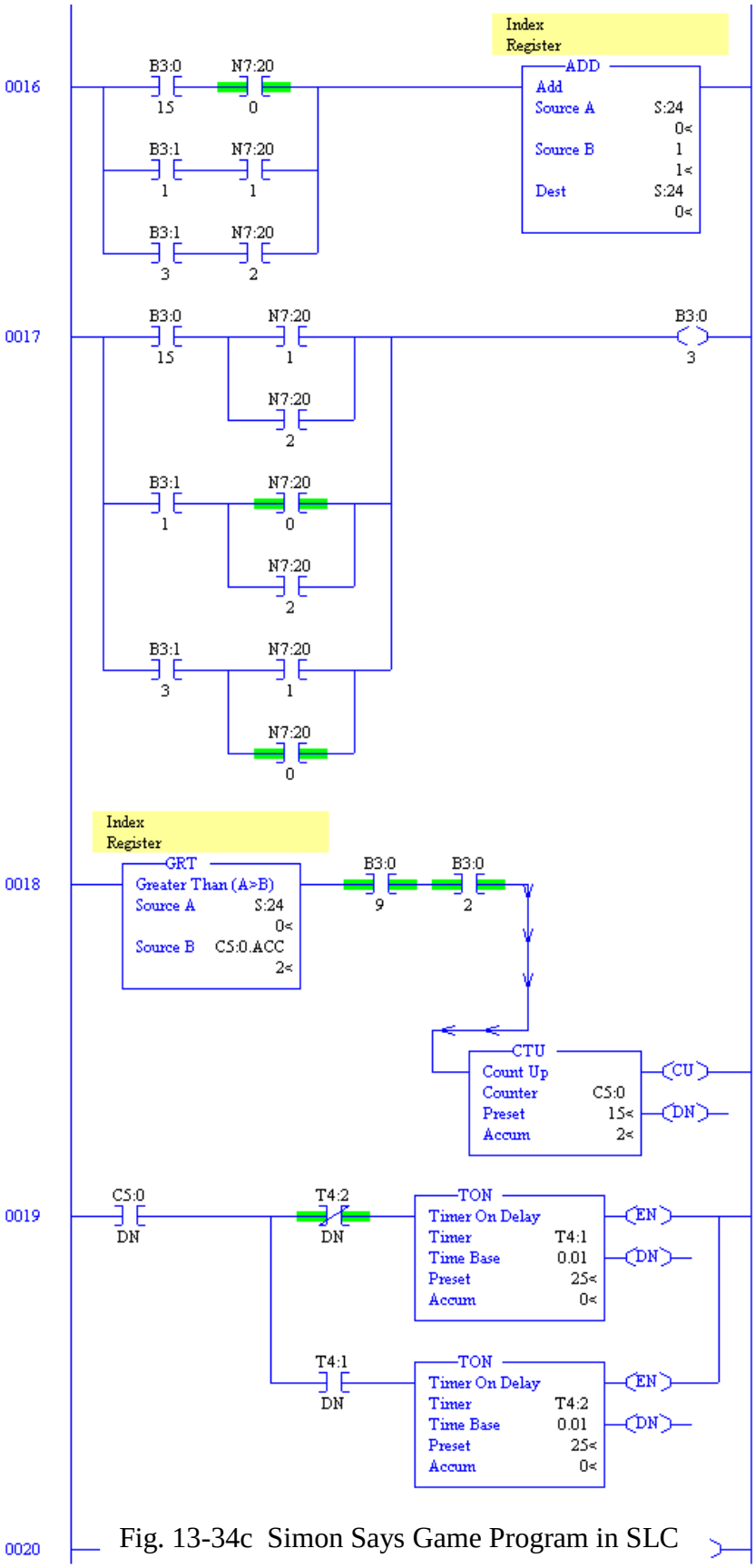


Fig. 13-34c Simon Says Game Program in SLC

In addition to the program, a table must be filled in starting at N7:0 in the N7 file.

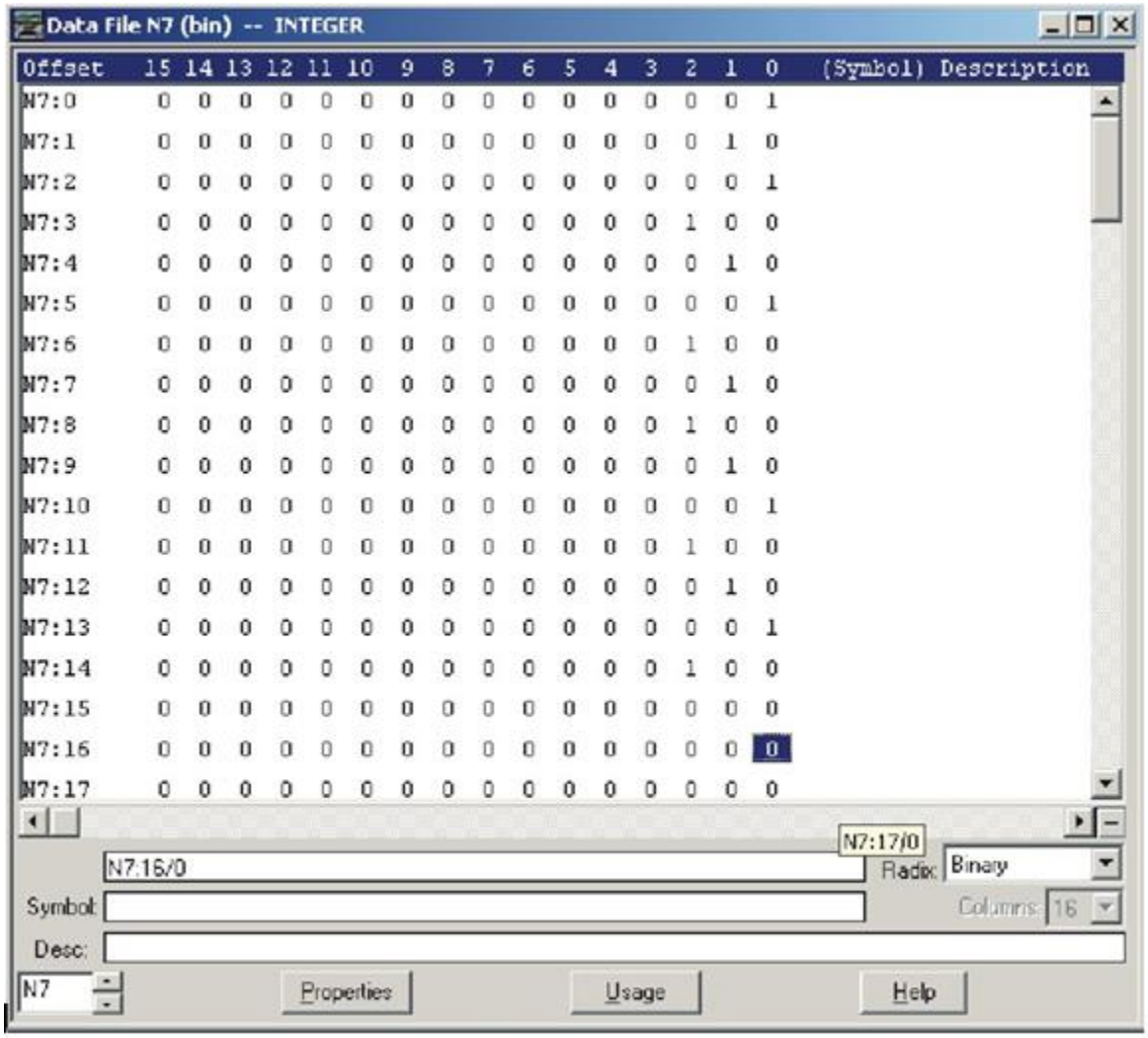
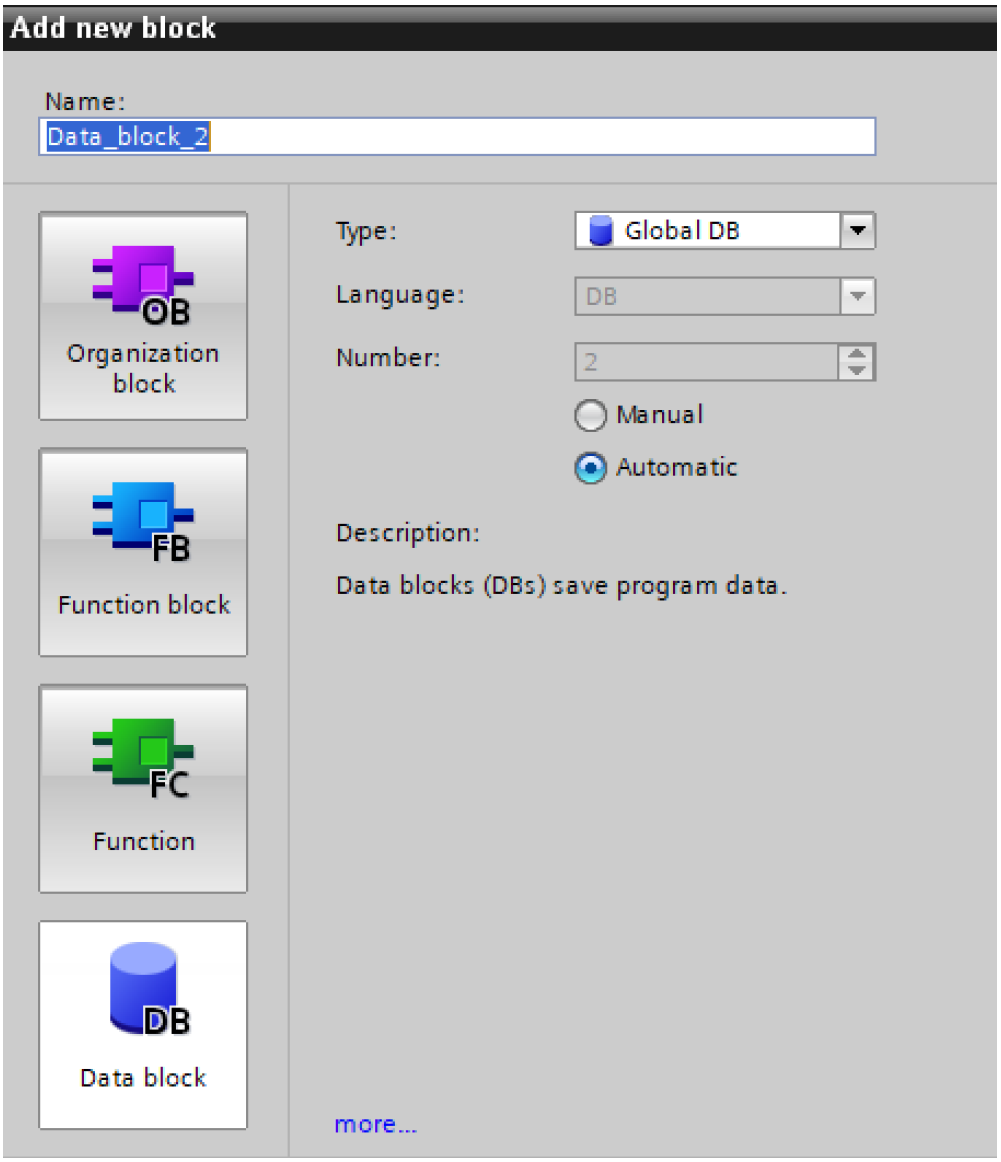


