Chapter 5 Control Task Basics

In his chapter, we begin programming a process. The first task is to conceptualize the problem, then move to describe a solution with combinational and then sequential logic. Combinational logic is simply And/Or logic. This chapter will discuss combinational logic and use the DeMorgan Theorem to practice the use of Ladder contacts and coils. We will turn to Chapter 6 to discuss sequential or memory logic. Addressing of variables is also formally introduced.

Modeling the Control Task

Most verbal descriptions of a technical task are not effective in their scope and are unreliable and not clear-cut. A technical sketch, on the other hand, is reliable but lacks description of the human and therefore may miss important details. Several approaches are the usual best approach to describing a process to be programmed. All these types of charts, descriptions, sketches, etc are best in describing the engineering model. Even a mathematical equation is acceptable to help the process.

The engineering model must be complete and exact. What is described must work in all circumstances and under all conditions and produce a safe result (that also, in this world, must make a profit).

A description of the engineering process may be described as follows:

Input (from customer)	Phases	Activities	Output (to customer)
Inquiry	Analysis	Problem Analysis Requirement Analysis Cost Calculation	Quotation
Order	Design	Requirements Definition/Design	Construction Documents
Approval of Design Documents	Implementation	Realization, Production including testing	Product
Delivery/Commissioning	Installation	Erection in operational environment	Useable Facility
Acceptance Commitment	Operation	Service	Customer benefits

Table 5-1 The Engineering Process

Several methods exist to describe a technical task. Some are more closely linked to the technical implementation such as Ladder Logic, Function Block Diagram, and a procedural language such as C or C++. It is always advisable to start with a drawing of the process with the inputs and outputs shown. A formal drawing may be prepared - referred to as a P&ID - to describe a process or an informal drawing such as the one below may be used.

The Juice Condenser

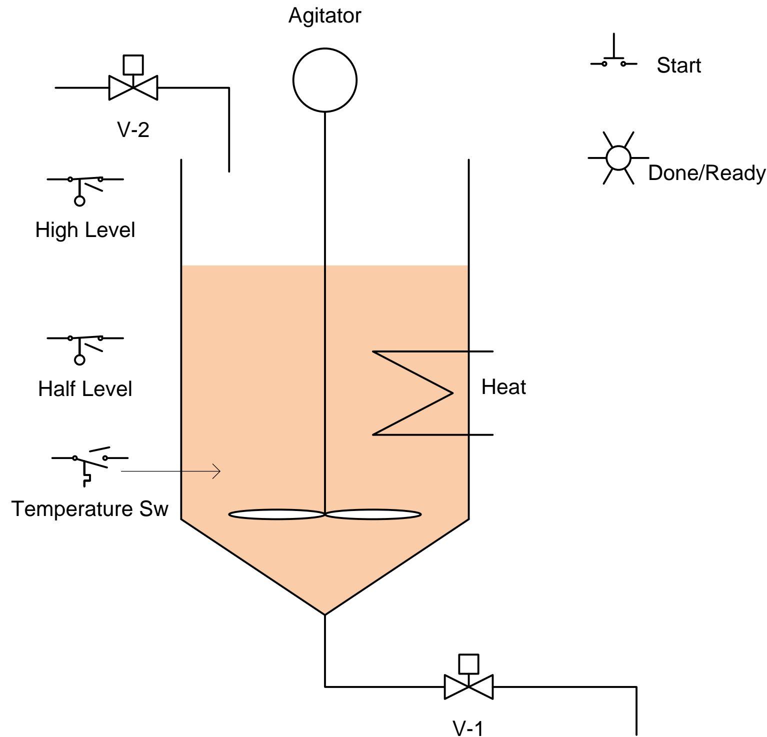


Fig. 5-1 The Juice Maker

A description of the above process is as follows:

For saving transportation cost for apple juice, the juice is condensed in a process of evaporation. The water is evaporated in the tank using heat. The process of the process includes the following steps:

1. Operator pushes the start pushbutton.
2. Valve V-2 opens and fills to the high level switch and then closes.
3. Heating occurs with the heat element on and stays on until the level reaches the half level or the temperature rises above 80° C. The temperature switch turns on when the temp reaches 80° C and turns off when the switch falls below 80° C.
4. Heating is enabled by the high level switch on and the agitator is always on as long as the half level switch is satisfied.
5. When the half level switch is not satisfied, the condensing process terminates and the tank empties through V-1. After the tank starts emptying, 30 seconds is timed and the tank is assumed to be emptied. The Done/Ready light is turned on and the next cycle is allowed via the Start button.

Timing diagrams may be used to describe the process. The drawing below is of the heat cycle once the vessel has been filled to high level until the condensate has been reduced to the middle level.

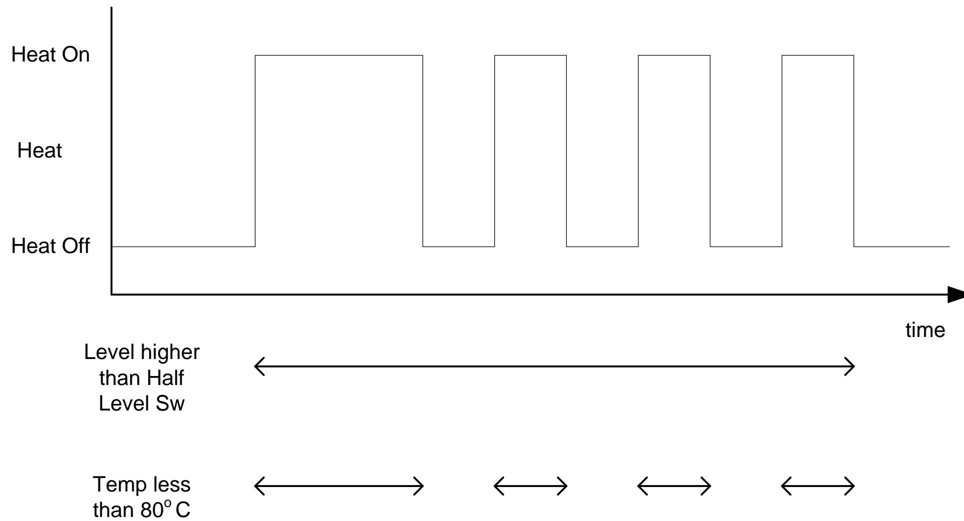


Fig. 5-2 Use of Timing Diagrams

Out of the verbal description, it is not clear whether the start button should be allowed to start an operation even though the tank is not empty. This is an error in the program if the condensed or partially condensed juice is not to be allowed to mix with fresh juice.

The use of Boolean or ladder logic expressions may aid or visualization of the process. For example, to fill the tank, V-2 must turn on. An expression for V-2 is:

V-2 = Start * "Done/Ready" * not High Level + (V-2 * not High Level)

Analysis of the switches must also occur. If a wire were to break, what would happen? Would the vessel overflow? Would the vessel "boil dry"? Questions such as these should be asked and the type of switch or control element specified accordingly.

A table for inputs and outputs should be created identifying each input (sensor) and output (actuator). The Function/State of each should be defined and a Signal Assignment must be given. The assignment is usually "1" unless the device is a device that is a device primarily concerned with stopping the process. Usually these devices are normally closed and open when exceeded. This may pertain to level, process/run or other stopping criterion. These tables are shown below:

Sensor	Function/State	Signal Assignment
Start Button	Start	1
High Level Switch	Level exceeded	0
Half Level Switch	Level exceeded	1
Temperature Switch	80° C exceeded	0

Table 5-2a Input Definitions

Actuator	Function/State	Signal Assignment
Agitator motor	Stirring/running	1
Fill Valve V-2	Fill tank	1
Flush Valve V-1	Empty tank	1
Heat	Heat juice	1
Done/Ready Indicator	Illuminated	1

Table 5-2b Output Definitions

After designing the states for the inputs and outputs, care must be taken to properly implement the switch and control actuator properly. Programs and wiring must be implemented with the correct condition to turn on or off each device.

Turning our attention to the various devices to be programmed, look first at the "Done/Ready" light (L1).

A boolean equation for this light could be:

not L1 = V-1 + Half + V-2

Using Boolean logic transfer function, DeMorgan's theorem,

L1 = not V-1 * not Half Level * not V-2

Timing Chart for various functions of Juice Condenser:

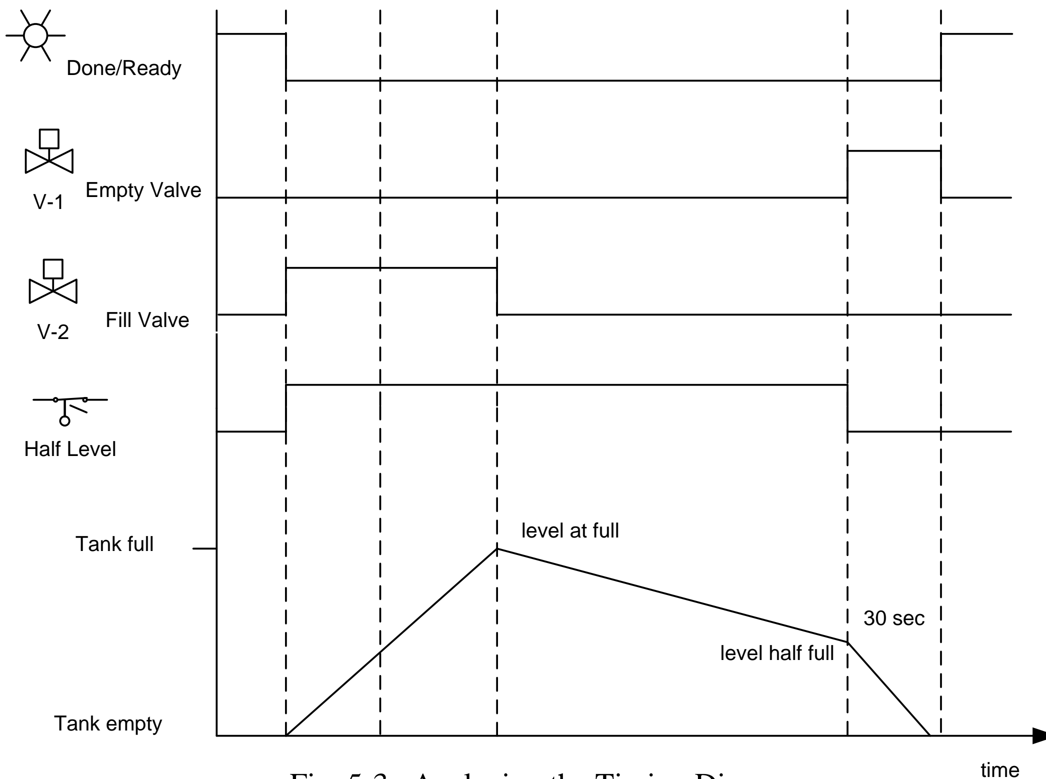
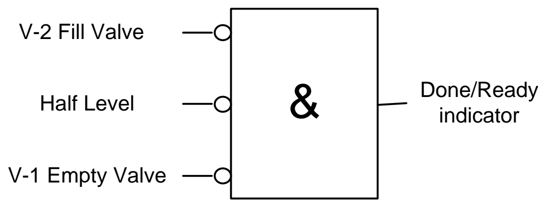
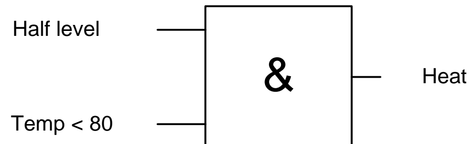


Fig. 5-3 Analyzing the Timing Diagram

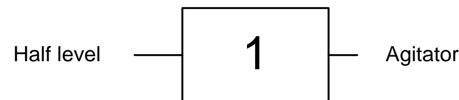
An expression for the Done/Ready indicator is as follows (in FBD):



The heater can be expressed as follows:



and the agitator motor as:



For the heat contactor, the hysteresis of the temperature sensor can be exploited. If this is not the case, a time delay would be needed to turn on and off the heat and avoid an on-off-on-off chatter that would pre-maturely fatigue the contactor and cause a fault.

The fill function can be programmed as follows:

V-2 = Start PB * Done/Ready* not High Level + (V-2 * not High Level)

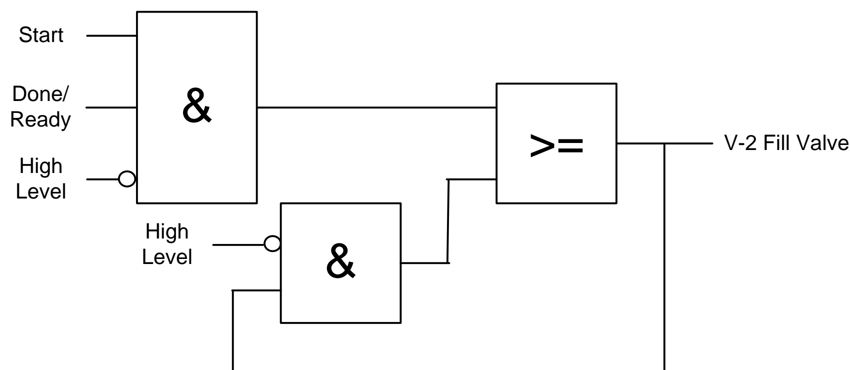


Fig. 5-4 FBD Logic for Fill Valve

Care must be taken when plotting timing diagrams to not imply a signal that can feed back on itself. While hard to understand, when such a circuit is implemented, it tends to not ever turn off but perpetually turn on and off at will. Every means should be used to eliminate such circuits. Time delays are a good way to circumvent such bad design.

We next turn to what to call variables in the two PLCs we use in this text. First is Siemens, then the older Allen-Bradley RSLogix 500 and then RSLogix 5000. The class uses only the Allen-Bradley RSLogix 5000 for A-B programs. Siemens uses the 1200 although it is equal to the 1500 for most applications.

Siemens Step 7 Basic Addressing

Memory Management:

The CPU provides the following memory areas to store the user program, data and configuration:

- Load memory is non-volatile storage for the user program, data and configuration. When a project is downloaded to the CPU, it is first stored in the Load memory area. This area is located either in a memory card (if present) or in the CPU. This non-volatile memory area is maintained through a power loss. The memory card supports a larger storage space than that built-in to the CPU. In class, the memory card is not used but will be valuable in applications in the factory environment.

- Work memory RAM is volatile storage for some elements of the user project while executing the user program. To improve system performance, the CPU copies some elements of the project from load memory into work memory. This volatile area is lost when power is removed, and is restored by the CPU when power is restored.

- Retentive memory is non-volatile storage for a limited quantity of work memory values. The retentive memory area is used to store the values of selected user memory locations during power loss. When a power down occurs, the CPU by design has enough hold-up time to retain the values of a limited number of specified locations. These retentive values are then restored upon power-up.

To view the memory usage for the current project, right-click on the CPU and choose “Resources”. To view the memory usage for the current PLC, double-click “Online and diagnostics” in the tree, expand “Diagnostics” and choose “Memory”.

Several options are available for storing data during execution of the user program:

1. Memory locations including (I), outputs (Q), bit memory (M), data block (DB), and local or temporary memory (L). User programs may read or write to any of these locations.

2. Data block (DB) locations store data for code blocks. Data stored in a DB is not deleted when execution of the associated program block ends. Two categories exist:

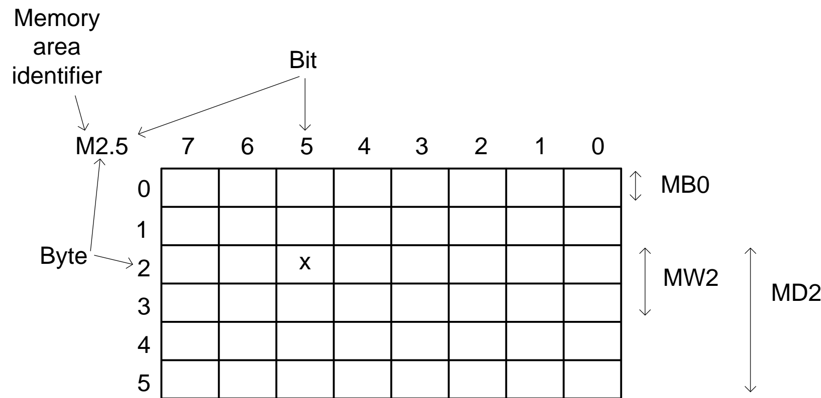
    a. Global DB: data usable by other blocks
    b. Instance DB: data usable only by a specific FB

3. Temporary memory: This is local memory (L) that is allocated when a code block is called and is lost after the block ends. This memory is reallocated to other blocks as needed.

4. Referenced addresses such as I and Q which access specific addresses in the processor's image. To access the input for an immediate read, include :P with the address. For example, I0.3 is read at the beginning of the scan while I0.3:P is read immediately.

Siemens Byte Address Format

Word and bit address can be used for addresses in Siemens' processors. With the 1200 processor, the TIA portal supports symbolic programming. The following addressing scheme is used only if desired for a specific reason. Data can be accessed in any of the memory areas (I, Q, M, DB and L) as byte, word or double word using this format. If a symbol uses 8 bits, use the B format. If two bytes are used, use the W format. If 4 bytes are used, use the D format.



A review of addresses for the S7-1200:

Type	Description	Notation	Example
Process Image output	The program writes values from the process image output table to output modules at the beginning of scan	Q QB QW QD	Q0.0 QB2
Process Image Input	The program writes input values from the input modules and saves them in a process image input table at beginning of scan	I IB IW ID	I0.0
Bit Memory	This is storage for internal memory	M MB MW MD	M0.0
Data Block	Data block storage	DBX DBB DBW DBD	
Local Data	Temporary storage of block	L LB LW LD	
I/O Immediate	Direct access/immediate read or write	<tag>:P	

Table 5-4 S7-1200 Memory Table Types

Addressing the I/O of the Siemens 1214C Processor

1. CPU input bits are addressed from I0.0 to I0.7 and I1.0 to I1.5 (14 points total)
2. CPU output bits are addressed from Q0.0 to Q0.7 and Q1.0 to Q1.1 (10 total points)
3. CPU analog inputs are addressed by words IW64 and IW66 (two analog points).

Data Types for the S7-1200:

Data type	Size
Bool	1 bit
Byte	8 bits
Word	16 bits
DWord	32 bits
Char	8 bits
Sint (short integer)	8 bits (-128 to 127)
USInt (unsigned short integer)	8 bits (0 to 255)
Int (integer)	16 bits (-32768 to 32767)
UInt (unsigned integer)	16 bits (0 to 65535
Dint (double integer)	32 bit (-2,147,483,68 to 2,147,483,647)
UDInt (unsigned double integer)	32 bit (0 to 4,294,967,295)
Real (real)	32 bit (+/- 1.18 x 10^{-38} to +/- 3.40x 10^{38})
LReal (long real)	64 bits (+/- 2.23 x 10^{-308} to +/- 1.79x 10^{308})
Time (time)	Defined later
String (character string)	Variable
DTL (data and time long)	Defined later

Table 5-5 S7-1200 Data Types

For example, the addresses for inputs on the S7-1200 are shown below. Typical addresses are included. Fourteen points in all are addressable:

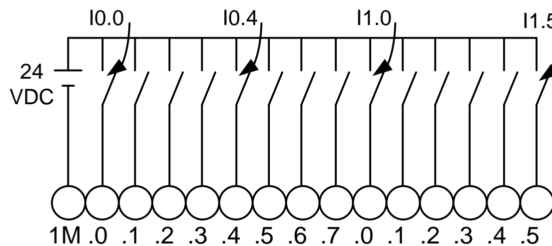


Fig. 5-5a Input Wiring

Outputs are shown in a similar fashion:

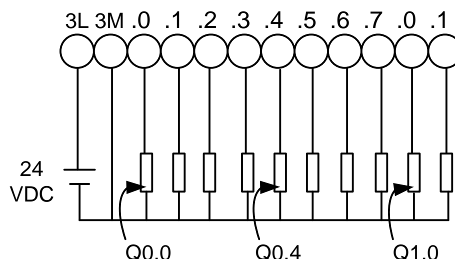


Fig. 5-5b Output Wiring

Configuration of tags is required. To configure a Boolean input, choose Bool and I. For address, allow the program to assign the next address or you may choose a specific address manually:

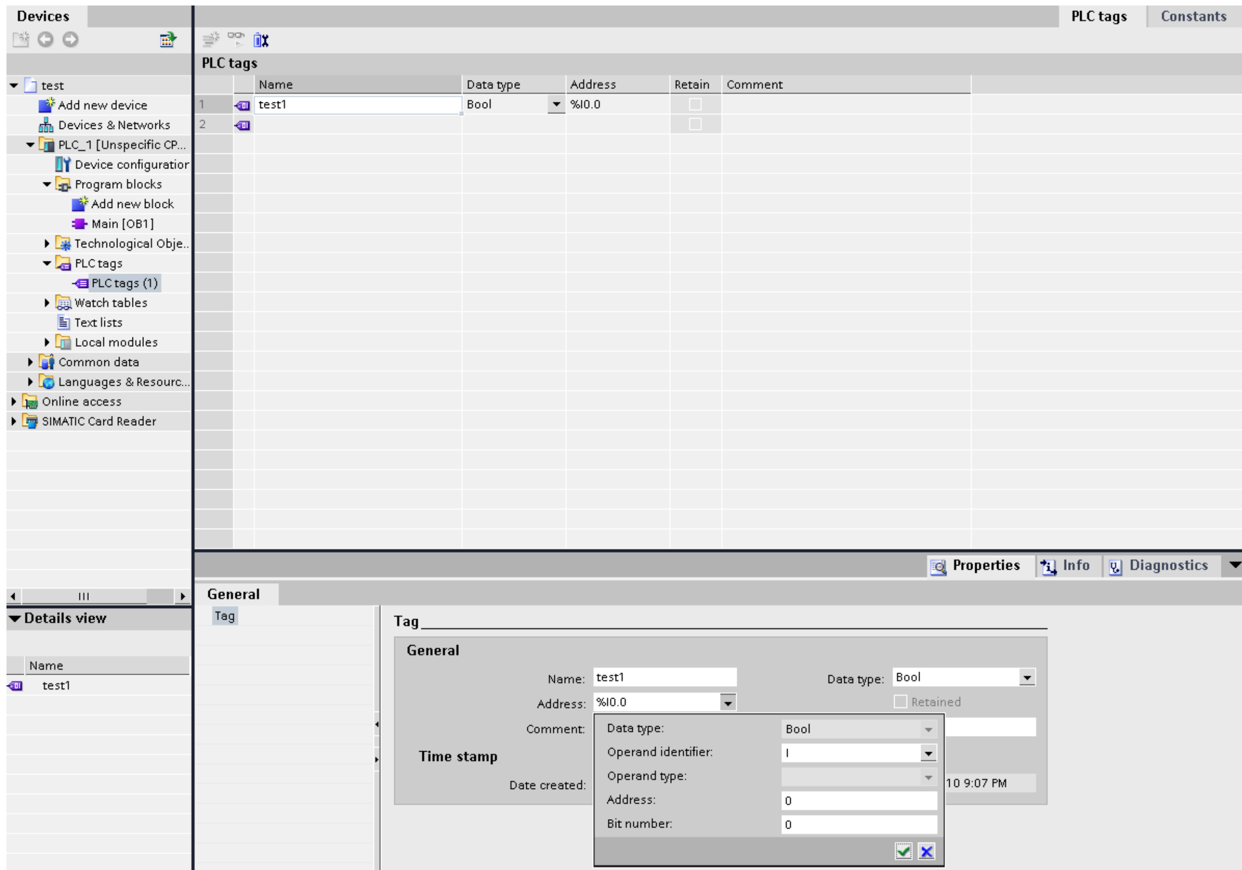


Fig. 5-6 Addressing a Bool

The address can be changed in the Operand type box. I represents Input, Q represents Output, and M represents internal memory. Internal memory may be designated as Bool, Byte, Word, etc.

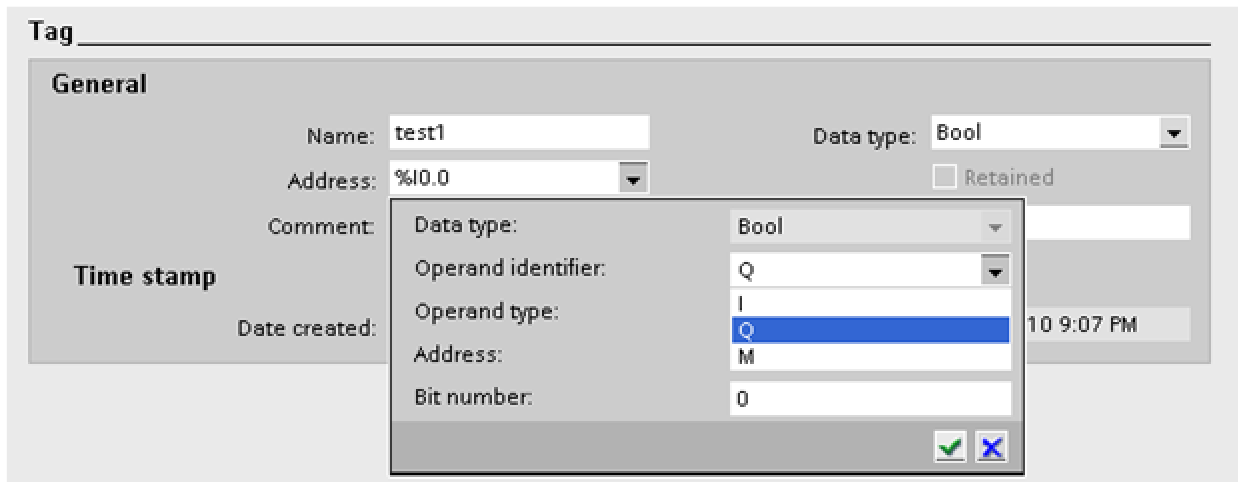


Fig. 5-7 Choosing the Operand Identifier

The contact can be addressed by right-click access to the contact when being programmed. An internal tag address is obtained by accessing M0.0, M0.1, M0.2 through the first byte, then M1.0, M1.1, etc.

Do not choose an immediate bit.

Rather, use the global bit which picks a bit in the M table. Immediate bits are only to be used to pass information in the scan. These bits are reset to 0 at the end of scan, thus losing their value for future use.

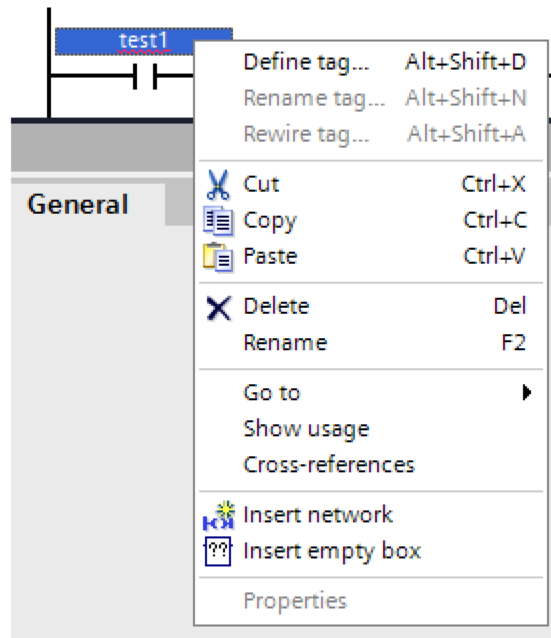


Fig. 5-8 Defining the Tag

The address is automatically assigned if not chosen by the programmer.

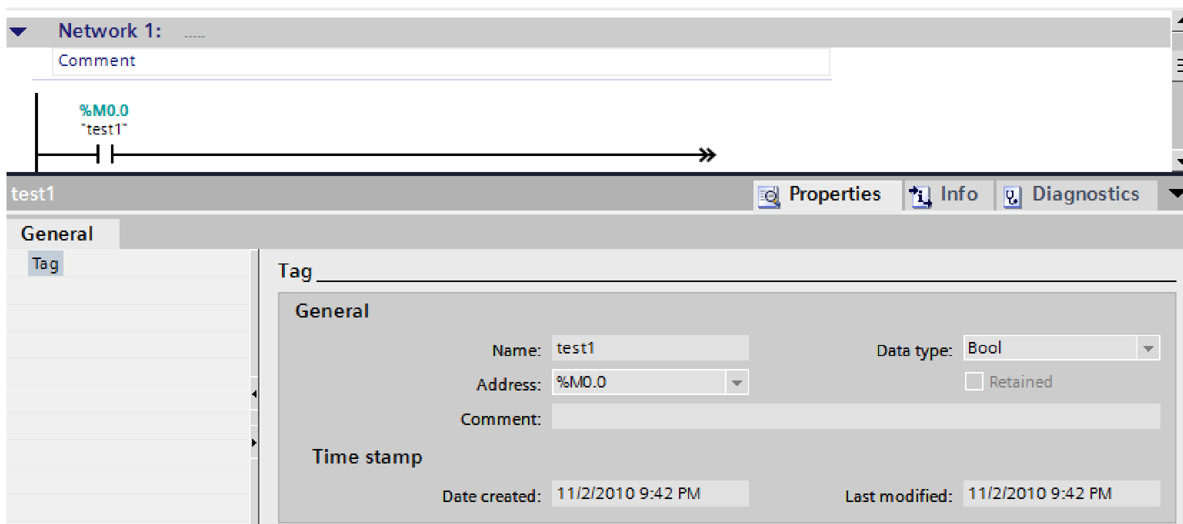


Fig. 5-9 Completing a Tag's Address

Addition of a second contact automatically allocates the next bit in Word 0, M0.1.

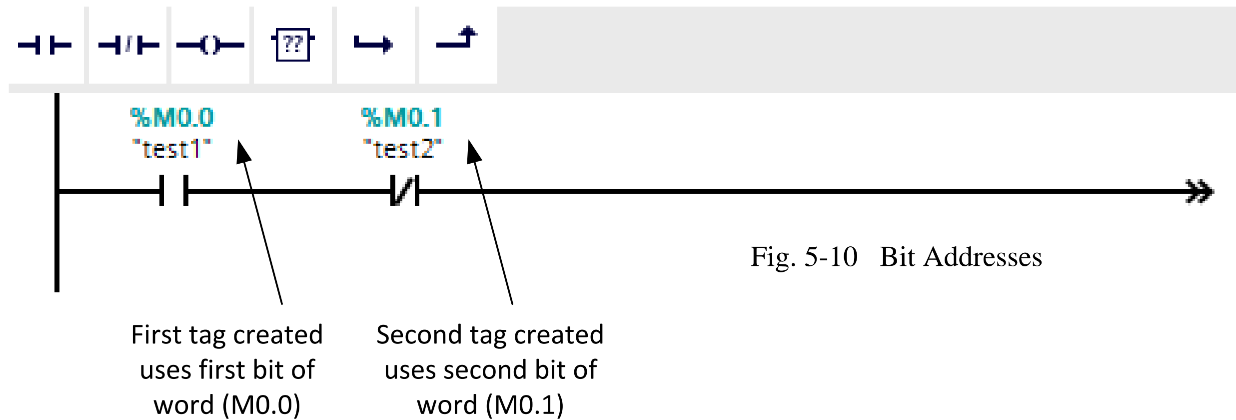


Fig. 5-10 Bit Addresses

Addition of an instruction requiring byte-length instructions shows the address assignment. Byte and word assignment starts with an even byte address. The following instruction shows a compare statement with the byte MB2 compared with MB3.

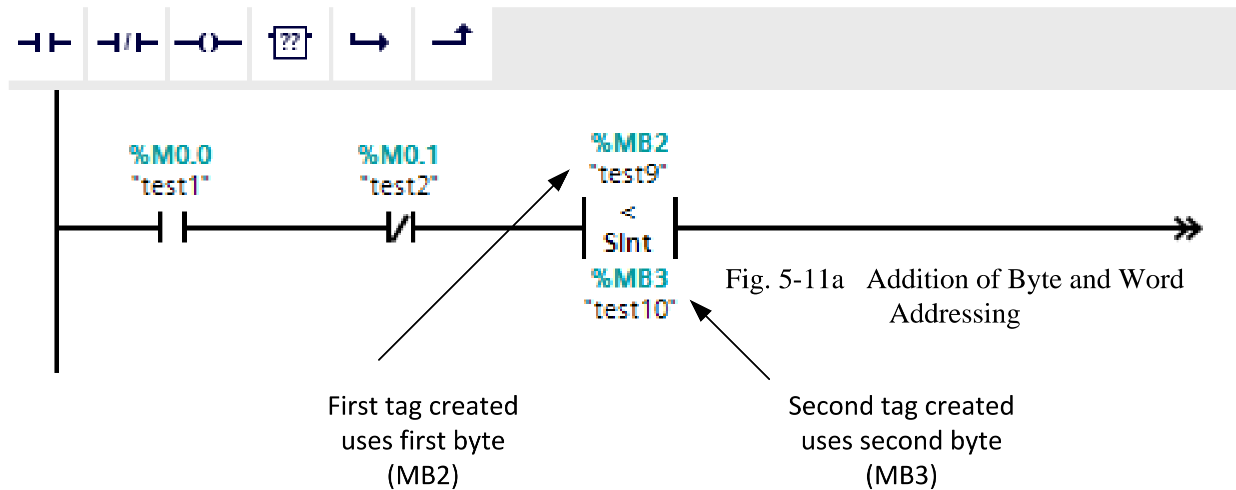


Fig. 5-11a Addition of Byte and Word Addressing

Programming of the instruction shows the designation of Data type as SInt and the Section as Global Memory.