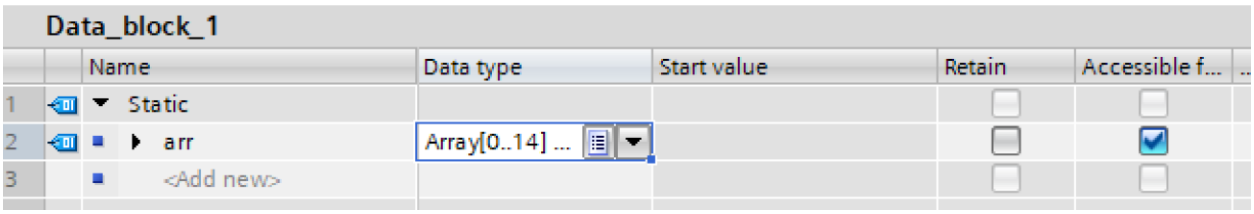
Fig. 13-35 Simon Says Game Program in SLC

The file must also be inserted for the game to work correctly. Set the Radix to *Binary* and set each word from N7:0 to N7: with either a 1 in the bit 0, 1, 2 or 3 position. The game will be played using these entries.

To implement the array and start the Simon program, the following is to be implemented. First, to build an Array with the Siemens processor, add a new Data Block.



In the Data Block, add an array. We call it 'arr' and give it 15 locations. For type, we can chose either 'Byte', 'Int' or 'Dint'. We will choose 'Byte'.



Finally, expand the table and in the ‘Start value’ column, insert a series of integers.

Data_block_1

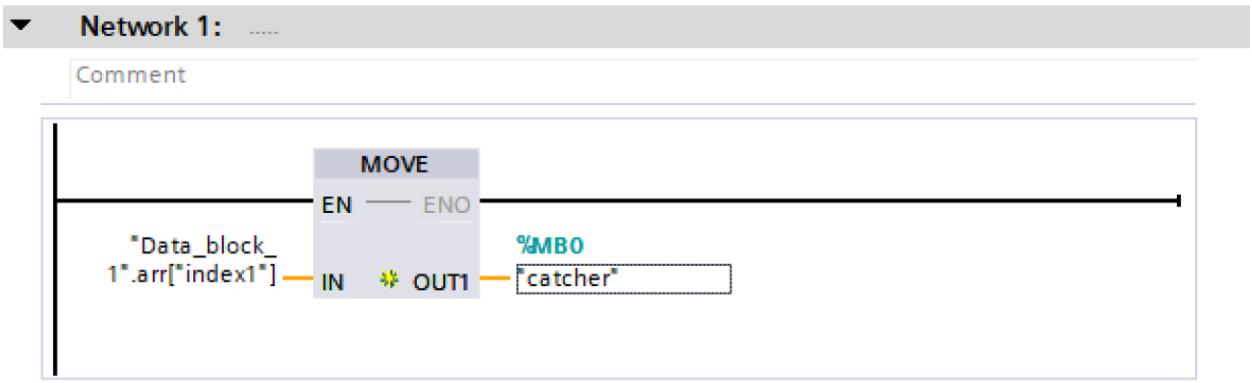
	Name	Data type	Start value	Retain	Accessible f...
1	Static			☐	☐
2	arr	Array[0..14] ...		☐	☑
3	arr[0]	Byte	16#0	☐	☑
4	arr[1]	Byte	16#0	☐	☑
5	arr[2]	Byte	16#0	☐	☑
6	arr[3]	Byte	16#0	☐	☑
7	arr[4]	Byte	16#0	☐	☑
8	arr[5]	Byte	16#0	☐	☑
9	arr[6]	Byte	16#0	☐	☑
10	arr[7]	Byte	16#0	☐	☑
11	arr[8]	Byte	16#0	☐	☑
12	arr[9]	Byte	16#0	☐	☑
13	arr[10]	Byte	16#0	☐	☑
14	arr[11]	Byte	16#0	☐	☑
15	arr[12]	Byte	16#0	☐	☑
16	arr[13]	Byte	16#0	☐	☑
17	arr[14]	Byte	16#0	☐	☑

These integers determine the game pattern. The same number may not be repeated but any series is acceptable. The game will play the same series each time the game is played.