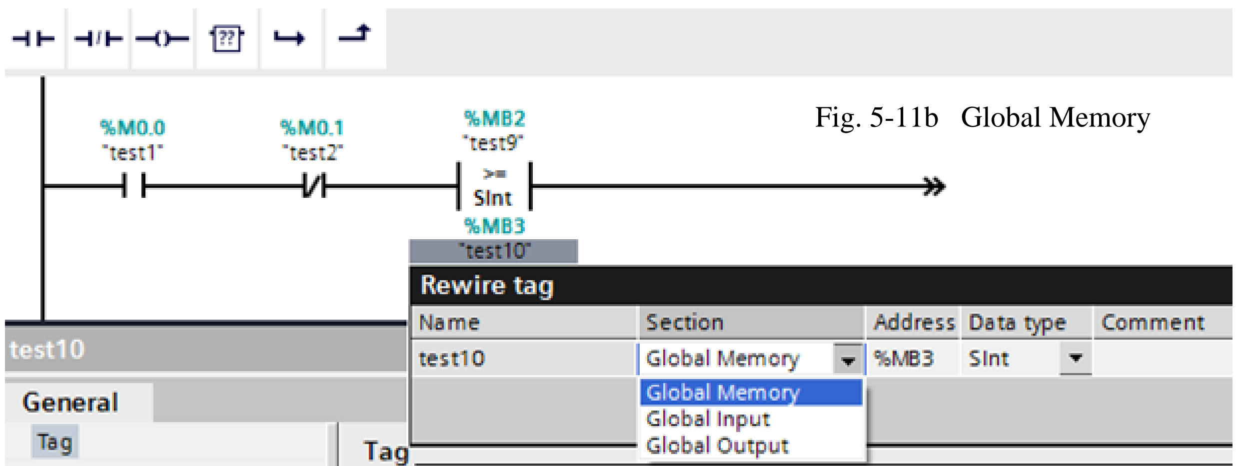


Fig. 5-11b Global Memory

Other Siemens address examples will be examined later in the text. The main addition in a later chapter describes the addressing of the FB or Function Block. The DB or data block associated with the FB carries the provision of static variables. These variables are not attached to M tags bur are saved in memory for future use. They act in a manner similar to the A-B RSLogix 5000 variables. They can be created, named, and used without worry as to a M tag or double usage in the program. These variables are valid only in the FB in which they reside.

Allen-Bradley Addressing

Next we examine the addressing of Allen-Bradley. The SLC PLC as well as the PLC-5 shared a common addressing structure referenced as RSLogix 500. The addressing grouped a type of data in a 'File'. These files were then accessed by program statements similar to Siemens. The bytes of logic could be addressed in various ways but programmers were 'encouraged' to address a type of data for a particular instruction. For example, a byte or word could be addressed as a complete entity or individually as bits. This architecture still carries over with the newer Allen-Bradley architecture as well as with the Siemens architecture. A common feature is the ability to access a word of data as either an entire word or as individual bits. The RSLogix 5000 family of processors will be discussed in a later section.

This is an older addressing architecture than in the more modern PLCs. However, due to the large number of legacy PLC machines in existence, especially in older processes or factories, the addressing scheme is important to be introduced. This addressing structure will be introduced here and used later to introduce some programs to be transferred to RSLogix 5000 or Siemens. Either will require some translation to the newer language from this legacy language. This addressing structure has a similar structure to the Siemens' M Table. The variables can be double used. This gives programmers many problems in that the double use of a variable may be very difficult to debug. This is a common problem with early programs in the A-B structure as well as the Siemens M table.

A-B's RSLogix500 Addressing

Data Table Layout of the SLC Processors is shown below:

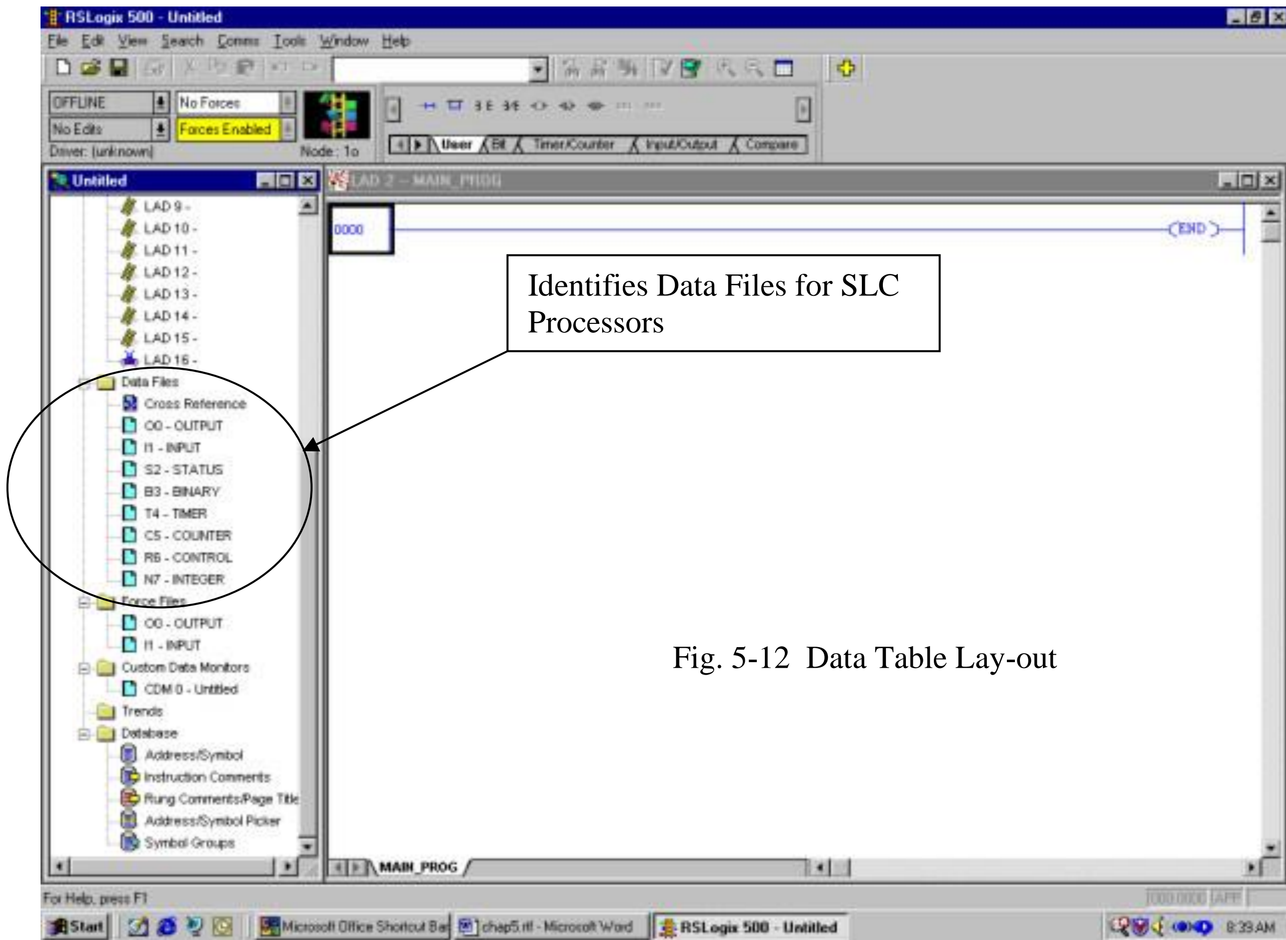


Fig. 5-12 Data Table Lay-out

Data Files in the System Tree lists files available for the MicroLogix controller. The MicroLogix is one of the SLC processors. It has a somewhat limited file structure but is of the same type as all SLC processors. Files listed below are general for all SLC processors:

O0 - Output
I1 - Input
S2 - Status
B3 - Binary
T4 - Timer
C5 - Counter
R6 - Control
N7 - Integer
F8 - Floating Point
ST

Each of the file types may be displayed by double-clicking on the file icon.

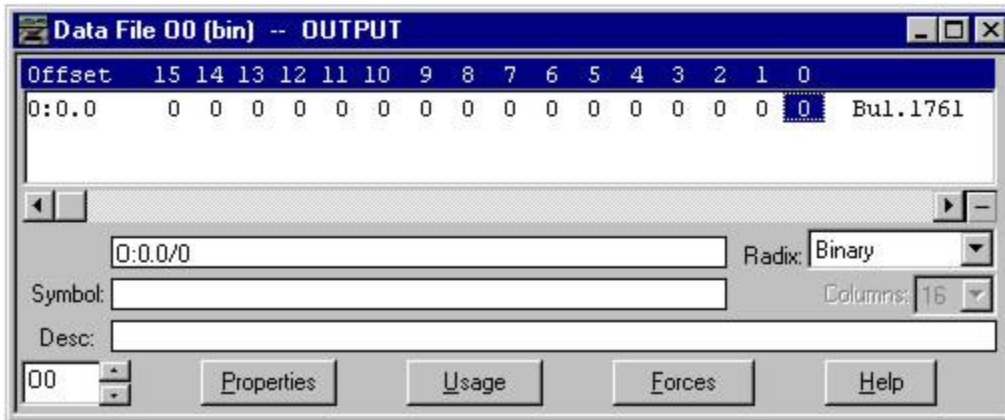


Fig. 5-13 Example of Output Data File

Double click on the O0 – Output icon. The output file for the MicroLogix 1000 is displayed. Notice the addresses:

O:0.0/0 first output (Output 0)
O:0.0/1 second output (Output 1)
… …
… …
O:0.0/15 last output (Output 15)

When referencing an output in a program, set the appropriate bit in this table by turning on either a coil using the OTE coil or OTL latch coil.

Double-clicking the Input file yields similar results with the Input file. Notice addresses are displayed on the MicroLogix controller for the appropriate input and output.

When output coil **O:0/0** is on, turns on bit in Output Table above

Fig. 5-14 Coil in Program Energized

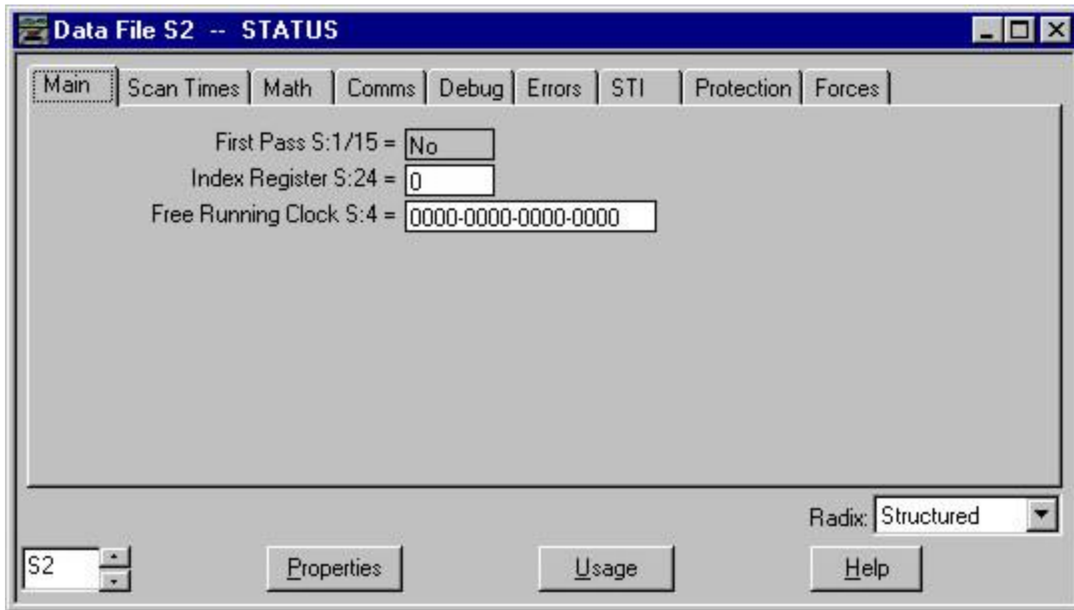


Fig. 5-15   Example of Status Data File

Double click on S2 – Status File Icon. The Status File for the MicroLogix 1000 is displayed.

Notice the Errors Tab. If the processor faults, the error tab shows the reason for the fault.

The Status File is laid out using addresses starting with "S". For example, S:1/15 is the address of the first pass bit. The first pass bit may be used in program control to initialize programs that need to be set to a particular condition as the processor first turns on (to run mode) or as the processor recovers after a power interruption.

The index register is accessed using the word S:24. Indexing will be discussed in a later chapter. Other words and bits may also be accessed using the 'S' table. A complete list is found in one of the appendices of the AB SLC Reference Manual. Status tables vary from processor to processor, with the more powerful processors having the more sophisticated status table layouts.

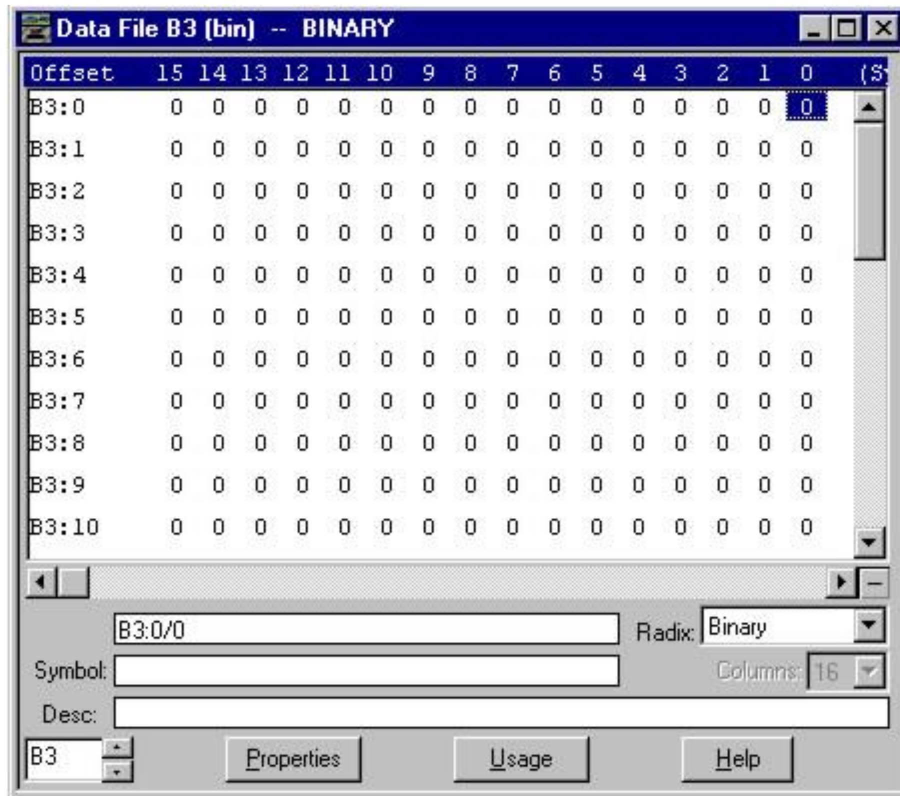


Fig. 5-16 Example of Binary Data File

Double click on B3 – Binary File Icon. The Binary File for the MicroLogix 1000 is displayed.

The Binary File is one of two files storing integers or bits. The other table is the Integer (N7) File. Binary Files are to be used primarily as bit storage tables. They may, however, be used to store numbers. Integer Files are to be used primarily as integer storage tables. Integer files may, however, be used to store bit information.

To enter bit information as an address in an instruction, enter:

B3:0/0 or B3/0

The bits are numbered sequentially starting with 0 and ranging the entire length of the table. Enter bits in either format for the B3 file:

B3:0/0	or	B3/0
B3:0/15	or	B3/15
B3:1/0	or	B3/16
B3:1/15	or	B3/31
B3:2/0	or	B3/32
B3:2/15	or	B3/47

Example of Word/Bit and /Bit addressing in B3:

B3:x/x:

B3:0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B3:1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B3:2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

B3:2/11 (word/bit format)

equal to:

B3/

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

B3/43 (/bit format)

The example of B3:2/11 is equivalent to B3/43. As can be seen, the word/bit address is equal to the bit address for the first word. Then the bit address adds 16 for each word down the table.

Example exercise of finding a Word/bit or /bit address:

Find the bit address equivalent to B3:4/12

Solution:

Notice that for each 16 bit word, the displacement increases by 16 for the bit address. So, for word 4, add 4*16 (=64) to the bit address (12). The result is 4*16 + 12 or 64 + 12 = B3/76

Data File T4 -- TIMER

Offset	EN	TT	DN	BASE	PRE	ACC	(Symb
T4:0	0	0	0	.01 sec	0	0	
T4:1	0	0	0	.01 sec	0	0	
T4:2	0	0	0	.01 sec	0	0	
T4:3	0	0	0	.01 sec	0	0	
T4:4	0	0	0	.01 sec	0	0	
T4:5	0	0	0	.01 sec	0	0	
T4:6	0	0	0	.01 sec	0	0	
T4:7	0	0	0	.01 sec	0	0	
T4:8	0	0	0	.01 sec	0	0	
T4:9	0	0	0	.01 sec	0	0	
T4:10	0	0	0	.01 sec	0	0	

T4:0/EN Radix:
Symbol: Columns: 6
Desc:
T4 Properties Usage Help

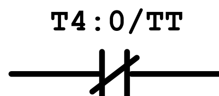
Fig. 5-17 Example of Timer Data File

Double click on T4 – Timer File Icon. The Timer File for the MicroLogix 1000 is displayed.

Timers, Counters, and Control Files are similar in that each is used to hold the information pertaining to the function it supports. For instance T4:0 is the table of information used to support the timer entered in the program using the T4:0 address. Only one timer may be used for each address.

Data useful to timers are the pre-set time, accumulated time, and coils reporting the status of the timer enabled, timer timing, or timing done.

Bits used in the timer may be used in programming. For instance,



will conduct when the timer T4:0 is timing but not at preset. Contacts from timers are used to control functions in which timing functions are needed. Presets and Accumulated values may be accessed using the format: `T4:0.ACC, T4:0.PRE`

Data File C5 -- COUNTER

Offset	CU	CD	DN	OV	UN	UA	PRE	ACC	(Sy
C5:0	0	0	0	0	0	0	0	0	
C5:1	0	0	0	0	0	0	0	0	
C5:2	0	0	0	0	0	0	0	0	
C5:3	0	0	0	0	0	0	0	0	
C5:4	0	0	0	0	0	0	0	0	
C5:5	0	0	0	0	0	0	0	0	
C5:6	0	0	0	0	0	0	0	0	
C5:7	0	0	0	0	0	0	0	0	
C5:8	0	0	0	0	0	0	0	0	
C5:9	0	0	0	0	0	0	0	0	
C5:10	0	0	0	0	0	0	0	0	

C5:0/CU Radix:

Symbol: Columns: 8

Desc:

C5 Properties Usage Help

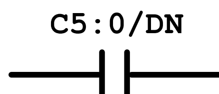
Fig. 5-18 Example of Counter Data File

Double click on C5 – Counter File Icon. The Counter File for the MicroLogix 1000 is displayed.

Timers, Counters, and Control Files are similar in that each is used to hold the information pertaining to the function it supports. For instance C5:0 is the table of information used to support the counter entered in the program using the C5:0 address. While only one timer may be used for each address, both an up-counter and a down-counter may be used by one C5 address.

Data useful to counters are the preset count, the accumulated count, and coils reporting the status of the counter counting up, counting down, done, overflow, underflow or update accumulator (used in high speed counter applications only).

Bits used in the counter may be used in programming. For instance,



will conduct when the counter C5:0 is counted to preset. Contacts from counters are used to control functions in which counting functions are needed. Presets and Accumulated values may be accessed using the format: `C5:0.ACC, C5:0.PRE`

Data File R6 -- CONTROL

Offset	EN	EU	DN	EM	ER	UL	IN	FD	LEN	POS
R6:0	0	0	0	0	0	0	0	0	0	0
R6:1	0	0	0	0	0	0	0	0	0	0
R6:2	0	0	0	0	0	0	0	0	0	0
R6:3	0	0	0	0	0	0	0	0	0	0
R6:4	0	0	0	0	0	0	0	0	0	0
R6:5	0	0	0	0	0	0	0	0	0	0
R6:6	0	0	0	0	0	0	0	0	0	0
R6:7	0	0	0	0	0	0	0	0	0	0
R6:8	0	0	0	0	0	0	0	0	0	0
R6:9	0	0	0	0	0	0	0	0	0	0
R6:10	0	0	0	0	0	0	0	0	0	0

R6:0/EN
Radix:
Symbol:
Columns: 10
Desc:
R6
Properties
Usage
Help

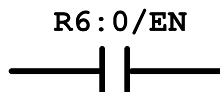
Fig. 5-19 Example of Control Data File

Double click on R6 – Control File Icon. The Control File for the MicroLogix 1000 is displayed.

Timers, Counters, and Control Files are similar in that each is used to hold the information pertaining to the function it supports. For instance R6:0 is the table of information used to support the Control entered in the program using the R6:0 address. Control functions are used in a number of different instructions. Most instructions are advanced instructions and are not used in the lower level course.

Data useful to control blocks includes length, pointer position and status bits for instructions such as shift registers, sequencers, and ASCII write.

Bits used in the counter may be used in programming. For instance,



will conduct when the Control R6:0 is enabled. Contacts from Control Blocks are used to control instructions in which the Control function is required.

Data File N7 (dec) -- INTEGER

Offset	0	1	2	3	4	5	6	7	8	9
N7:0	0	0	0	0	0	0	0	0	0	0
N7:10	0	0	0	0	0	0	0	0	0	0
N7:20	0	0	0	0	0	0	0	0	0	0
N7:30	0	0	0	0	0	0	0	0	0	0
N7:40	0	0	0	0	0	0	0	0	0	0
N7:50	0	0	0	0	0	0	0	0	0	0
N7:60	0	0	0	0	0	0	0	0	0	0
N7:70	0	0	0	0	0	0	0	0	0	0
N7:80	0	0	0	0	0	0	0	0	0	0
N7:90	0	0	0	0	0	0	0	0	0	0
N7:100	0	0	0	0	0					

N7:0 Radix: Decimal
Symbol: Columns: 10
Desc:
N7 Properties Usage Help

Fig. 5-20 Example of Integer Data File

Double click on N7 – Integer File Icon. The Integer File for the MicroLogix 1000 is displayed.

The Integer File is one of two files storing integers or bits. The other file is the Binary (B3) File. Integer Files are to be used primarily as word storage tables. It may, however, be used to store individual bits. Binary files are to be used primarily as bit storage tables. Binary files may, however, be used to store integer information.

To enter bit information as an address in an instruction, enter:

N7:0/0 (N7/0 is not allowed in bit only mode as is B3 bit information.)

To enter word length information as an address in an instruction, enter:

N7:0

Addresses range from N7:0 to N7:105 in the table above. Notice the Radix in the lower right-hand corner of the display. To view the table in binary format, change the radix to *binary*. Other choices are *hexadecimal/BCD, octal, decimal* and *ASCII*.

Word length for N7 and B3 files is 16 bit length.

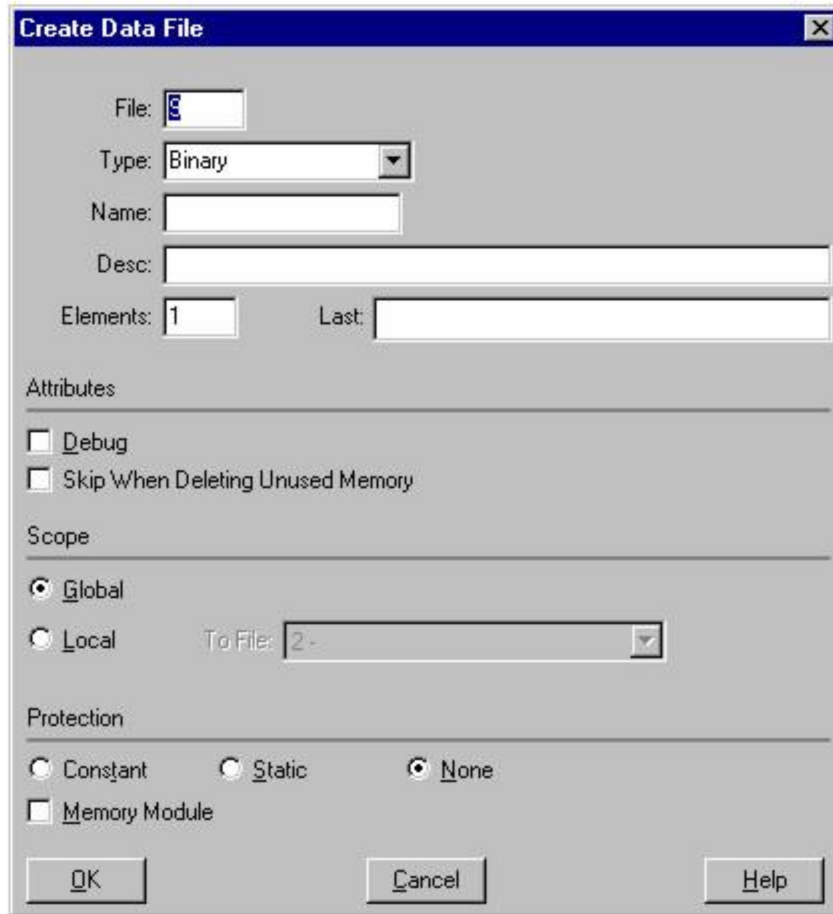


Fig. 5-21 Example of Creation Screen for New Data File

While the MicroLogix 1000 does not allow the addition of other data files, larger processors do allow the addition of data files and the expansion of existing data files. For instance, with the SLC 5/03, by right-clicking on Data Files, and selecting *New*, the table in Fig. 5-21 appears. With it, the user is prompted to enter the name and type of a new data file. With the SLC processors, 255 data files is the upper limit.

A-B's RSLogix5000 Addressing

RSLogix 5000 gives a view of the processor similar to RSLogix or RSLoigx 500 which are used to program the PLC 5 and SLC processors respectively. The view of the ladder programming area is similar as well as the general layout of the system tree at left.

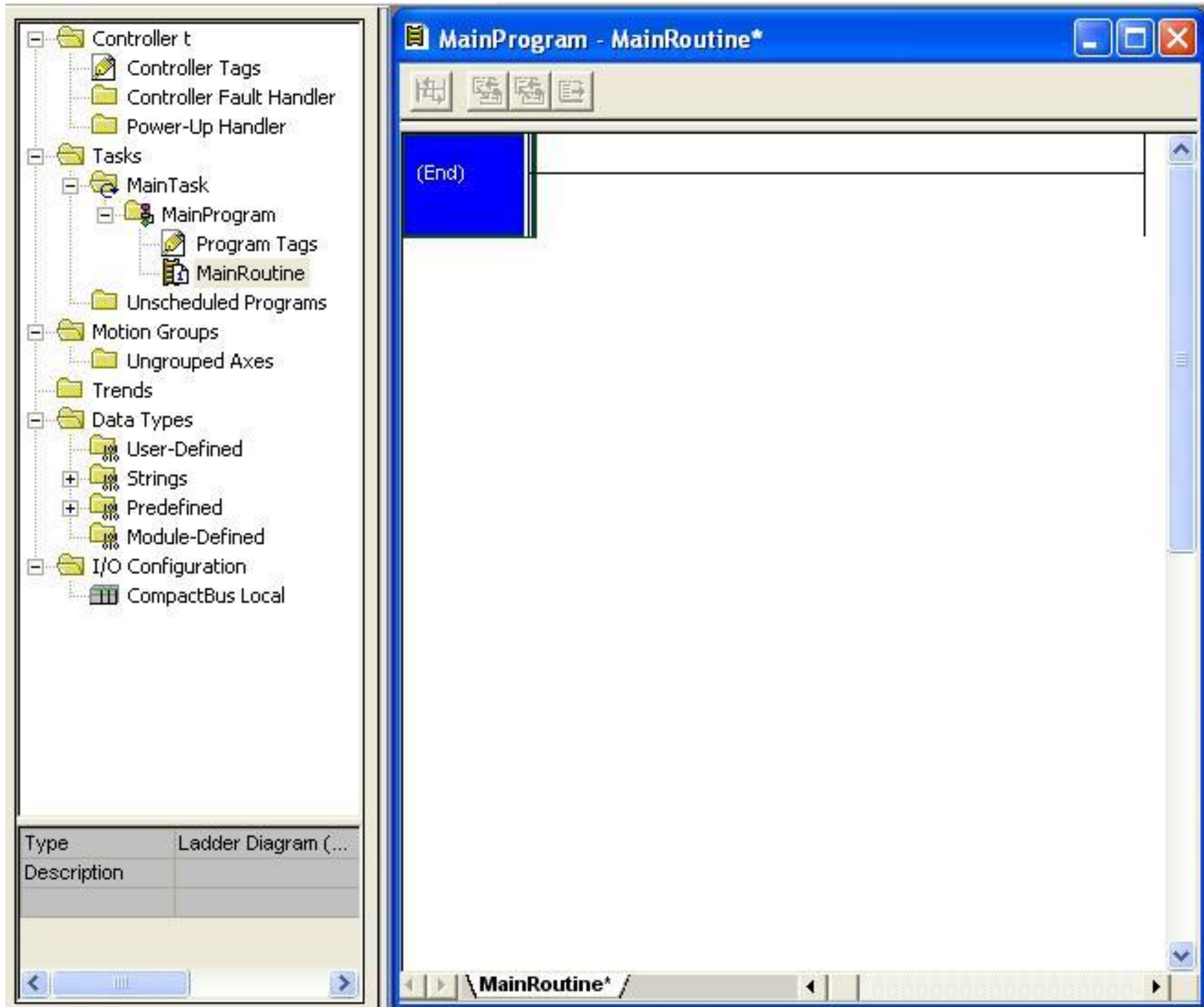


Fig. 5-22 Example RSLogix 5000 Screen

Notice that data files are not saved by type as in RSLogix 500 but by scope. They may be stored as Controller Tags or as Program Tags. Program tags are used only in the program *MainRoutine* or one of its subroutines. If only one program is created under *Tasks*, then the program tags become the database for the entire project except for data tags associated with inputs and outputs (I/O).

Creation of program tags is accomplished by typing a name for each tag. Tags are entered in the Program Tags menu under the *MainProgram* header.