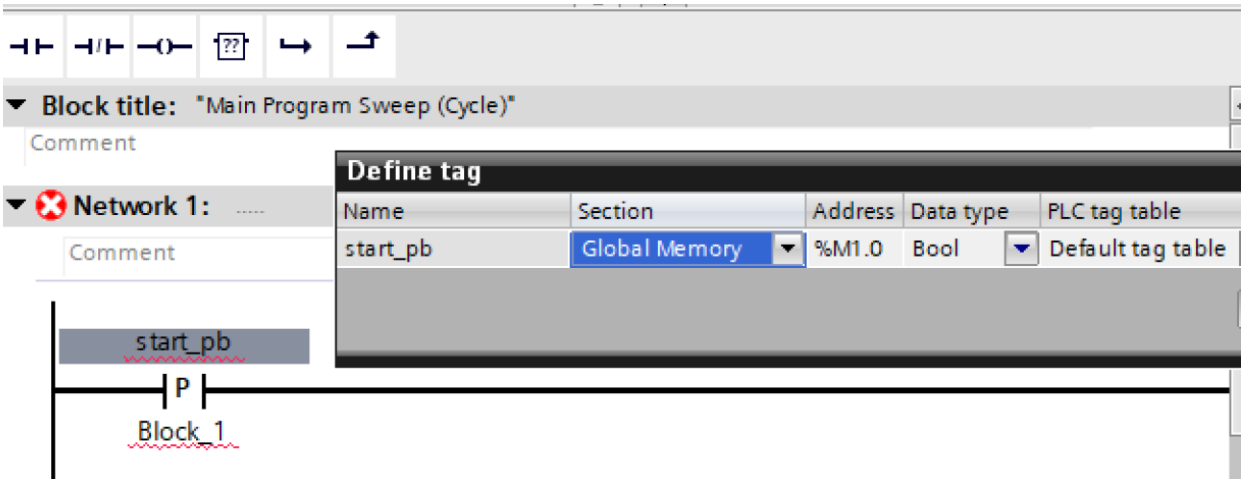
Data_block_1

	Name	Data type	Start value	Retain	Accessible f...
1	Static			☐	☐
2	arr	Array[0..14] of Byte		☐	☑
3	arr[0]	Byte	1	☐	☑
4	arr[1]	Byte	2	☐	☑
5	arr[2]	Byte	4	☐	☑
6	arr[3]	Byte	8	☐	☑
7	arr[4]	Byte	1	☐	☑
8	arr[5]	Byte	2	☐	☑
9	arr[6]	Byte	4	☐	☑
10	arr[7]	Byte	8	☐	☑
11	arr[8]	Byte	1	☐	☑
12	arr[9]	Byte	2	☐	☑
13	arr[10]	Byte	4	☐	☑
14	arr[11]	Byte	8	☐	☑
15	arr[12]	Byte	1	☐	☑
16	arr[13]	Byte	2	☐	☑
17	arr[14]	Byte	4	☐	☑

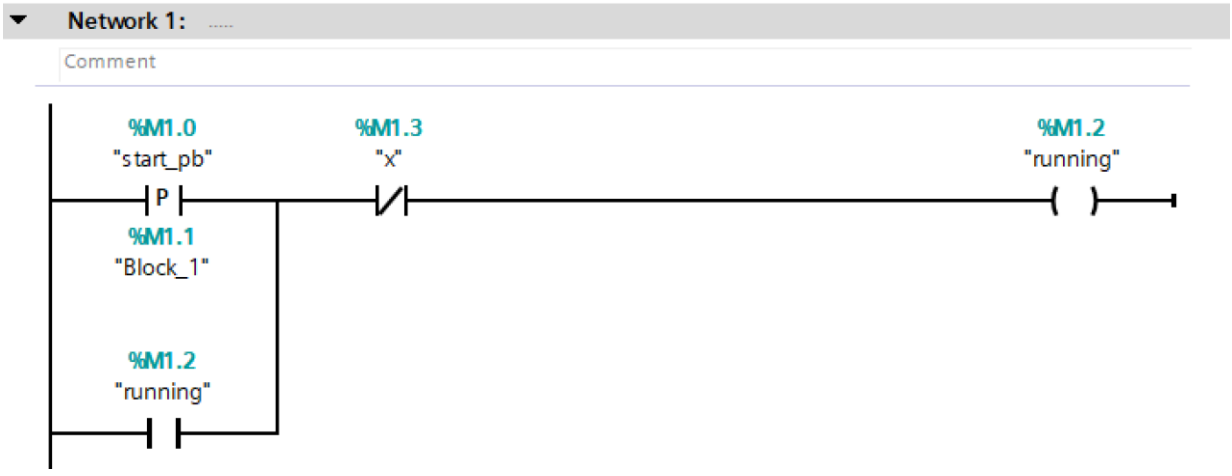
The following shows the construct of the ‘Move’ instruction using a file to word format:



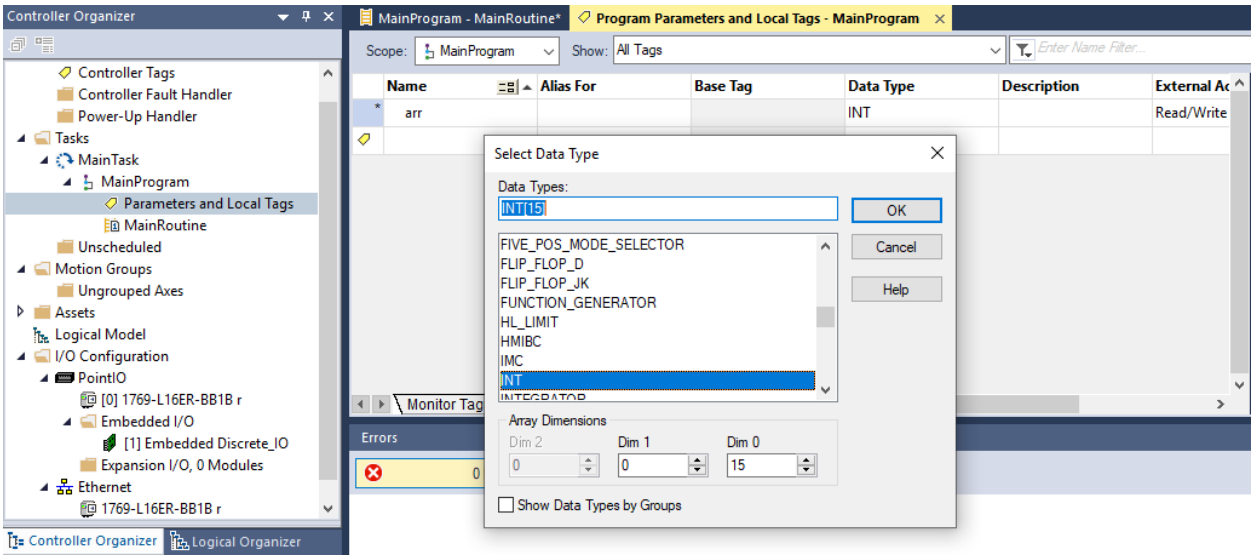
The following shows the construction of a ‘one shot’ or ‘edge trigger’ instruction.