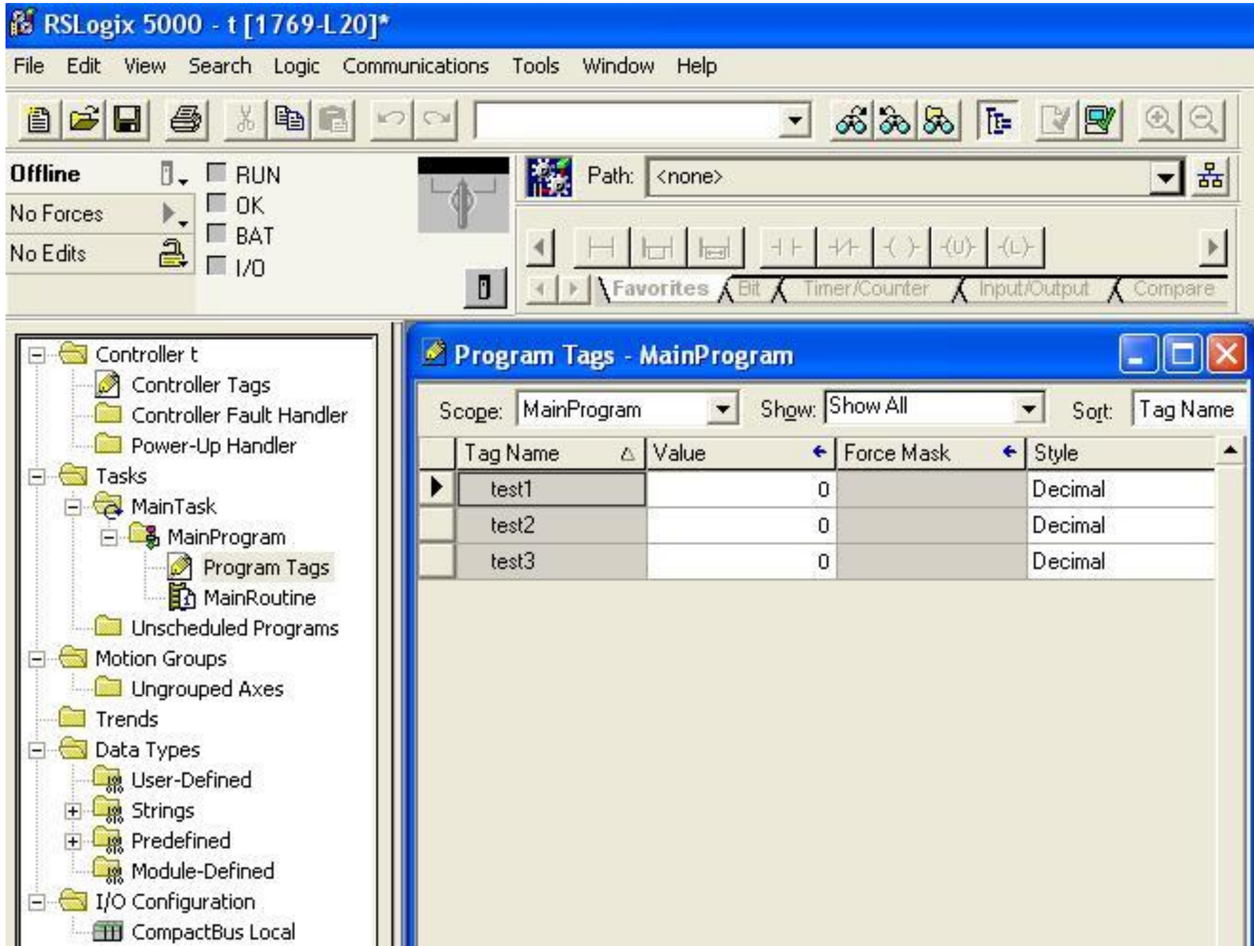


Fig. 5-23   Creation in Main Program

In the example above, tags *test1*, *test2*, and *test3* are being created for use in the MainProgram tag database. These tags must be created before being used in the logic of MainRoutine or a subroutine of MainRoutine. Names may have alpha or numeric content. No spaces are allowed. Very long names may be created. However, consideration must be taken of the HMI program to be interfaced to the processor. If the HMI program limits length to 30 characters, the program tags should also be limited to 30 characters. Some popular brands of HMI limit the tag to approximately 30 characters.

The task of entering database points takes time and should be planned before starting. A popular approach to tag generation incorporates using Excel to generate tags and then import the tags from a CSV (comma-separated-variable) file. Tags may also be exported from RSLogix 5000 to Excel for modification and editing.

Creation of programs is similar to RSLogix 500 (SLC) or RSLogix (PLC-5). The program below shows a simple seal circuit using the three Boolean tags created earlier.

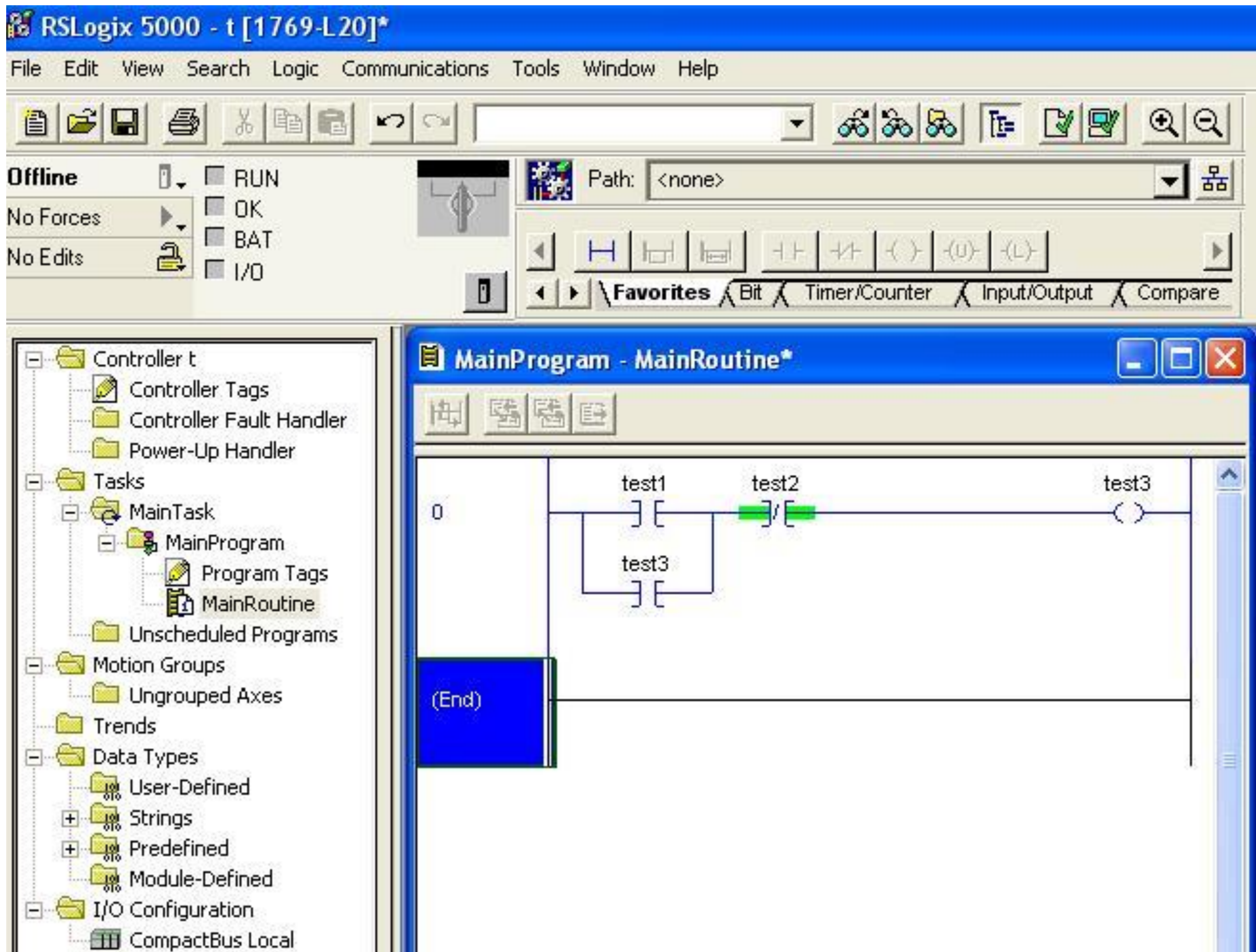


Fig. 5-24 Program Creation in MainRoutine

Once tags are created, they may be used in a program. From the figure above, tags *test1*, *test2*, and *test3* were configured as boolean tags and can be used as discrete bits in logic. This program shows a simple seal circuit using the three bits. The three bits are similar to the SLC architecture except that names in RSLogix and RSLogix 500 included a fixed name such as B3:x/y and an attached comment (if desired). Naming of bits as shown above is left to the creativity of the programmer. If it is desired that the name to be used is B3_3_4, then that tag must be created and programmed. It is better, however, to use names that have meaning to the process. The naming of bits to use in a ladder program should be done carefully since others will probably be reading and troubleshooting the program at some future date.

A great variety exists in the new data selection table. As can be seen in Fig. 5-25 below, the number of data types has been greatly expanded.

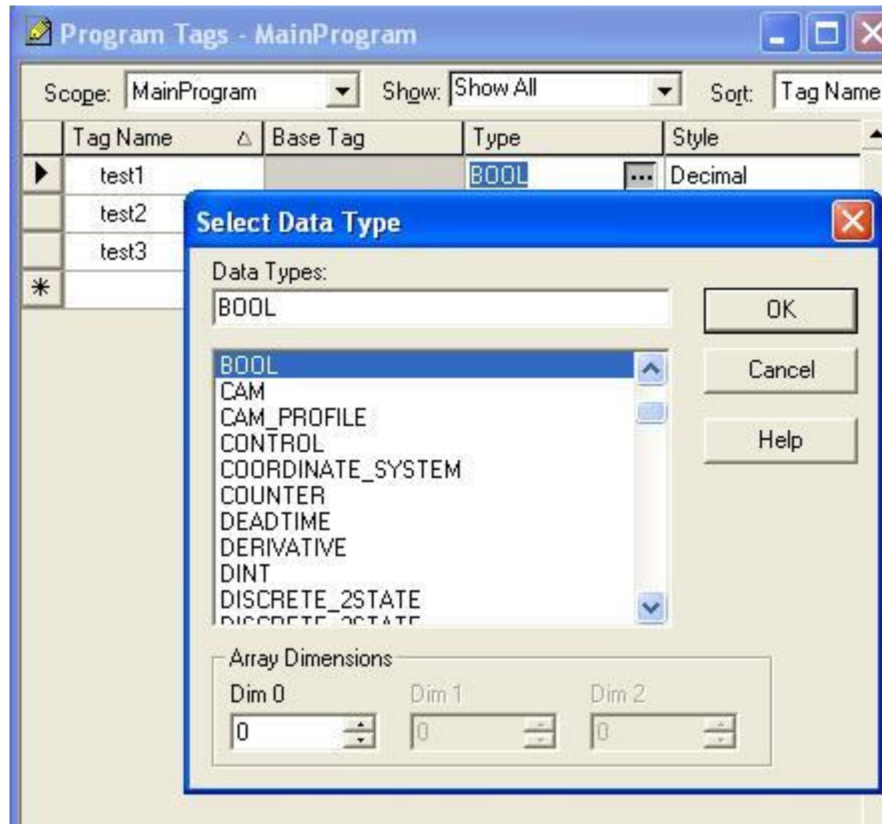


Fig. 5-25 Selection Table for Tag Name

Data selection is not limited to a few common data types. From the manuals listed for ControlLogix and CompactLogix processors is a new area for PLC involvement - Motion Instructions. Data types for motion instructions plus a more advanced set of process control instructions are available.

Timers and counters are similar to the SLC architecture. The figure shown below shows an expanded view of the timer variable *time1*.

time1		TIMER	
time1.PRE		DINT	Decimal
time1.ACC		DINT	Decimal
time1.EN		BOOL	Decimal
time1.TT		BOOL	Decimal
time1.DN		BOOL	Decimal
time1.FS		BOOL	Decimal
time1.LS		BOOL	Decimal
time1.OV		BOOL	Decimal
time1.ER		BOOL	Decimal

Fig. 5-26 Timer Data Type Expanded

For timers and counters, all addresses are created with the original entry. The timer *time1* was created as a timer variable. Variables created in the database with the creation of *time1* include:

time1.PRE	*time1* preset value
time1.ACC	*time1*accumulated value
time1.EN	*time1* Enable bit
time1.TT	*time1* Timer Timing bit
time1.DN	*time1* Done bit

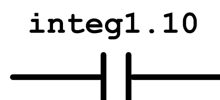
Other bits are used in non-ladder programming routines. Other languages such as Function Block Diagram (FBD) use the other bits listed (.FS, .LS, .OV, and .ER).

Integers are created in similar fashion to boolean and other data types. The figure below shows an integer variable created as *integ1* and displayed as both an integer (*int* datatype) and as 16 discrete boolean bits. The data can be used in either form.

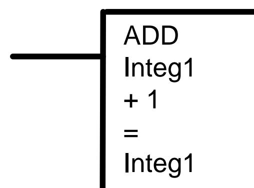
⊟ integ1		INT	Decimal
integ1.0		BOOL	Decimal
integ1.1		BOOL	Decimal
integ1.2		BOOL	Decimal
integ1.3		BOOL	Decimal
integ1.4		BOOL	Decimal
integ1.5		BOOL	Decimal
integ1.6		BOOL	Decimal
integ1.7		BOOL	Decimal
integ1.8		BOOL	Decimal
integ1.9		BOOL	Decimal
integ1.10		BOOL	Decimal
integ1.11		BOOL	Decimal
integ1.12		BOOL	Decimal
integ1.13		BOOL	Decimal
integ1.14		BOOL	Decimal
integ1.15		BOOL	Decimal

Fig. 5-27 Expanded Integer Word

Addressing is either in word or bit format. The table above displays an integer *integ1* as either a word or as 16 discrete bits. The bits may be used to turn on contact closures if required. For example:



Or the word *integ1* may be used in a math statement:



This statement adds 1 to the integer *integ1* each time the block is executed.

Floating point or REAL numbers may also be specified. The window below shows the creation of a real variable *real1*.

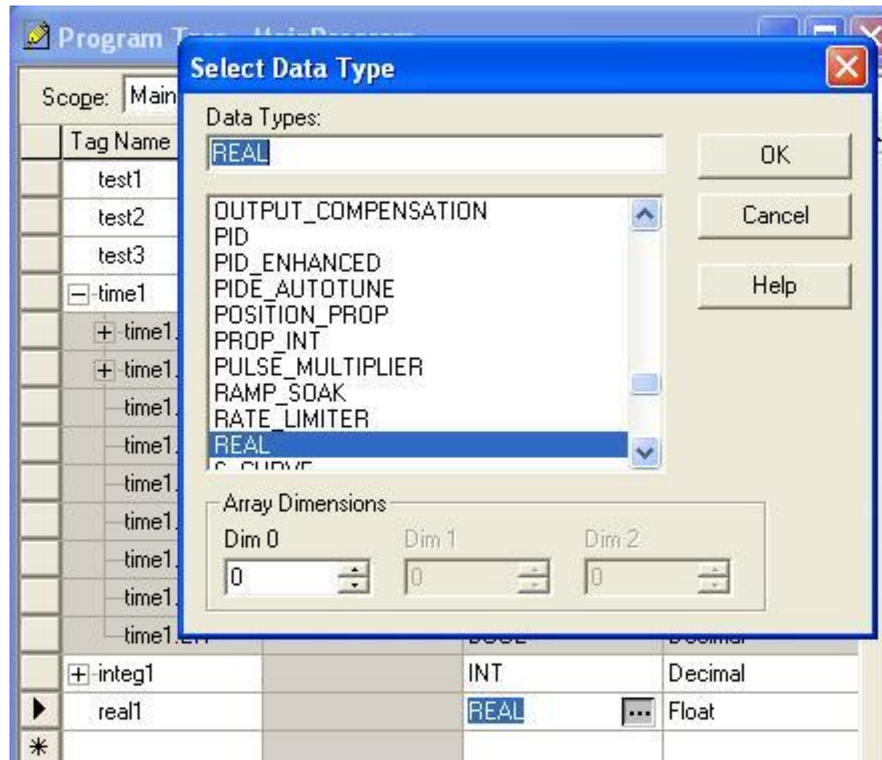


Fig. 5-28a   Real/Floating Point Number

Real or floating point variables are used to represent signed numbers in most math calculations. They are useful in calculations in that overflow is very rare and accuracy is very good. Notice that the choice INT or REAL allows the use of non-zero dimensioned arrays. This choice allows the use of tables and instructions that store numbers based on an index.

Fig. 5-28b   Table of Real Numbers

This table shows the array of real numbers stored as real1[0] through real1[19]. Data may be manipulated using a double integer as the pointer to reference a specific entry in the table.

The window below shows the creation of a subroutine. Notice that two subroutines (sub1 and sub2) exist and that a new subroutine (*sub3*) is being created in this screen. Notice that with the SLC architecture, ladder programs were referenced as Lad2, Lad3, etc.

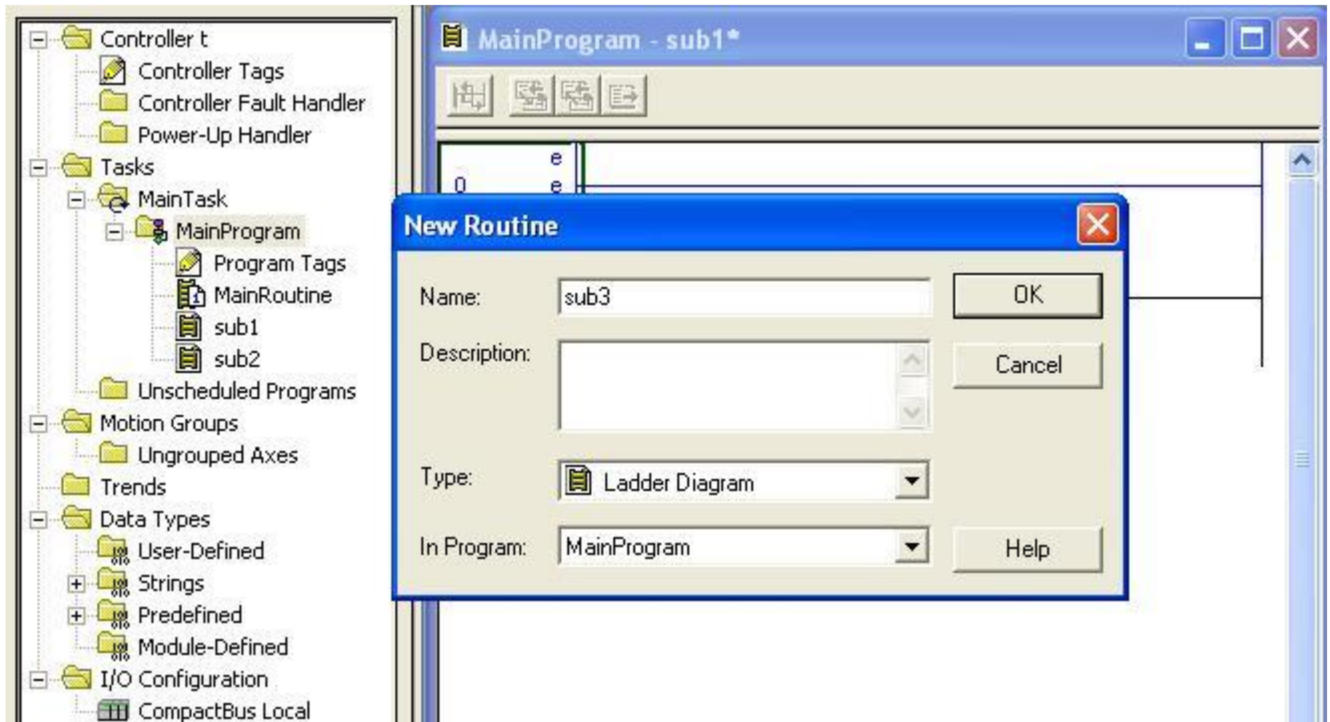


Fig. 5-29 Creating Subroutines

In the program MainProgram, two subroutines are already created and a third routine sub3 is presently being added. Notice the box Type. There may be only one choice, Ladder Diagram, or a number of different choices if the software is enabled to allow programming in another language.

Total programming languages for the PLC portion of the Logix 5000 family include:

- Relay Ladder
- Structured Text
- Function Block
- Sequential Function Chart

Each language has advantages that may be exploited in programming a subroutine. The relay ladder subroutines must include a *return* block to end the subroutine and return to the calling program. Other languages such as Function Block do not require the return block.

Inputs and outputs are added to the ControlLogix or CompactLogix processors by building the table under I/O Configuration. Each card is either recognized by the software or added by hand. The 16 point input card, 1769-IA16/A, has been added in slot [1] to the processor. Addressing for the input is shown in the controller tags list at right in the figure.

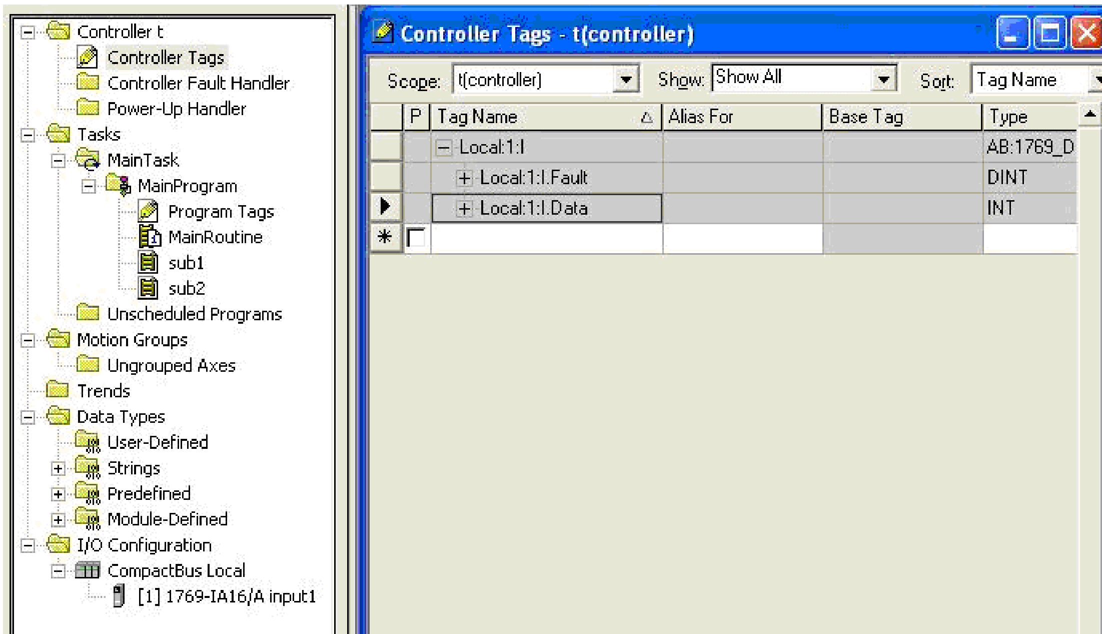


Fig. 5-30a Adding Inputs and Outputs

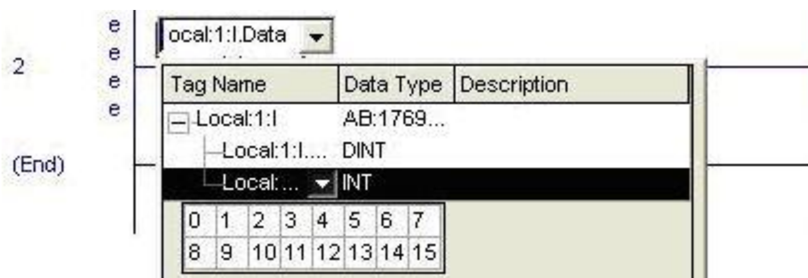


Fig. 5-30b Programming I/O

Fig. 5-30c Input Referenced

DeMorgan's Theorem

We turn again to development of the logic of programming in Ladder Logic. From a course in Digital Logic, DeMorgan's Theorem is studied and found useful to negate or invert Boolean and now ladder logic. The three basic functions of DeMorgan's Theorem are:

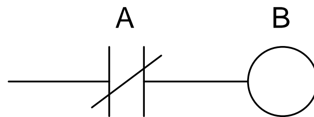
a) Not(Not(X)) = X

b) Not(X or Y) = Not(X) and Not(Y)

c) Not(X and Y) = Not(X) or Not(Y)

These three functions relate directly to ladder logic in that the following apply in ladder logic:

a)



A = C since Not(Not(A)) = A

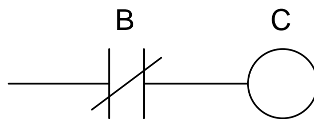
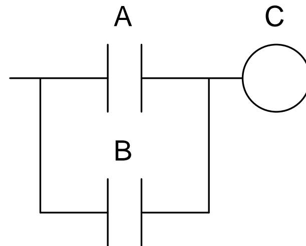


Fig. 5-31a DeMorgan Negation

b)



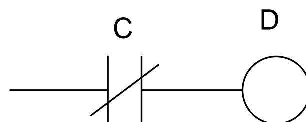
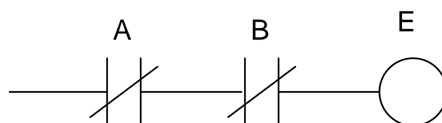


Fig. 5-31b DeMorgan NOR



Since Not((A or B)) = (Not A and Not B), (D = E)

c)

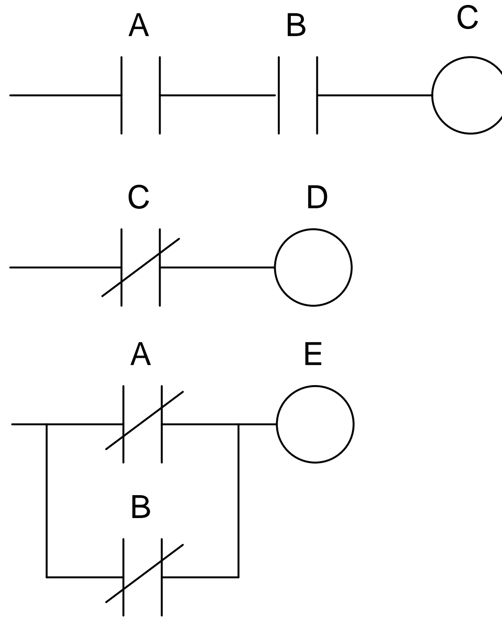


Fig. 5-31c DeMorgan NAND

Since Not((A or B)) = (Not A and Not B),
(D = E)

DeMorgan's Inversion Theorem is useful in relay logic when inverting logic without using a relay coil. It is used especially when negative logic is desired for a program but the logic becomes very complicated. It may be easier for the programmer to visualize positive logic and invert the logic rather than working only with negative logic.

The following example shows the inversion principle at work. Notice the piecewise approach to solving the problem left-to-right

Example: Invert the following ladder logic.

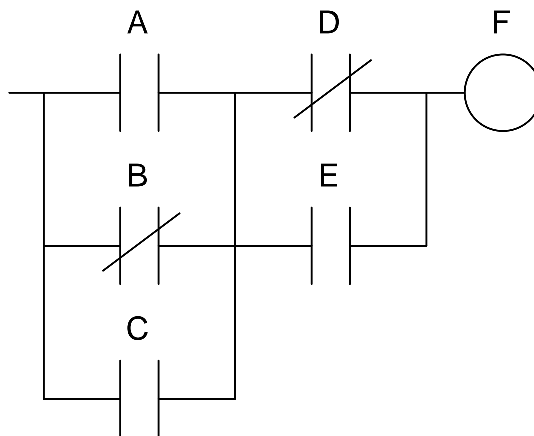


Fig. 5-32a Combination Ladder Diagram

First notice that the circuit can be broken into two separate ‘and’ functions:

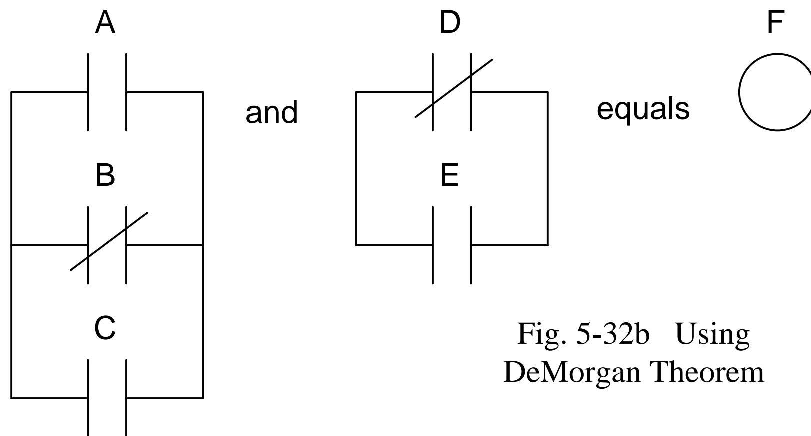


Fig. 5-32b Using DeMorgan Theorem

The negative of the left three contacts is:

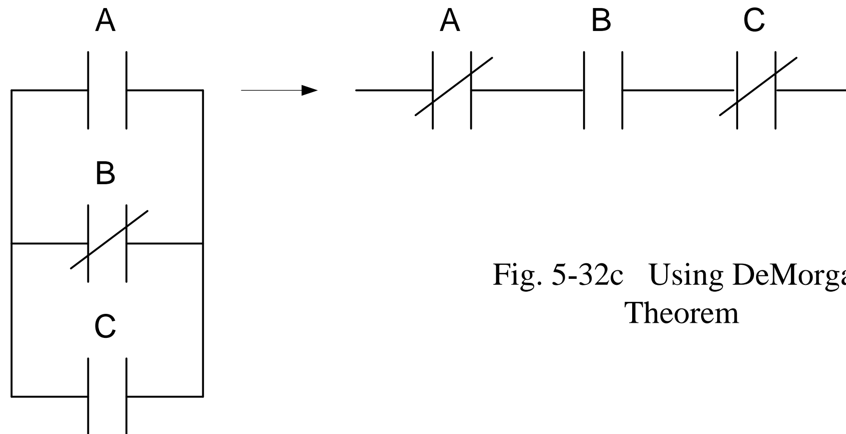


Fig. 5-32c Using DeMorgan Theorem

The negative of the two right contacts is:

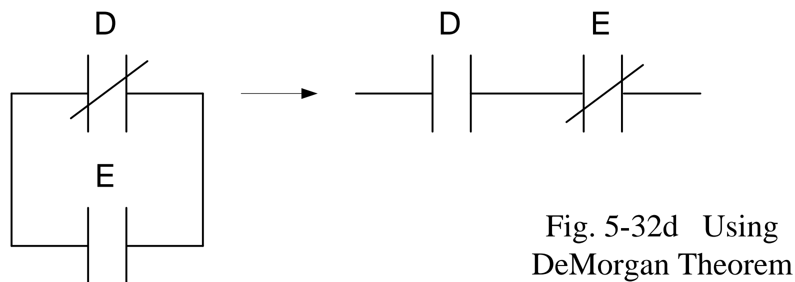


Fig. 5-32d Using DeMorgan Theorem

The two parts are combined as an *or* combination since the original function was *anded* together.

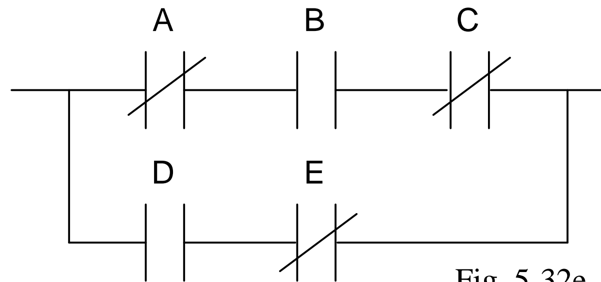


Fig. 5-32e Using DeMorgan Theorem

Thus the inverse of the original function may be found. Function inversion is an important concept to practice and will help give experience writing ladder logic. Of course, the same principles developed in Digital Logic can be applied, that is, convert the Ladder program to Boolean, find the inverse using Boolean DeMorgan rules and then convert back to Ladder. Either approach is acceptable.

Combination Ladder Logic

Examples of branches of ladder logic can be used to practice writing ladder logic. Combinations of normally open and normally closed contacts in various configurations are shown.

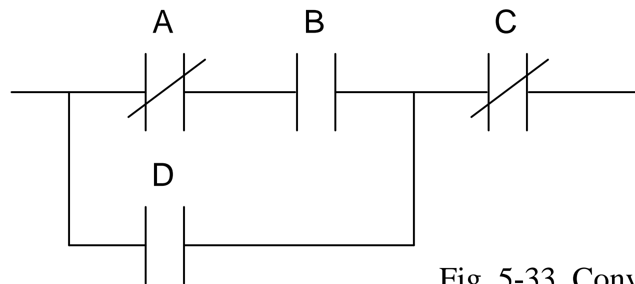


Fig. 5-33 Converting Ladder to Boolean

This is read: ((Not A and B) or D) and (Not C)

Another example of converting Ladder to Boolean:

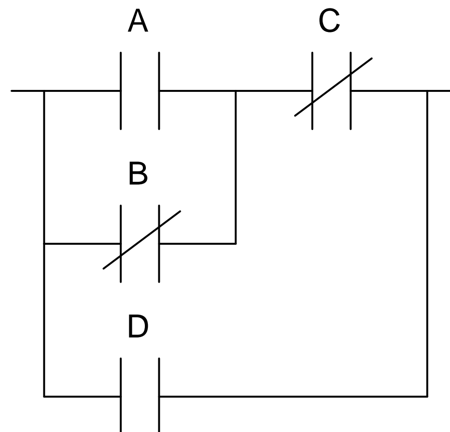


Fig. 5-34 Converting Ladder to Boolean

This is read: ((A or B) and (Not C)) or D:

If A, B, C, and D are used to represent real-life components, it could be read :

((Limit Switch 1 (A) or Limit Switch 2 (B)) and (Not Limit Switch3 (C)) or PushButton 4 (D))

Example: Converting Boolean to ladder logic:

Write the following Boolean statement in ladder logic:

A and (Not B) equal output C

Solution:

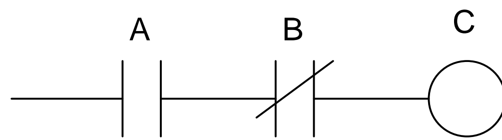


Fig. 5-35 Converting Boolean to Ladder

Example:

Write the following Ladder logic in Boolean:

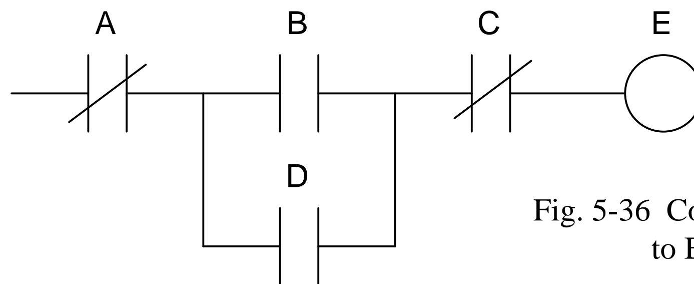


Fig. 5-36 Converting Ladder to Boolean

Solution:

(Not A and (B or D) and Not C) equals output E

Evaluating State Tables

State tables are also an important concept in Boolean Logic. We evaluate the following rung of logic for the state of output G for every state of the inputs A – F.