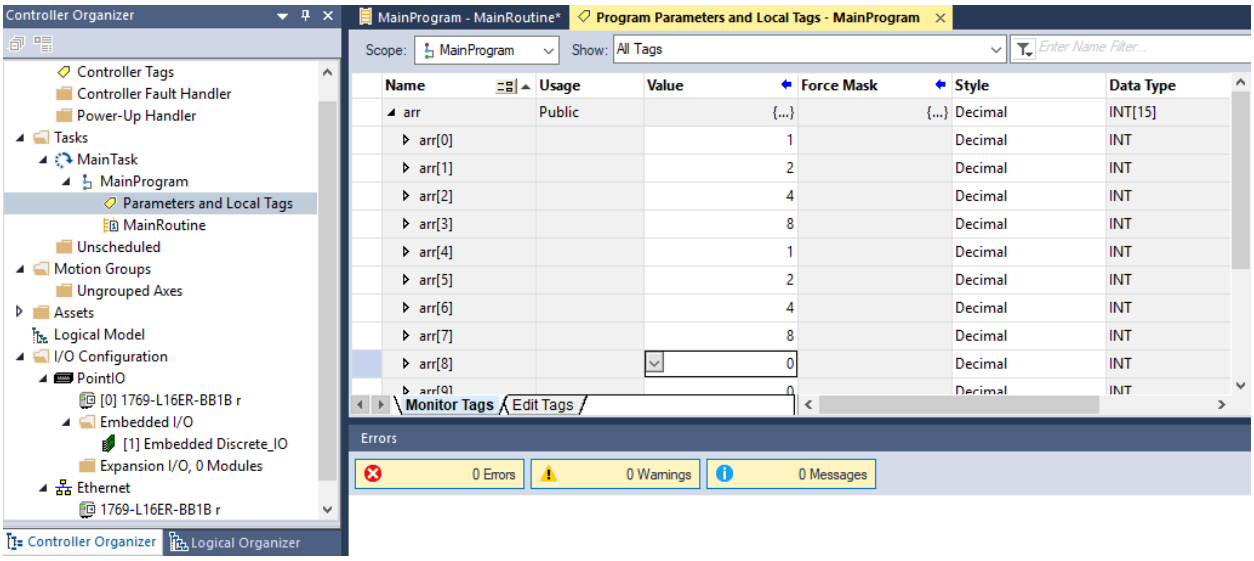
The first rung of the program shows the ‘running’ of the game being set. Events that terminate the game are ‘x’ instructions.



With Allen-Bradley, the instructions are similar to Siemens for building an array. We do not need to include a Data Block, however.



Populating the table is shown here:



The ‘Move’ block:

And the Game Start Logic:

Lab 13.2 Whack-a-Mole

Design a Whack-a-Mole game using only 4 lights. The game is to react to the light by pushing that particular button before the light turns off. Construct the game so that the time between lights is pseudo-random (you pick various numbers) and the next light to turn on is pseudo-random (again, you pick). Count the steps (light turn-ons) and stop at 30. If the person playing the game is successful 10 of the 30 times, blink all the lights a number of times. The steps are repeated each time the game is played. The values for buttons and time delays are to be **stored in a table and re-used.**

Use a button not part of the game to start the game.

The layout is as follows:

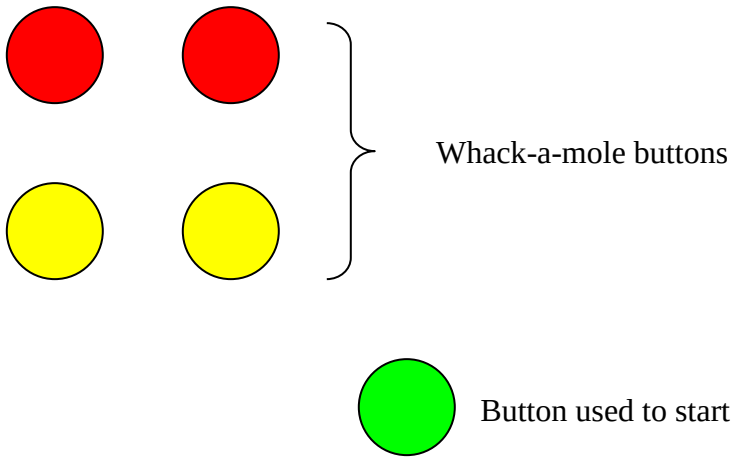


Fig. 13-36

Lab 13.2A Automatically rotate through 3 different sets of data tables.

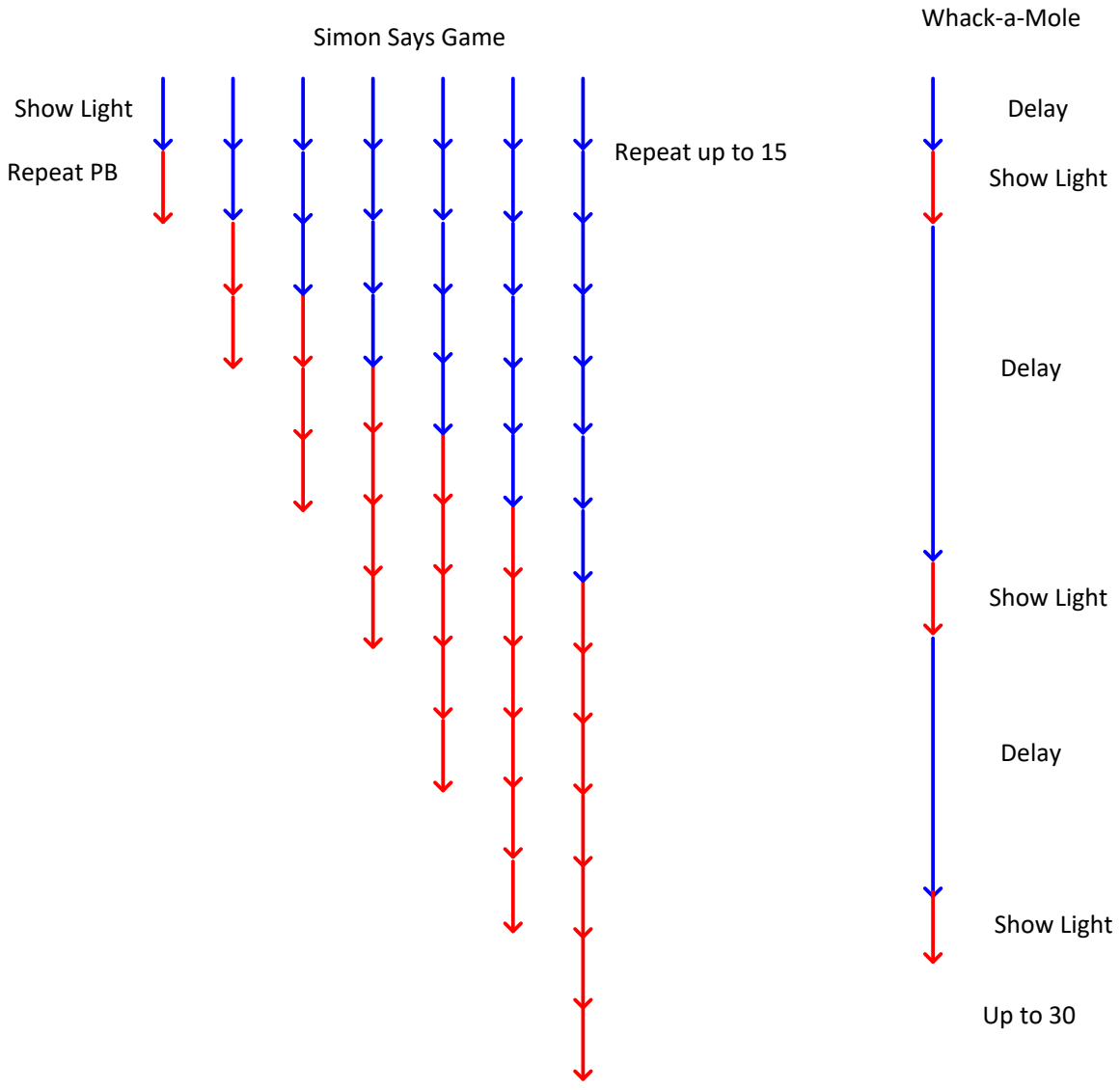
Lab 13.2B Create a teach mode using a separate button to teach the game a sequence of steps which can then be played.

Lab 13.2C Add a table of results including whether the player hit the light while the light was

on and how long the response was delayed from when the light first turned on. Results for each hit are to be saved sequentially in the table.