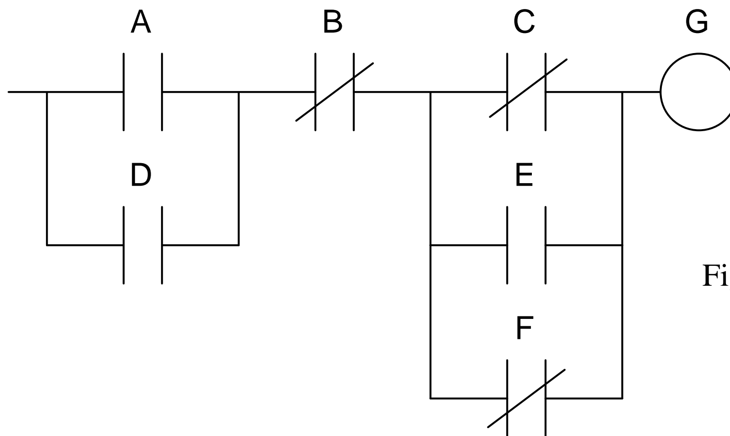


Fig. 5-37 Evaluating Truth or State Table

Since there are six different input elements in this diagram, there are 2^6 or 64 different states for the inputs. The objective is to find the state of the output G for each condition of A through F. Notice that large groups of outputs can be filled in with the knowledge that the output G will be off as long as B is on.

	A	B	C	D	E	F	Output G
0	0	0	0	0	0	0	
1	0	0	0	0	0	1	
2	0	0	0	0	1	0	
3	0	0	0	0	1	1	
4	0	0	0	1	0	0	
5	0	0	0	1	0	1	
6	0	0	0	1	1	0	
7	0	0	0	1	1	1	
8	0	0	1	0	0	0	
9	0	0	1	0	0	1	
10	0	0	1	0	1	0	
11	0	0	1	0	1	1	
12	0	0	1	1	0	0	
13	0	0	1	1	0	1	
14	0	0	1	1	1	0	
15	0	0	1	1	1	1	
16	0	1	0	0	0	0	
17	0	1	0	0	0	1	
18	0	1	0	0	1	0	
19	0	1	0	0	1	1	
20	0	1	0	1	0	0	
21	0	1	0	1	0	1	
22	0	1	0	1	1	0	
23	0	1	0	1	1	1	
24	0	1	1	0	0	0	
25	0	1	1	0	0	1	
26	0	1	1	0	1	0	
27	0	1	1	0	1	1	
28	0	1	1	1	0	0	
29	0	1	1	1	0	1	
30	0	1	1	1	1	0	
…	.	.	.	.	.	.	…
60	1	1	1	1	0	0	
61	1	1	1	1	0	1	
62	1	1	1	1	1	0	
63	1	1	1	1	1	1	

Table 5-5 Example of Truth or State Table

Solution of State Table Row 0:

Starting at row or state 0, all inputs are off:

Solution for row 0:

Since both A and D are 0 and A and D are normally open, the circuit does not conduct through either A or D. If the circuit conducts past either A or D, however, the circuit would work since B would conduct with B = 0 or off and C or F would conduct allowing G to turn on.

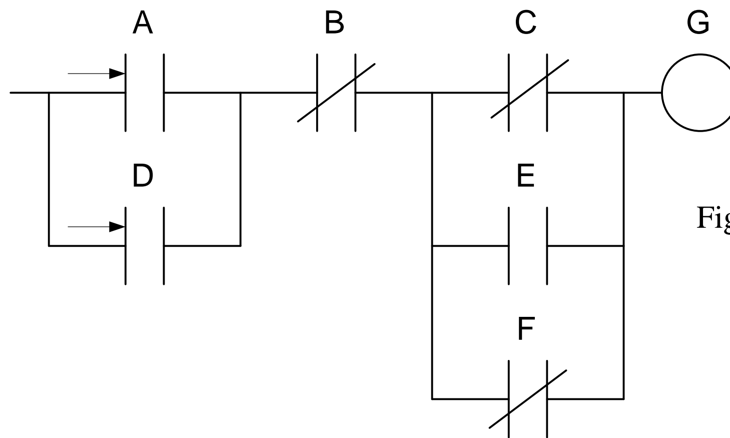


Fig. 5-38 Evaluation of Ladder Rung

	A	B	C	D	E	F	Output G
0	0	0	0	0	0	0	0

As an exercise, fill in all 63 remaining entries of the table.

It is accepted practice that not all states are analyzed, especially when programming the Boolean equation in Ladder or FBD.

Just a few states of a logic rung are usually all that are needed to correctly analyze a circuit.

Save time by **not** constructing a State Table as an analysis tool. It is almost never done in the analysis or troubleshooting of a program.

A Common Programming Error

Early programming efforts may lead to the following error: While all errors cannot possibly be predicted, this one is a common error in many students' early programming efforts. RSLogix 500 is used as the programming language.

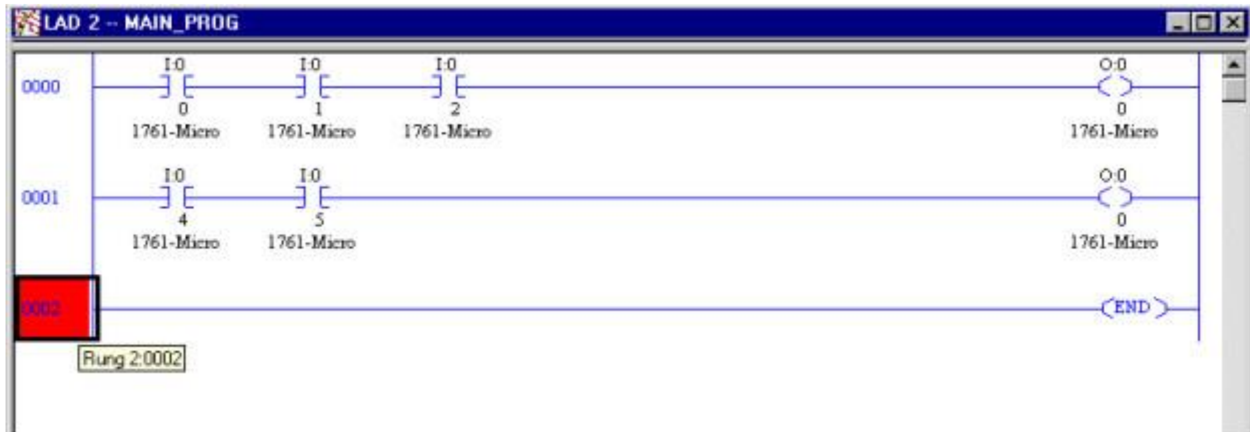


Fig. 5-39 Double Use of Coils

First the programmer enters the first rung to turn on the coil O:0/0. Then later the second rung is also entered to turn on the same output O:0/0. This does not work! The scanning nature of the program will turn on the first output as set by the conditions of rung 0. Then rung 1 is solved and the same output is written over with conditions set by rung 1.

The general rule is: Last coil wins!

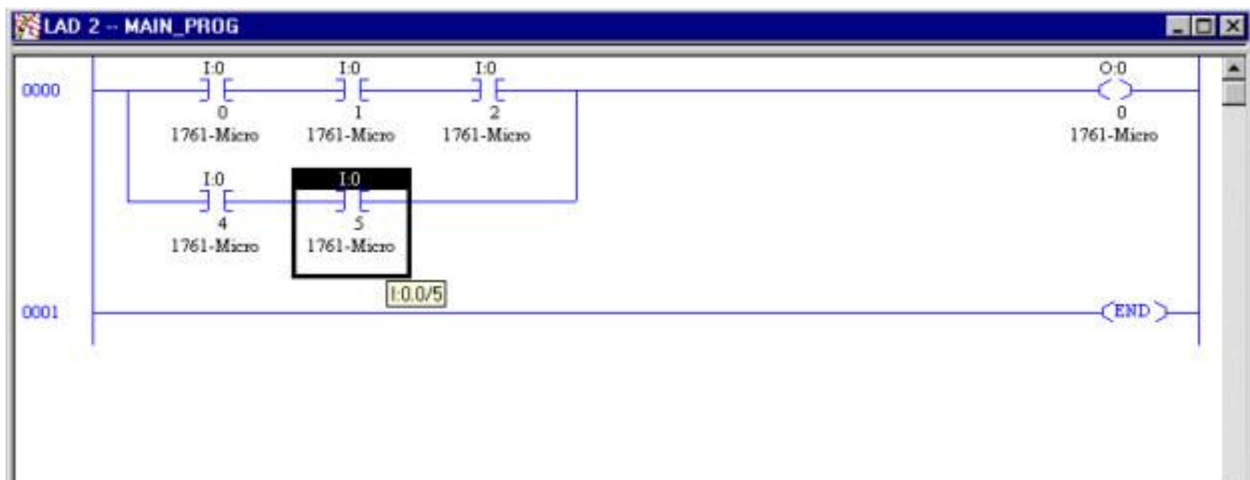


Fig. 5-40 Solution of Double Coil Problem

The combination of the two rungs into one is required to allow both earlier rungs to work properly. The concept of double-use of a coil will be discussed later and exceptions will appear that allow its use. However, be careful! The concept of not double-using a coil is important in all PLC programming including Allen-Bradley as well as Siemens.

An Example of Combination Logic

An example problem will help with development of combinational logic. There are no memory circuits necessary in the problem. Design logic to control the process:

Problem Statement:

A car wash with two bays has a pump supplying water pressure to the spray heads. If both bays are in use or if one bay requires a second set of heads for a tall truck, a second pump is required. The Tall truck request is made via a selector switch for each bay. If both bays are in use with a tall truck in one or both bays, a third pump is required.

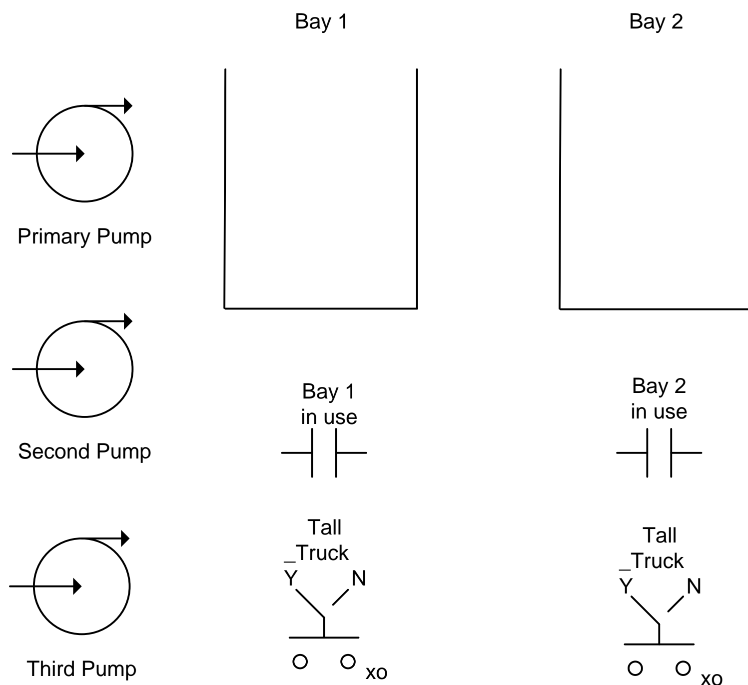


Fig. 5-41 Car Wash

First to be completed are the definition tables:

Definition of Inputs: Table 5-6a

Sensor	Function/State	Signal Assignment
contact	Bay 1 in use	1 (NO)
contact	Bay 2 in use	1 (NO)
selector switch	Tall Truck in Bay 1	1 (NO)
selector switch	Tall Truck in Bay 2	1 (NO)

Definition of Outputs: Table 5-6b

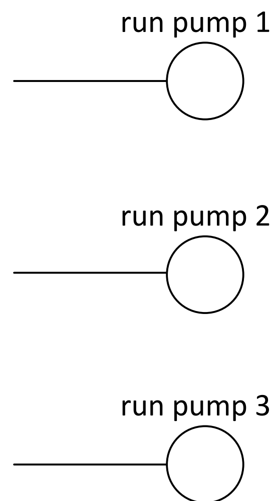
Actuator	Function/State	Signal Assignment
motor starter	run pump 1	1
motor starter	run pump 2	1
motor starter	run pump 3	1

Write Boolean equations for the above:

(left as exercise)

Convert the Boolean equations to Ladder Logic:

First, picture the logic from what needs to be accomplished:

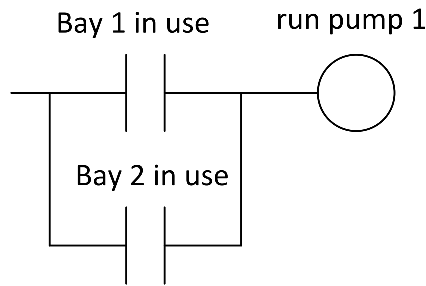


Then begin to formulate what is necessary to turn on each pump. If you can say the function, then writing it is easy.

Say "to run pump 1, either bay 1 is in use or bay 2 is in use". If you agree that this is what is necessary to run the first pump, then picture the function in boolean or ladder:

(Bay 1 in use) + (Bay 2 in use) = run pump 1

Here "+" is read "or". Next convert the Boolean statement to ladder logic as the following:

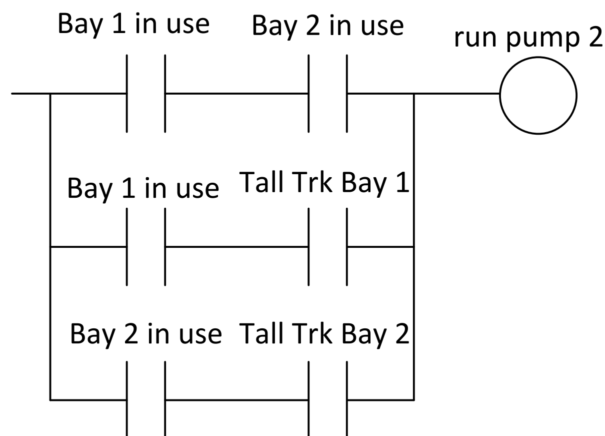


Then, concentrate on pump 2 and describe all conditions necessary to turn on this pump. Go back to the original problem statement to draw from:

"If both bays are in use or if one bay requires a second set of heads for a tall truck, a second pump is required."

Describe this statement in boolean and ladder:

((Bay 1 in use) * (Bay 2 in use)) + (Bay 1 in use)*(Tall truck in bay 1)+ (Bay 2 in use)*(Tall truck in bay 2) = run pump 2



The third pump will be left as an exercise. Each input must be properly identified and addressed as either an input from the wired input list or from the logic developed in the program. A-B and Siemens require the addressing follow their convention for this. Internal logic may use names that are may be the same as the other since a tag may be used with the user's designation. Programs may look very similar with internal logic used since the tags may be the same. It is up to the programmer to choose names that are descriptive for the eventual user to be able to understand quickly the logic.

Summary

This chapter begins the programming process. First, the decision as to an I/O list must be defined. Then various statements are made that define the logic. If Boolean logic is used in the process, the result is a program that eventually is translated to Ladder or FBD to incorporate into the program. Addresses are assigned to the inputs, outputs and internal logic. If bit logic is used, the tags are assigned “bool” as the type. Other types are discussed in later chapters.

The addressing of Siemens S7-1200, A-B SLC (RSLogix 500) and A-B Compact (RSLogix 5000) are discussed. The two that are used in the labs are S7-1200 and A-B Compact. The older SLC architecture is used as a reference for those who may still need to maintain this type of system.

Logic statements are designed in ladder and analyzed. The analysis includes DeMorgan negation. This method is used primarily to familiarize the student with ladder statements and require the student to analyze the ladder statement in a logical manner. The student may first convert the ladder to Boolean, negate the Boolean and then convert back or they may opt to convert the ladder directly. Either approach is accepted.

Other examples show the conversion from Boolean to ladder and from ladder to Boolean. Truth tables are also introduced but not encouraged. All these examples are designed to show the student similarities between the Ladder (and FBD) logic developed in the PLC and the logic from the student’s digital experiences.

This chapter began with an example of a juice maker. While the logic can be developed in this chapter, it will emerge that the logic for the juice maker requires additional understanding of memory circuits to complete the logic for this problem. That logic awaits the student in the next chapter.

Exercises

1. *A car wash with two bays has a pump supplying water pressure to the spray heads. If both bays are in use or if one bay requires a second set of heads for a tall truck, a second pump is required. The Tall truck request is made via a selector switch for each bay. If both bays are in use with a tall truck in one or both bays, a third pump is required.

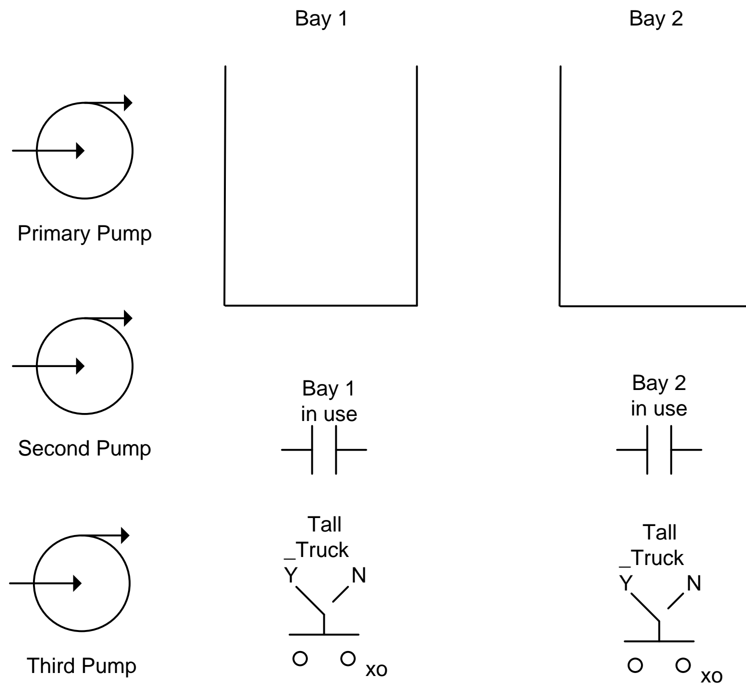


Fig. 5-41 Car Wash

Definition of Inputs: Table 5-6a

Sensor	Function/State	Signal Assignment

Definition of Outputs: Table 5-6b

Actuator	Function/State	Signal Assignment

Write Boolean equations for the above:

Convert the Boolean equations to Ladder Logic:

2. *A design change was noticed by the engineer who saw that the xo on the selector switch for Bay 2's Tall Truck Selector switch was really wired ox. Would this change affect the Input Signal Assignment, Boolean equations, or ladder logic? If so, how?

3. Which pump of problem #1 will be on the most? Is this a good design? Describe how you would change the design if you do not agree that this is a good design.

4. Develop a logic statement in Boolean, FBD and Ladder to turn on an output when switch X and switch Y are energized or when switch Z is energized:

5. Develop a logic statement in Boolean, FBD and Ladder to turn on an output when switch U is on or when <u>only</u> one of the two following is on: V energized, W energized.

6. Develop a logic statement in Boolean, FBD and Ladder to energize the engine start circuit when the key is in the ignition, all (3) front-passenger seat belts are engaged (in which a person is sitting), and all doors are closed (assume a 4-door car).

7. *Use the following ladder rung to answer the following three questions:

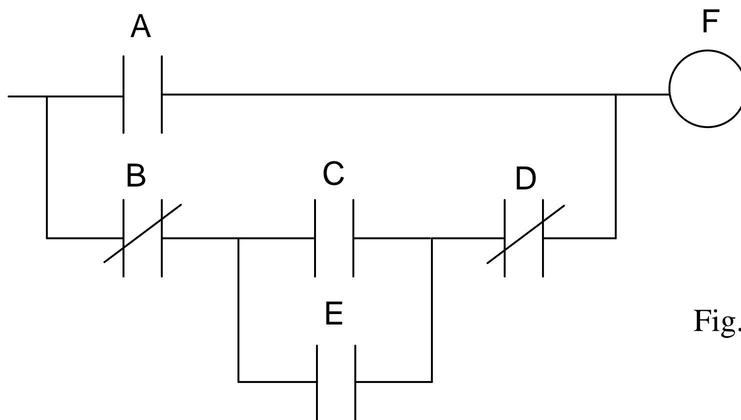


Fig. 5-42

a. Write the DeMorgan of the circuit above using ladder format.
b. Write the Boolean equivalent of the circuit above.
c. Create a truth table and find the state of F for each condition of A-E.

8. *Use the following ladder rung to answer questions a and b of #7, above:

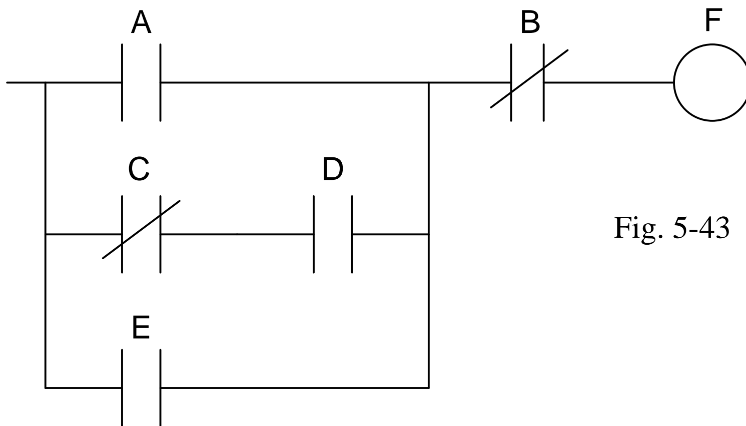


Fig. 5-43

9. *Use the following ladder rung to answer questions a and b of #7, above:

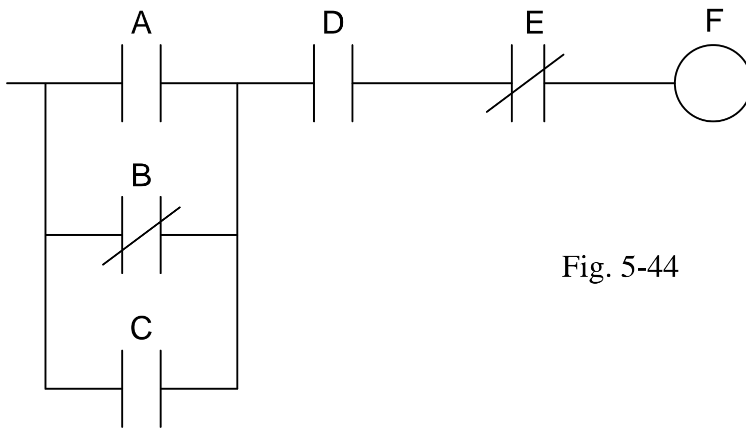


Fig. 5-44

10. *Use the following ladder rung to answer questions a and b of #7 above:

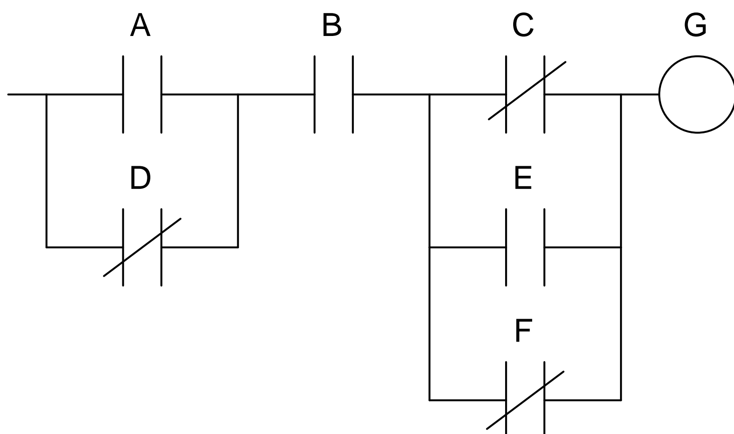


Fig. 5-45

Write as ladder logic the following Boolean expressions:

11. (a and not b and not c) equal output d

12. (a or not b) and (c or not d or not e) equal output f

13. *(a or not b or not c or not d) and (not e or not f) and (not g or h) equal output j

14. (a or b) and c equal output d

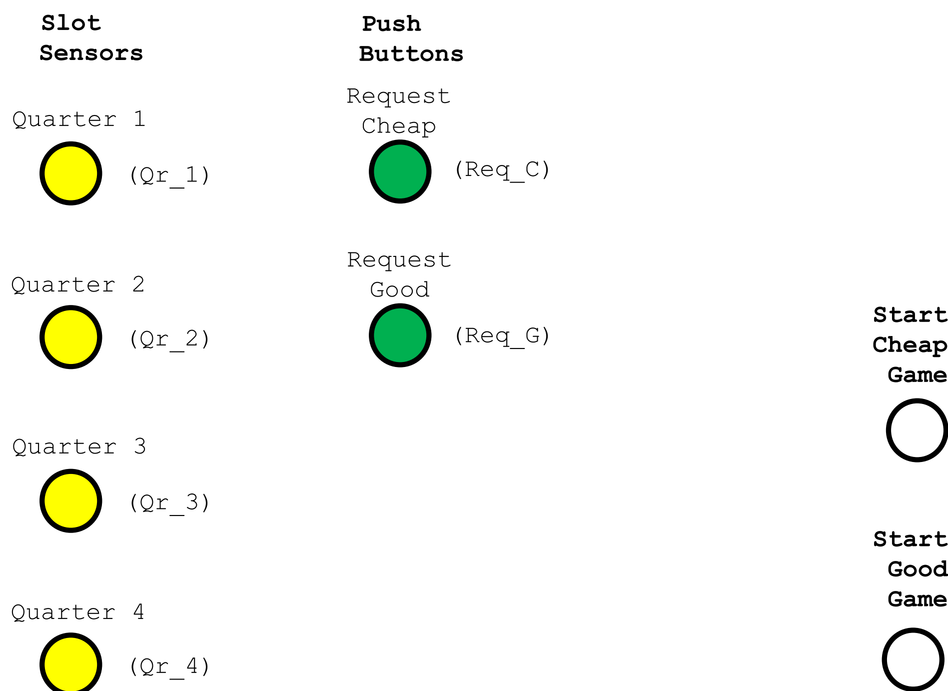
15. Convert the following Allen-Bradley's RSLogix 500's B3: addresses from word/bit format to bit format:

    a. B3:0/2
    b. B3:2/5
    c. B3:10/9
    d. B3:6/15

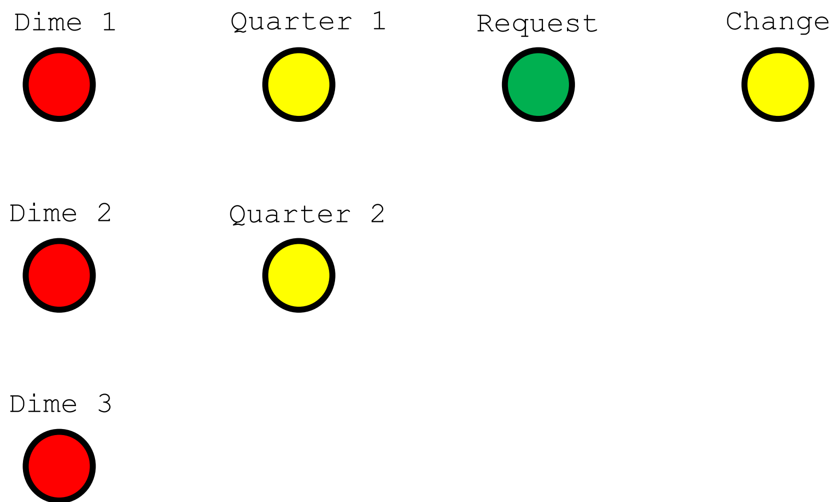
16. Convert the following B: addresses from bit format to word/bit format:

    a. B3/20
    b. B3/8
    c. B3/56
    d. B3/211

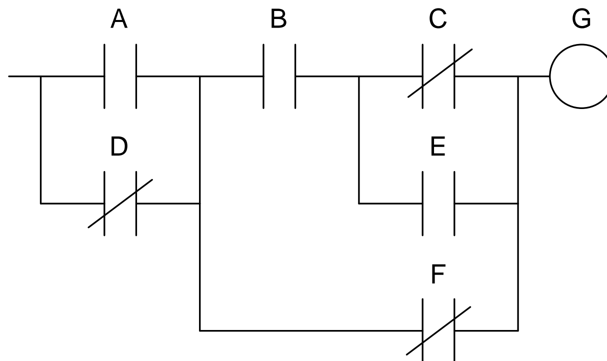
17. *The attached buttons and coin slot sensors are part of an arcade game. Two games are in the same arcade box. One is a cheap game and one is a good game. If the player inserts quarters in any three of the four slots marked quarter 1 through quarter 4, and pushes the **Request Cheap** button, the cheap game starts. If the player puts quarters in all four of the quarter slots and pushes the **Request Good** button, the good game starts. Program rungs to energize a coil for starting the cheap game and a coil for starting the good game. The cheap game does not start if all four quarter slots are filled. Assume all state assignments for the slot sensors and buttons are equal to 1.



18. *Write the logic necessary to turn on the Change light if 55 cents is required for a candy bar and the rules similar to those of the lab are adhered to, that is, that 1, 2 or 3 coins can be 'on' when the request is pushed with the first is on before the second which is on before the third. Be as complete as possible:



19. *Write the DeMorgan of the circuit below using ladder format.



Lab 5.1 The Coin Changer

Implement a program to control a coin changer. A coin changer is built to return change plus dispense a $.35 candy bar. No more than three coins are to ever be used (There is no need to count the number of coins entered). Coins to be used are dimes and quarters. Write a program to accept or reject the sale based on the coins rendered. Coins rendered are checked by inputs *on* using push buttons or selector switches when the Request Candy Bar button is pushed.

Assume dime 2 is not allowed until dime 1 is *on*. Assume dime 3 is not allowed until dime 2 is *on*. That is, dimes enter by filling the slot for dime 1, then dime 2 and finally dime 3.

The same sequence is used for quarters.

Inputs are as follows:

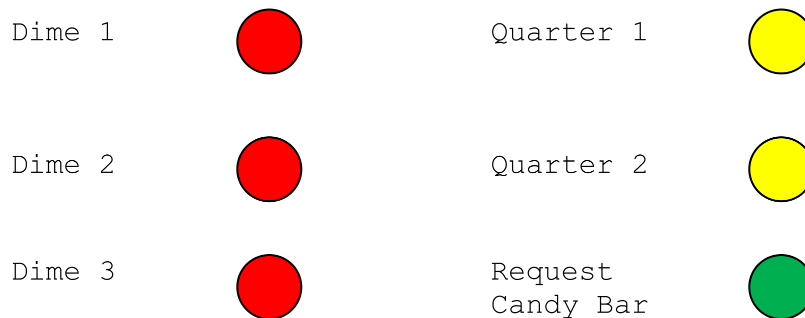


Fig. 5-45 Layout of I/O for Lab

Outputs are as follows:

Accept turns on with the Request Candy Bar input and enough money entered.
Change turns on with the Request Candy Bar input and an excess of money.
Reject turns on with the Request Candy Bar input when no money or not enough money is entered.

Option 1: Change the price to $.45 for the candy bar.

Option 2: Change the price to $.55 for the candy bar. Here 4 coins may be used. (including 1 nickel)

This work is licensed under a Creative Commons Attribution 4.0 International License.