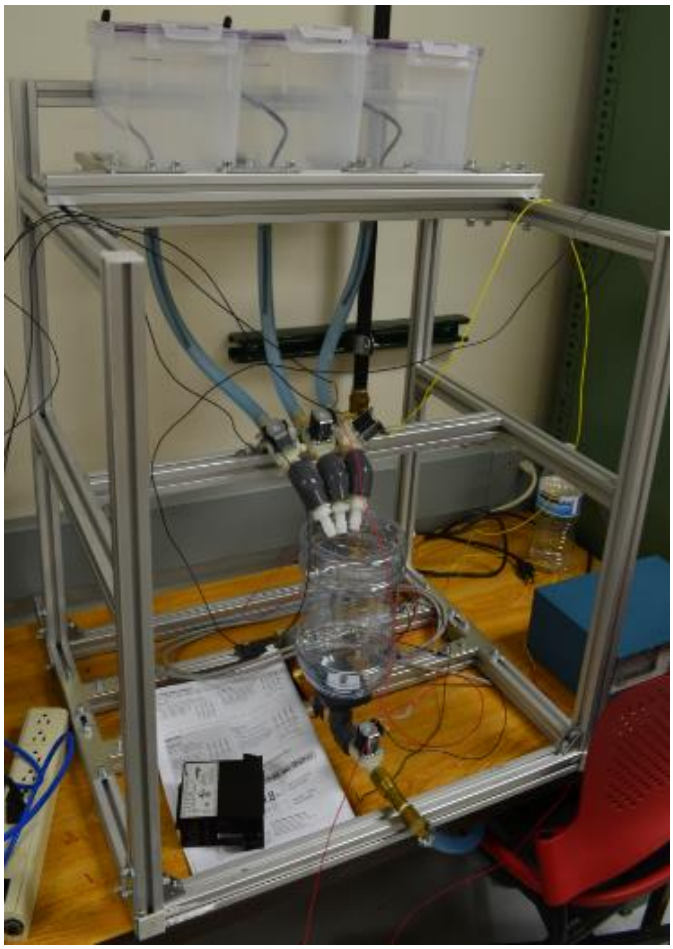
Lab 13.2C1 Implement 13.2C above with a UDT output table. Save sequential hit data for later display or analysis.

The structure of the two games is shown below:



Lab 13.3 Liquid Batch Lab

The lab was first built by a capstone group. Since then, it has been used by a two other capstone groups as well. It has been in storage at other times and is rather large.



Three Ingredient Batch System

Purpose of the Lab – The purpose of this lab is to introduce students to batching system design. Earlier lab exercises include games that simulate the same principles of design.

Scale Weighing

Scale Weighing Systems are an important part of a batch system. Siemens provides a load cell interface system for weighing applications called the SIWAREX WP231.

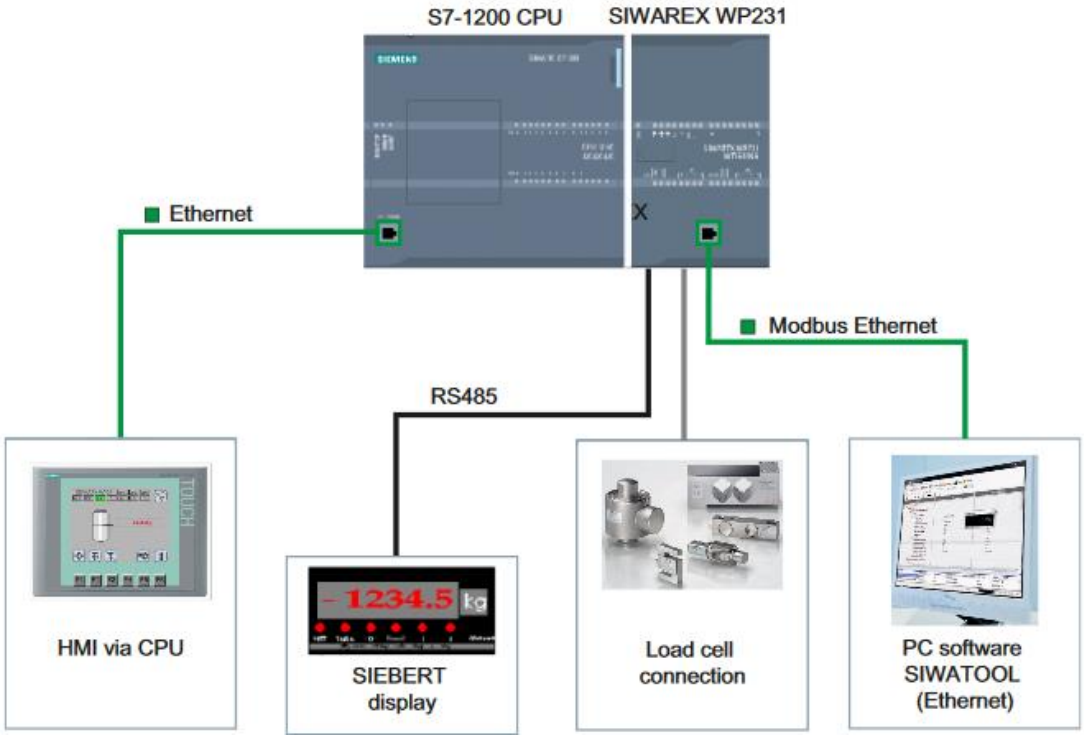
The electronic weighing system has the following characteristics as listed by Siemens:
“

- Uniform design technology and consistent communication in SIMATIC S7-1200
- Parameter assignment by means of HMI panel or PC
- Uniform configuration option in the SIMATIC TIA Portal

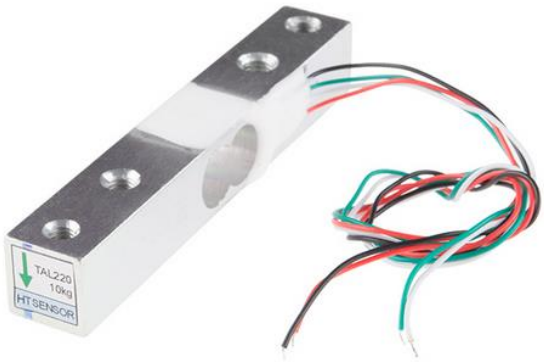
● Measuring of weight with a resolution of up to 4 million divisions
● High accuracy, 3000 d, legal for trade according to OIML R76
● Legal-for-trade display with SIMATIC operator panel or PC
● High measuring rate of 100/120 Hz (effective interference frequency suppression)
● Limit monitoring
● Flexible adaptation to varying requirements
● Easy calibration of the scales using the SIWATOOL program
● Automatic calibration is possible without the need for calibration weights
● Module replacement is possible without recalibrating the scales
● Use in Ex Zone 2 / ATEX approval
● Intrinsically safe load cell supply for Ex Zone 1 (SIWAREX IS option)
● Diagnostics functions"

SIWATOOL overview

SIWATOOL does not only offer support when you set the scale but also when you analyze the diagnostic buffer that can be saved after being read out of the module together with the parameters. The display of the current scale status can be configured.



A load cell is pictured in the figure below:



Also, the load cell can be terminated in a converter box similar to the Red Lion Strain/Load Cell Panel Meter pictured below. Since it can be used to interface to a PLC using an optional analog output, the Red Lion is used simply to pass through a signal from the scale to the PLC after linearization has occurred with the scale signal.

Red Lion PAXS0000 Strain/Load Cell Panel Meter

Red Lion – Mfr # PAXS0000 – Item # EW-68486-20

No Reviews Write the First Review

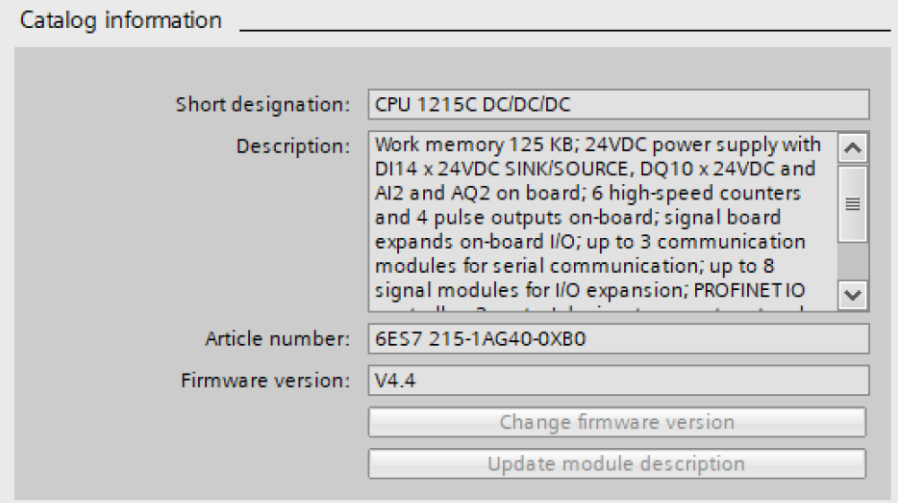
Upgradeable for customized solutions – Choose from temperature, process, voltage, and current inputs

- Process, Voltage, Current, Temperature, And Strain Gauge Inputs
- 6-Digit 0.56" Red Sunlight Readable Display
- Variable-intensity display for viewing in all lighting conditions
- 16 Point Scaling For Non-Linear Processprogrammable Function Keys/User Inputs
- 9 Digit Totalizer (Integrator) With Batching

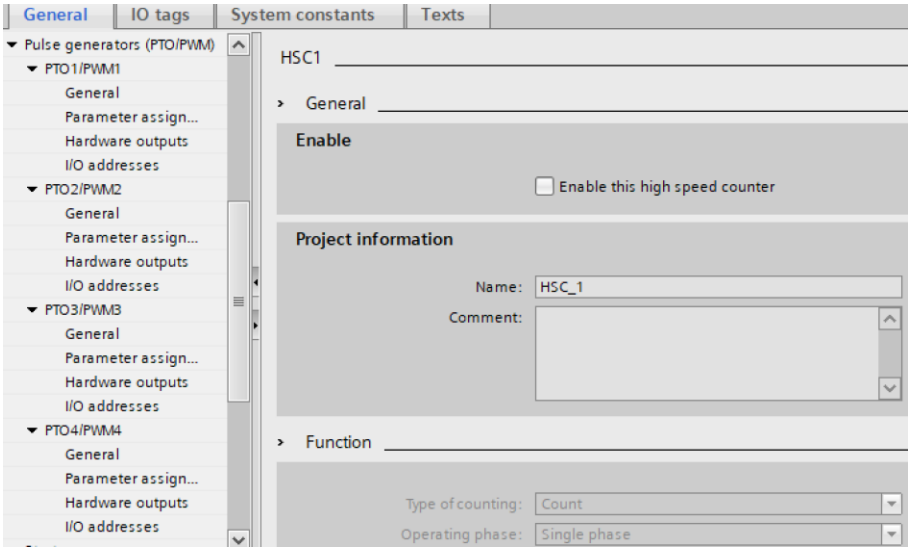
Lab 13.4 Simple Robotic Arm

Using the program from Ch. 8 for the servo arm, design a robotic arm that has four degrees of motion. The arm is to move in a set number of motions with a time delay between motions so as to pick up a coin and lay it down in another spot.

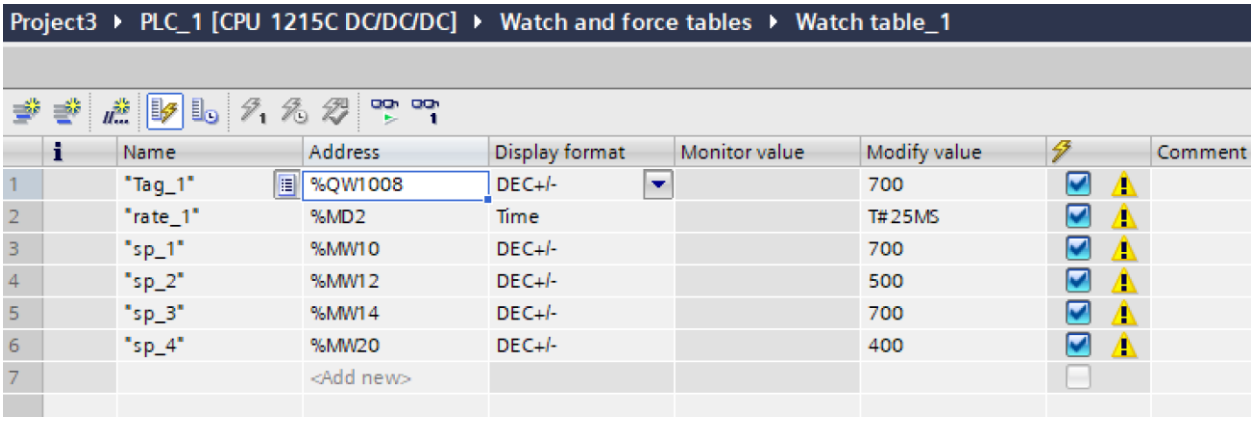
The four axes control the four servo motors in the robot with PWM output control. The processor is the S7-1215 DCDCDC model. The axes are set up similar to the code below. The program can be written to move between various settings for the sp_x variables for the four axes.



The PWM outputs are controlled through configuring PWM1, PWM2, PWM3 and PWM4 below:

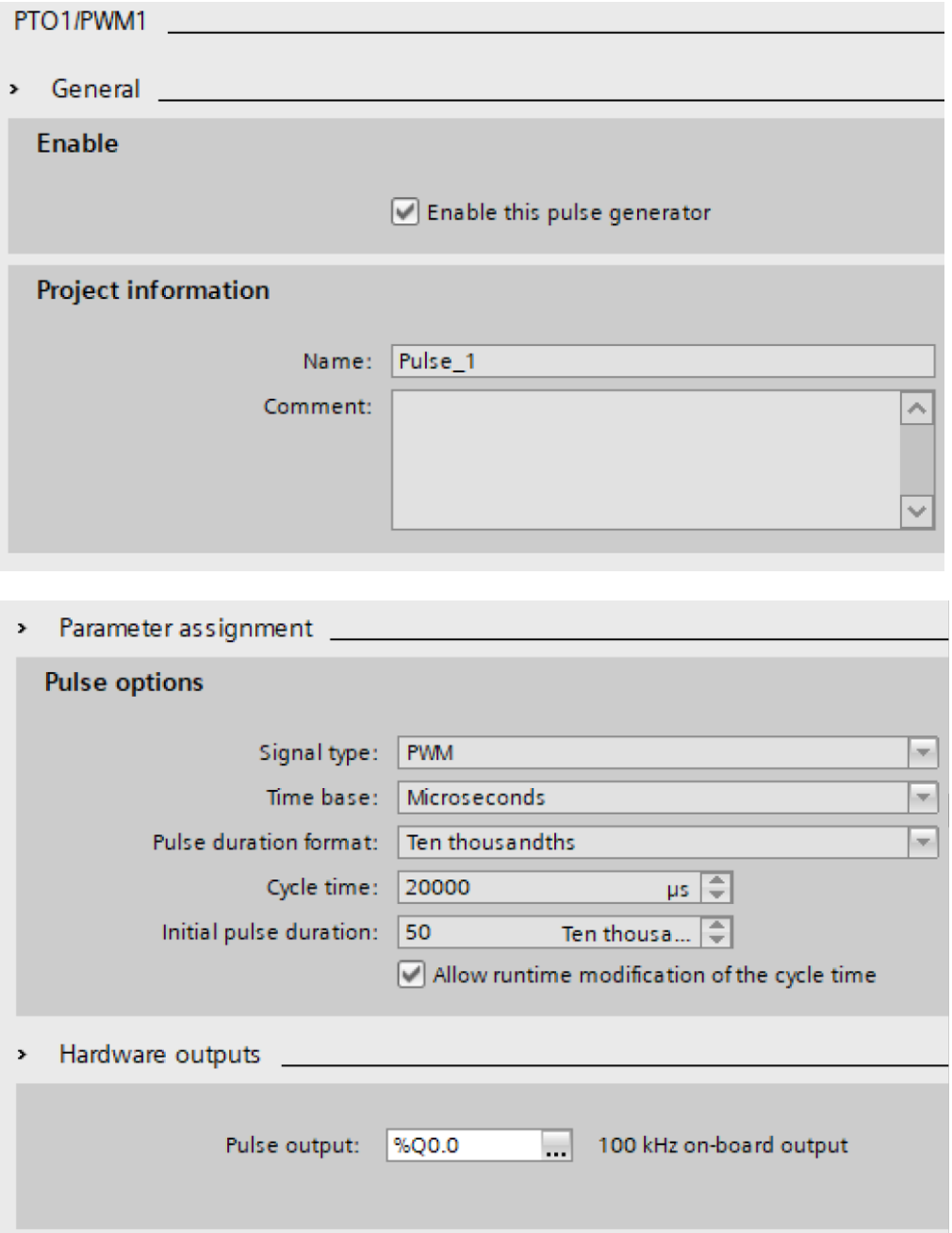


Variables that control movement are sp_1, 2, 3, and 4 (setpoints). The program may be tested by setting ena to 1 and test_1, 2, 3, and 4 to 1. The setpoint variables are the position points.



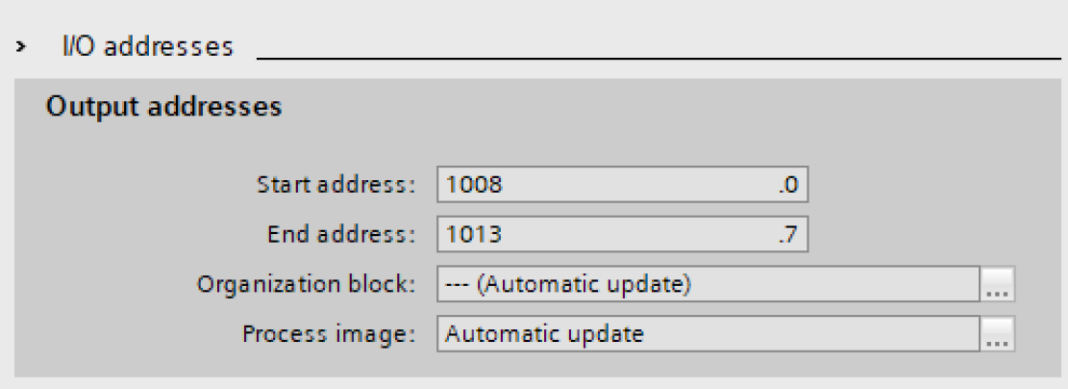
	Name	Address	Display format	Monitor value	Modify value	Comment
1	"Tag_1"	%QW1008	DEC+/-		700	
2	"rate_1"	%MD2	Time		T#25MS	
3	"sp_1"	%MW10	DEC+/-		700	
4	"sp_2"	%MW12	DEC+/-		500	
5	"sp_3"	%MW14	DEC+/-		700	
6	"sp_4"	%MW20	DEC+/-		400	
7		<Add new>				

Configuration of the PWM axes follow. They all resemble PWM1 shown below:



The other outputs are wired as follows:

Axis 2 – Q0.1
Axis 3 – Q0.3
Axis 4 – Q0.4



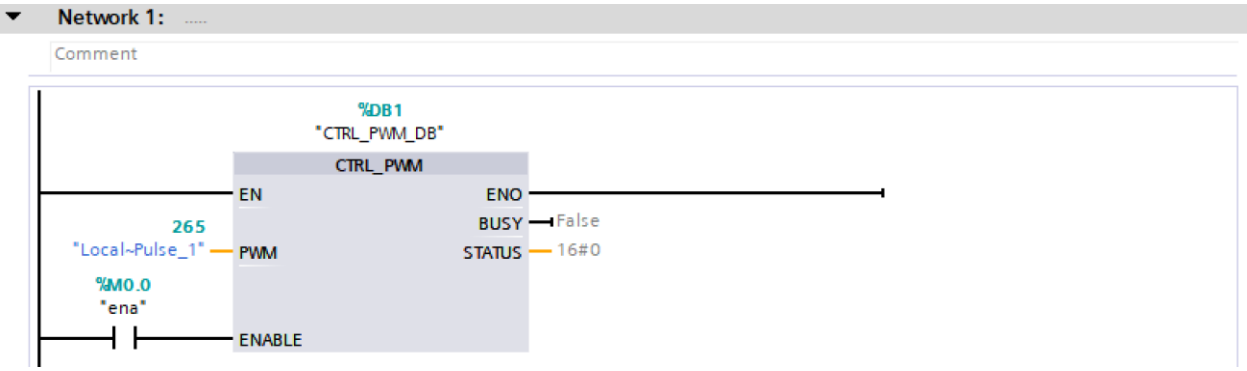
Output variables to be written to are:

Axis 1 - QW1008
Axis 2 – QW2
Axis 3 - QW1014
Axis 4 – QW8

Each variable is 16 bit (Word length).

Each axis must contain the following logic:

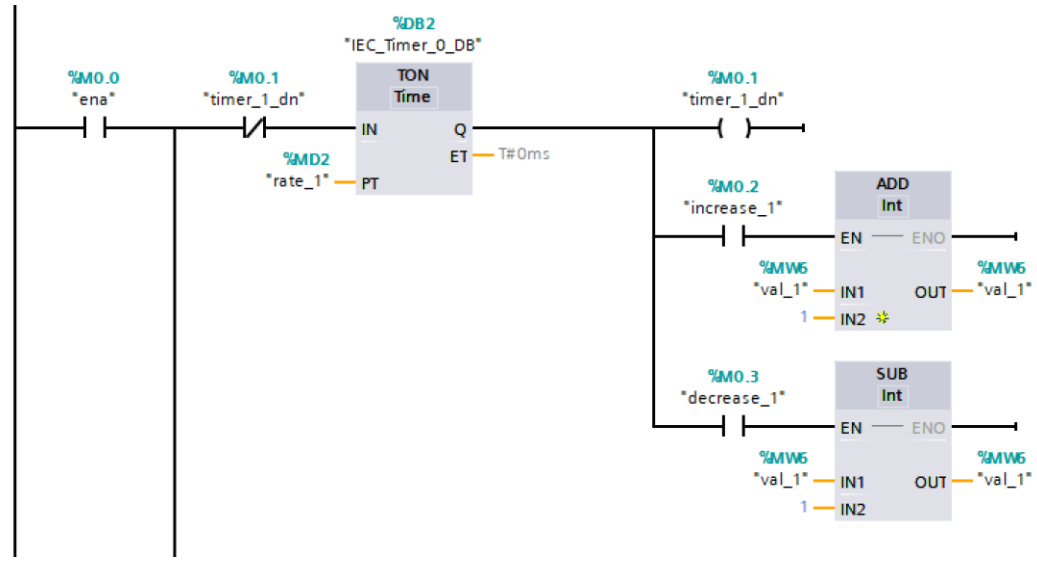
This instruction must be inserted to set up the PWM motion.

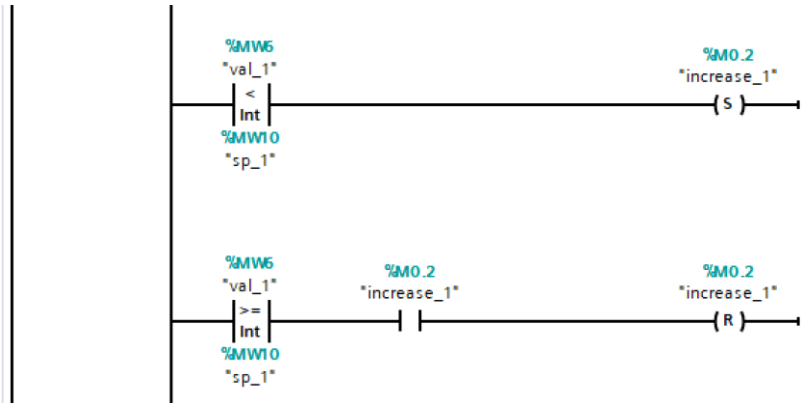


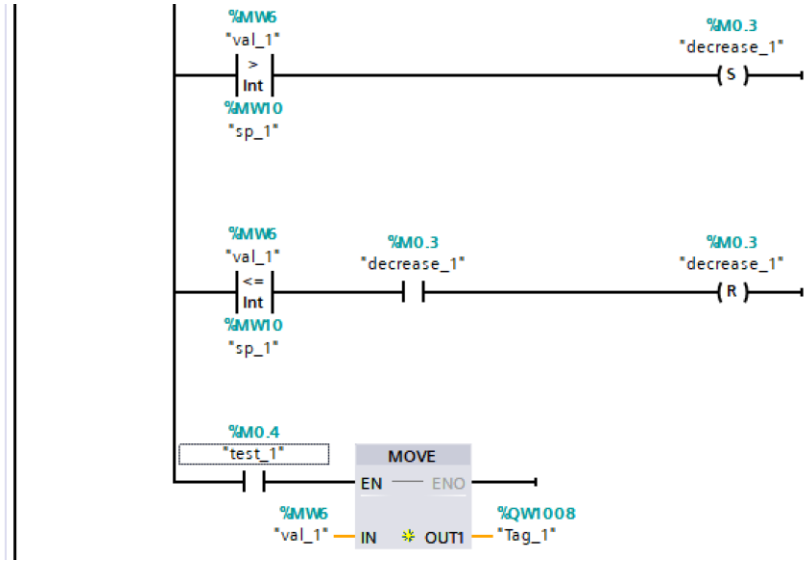
This lab may be programmed several different ways. The best would include arrays that have positional data for the various axes. As the robot moves through various positions, the robot program moves down through the list of position data at each position. There should also be a rate of movement for each of the axes (or perhaps just one common rate as programmed here). The rate of movement is important as well as sequence of the positional data.

At present, only one of these robotic arms is available. The 5 V separate power supply is extremely important as the current draw by the various axes is large.

The following logic is a timer that either increases or decreases the value to be output to the PWM. The setpoint determines whether the value is to grow or decrease.







The power supply shown at left is a large 5 V supply capable of supplying current to a number of different servo controllers.

The remainder of this lab is found in the Hybrid Lab Text's Ch. 31. The entire listing is found there for a program that takes a number (command) and moves through up to 9 different points to execute a robotic motion. There is a fifth array with the command to either pick up or put down a part using suction.

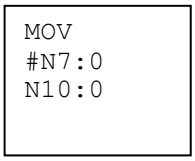
Exercises:

1. Write logic to move sequential data from a table starting at N7:20 to location B3:20. Use B3/9 to reset to the top of the file (B3/9 is a one-shot pulse). Use B3/10 to move the data (B3/10 is also a one-shot pulse).

   Use the instructions below to fill in changes in the table:

Source	Value	Destination	Value
N7:0	102	N10:0	0
N7:1	115	N10:1	0
N7:2	210	N10:2	0
N7:3	365	N10:3	0
N7:4	542	N10:4	0
N7:5	9	N10:5	0

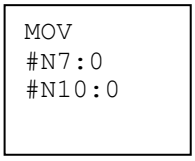
2..



and s:24 =2 →

Destination	Value
N10:0	
N10:1	
N10:2	
N10:3	
N10:4	
N10:5	

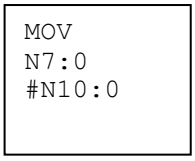
3.



and s:24 =4 →

Destination	Value
N10:0	
N10:1	
N10:2	
N10:3	
N10:4	
N10:5	

4.



and s:24 =3 →

Destination	Value
N10:0	
N10:1	
N10:2	
N10:3	
N10:4	
N10:5	

5. With the problems 2-4 above, rewrite the logic using Siemens' LAD equivalent instruction, Allen-Bradley's RSLogix 5000 LAD equivalent instruction.

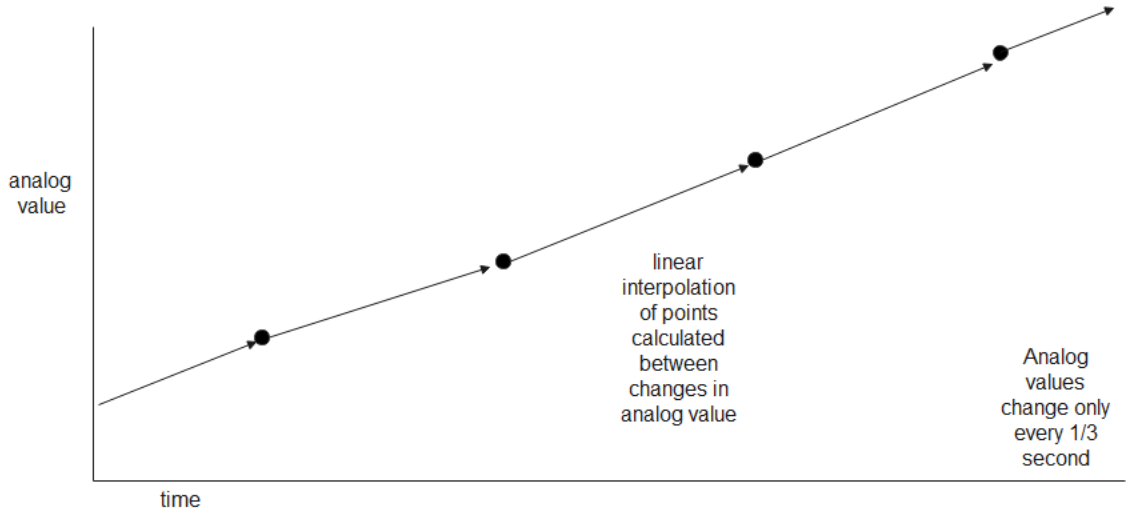
6. Rewrite rungs 9-14 of the Simon Says sample program using A-B RSLogix 5000 LAD code.

7. Review the following from the Chemicals & Petrochemicals Plant Automation Congress 2015 presentation "Batch Process Control Strategy"

8. Review a scale input method and comment on the interface between the scale system and the PLC.

9. Discuss implementation strategies for programming the batch project in Fig. 13-32.

10. Write a program to successfully turn off a solenoid based on a moving weight for the analog value when the updated values are only reported every 1/3 second. Use the following graph as a guide.



11. Using the A-B software and the Siemens software, create a UDT with at least five different variables and three different types.

12. Provide a network compiled and working using a Variant type variable.

Appendix

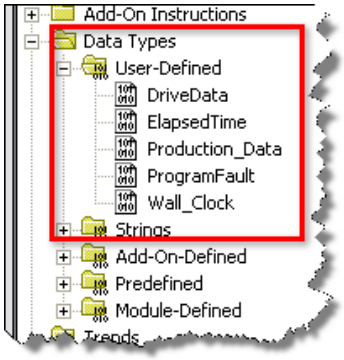
Linking PLC UDT Tags to HMI Faceplates and Pop-ups in TIA Portal V13 SP1 (DMC Corp)

A blog posted by Jason Mayes in Front Page, PLC, Automation, Siemens PLC, HMI and SCADA, WinCC

Please Review for Creating UDTs combining Logic and HMI

User Defined Data Types (UDTs) with A-B

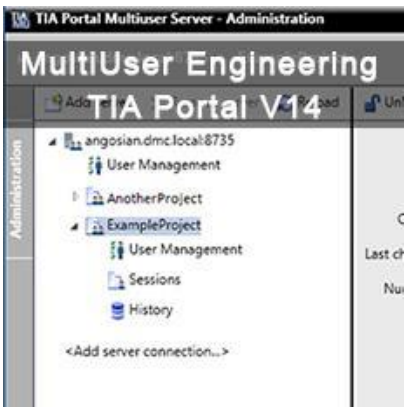
The Controller Organizer has a folder called Data Types > User-Defined with all the UDTs in the project.



Review this data type from A-B.

Leveraging Siemens MultiUser Engineering for TIA Portal

Posted by Gina Brooks-Zak in PLC, Automation, Siemens PLC, HMI and SCADA



Review this as well: