Appendix D - New Special Functions

- AB **ASCII to Binary**
  Convert an sequence of ASCII characters stored in from 1 to 8 Holding Registers into a binary number.

- BM **(Indirect) Block Move**
  Transfers a block of Holding Registers between the Extended Register (XR) memory and Holding Register (HR) memory.

- BU **Byte Unpack**
  Extracts the upper and lower bytes from a table of registers and places these bytes into separate Holding Registers. Used to unpack data from 8 bit analog input modules.

- BZ **Bit Zero**
  Zero's a column of bits from a table of registers.

- CB **Compress Bit**
  Extract a bit column from a table of registers and places these bits their own register(s).

- CX **Checksum**
  Calculate the CRC or LRC over a table of registers.

- ER **Extend Register**
  Allocate memory for Extended Registers (XR).

- LX **Limit Check**
  Verify that an input number is within a upper and lower range.

- PB **Pack Byte**
  Transfers bytes (either high or low byte) from a table of registers into a packed formated (two bytes per register) in another table. Used to format data for use by 8 bit analog output modules.

- SC **Scale**
  Convert a number from one range to another range.

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The ASCII to BINARY "AB" function will convert an ASCII number stored in a table of registers into a binary number. The number may contain imbedded commas, periods (decimal points) and plus (+) and minus (-) signs. This function will typically be used with the ASCII RECEIVE "AR" special function to assist in processing the data received by a serial port.

OP CODE

Op Code 96 defines the Literal (LT) as the AB function. Whether or not the Literal function should be used depends upon the capability of the your program loader. Refer to the introduction and LT function description in the Special Function section for programming details.

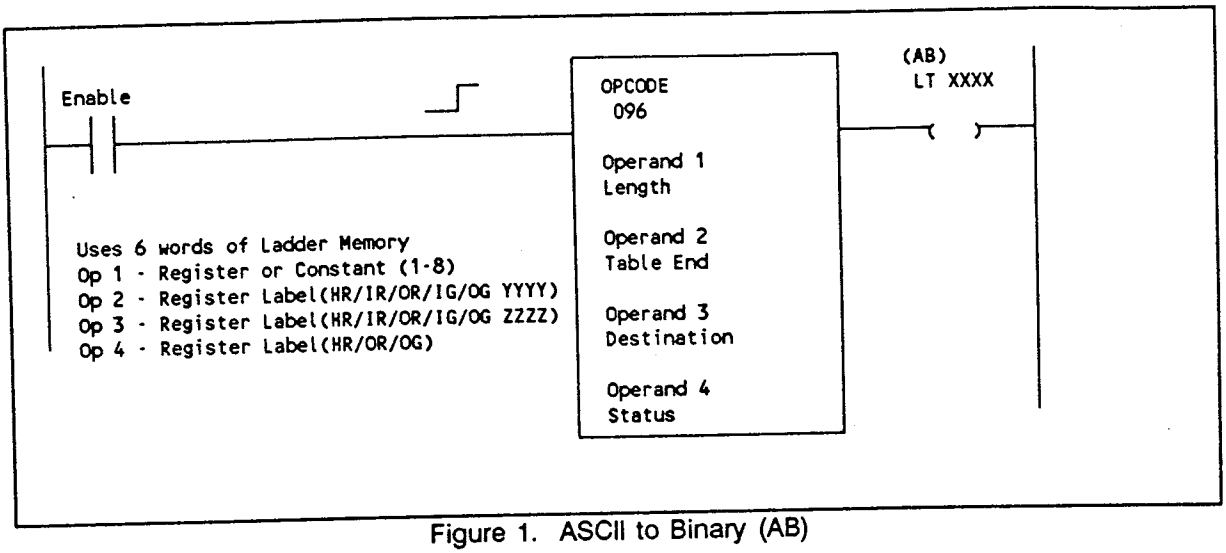


Figure 1. ASCII to Binary (AB)

AB - ASCII to BINARY

SPECIFICATION

OPERAND 1 - Table Length

The AB function only processes one ASCII character per Holding Register, a multi-digit ASCII character (12456, for example) must be stored in multiple Holding Registers. The TABLE LENGTH operand defines how many registers are being used to hold the ASCII characters. Valid number are 1 through 8, therefore the number of ASCII characters in the number can be between 1 and 8. The LENGTH operand can be a constant or it can be obtained from a HR, IR, OR, IG, or OG register. Storing the LENGTH in a register allows this operand to be changed under program control.

OPERAND 2 - Table End

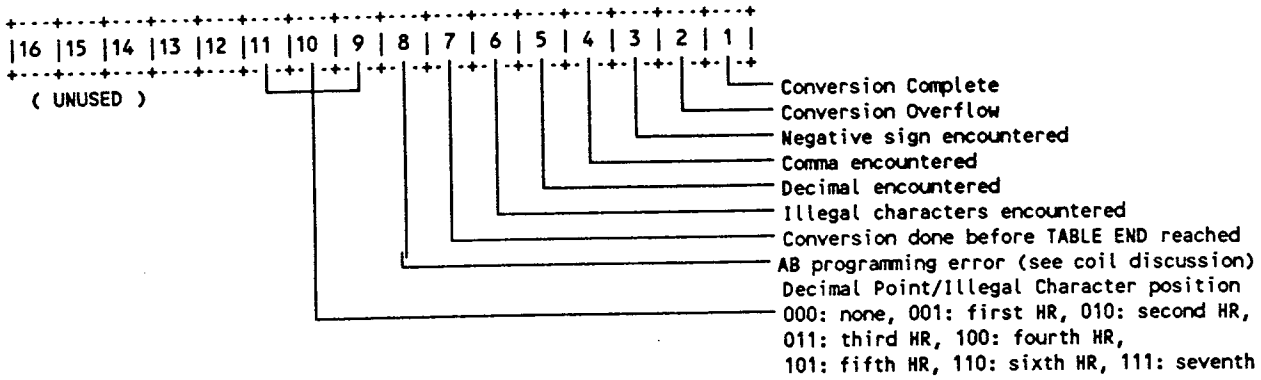
This operand holds a constant or register that contains a number. That number represents a HR reference that will be the last register in a table. That table is where the ASCII characters are stored. Each ASCII character takes one HR. Since the ASCII character is stored in the lower half (byte) of the register, the upper half (byte) of the register is not used. The number used this operand can be a constant value or it can be obtained from a HR, IR, OR, IG or OG register. Storing the TABLE END in a register allows this operand to be changed under program control.

OPERAND 3 - Destination

The result of the ASCII to Binary function is a binary number. This binary number is always placed in a Holding Register. The DESTINATION operand is a number of the Holding Register that is to receive the converted binary number. This number can be programmed as a constant or it can be obtained from a HR, IR, OR, IG or OG register. Storing the DESTINATION in a register allows this operand to be changed under program control.

OPERAND 4 - Status

The AB function informs the program if it was successful in performing the conversion. If the function failed, one or more status bits are set in a STATUS register. These status bits will help describe why the conversion was unsuccessful. The status register can also help troubleshoot where the offending ASCII character is located in the table of registers.



If a comma is encountered in the conversion, a status bit (bit 4) is set, but the conversion proceeds normally. A comma counts as a Holding Register when computing the TABLE LENGTH.

ENABLE CIRCUIT

An OFF to ON transition will cause the AB function to convert the ASCII characters found in the *low bytes* of the table defined by the TABLE END and TABLE LENGTH parameters and place the result in the Holding Register defined by DESTINATION. The STATUS register is updated to indicate whether the conversion was successful or if the conversion was terminated due to some problem.

COIL

The coil will turn on if the enable contact is energized and the conversion completes normally. The coil will remain off if any of the following occur:

- Enable contact not conducting
- Table Length > Table End
- Table End > Highest Holding Register Used
- Destination > Highest Holding Register Used
- Table Length < 1
- Table Length > 8
- Status programmed as a constant value
- Converted number exceeds 65535 (conversion overflow)
- Five characters are converted before the last register in table is encountered
- Illegal ASCII characters are found

ASCII Character	Hex Representation
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H
+	2BH
,	2CH
-	2DH
.	2EH

All other characters are considered invalid.

Figure 1 - Legal ASCII characters for AB

AB - ASCII to BINARY

Operation

When the AB function is enabled via an OFF to ON transition, the low byte of the first register of the table will be converted to binary (assuming that the byte is a valid ASCII character. Table 1 gives the valid ASCII characters). This converted binary value is stored in a temporary location. The AB function then reads and converts the value found in the low byte of the next higher register. This new value is added to 10 times the value stored in the temporary location and the result is stored in the same temporary location. The AB function then reads and converts the value found in the low byte of the next higher register. This converted value is added to 10 times the value in the temporary location. This sum is then stored back into the temporary location. The process continues until either all characters have been converted, or an illegal character is encountered. If an illegal character is encountered, a bit in the status register is set and a pointer to the offending HR reference is written into three bits of the same status register. Note that the least significant digit of the ASCII value is stored in the higher register numbers of the TABLE, while the most significant digit of the ASCII value is stored in the lower register numbers of the TABLE.

Example

Enter these values into an AB function (use LT with opcode 96 on your program loader)

Length	HR0001	HR0001	= 00006
Table End	HR0002	HR0002	= 00105
Destination	HR0003	HR0003	= 00106
Status	HR0004		

HR0100	- 0031h	(ASCII 1)
HR0101	- 002Ch	(ASCII ,)
HR0102	- 0032h	(ASCII 2)
HR0103	- 0033h	(ASCII 3)
HR0104	- 0034h	(ASCII 4)
HR0105	- 0035h	(ASCII 5)

When the ENABLE line is toggled from OFF to ON, the converted binary value should be written into HR0106 and the status register (HR0004) should show the first bit (conversion complete) and the fourth bit (comma encountered) as being set. The AB function coil should be energized to indicate successful completion.

HR0106	- 3939h (Decimal 12345)
HR0004	- 00000000 00001001

Figure 2 - AB Example

Applications

Frequently, the PC1200 serial port is connected to an external device (bar code reader, mass flow meter, ultrasonic rangers, weigh scales, etc.) that transmits a stream of data to the PLC in ASCII format. If the ASCII information contains numeric information, and the PLC is to perform math on that numeric information, then the ASCII numbers must be converted to binary.

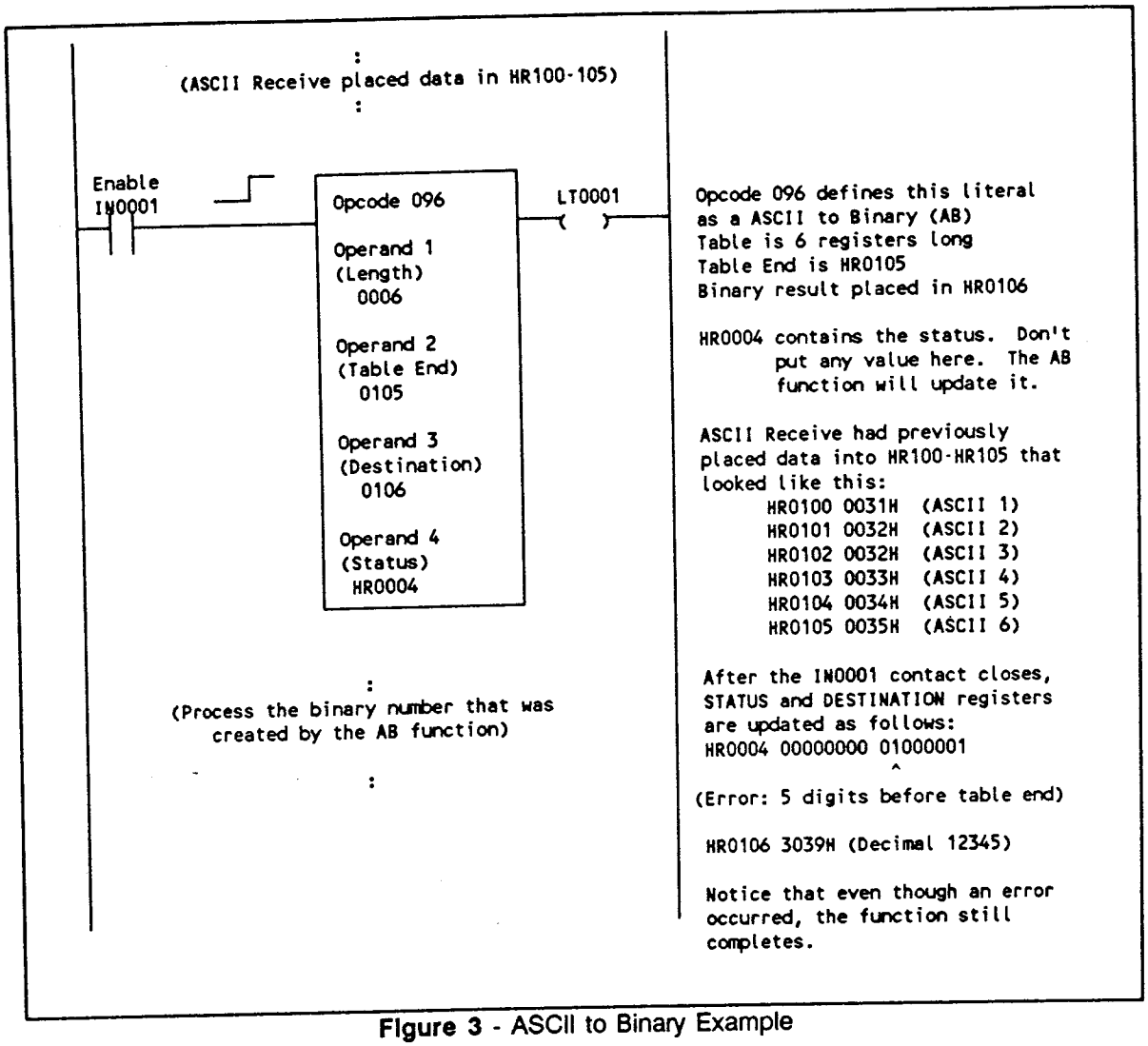


Figure 3 - ASCII to Binary Example

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Indirect Block Move "BM" special function is a more flexible version of the Block Transfer "BT" special function. It provides a mechanism to define several tables indirectly and transfer the contents in one of two ways. One transfer mode copies a source table to a destination table, while the second mode involves two source tables in a chained exchanged. The indirect definition of the tables allows for redefinition from other special functions during the program execution and provides access to the indirect Holding Register memory defined by the Extend Register special function. Only Holding Register memory is exchanged with this function.

OP CODE

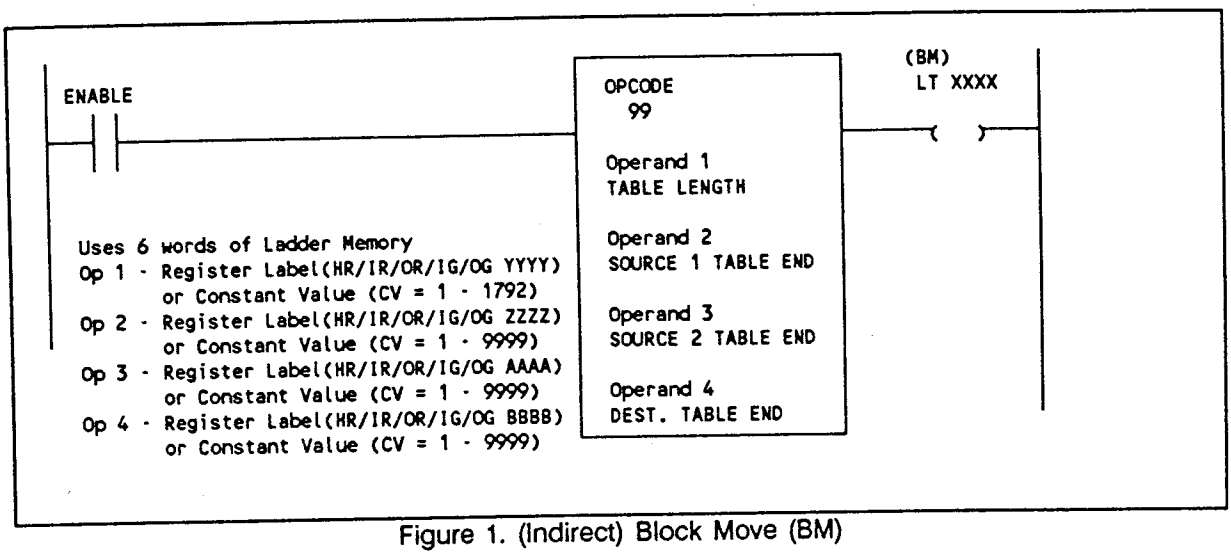


Figure 1. (Indirect) Block Move (BM)

BM - (INDIRECT) BLOCK MOVE

SPECIFICATION

OPERAND 1 - TABLE LENGTH

The TABLE LENGTH defines the length of the Source or Destination Tables indirectly through the contents of a register or directly with a Constant Value. Allowable table lengths range from 1 to 1792 in magnitude. Operand 1 will be considered invalid if outside of this range.

OPERAND 2 - SOURCE TABLE 1 END

The SOURCE TABLE 1 END defines the end of Holding Register Source Table #1 indirectly with the contents of a register or directly with a Constant Value. The absolute range of this operand is between 1 and 9999. In addition the value for this operand must be greater or equal to the TABLE LENGTH and must be less than or equal to the Highest Holding Register Used (HHRU). Operand 2 will be considered invalid if these conditions are not satisfied.

OPERAND 3 - SOURCE TABLE 2 END

The SOURCE TABLE 2 END defines the end of Holding Register Source Table #2 indirectly with the contents of a register or directly with a Constant Value. The absolute range of this operand is between 1 and 9999. In addition the value for this operand must be greater or equal to the TABLE LENGTH and must be less than or equal to the Highest Holding Register Used (HHRU). Operand 3 will be considered invalid if these conditions are not satisfied.

OPERAND 4 - DESTINATION TABLE END

The DESTINATION TABLE END defines the end of the Holding Register Destination Table indirectly with the contents of a register or directly with a Constant Value. The absolute range of this operand is restricted between 1 and 9999. In addition the value for this operand must be greater or equal to the TABLE LENGTH and must be less than or equal to the Highest Holding Register Used (HHRU). Operand 4 will be considered invalid if these conditions are not satisfied.

ENABLE CIRCUIT

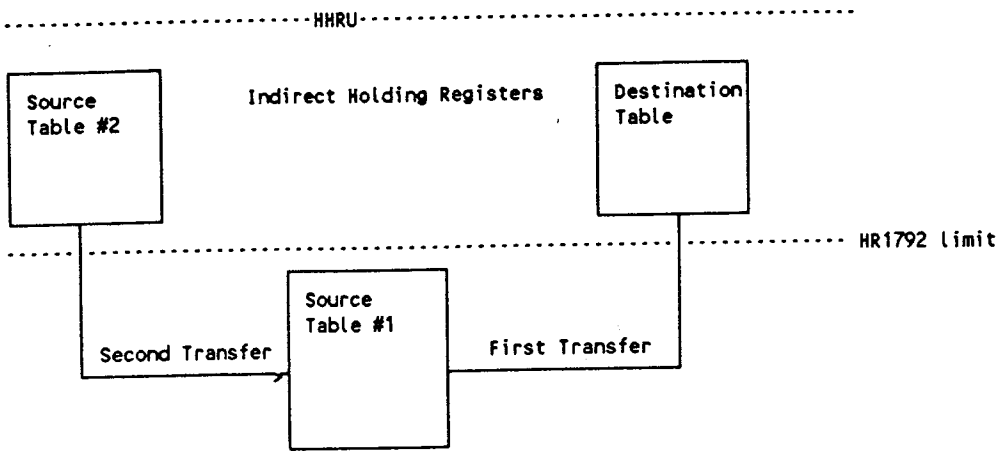
When the BM ENABLE CIRCUIT conducts, the operands will be checked for valid range and if ok the transfer will occur.

COIL

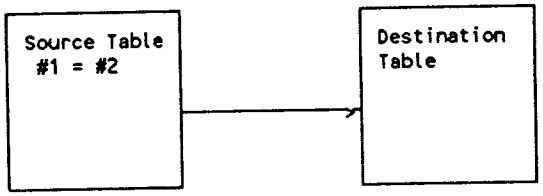
The output coil is used to indicate programming errors. On any scan that the ENABLE CIRCUIT solves true and at least one of the operands is invalid, the output coil will turn on and the tables will remain unmodified.

Operation

On each scan that the ENABLE CIRCUIT solves true and all operands are valid, the contents of Source Table #1 is copied to the Destination Table. After this first transfer, Source Table #2 is copied to the block that originally held Source Table #1. The primary intent of the three tables exchange is to provide a mechanism in exchanging memory with the indirect Holding Register space.



If the two source tables are defined equal where SOURCE TABLE 1 END = SOURCE TABLE 2 END, then only one transfer occurs from the combined Source Table to the Destination Table.



In order to create a "Table Swap", two BM functions are required. See following application section. Defining the DESTINATION TABLE END equal to one of the SOURCE TABLE 1 END or SOURCE TABLE 2 END is unsupported and will give unpredictable results. Special checks are made so that if the destination table overlaps the source table the data will be read from the source before it is overwritten.

The BM special function can handle large tables, up to 1792 long, in short time periods by using a specialized Direct Memory Access (DMA) function of the CPU. 1.8 milliseconds are required for a one way transfer of 1792 Holding Registers, 3.6 milliseconds for a two way transfer. If source power to the PLC were to fail in the middle of a BM transfer the operation will complete. This will prevent block tearing, however the user should have some means of labeling the block to keep track of which block is which. In the remote case that the DMA function were to fail either during power down or normal BM operation, bit 9 of the Fault Register will be set to indicate hardware failure.

The BM function is active every scan that the ENABLE CIRCUIT solves true. This means that the BM function should be controlled on a scan-by-scan basis, otherwise tables may be unexpectedly overwritten in the dual table exchange mode. For example, after three scans with the ENABLE CIRCUIT conducting, all three tables will contain the contents of Source Table #2 if no other action is taken.

BM - (INDIRECT) BLOCK MOVE

Applications

Three examples will illustrate the capability of the BM special Function. Single block transfer example. Results from calculations exist in HR0041 to HR0050, application requires ten running samples space by one second intervals to be kept in HRs 101 to 200.

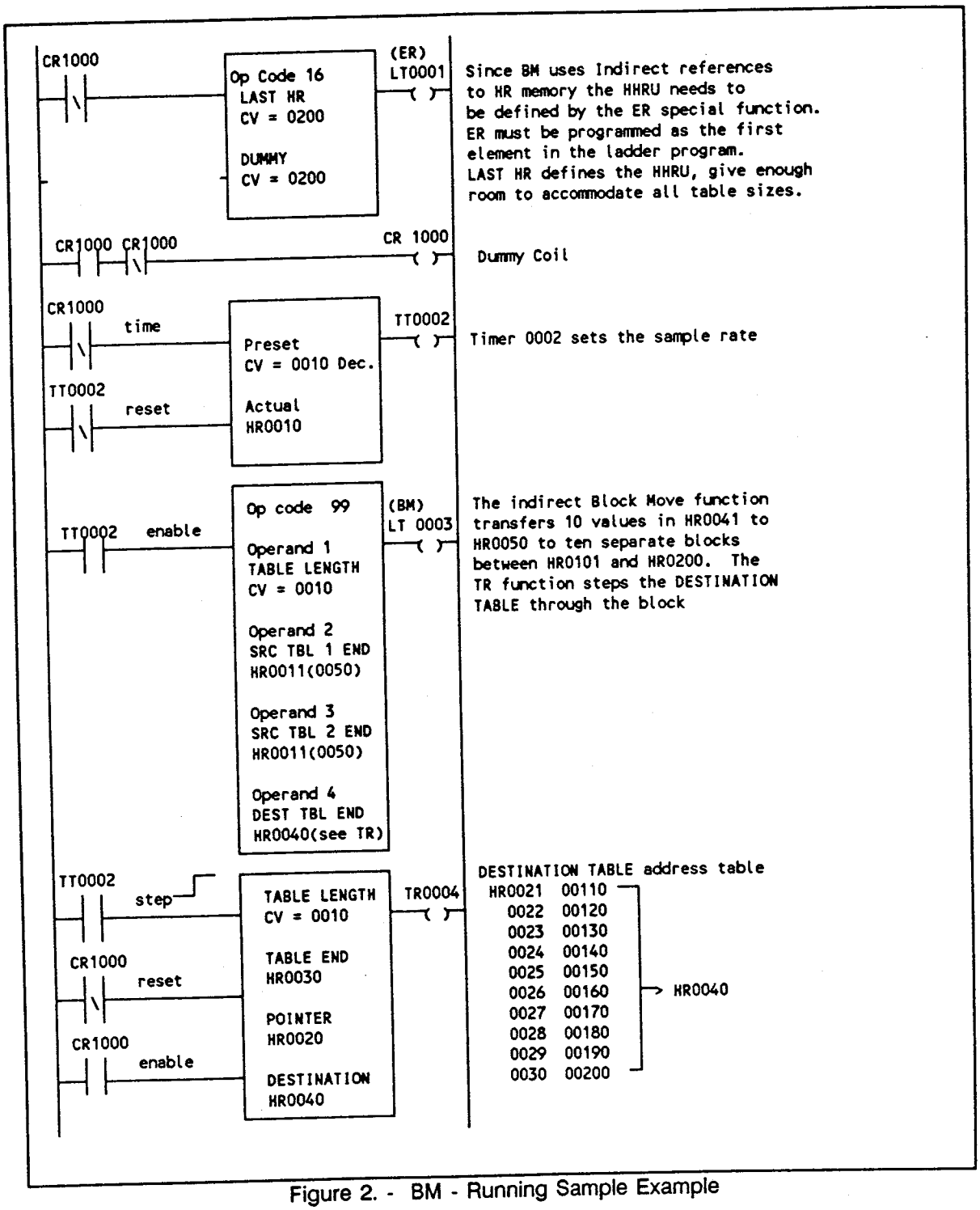


Figure 2. - BM - Running Sample Example

Applications Cont.

Double and Single block transfer example. Based on IN0001, 200 registers are exchanged across the 1792 Holding Register boundary.

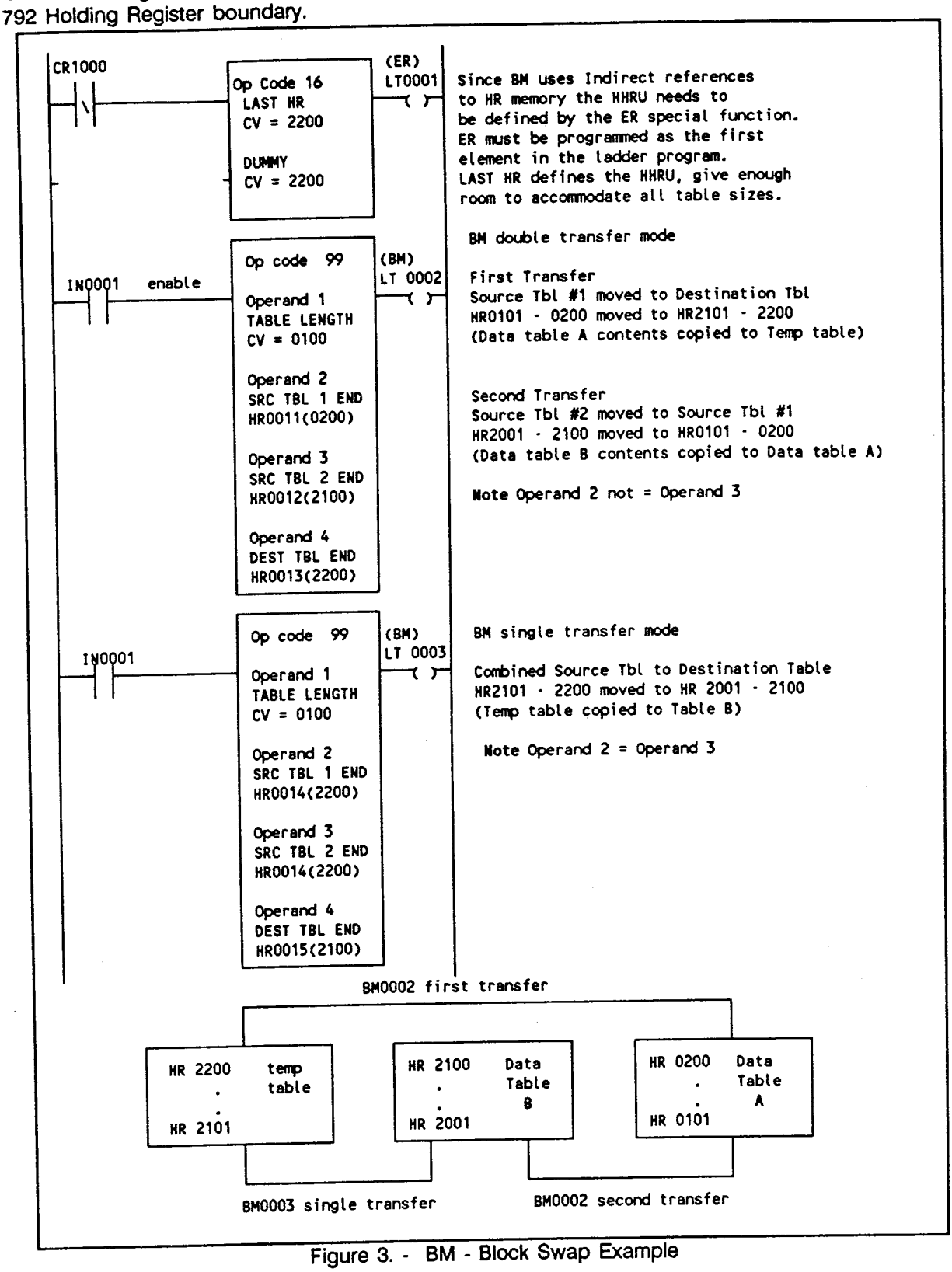


Figure 3. - BM - Block Swap Example

BM - (INDIRECT) BLOCK MOVE

Applications Cont.

A carwash is broken into 500 zones represented by a 500 registers table. 16 car wash options brought in from Input Group one need to be moved through the table register-by-register when pulses on IN0017 indicate the advance of the car to be washed. Bit picks of the registers in the table indicates when to turn on the selected options. Several N-bit Shift Registers could be chained together to strobe the information through the table, however a cleaner implementation would be to use the BM function with Source and Destination overlapped with an offset of one register. Because the table location is static in memory all operand references will use Constant Values.

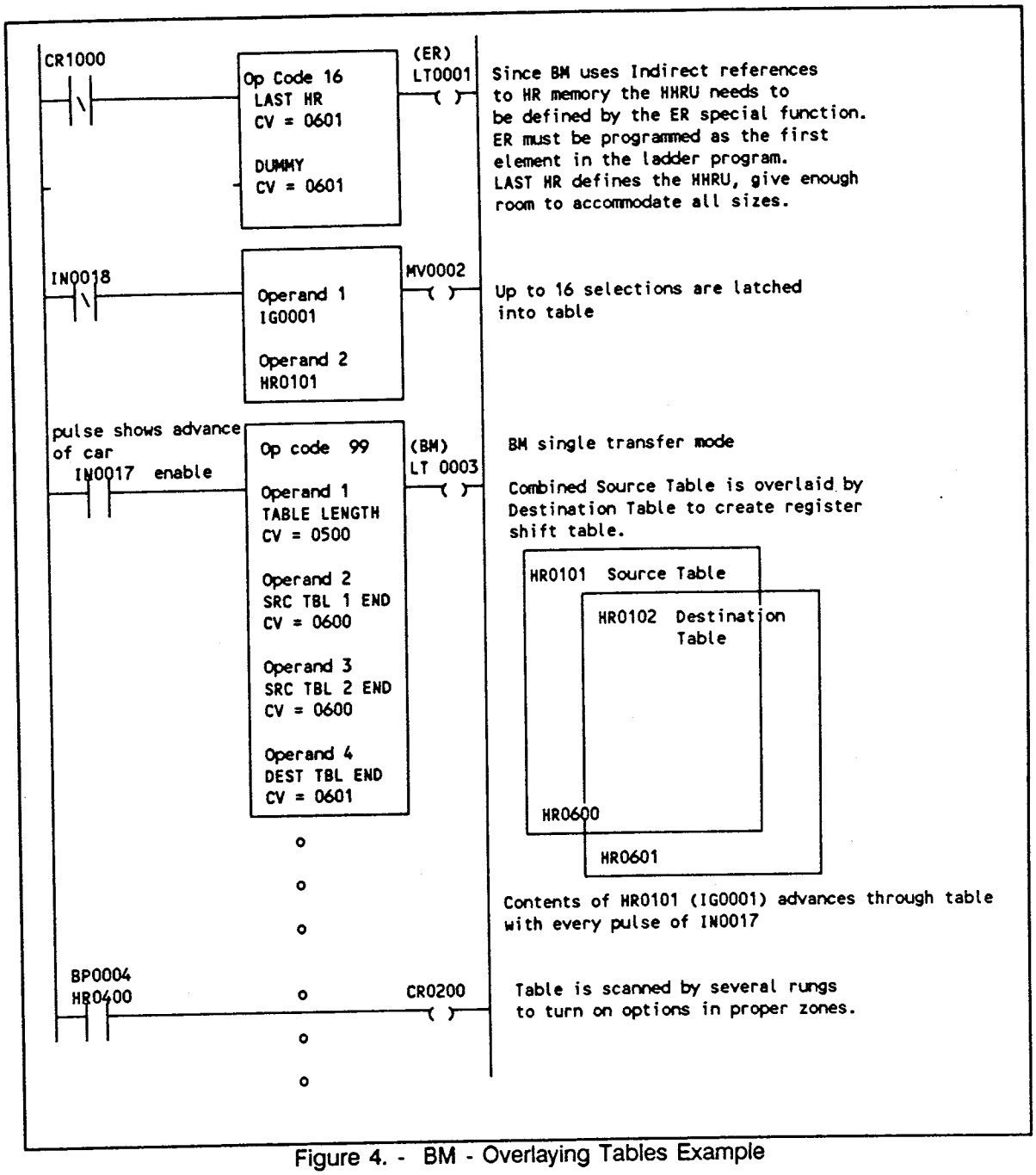


Figure 4. - BM - Overlaying Tables Example

Applications Cont.

The most powerful implementation of the BM special function is to create a Holding Register bank switching scheme. Two or more sections of the ladder program can re-use the same block of Holding Registers for reference purposes but operate on an offset table of registers transferred in with the BM special function. This example defines the following memory segments:

HR0001 - HR1000 - Common shared HR memory space = table A
HR1001 - HR1792 - Unshared direct HR memory *
HR1793 - HR2000 - Spare indirect memory area
HR2001 - HR3000 - HRs for first half of program = table B
HR3001 - HR4000 - HRs for second half of program = table C

Tables B and C will contain identification tag words, HR2001 contains 00001 and HR3001 contains 00002. These ID tags are used to insure proper table position on transition into run. A power down in the middle of the program execution could leave tables swapped if this precaution were not taken. The following is a flow chart showing a bank switch scenario.

Set HHRU = HR4000 with ER function

Test for current block
HR0001(XXXX) = 00001 ?
IF NO perform single transfer BM
Source #1 = table B
Source #2 = table B
Destination = table A

Perform First half of program using values from HR2001 to HR3000. Ladder is programmed with operand references HR0001 - HR1000.

Perform Bank switch of tables B and C with A using a Double transfer form of BM
Source #1 = table A
Source #2 = table C
Destination = table B

Perform Second half of program using values from HR3001 to HR4000. Ladder is programmed with operand references HR0001 - HR1000.

Perform Bank switch of table B and C with A, to save latest block and prepare for next ladder scan, using a double transfer form of BM.
Source #1 = table A
Source #2 = table B
Destination = table C

End of Ladder - Repeat

* All overhead functions including the BM special functions should define operands with the unshared HR memory HR1001 to HR1792.

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Byte Unpack "BU" special function is a wholesale extension of the Move Byte special function acting to separate byte information from a contiguous table of registers or groups. Four "Unpacking" modes are selected through the CONFIGURATION operand to define how bytes in the Source Table will be split and copied to the Destination Table. The Source Table is split byte wise and copied low byte first, then high byte, to the Destination Table one byte per register. Therefore, the destination is always twice as long as the source. The Pack Byte is the logical inverse of the Byte Unpack special function.

OP CODE

Op Code 101 defines the Literal (LT) as the BU function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the Introduction and LT function description in the Special Function section for LT programming details.

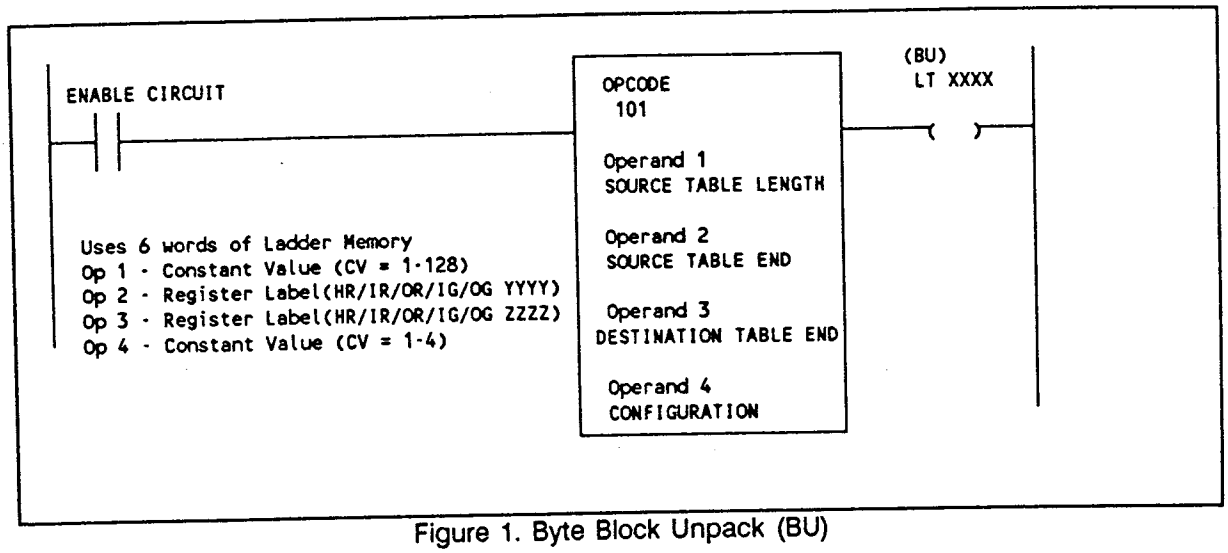


Figure 1. Byte Block Unpack (BU)

SPECIFICATION

OPERAND 1 - SOURCE TABLE LENGTH

The SOURCE TABLE LENGTH is a Constant Value that defines the number of registers in the Source Table. This parameter may not be less than 1 or greater than 128. Indirectly it defines the Destination Table Length as 2 X the SOURCE TABLE LENGTH (destination maximum length = 256). Operand 1 will be considered invalid if it is not a constant or if it is outside of the accepted range

BU - BYTE UNPACK

OPERAND 2 - SOURCE TABLE END

The SOURCE TABLE END defines the type and address number of the last register in the Source Table from which information is being copied. This operand may be an HR, IR, OR, IG, or OG. Address labels may not be less than the SOURCE TABLE LENGTH, in magnitude, or greater than the highest reference allowed for the type chosen. Operand 2 will be considered invalid if it violates the address restrictions or if it is programmed as a constant.

OPERAND 3 - DESTINATION TABLE END

The DESTINATION TABLE END defines the type and address of the last register in the table where information is being copied to. It is specified by an HR, IR, OR, IG or OG. Address values may not be less than SOURCE TABLE LENGTH * 2, in magnitude, or greater than the highest reference allowed for the type chosen. Operand 3 will be invalid if it violates the address restrictions or if it is programmed as a constant.

The Source and Destination Table should not overlap. No checks are made to detect overlapping tables and no measures are taken to avoid writing over the Source Table before it is read.

OPERAND 4 - CONFIGURATION

The CONFIGURATION, programmed as a Constant Value 1 - 4, specifies the manner in which the Byte Unpack operation will be performed. Values outside of this range or programmed other than a constant will be considered invalid. The definition of the CONFIGURATION selection is given in table 1.

CONFIGURATION Number	Half of destination used	Action on unused half
1	Low Byte	High Zeroed
2	Low Byte	High Untouched
3	High Byte	Low Zeroed
4	High Byte	Low Untouched

Table 1. CONFIGURATION Definition

ENABLE CIRCUIT

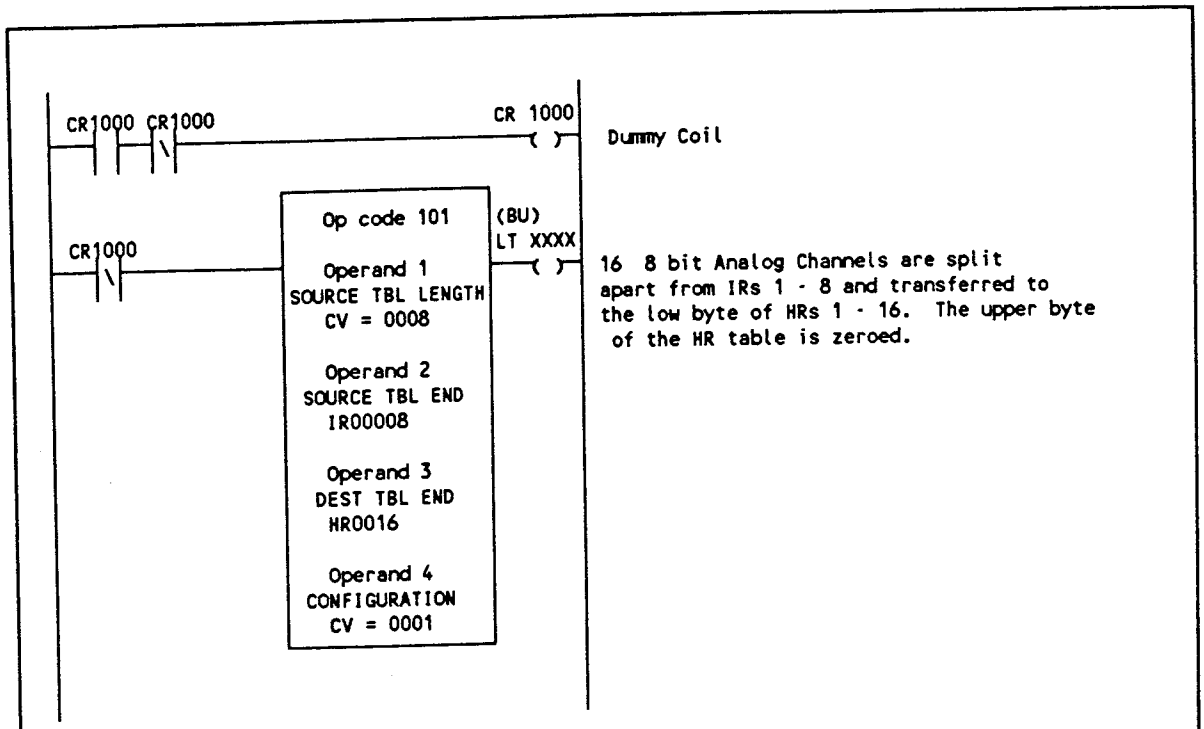
When the ENABLE CIRCUIT conducts, data is copied from the Source Table to the Destination Table according to the CONFIGURATION.

COIL

The output coil is used to indicate programming errors. On any scan that the ENABLE CIRCUIT solves true, and at least one of the operands is invalid, the output coil will turn on and the Destination Table is not modified. If operands are valid the coil will remain off.

Applications

Typical applications include: Splitting the stacked eight bit analog channels from input registers, Splitting ASCII characters for easier manipulation, or any application which treats byte information in a wholesale fashion. Figure 2 shows an example where 16 eight bit Analog channels are brought in through IR 1-8 in a stacked mode and then broken apart with the BU special function into a HR table. This example could be expanded to include all 128 IRs.



The information will transfer as follows (contents shown in Hex for byte separation) :

	Source Table High	Low		Destination Table High	Low
IR0001	01	02	HR0001	00	02
	03	04		00	01
	05	06		00	04
	07	08		00	03
	11	12		00	06
	13	14		00	05
	15	16		00	08
IR0008	17	18		00	07
				00	12
				00	11
				00	14
				00	13
				00	16
				00	15
				00	18
			HR0016	00	17

Figure 2. - BU Analog channel example

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Bit Zero "BZ" special function is a wholesale extension of the Bit Clear coil. A column of bits from the defined table are zeroed when ever the enable circuit is true. The intent of the function is to clear flags such as Out-of-Range or negative sign, once acknowledged. The CB function becomes a natural prefix to the BZ function for Input Register manipulation.

OP CODE

Op Code 53 defines the Literal (LT) as the CB function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the introduction and LT function description in the Special Function section for LT programming details.

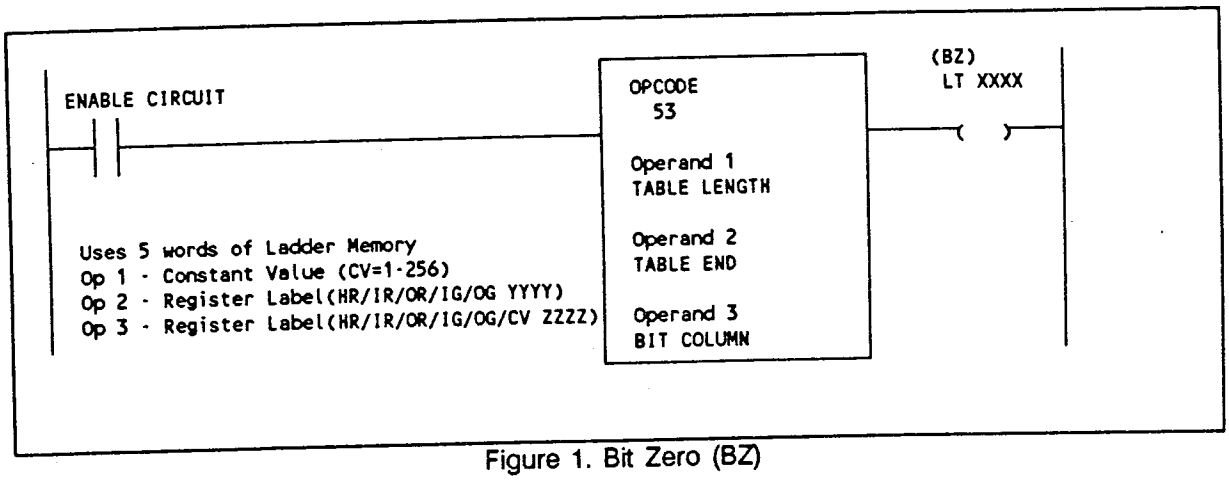


Figure 1. Bit Zero (BZ)

BIT ZERO - BZ

SPECIFICATION

OPERAND 1 - TABLE LENGTH

The TABLE LENGTH, along with the TABLE END, define a table in PLC memory in which a BIT COLUMN will be zeroed. The TABLE LENGTH is defined by a Constant Value between 1 and 256. Operand 1 will be invalid if it is not in this range or programmed as a constant.

OPERAND 2 - TABLE END

The TABLE END defines the type and address of the last register in the table in which the bit information is to be zeroed. This operand may be programmed as an HR, IR, OR, IG or OG. Address labels may not be less than the TABLE LENGTH, in magnitude, or greater than the highest reference allowed for the type chosen. Operand 2 will be considered invalid if it violates the address restriction or if it is programmed as a constant.

OPERAND 3 - BIT COLUMN

The BIT COLUMN, entered as a Constant Value 1 - 16, specifies which bit position out of each register is to be zeroed. Values outside of this range or programmed other that a constant will be considered invalid.

ENABLE CIRCUIT

When the BZ ENABLE CIRCUIT conducts, bit data is zeroed in the defined table according to the Bit Column selection.

COIL

The output coil is used to indicate programming errors. On any scan that the ENABLE CIRCUIT solves true and at least one of the operands is invalid, the output coil will turn on and the table is not modified. If operands are valid the coil remains off.

BZ Operation

Table 1 shows the action of the BZ special function with before and after table status.

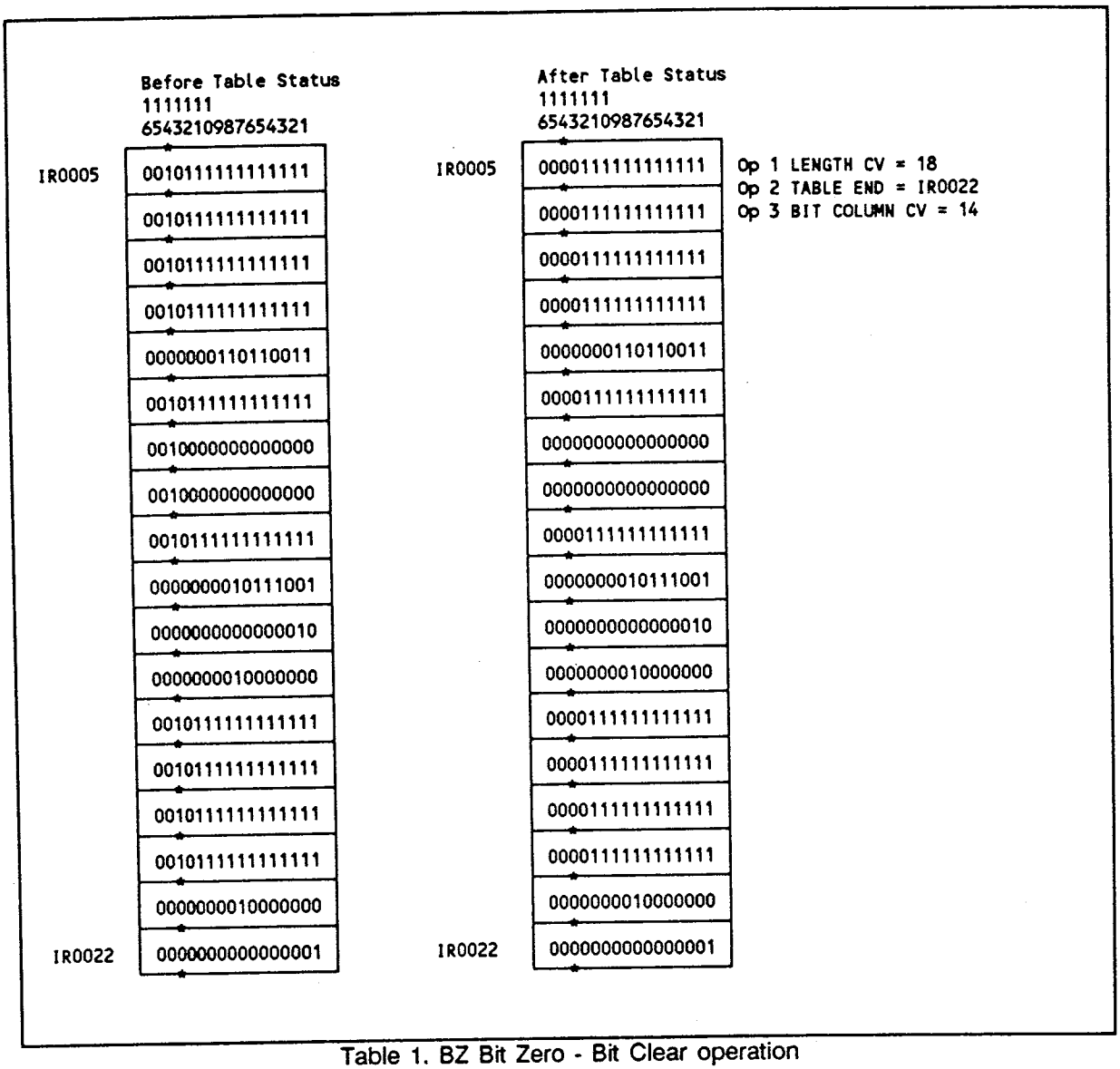


Table 1. BZ Bit Zero - Bit Clear operation

BIT ZERO - BZ

Applications

After the CB special function captures the "Out-of-Range" flags from 18 IRs they can be removed with the BZ special function so as not to effect the converted value.

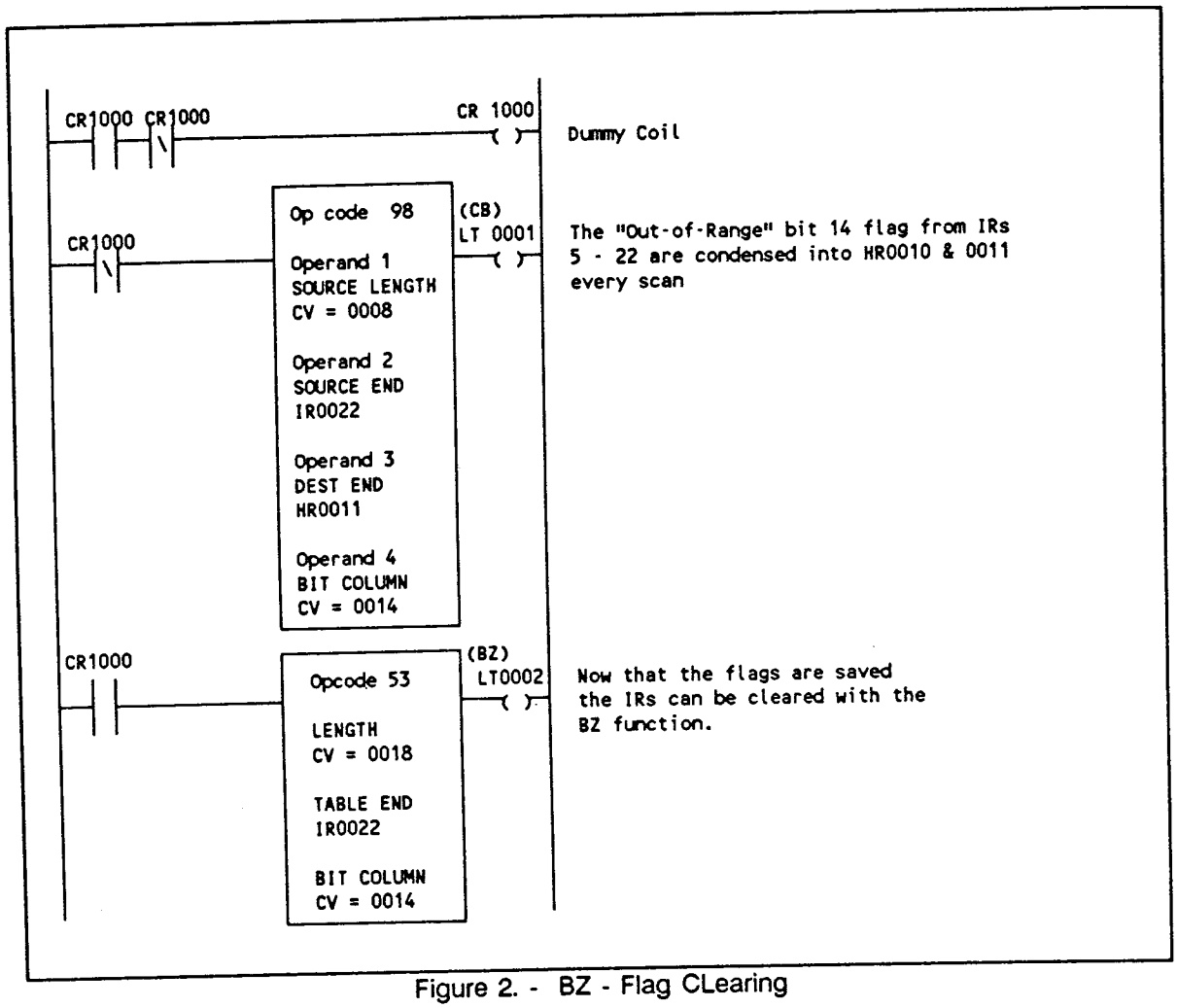


Figure 2. - BZ - Flag CLearing

NOTE:

When using the BZ function on I/O image types IR and IG, note that the function will have an effect starting at the point of execution and then holds until the end of scan. Afterwards the I/O update occurs resetting any cleared bits. Since the PC1200 answers communication requests after this update any monitoring device will show the cleared bits reset. If you desire to monitor the effect of the BZ function on these types use a BT function, after BZ, to transfer the table to HR memory for monitoring.

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Compress Bit "CB" special function is a wholesale extension of the Bit Pick contact. A column of bits from the Source Table are copied to a Destination Table in a condensed format. One bit per register is transferred from the Source Table to the Destination Table. Bit "X" in the first register of the Source Table is copied to the first bit of the first register of the Destination Table - bit "X" in the second register of the Source Table is copied to the second bit in the first register of the Destination Table and so on.

The intent is to transfer flags to a condensed table to be stored and transferred with greater ease. The Destination Table form is compatible with the Search Matrix and Bit Operate special functions for wholesale flag detection operations.

OP CODE

Op Code 98 defines the Literal (LT) as the CB function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the introduction and LT function description in the Special Function section for programming details.

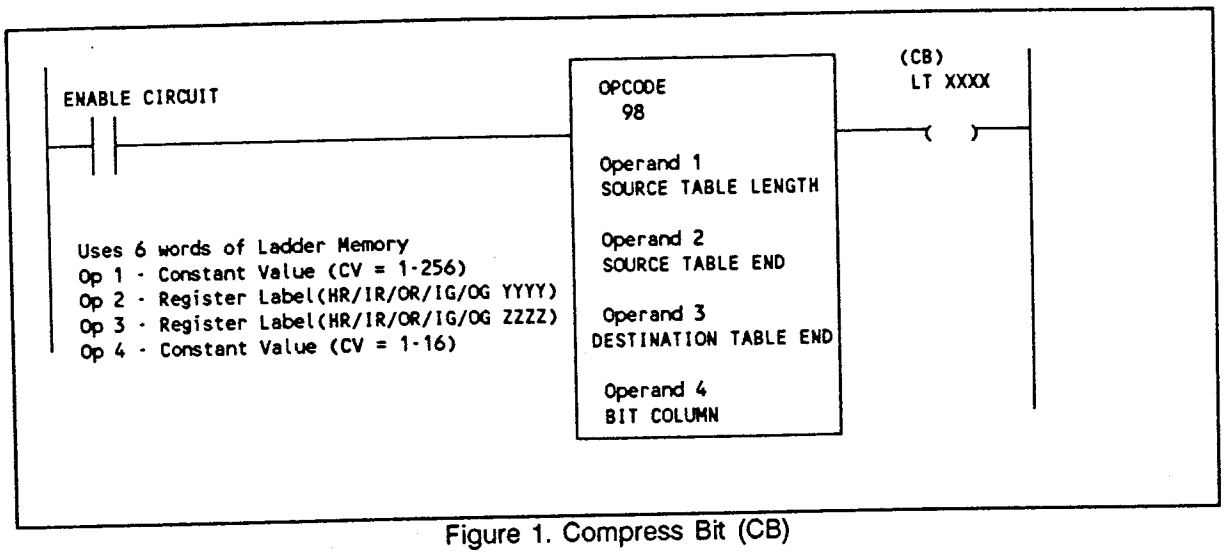


Figure 1. Compress Bit (CB)

CB - COMPRESS BIT

SPECIFICATION

OPERAND 1 - SOURCE TABLE LENGTH

The SOURCE TABLE LENGTH, along with SOURCE TABLE END, defines a table in memory from which a column of bits will be copied. The table length is defined by a Constant Value between 1 and 256. Operand 1 will be invalid if it is not in this range or programmed as a constant.

OPERAND 2 - SOURCE TABLE END

The SOURCE TABLE END defines the type and address of the last register in the Source Table from which the bit information is being copied. This operand may be programmed as an HR, IR, OR, IG or OG. Address labels may not be less than the SOURCE TABLE LENGTH, in magnitude, or greater than the highest reference allowed for the type chosen. Operand 2 will be considered invalid if it violates the address restrictions or if it is programmed as a constant.

OPERAND 3 - DESTINATION TABLE END

The DESTINATION TABLE END defines the type and address of the last register in the table where information is being copied to. The legal types are HR, IR, OR, IG, or OG. Address labels for this operand may not be less than SOURCE TABLE LENGTH + 16, in magnitude, rounded to the next full number. Bits required to complete a full register in the Destination Table are not altered in the transfer. Operand 3 will be considered invalid if it violates the address restriction or if it is programmed as a constant.

The Source and Destination Tables should not overlap. No checks are made to detect overlapping tables and no measures are taken to avoid writing over the Source Table before it is read.

OPERAND 4 - BIT COLUMN

The BIT COLUMN, entered as a Constant Value 1 - 16, specifies which bit position out of each register will be copied. Values outside of this range or programmed other that a constant will be considered invalid.

ENABLE CIRCUIT

When the CB ENABLE CIRCUIT conducts, bit data is copied from the source table to the Destination Table according to the BIT COLUMN selection.

COIL

The output coil is used to indicate programming errors. On any scan that the ENABLE CIRCUIT solves true and at least one of the operands is invalid, the output coil will turn on and the Destination Table is not modified. If operands are valid the coil remains off.

CB Operation

Table 1. shows the result of a CB special function copying bit 14 from each IR in the Source Table and copying them to the Destination Table. Since the Source Table is not an even multiple of 16, the Destination Table has un-referenced bits 3 through 16 in HR0011. The effected bits in the tables are being represented by alphabetic characters to give a sense to the translation direction. In actual practice, they would be binary values of "0" or "1".

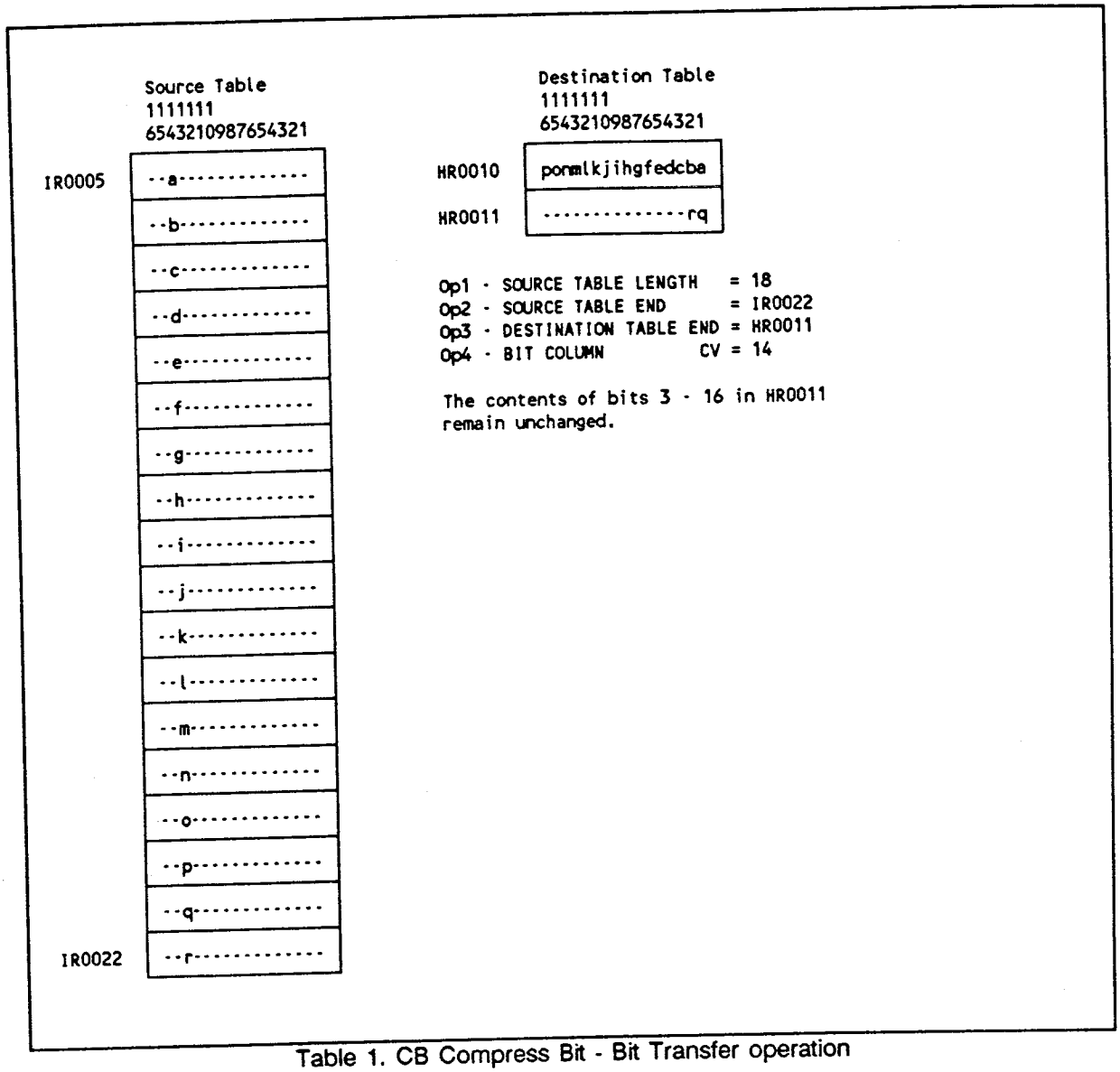


Table 1. CB Compress Bit - Bit Transfer operation

CB - COMPRESS BIT

Applications

The compressed form of the Destination Table is desirable for use with other Westinghouse special functions to annunciate flags.

Using table 1 for reference, the following example shows 18 Input Register from NL-1052 modules where the fourteenth bit is the "Out-of-Range" flag. These flags are being copied and saved in Destination Table HR0010 - HR0011. In order to display if any channels are out-of-range, the XM special function scans the Destination Table. The XM coil will go high upon detecting any flags set high. To further refine which channel is the offender the SM special function can step through the Destination Flag Table showing flags set active through the contents of the BIT REGISTER.

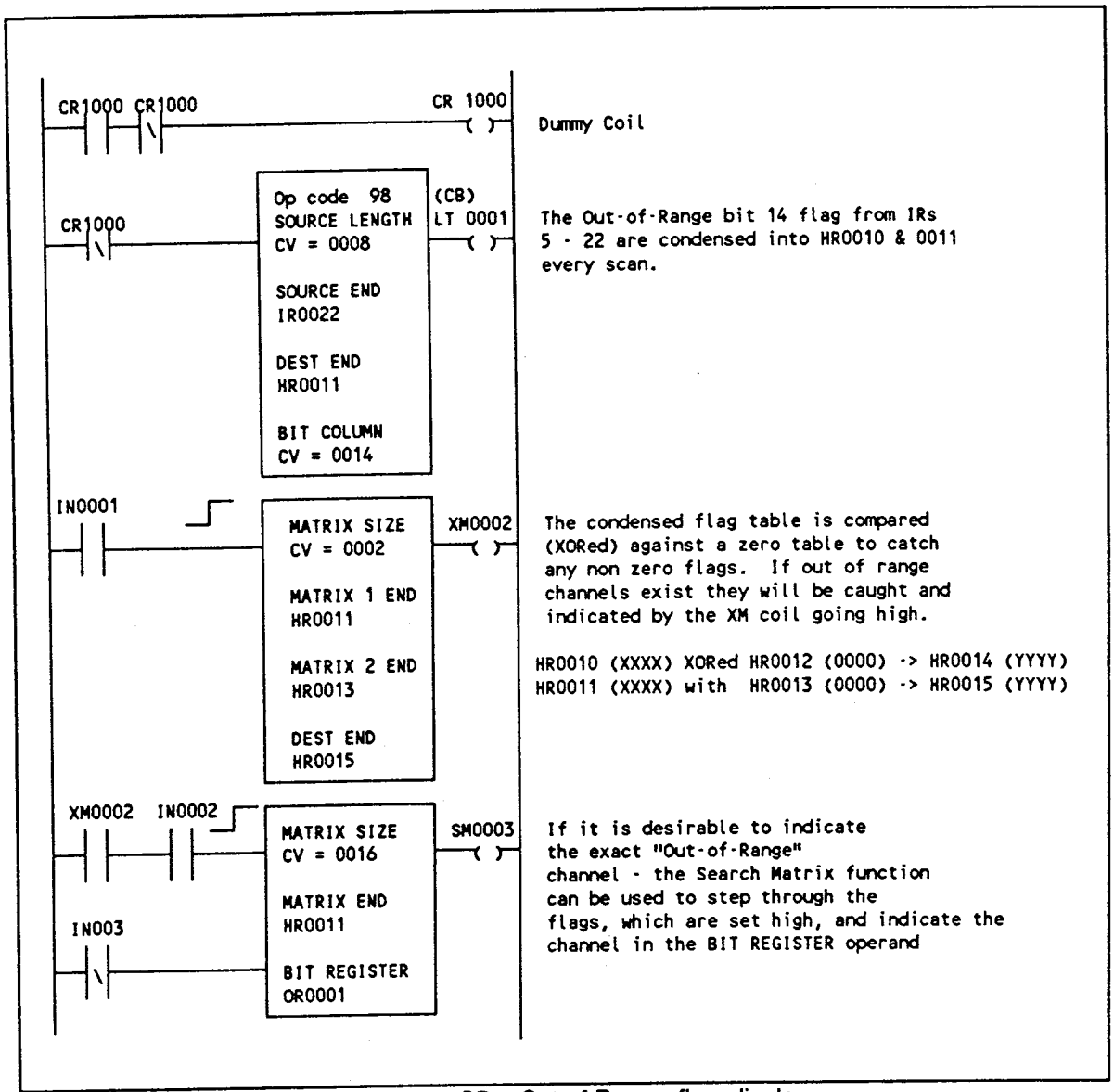


Figure 2. - CB - Out-of-Range flag display

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Checksum function has two main uses. The first use is in average (mean) calculations. To calculate an average of a table of numbers, you first sum the table of numbers together, then divide by the number of numbers in the table. The Checksum function simplifies this calculation by performing this table addition for you.

The second use of the Checksum function involves serial data communications. A checksum is frequently appended to the end of a stream of data bytes to assist in validating the data bytes. A receiving station is expected to duplicate the checksum calculation and compare the results of its calculation with the checksum that was appended to the incoming message. If they don't match, the incoming message is likely corrupted and it is discarded.

This Checksum function will calculate either a Longitudinal Redundancy Check (LRC) or a Cyclic Redundancy Check (CRC-16). The CRC-16 calculation is the same that is used by several common protocols including Modbus RTU. The LRC calculation is the same that is used by Westinghouse "6 Byte" protocol and Modbus ASCII.

OP CODE

Op Code 100 defines the Literal (LT) as the CX function. Whether or not the Literal function should be used depends upon the capability of the your program loader. Refer to the introduction and LT function description in the Special Function section for programming details.

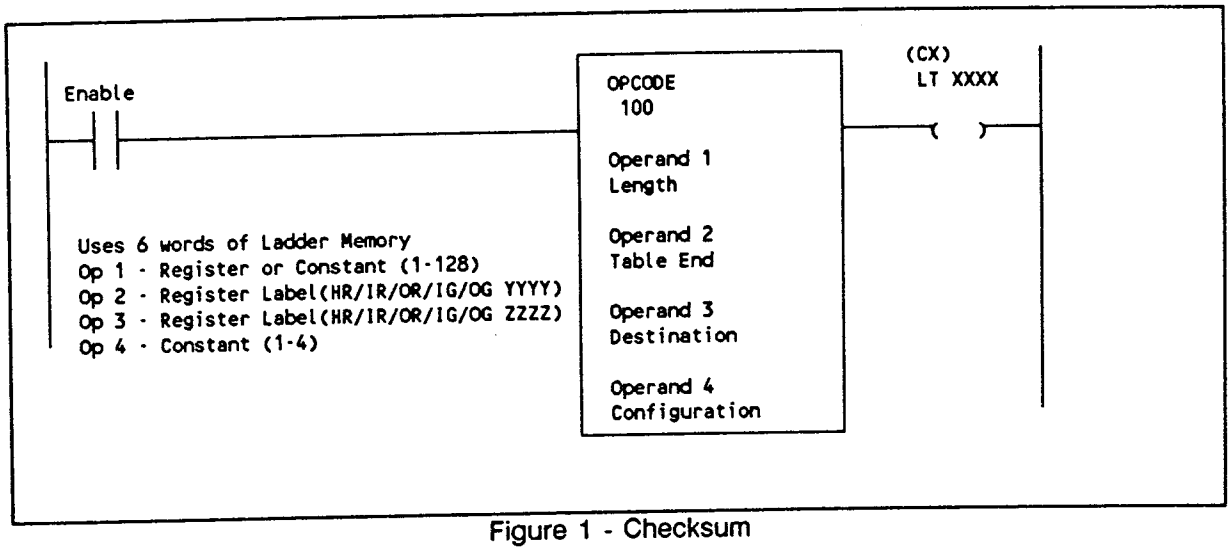


Figure 1 - Checksum

SPECIFICATION

OPERAND 1 - Table Length

The CX TABLE LENGTH operand contains a number that specifies how many registers are used in the checksum calculation. This number may either be a constant or it may be located in an HR, IR, OR, IG or OG register. If the number is stored in a register, a ladder program can manipulate it during runtime.

OPERAND 2 - Table End

The TABLE END operand is a number that represents a HR reference that will be the last register in a table. That table is where the data is stored. The number can either be a constant or it can be located in an HR, IR, OR, IG or OG register. The largest number (constant or value stored in a register) cannot exceed 1792 or the highest Holding Register used in the ladder program, whichever is smaller. The smallest number cannot be less than the LENGTH of the table.

OPERAND 3 - Destination

The DESTINATION operand points to an HR, OR or OG that is to receive the checksum.

OPERAND 4 - Configuration

The CONFIGURATION defines the type of checksum to be performed. It is specified as a constant value in the range of 1 to 4.

Configuration Number	Operation of CX Function
1	16 bit LRC
2	16 bit CRC taken low byte first then high byte
3	16 bit CRC taken high byte first then low byte
4	16 bit CRC taken from HR's low byte only

The LRC configuration treats the source table as 16 bit unsigned integers, while the three CRC modes treat the source table as a group of bytes. Algorithms for the LRC and CRC are described later in this section.

ENABLE CIRCUIT

On every scan that the ENABLE contact (or contact matrix) solves true, the CX special function will compute the checksum of the table defined by TABLE END and TABLE LENGTH using the method dictated by CONFIGURATION. The result of this operation will be placed in the register or group specified by DESTINATION.

COIL

The output coil is used to indicate programming errors. On any scan that the enable contact is true and at least one of the operands is invalid, the output coil will turn on. Otherwise, the coil will be off. On any scan that the coil is turned on, the destination register is not modified.

Operation

LRC Algorithm

The LRC is computed by the following procedure:

1. Set a 16 bit register to zero
2. Add a word to this register and discard any carry past the 16th bit
3. Repeat step 2 with the subsequent words (registers) until all words have been summed.

The following example illustrates the process for a sequence of words 'FA03h 10FCh 0010h 0501h':

Initialize Register		0000h
Add first	+	FA03h
	=	FA03h
Add second	+	10FCh
	=	0AFFh
Add third	+	0010h
	=	1B0fh
Add fourth	+	0501h
	=	1010h
Resulting LRC	=	1010h

CRC Algorithm

The CRC algorithm is computed by the following procedure:

1. Set a 16 bit register to all 1's.
2. Exclusive OR a byte with the low order byte of that register.
3. Shift the register one bit to the right.
4. If the bit that was shifted out (FLAG) was a 1, then exclusive OR the number '1010 0000 0000 0001' with the register.
5. Repeat steps 3 and 4 until 8 shifts have been performed.
6. Repeat steps 2 through 5 for the next byte until all bytes have been processed.

The following example illustrates this process for the sequence of bytes '02h 07h':

	LSB	MSB	FLAG
Initial register	1111 1111	1111 1111	
XOR with '02h'		0000 0010	
	1111 1111	1111 1101	
After Shift 1	0111 1111	1111 1110	1
XOR	1010 0000	0000 0001	
	1101 1111	1111 1111	
After Shift 2	0110 1111	1111 1111	1
XOR	1010 0000	0000 0001	
	1100 1111	1111 1110	
After Shift 3	0110 0111	1111 1111	0
After Shift 4	0011 0011	1111 1111	1
XOR	1010 0000	0000 0001	
	1001 0011	1111 1110	
After Shift 5	0100 1001	1111 1111	0
After Shift 6	0010 0100	1111 1111	1
XOR	1010 0000	0000 0001	
	1000 0100	1111 1110	
After Shift 7	0100 0010	0111 1111	0
After Shift 8	0010 0001	0011 1111	1
XOR	1010 0000	0000 0001	
	1000 0001	0011 1110	
XOR with '07'		0000 0111	
	1000 0001	0011 1001	
After Shift 1	0100 0000	1001 1100	1
XOR	1010 0000	0000 0001	
	1110 0000	1001 1101	
After Shift 2	0111 0000	0100 1110	1
XOR	1010 0000	0000 0001	
	1101 0000	0100 1111	
After Shift 3	0110 1000	0010 0111	1
XOR	1010 0000	0000 0001	
	1100 1000	0010 0110	
After Shift 4	0110 0100	0001 0011	0
After Shift 5	0011 0010	0000 1001	1
XOR	1010 0000	0000 0001	
	1001 0010	0000 1000	
After Shift 6	0100 1001	0000 0100	0
After Shift 7	0010 0100	1000 0010	0
After Shift 8	0001 0010	0100 0001	0
Result of CRC is	1 2	4 1	hex

Figure 4 - CRC Calculation

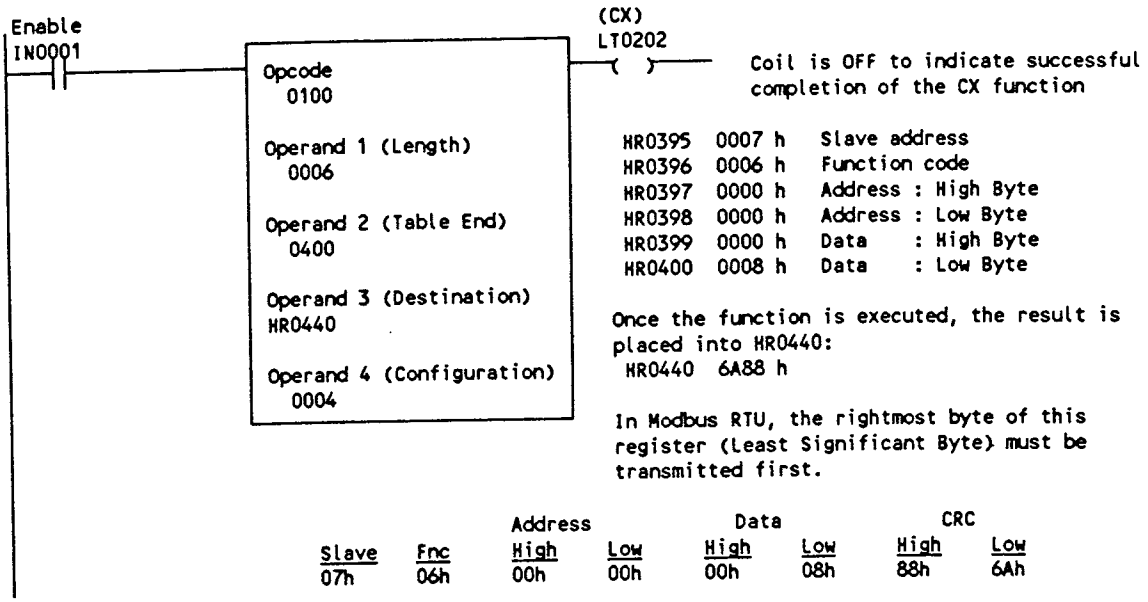
Applications

The PC1200 can be programmed to act as a Modbus Master if the proper sequence of bytes are transmitted out the serial port of the PLC. This example shows how a Modbus message (write 0008 into Holding Register 1 of slave number 7) could be sent out of the PC1200 Port B. The portion of the code that processes the response (ASCII Receive, etc.) has been omitted for clarity. Contact Westinghouse for assistance or a copy of the entire Modbus Master program.

A Modbus message is constructed of 5 parts: Address, Function Code, PLC address, Data and CRC
The bytes that must be sent for the example are:

		Address		Data		CRC	
Slave	Fnc	High	Low	High	Low	High	Low
07h	06h	00h	00h	00h	08h	??	??

In order to properly complete the Modbus message, the CRC must be calculated:



		Address		Data		CRC	
Slave	Fnc	High	Low	High	Low	High	Low
07h	06h	00h	00h	00h	08h	88h	6Ah

We now have the complete message that must be sent. The complete data is configured into the following Holding Registers. What still must be done is to add the ASCII Transmit (AT) "B-Codes" into this table of registers to force the AT function to send raw binary data rather than process the data as ASCII characters and "embedded data codes (B-Codes)".

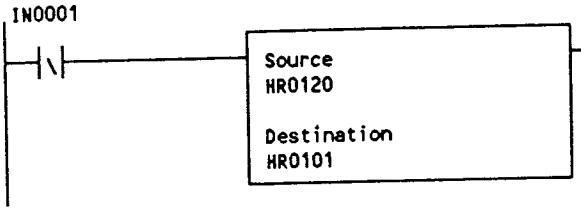
```
HR0393  0000 h  (not used, fill in with B-Code later)
HR0394  0000 h  (not used, fill in with B-Code later)
HR0395  0007 h  (Slave we are trying to talk to)
HR0396  0006 h  (Modbus Function Code.  06h means write to Holding Register)
HR0397  0000 h  (High Byte of PLC Address.  PLC address is one less than HR number)
HR0398  0000 h  (Low Byte of PLC Address.)
HR0399  0000 h  (High Byte of Data to write to PLC)
HR0400  0008 h  (Low Byte of Data to write to PLC)
HR0401  0088 h  (CRC High as calculated by CX)
HR0402  006A h  (CRC Low as calculated by CX)
HR0403  0000 h  (not used, fill in with B-Code later)
```

Figure 5 - Modbus Master Example

Notice that we stored the data into registers one byte at a time rather than using the full 16 bits (which would have allowed us to store 2 bytes per register). While it is true that using half a register is a bit wasteful, it may be easier to visualize what each byte is used for if we split them into two registers. Using one byte per register also makes sending odd number of bytes easier with the ASCII Transmit function (since the AT function allows us to define the number of registers, not bytes, transmitted. Hence, sending half a register would not be straight forward).

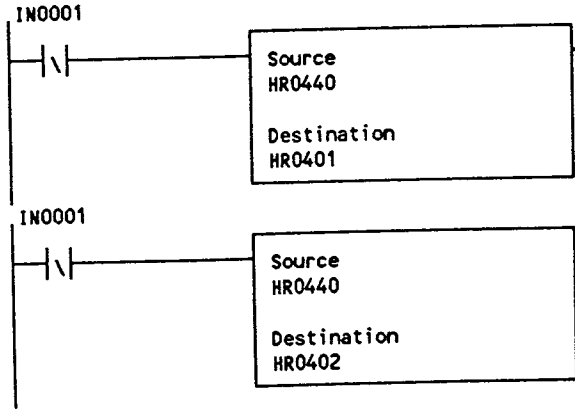
The next task is to send this complete message out the serial port. For this example we will say that a modem is connected to PORT B of the NL-1075B I/O module. In order to send a raw binary message with the ASCII Transmit function we use the BAxx "B-Code". Refer to your PLC Systems Manual for more information on programming the AT function. Manually preset HR0393 and HR0394 with the "B-Code" that instructs the AT function to send the lower byte of a table of registers. Place the B800 "End of Message" B-Code into HR403.

```
HR0393  BA03 h  (Transmit low byte only of the following table)
HR0394  0008 h  (Table to send is 8 registers long)
HR0395  0007 h  (Slave we are trying to talk to)
HR0396  0006 h  (Modbus Function Code.  06h means write to Holding Register)
HR0397  0000 h  (High Byte of PLC Address.  PLC address is one less than HR number)
HR0398  0000 h  (Low Byte of PLC Address.)
HR0399  0000 h  (High Byte of Data to write to PLC)
HR0400  0008 h  (Low Byte of Data to write to PLC)
HR0401  0088 h  (CRC High Byte calculated from CX function)
HR0402  0000 h  (CRC Low Byte calculated from CX function)
HR0403  B800 h  (AT End-Of-Message "B-Code")
```



Preset HR0120 to 00393 decimal when IN0001 is de-energized.
HR0101 is the pointer register used by the ASCII Transmit function to point to the beginning of the table of data we want to send.

We use the Move Byte (MB) function to transfer the individual bytes from the CX function into individual bytes (Low Byte of HR440 -> HR401, High Byte of HR440 -> HR402). Alternately, the Block Unpack (BU) function could have been used instead of two separate MB functions. The reason that we unpack the CRC (instead of simply inserting a "B-Code" to tell the AT function to sent the CRC as a binary number, is because the AT function stops sending characters when a new B-Code is encountered. The AT function then processes the AT function on the next scan. Since the AT function will momentarily stop sending characters, the Modbus receiver may incorrectly assume that an "End of Message" timeout has been received. Modbus defines an "End of Message" when 3.5 character times elapse without data reception. One character time is bits in characters/baud. (i.e. 1 start bit plus 8 data bits plus 1 parity bit plus 2 stop bits equals 12 bits. Divide 12/9600 = 1.25 mS per character. 3.5 character times = 4.375 mS).



Move the lower byte of the CRC to HR401 in the AT PRINT table.

Move the upper byte of the CRC to HR402 in the AT PRINT table.

Figure 6 - Modbus Master Continued

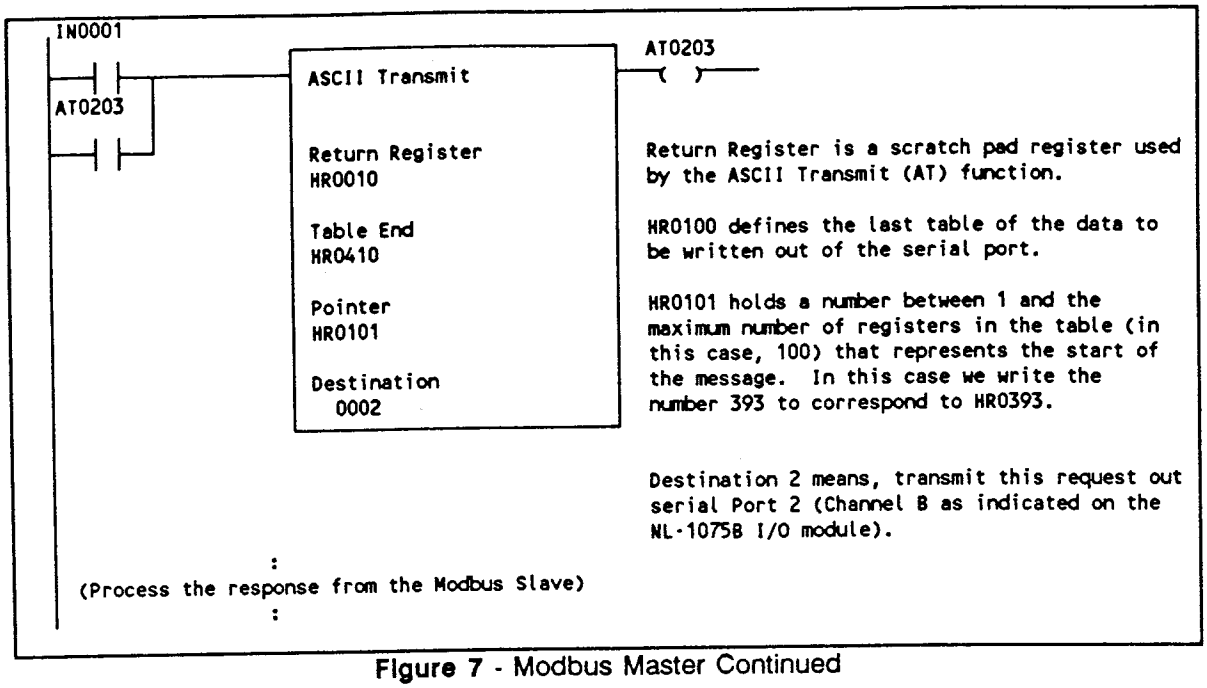


Figure 7 - Modbus Master Continued

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Extend Register "ER" special function is the ladder window into the Highest Holding Register Used (HHRU) Executive Parameter. It provides a mechanism to expand beyond the direct HHRU address reference limits of 1792, up to the End Of Program "EOP" word or HR9999 maximum. This expanded Holding Register memory can be accessed indirectly by the Indirect Block Move "BM", ASCII Transmit and Receive "AT" and "AR", Port Transmit "PT", ASCII to Binary Convert "AB" and Check Sum "CX" special functions.

The indirect Holding Register memory can be used to store recipes, presets and other batch parameters. In addition this memory is ideal for ASCII or Port Transmit communication buffers and work space.

OP CODE

Op Code 16 defines the Literal (LT) as the ER function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the introduction and LT function description in the Special Function section for programming details.

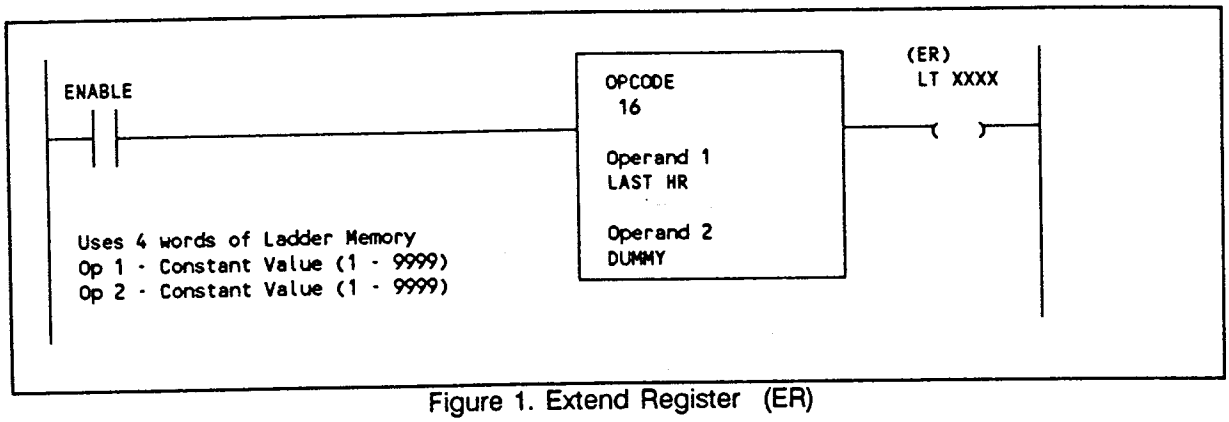


Figure 1. Extend Register (ER)

ER - EXTEND REGISTER

SPECIFICATION

OPERAND 1 - LAST HR

LAST HR defines the Highest Holding Register Used or the HR upper limit. It is programmed as a Constant Value between 1 and 9999 maximum. The HHRU must be lower in memory than the End Of Program (EOP) definition.

OPERAND 2 - DUMMY

A Constant Value between 1 and 9999 which must be programmed to satisfy the special function format of 2 operands (LT op codes 16-47). It is not used by the function, however for clarity it is suggested to be made equal to Operand 1.

ENABLE CIRCUIT

The ENABLE CIRCUIT has no function in this special function.

COIL

The ER Coil will follow the status of the ER enable circuit. It does not give any information as to the status of the ER function.

OPERATION

The ER special function must be the first function in the program to insure that the HHRU is properly defined. Failure to follow this rule will cause a User Software fault condition bit 7. This function is executed on the first scan after a transition into Run. Operand one is used to define the Highest Holding Register Used and reserve a block of memory. Until the ER function is executed, the user cannot access Holding Registers above the standard limit of 1792 with other functions such as BM, AB, CB etc. During ER execution the value defined by Operand one is compared to the present value of HHRU. If the ER value is larger then the HHRU is overwritten with the new value.

Once the ER function is executed the only way to allocate more or less memory for Holding Register storage is to, transition to STOP/PROGRAM mode, reprogram operand 1, and then transition back into RUN mode. Once the HHRU has been increased, by ER, beyond 1792 it cannot be reduced below 1792 without reprogramming the function and using the register re-pack function of the program loader to compress the HHRU down to the smaller limit.

If a program containing an ER function is down-loaded to the PC1200, the actual number of HHRU for this program will not be calculated by ER until the unit is put into RUN mode. The user should activate the ER function after down-loading and before editing such a program. The program loader requires this HHRU reference for file recording and program bounds.

If operand 1 is not programmed as a constant or if operand 1 is the reference number of a register above the EOP, or if the ER function is not programmed as the first rung of the program, or if the enable contact matrix is made up of more than one contact, the PC1200 will be put into FAULT with a User Software Error - fault bit 7.

Note - Only PC1200s with 16K of user memory have sufficient memory to set the HHRU to 9999. The user memory holds both Holding Registers and the ladder program. See Section 6 of the 1000 Series Systems Manual for user memory definition.

Applications

A control engineer wants to design a PC1200 system where 4K of memory will be allocated to recipes. It has been determined that 2K is enough for the ladder and that the base reference up to 1792 Holding Registers will be used as well. Adding these segments together is slightly below 8K and therefore a PC1200-X042 is selected. The total register allocation will be 1792 plus 4K or 5888 words. As the first programming action the engineer programs an ER special function with the first operand equal to 5888. The PC1200 is then put into RUN mode to set the HHRU equal to operand 1. From this point on the holding register block is marked and the rest of the program can be structured.

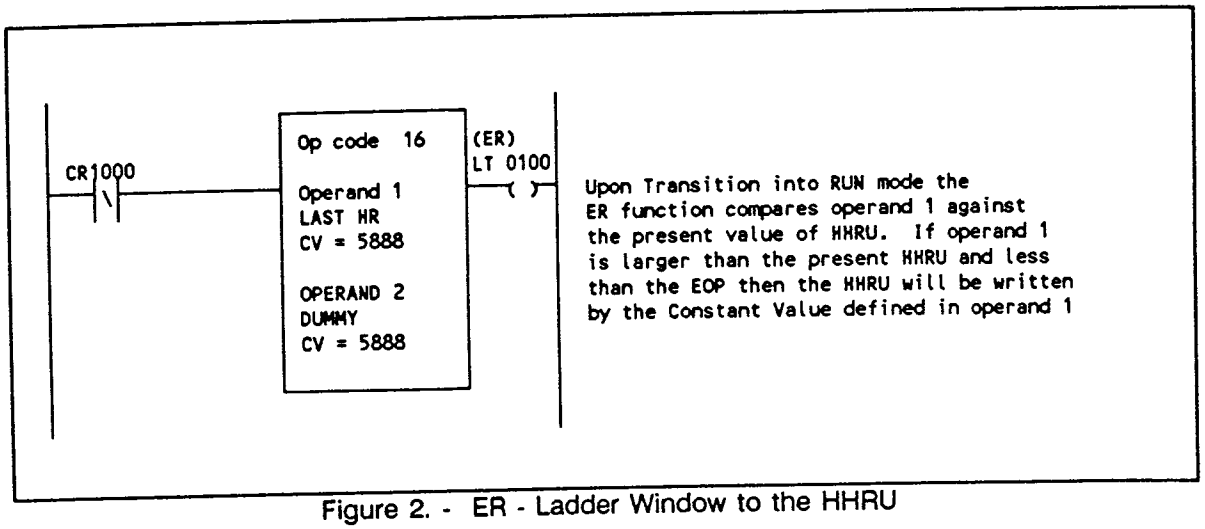


Figure 2. - ER - Ladder Window to the HHRU

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Limit Check "LX" special function determines if the value stored within a register or group is within specified limits.

OP CODE

Op Code 54 defines the Literal (LT) as the LX function. Whether or not the Literal function should be used depends upon the capability of you program loader. Refer to the introduction and LT function description in the Special Function section for LT programming details.

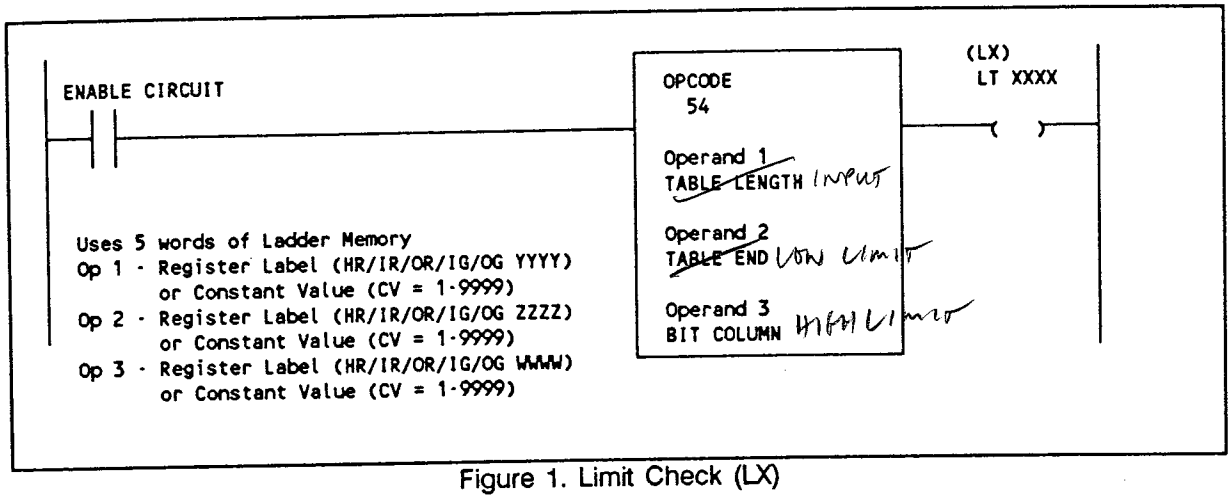


Figure 1. Limit Check (LX)

LX - LIMIT CHECK

SPECIFICATION

OPERAND 1 - INPUT

The INPUT defines the type and address of the register which contains the tested value or it can be defined as a Constant Value. Legal program definitions are HR, IR, OR, IG, OG, or Constant Value.

OPERAND 2 - LOW LIMIT

The LOW LIMIT defines the type and address of the register which contains the lower bound to which the INPUT is compared or it can be defined as a Constant Value. Legal program definitions are HR, IR, OR, IG, OG or Constant Value.

OPERAND 3 - HIGH LIMIT

The HIGH LIMIT defines the type and address of the register which contains the upper bound to which the INPUT is compared. Alternatively it can be programmed as a Constant Value. Legal program definitions are HR, IR, OR, IG, OG or Constant Value.

ENABLE CIRCUIT

When the LX ENABLE CIRCUIT conducts the equation LOW LIMIT ≤ INPUT ≤ HIGH LIMIT and LOW LIMIT ≤ HIGH LIMIT is tested.

COIL

If the ENABLE CIRCUIT is energized and the above equation solves true the coil will be energized. If the enable is un-energized or LOW LIMIT > HIGH LIMIT the coil will remain off.

Applications

Frequently it is desirable to know if a measured value stays within safe limits. The following example of figure 2 compares the temperature value coming in on an Input Register against safe limits for a refrigeration system. If the temperature falls outside of the specified limits an alarm is set.

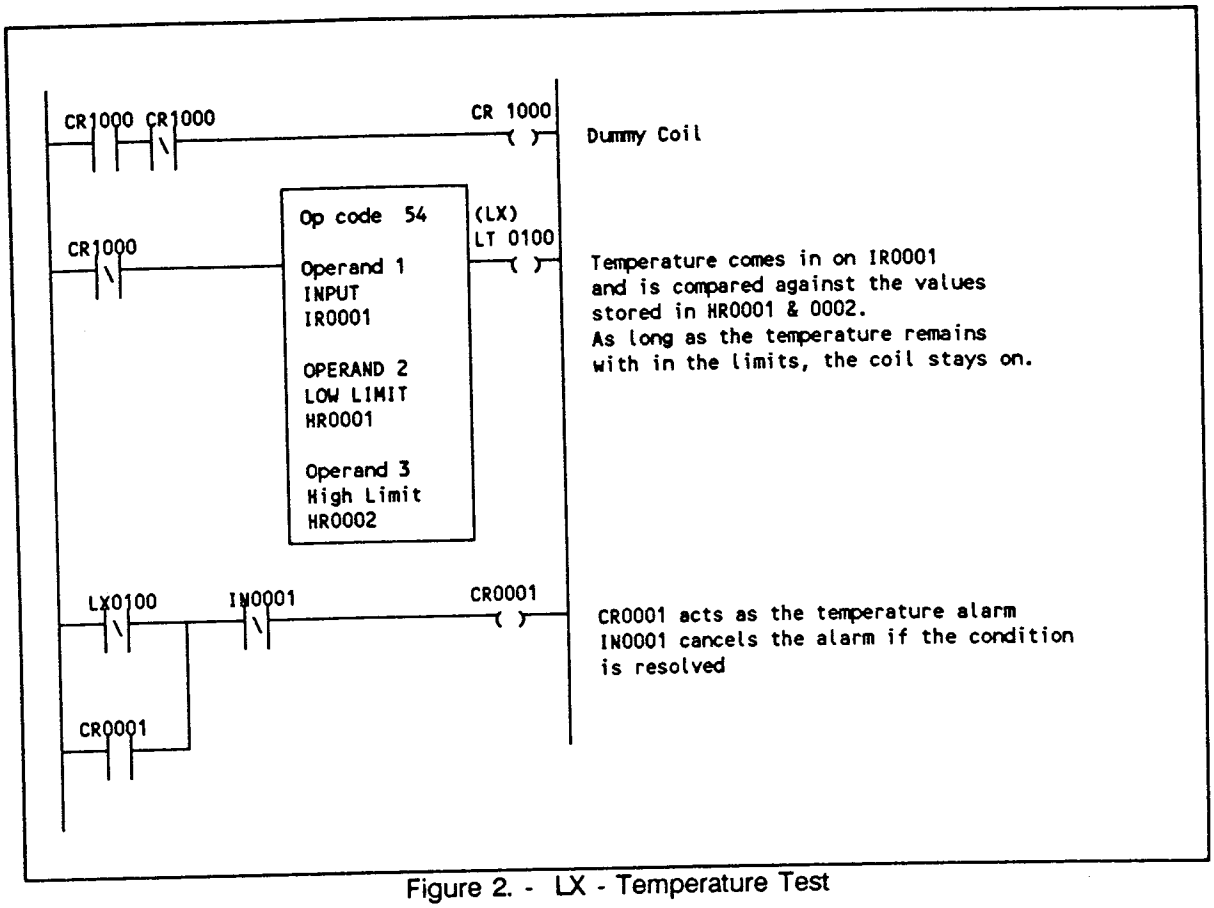


Figure 2. - LX - Temperature Test

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Pack Byte "PB" special function is a wholesale extension of the Move Byte special function acting to condense byte information into a contiguous table of registers or groups. Four "Packing" modes are selected through the CONFIGURATION operand to define which byte, high or low, in the Source Table will be picked and copied to the Destination Table, stacked two bytes per register. This is a reduction operation and therefore the destination will always be roughly one half the size of the source (an odd length source necessitates an un-referenced extra byte in the destination). The Byte Unpack "BU" special function is the logical inverse of the Pack Byte special function.

OP CODE

O Code 102 defines the Literal (LT) as the PB function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the introduction and LT function description in the Special Function section for LT programming details.

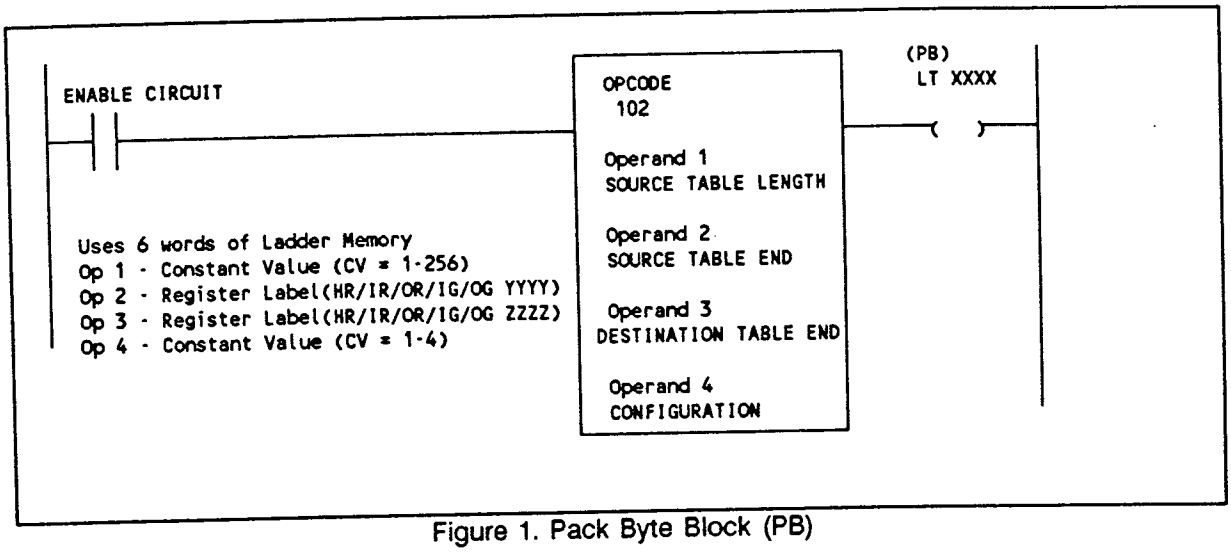


Figure 1. Pack Byte Block (PB)

SPECIFICATION

OPERAND 1 - SOURCE TABLE LENGTH

The Source Table Length is specified as a Constant Value that defines the number of registers in the source table. This parameter may not be less than 1 or greater than 256. Indirectly it defines the Destination Table as 1/2 X the SOURCE TABLE LENGTH, rounded to the next whole number. Operand 1 will be considered invalid if it is not a constant or if it is outside of the accepted range

OPERAND 2 - SOURCE TABLE END

The SOURCE TABLE END defines the type and address of the last register in the Source Table from which information is being copied. This operand may be programmed as an HR, IR, OR, IG or OG. Address labels may not be less than the SOURCE TABLE LENGTH, in magnitude, or greater than the highest reference allowed for the type chosen. Operand 2 will be considered invalid if it violates the address restrictions or if it is programmed as a constant.

OPERAND 3 - DESTINATION TABLE END

The DESTINATION TABLE END defines the type and address of the last register in the table where information is being copied to. The legal types are HR, IR, OR, IG or OG. Address labels for this operand may not be less than SOURCE TABLE LENGTH + 2, in magnitude, rounded to the next whole number. Operand 3 will be considered invalid if it violates the Address restriction or if it is programmed as a constant.

The Source and Destination Tables should not overlap. No checks are made to detect overlapping tables and no measures are taken to avoid writing over the Source Table before it is read.

OPERAND 4 - CONFIGURATION

The CONFIGURATION, programmed as a Constant Value 1-4, specifies the manner in which the Pack Byte operation will be performed. Bytes are always copied from the Source Table's low byte first, then high byte. Values outside of (CV = 1-4) this range or programmed other than a constant will be considered invalid. The definition of the CONFIGURATION selection is given in table 1.

If the Source Table has an odd length, then the last register in the destination will have one byte unaltered. If the CONFIGURATION is CV= 1 or 3 then the upper byte will be left unaltered. Alternatively if the CONFIGURATION is CV = 2 or 4 then the lower byte will be left unaltered.

CONFIGURATION Number	Half of Source used	Order of filling Destination
1	Low Byte	1st->Low / 2nd->High
2	Low Byte	1st->High / 2nd->Low
3	High Byte	1st->Low / 2nd->High
4	High Byte	1st->High / 2nd->Low

Table 1. CONFIGURATION Definition

ENABLE CIRCUIT

When the PB ENABLE CIRCUIT conducts, data is copied from the Source Table to the Destination Table according to the CONFIGURATION.

COIL

The output coil is used to indicate programming errors. On any scan that the ENABLE CIRCUIT solves true and at least one of the operands is invalid, the output coil will turn on and the Destination Table is not modified. If operands are valid the coil will remain off.

Applications

Typical applications include: Condensing eight bit analog channels 2 per Output Register to save on addressing, Condensing ASCII characters for use with ASCII Transmit, or any application which treats byte information in a wholesale fashion.

Figure 2 shows an example where 15, eight bit, Analog channels are condensed and sent out through ORs 1-8 in a stacked mode. This example could be expanded to include all 128 ORs.

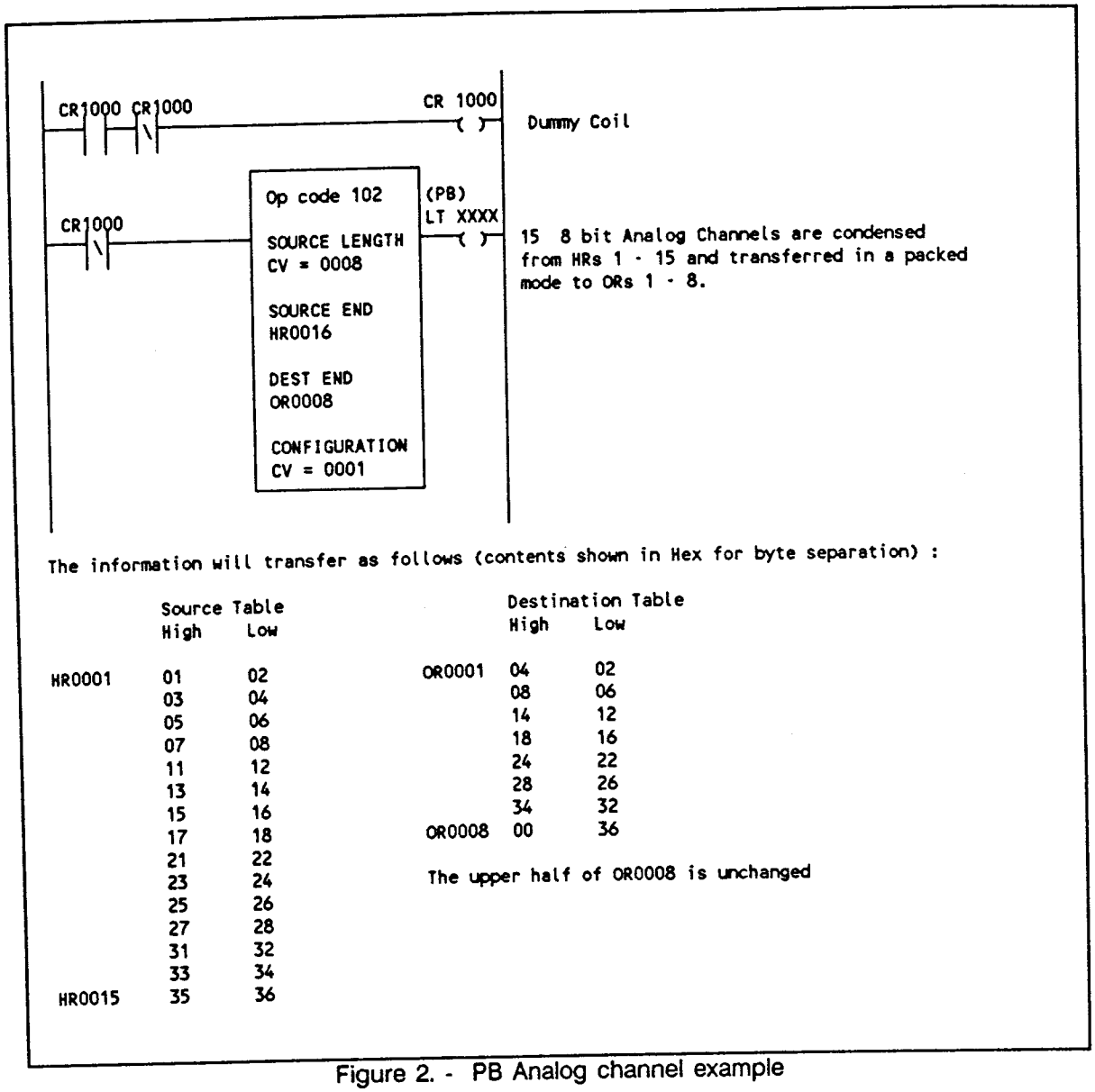


Figure 2. - PB Analog channel example

PC-1100-X01Y: NOT SUPPORTED	PC-1100-X05Y: NOT SUPPORTED
PC-1100-X02Y: NOT SUPPORTED	PC-1200-X02Y: SUPPORTED (≥ V1.6)
PC-1100-X03Y: NOT SUPPORTED	PC-1200-X04Y: SUPPORTED (≥ V1.6)

DESCRIPTION

The Scale "SC" special function converts an INPUT range of values into an OUTPUT range of values. The translation is accomplished by entering the limits of the INPUT and OUTPUT ranges. The result is equivalent to a Y = mX + b translation. In addition to saving memory over use of standard math functions, this function checks for INPUT out-of-range, calculates the scaling factor "m" and has a built-in rounding routine.

OP CODE

Op Code defines the Literal (LT) as the SC function. Whether or not the Literal function should be used depends upon the capability of your program loader. Refer to the introduction and LT function description in the Special Function section for programming details.

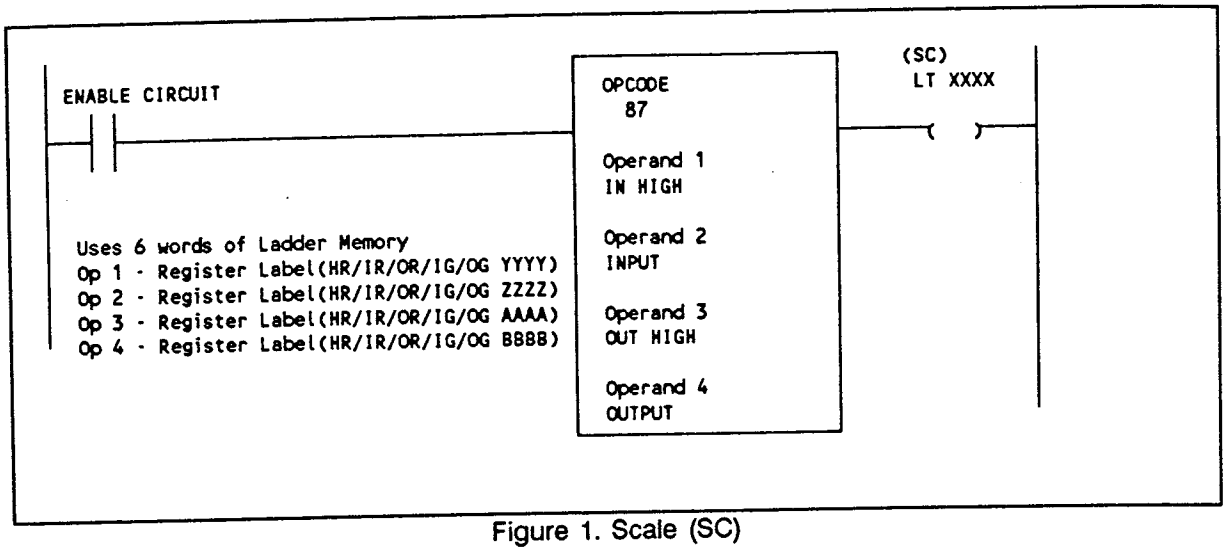


Figure 1. Scale (SC)

SC - SCALE

SPECIFICATION

OPERAND 1 - IN HIGH

The IN HIGH operand defines the type and address of a register pair containing the INPUT boundaries. The IN HIGH register is defined directly by operand 1 while the IN LOW register is referenced indirectly as the preceding memory location (operand 1 address minus one).

The allowable range for the IN HIGH and IN LOW operands is 0 - 65535. To have meaning however, IN HIGH should be greater than IN LOW. The legal register types are HR, IR, OR, IG, or OG. IN HIGH will be considered invalid if it is programmed as a constant or if there is no valid address preceding it (i.e. IN HIGH programmed as IG0001) for the implied In Low operand. In addition, address labels may not be greater than the highest allowed reference for the type chosen.

OPERAND 2 - INPUT

The INPUT operand defines the type and address of the register which contains the value to be scaled. The legal register types are HR, IR, OR, IG, or OG. The range of this operand is intended to stay within the bounds of IN HIGH and IN LOW. If the bounds are exceeded, the OUTPUT will latch at the appropriate boundary and the SC coil will turn on. If the operand is programmed as a constant it will be considered invalid.

OPERAND 3 - OUT HIGH

The OUT HIGH operand defines the type and address of a register pair which contain the OUTPUT boundaries. The OUT HIGH register is defined directly by operand 3 while the OUT LOW register is referenced indirectly as the preceding memory location (operand 3 address minus one).

The allowable range for the OUT HIGH and IN LOW operands is 0 - 65535. To have meaning however, OUT HIGH should be greater than OUT LOW. The legal register types are HR, IR, OR, IG, or OG. OUT HIGH will be considered invalid if it is programmed as a constant or if there is no valid address preceding it (i.e. OUT HIGH programmed as IG0001) for the implied OUT LOW operand. In addition, address labels may not be greater than the highest allowed reference for the type chosen.

OPERAND 4 - OUTPUT

The OUTPUT is the register where the result of the scaling special function is placed. This value will always be in the range of OUT LOW and OUT HIGH. Under certain invalid conditions defined later in function operation, no operation occurs (OUTPUT is unchanged) and the SC coil is set high. The legal register types are HR, IR, OR, IG, or OG. This operand will be considered invalid if programmed as a constant.

ENABLE CIRCUIT

On every scan that the ENABLE CIRCUIT solves true, the SC function is active. If all of the operands are programmed in a valid manner and the operand contents test valid, then the value specified by the INPUT is scaled and the scaled value is stored in the OUTPUT operand.

SC OPERATION

Under valid conditions the Input is scaled using the equation Y = mX + b in the following form:

*Y = OUTPUT = (INPUT - IN LOW) * Scale factor + OUT LOW where:*

$$m = \text{Scale Factor} = \frac{\text{OUT HIGH - OUT LOW}}{\text{IN HIGH - IN LOW}}, \quad \text{(bias) } b = \text{OUT LOW}, \quad X = \text{INPUT}$$

The actual order of operation that will be performed to calculate the OUTPUT are as follows:

*OUTPUT = {[(INPUT - IN LOW) * (OUT HIGH - OUT LOW)] / (IN HIGH - IN LOW)} + OUT LOW*

The result of the divide will be rounded up if the remainder is greater than or equal to 0.5, otherwise the remainder will be truncated. If because of round off, the OUTPUT is outside the range specified by OUT HIGH and OUT LOW, the OUTPUT will be set to the nearest limit.

RESTRICTION ON OPERATION

INPUT Range
If the INPUT is less than the value of IN LOW, the output will be set to OUT LOW. If the INPUT is greater than IN HIGH, the OUTPUT will be set to OUT HIGH.

Limits
The boundaries should solve the relationship - *(IN LOW < IN HIGH) and (OUT LOW ≤ OUT HIGH)* - true. If the relationship is false the setup will be considered invalid and no scaling will be performed and the output will remain unchanged.

COIL

The coil is used to signal alarms. If abnormal conditions occur such as invalid operands, boundaries are setup with invalid comparisons, or the Input is out-of-range, the coil is set high. Under valid conditions the coil will remain off. In summary the coil will turn on only if the contact enable matrix solves true and if any of the following conditions exist:

1) Operands are defined invalid
   a) Constant used
   b) Address reference higher than maximum allowed for type
   c) Operand 1 or 3 is programmed with no room for implied operands
2) An illegal boundary set-up is used
   a) IN LOW > IN HIGH or OUT LOW > OUT HIGH
   b) IN LOW = IN HIGH (Scaling factor would have a divide by zero)
3) INPUT out of range - INPUT < IN LOW or INPUT > IN HIGH

SC - SCALE

Applications

A field transducer supplies a 4-20 mA current loop signal to an NL-1052 Analog to Digital convertor module. The current signal is proportional to the actual process temperature where 4 mA equals 100 degrees F and 20 mA equals 350 degrees F. The 12 bit resolution NL-1052 module converts this signal into a number between 0 to 4000 and places this value into an Input Register during the I/O update scan.

The ladder program can use the raw binary value for monitor or control purposes, however the control engineer wants to display the temperature in Engineering units.

Since the current loop link in the information chain is proportional to the temperature and the converted Input Register value, its effect cancels out of the conversion equation. Graphically figure 2. shows the relationship. In this example the temperature and the Input Register converted binary value are represented by the following proportion:

$$\frac{Y \text{ (Output degrees F)}}{350 - 100^\circ F} = \frac{\text{Input Register Value}}{4000 - 0}$$

At an Input Register Value of "0" the temperature equals 100 which creates the bias term "b" and completes the equality into the recognizable Y = mX + b form

$$Y = \{\text{Input Register Value} * [(350-100^\circ F) / (4000 - 0)]\} + 100^\circ F$$

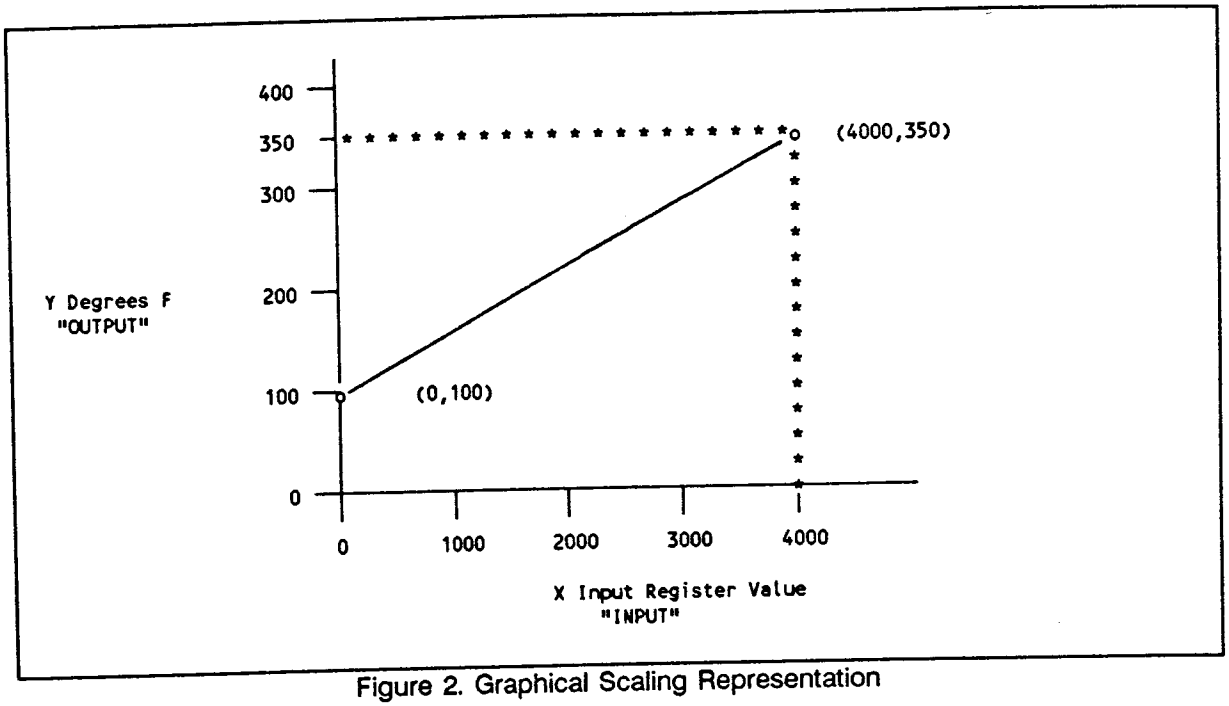


Figure 2. Graphical Scaling Representation

Application cont.

All of this math becomes somewhat transparent with the Scale special function. All that is necessary for the user to know is the upper and lower bounds of the INPUT conversion (4000 and 0) and the upper and lower bounds of the corresponding Engineering units (350 and 100). Figure 3 continues the example of an INPUT coming in on IR0001 and scaled Output sent to HR0010. A BCD conversion is performed on HR0010 to create an LED display compatible format on OR0001. The Control Engineer decides to represent the engineering units in tenths of a degree Fahrenheit so as to expand the range and save the fractional information. Therefore 3500 and 1000 are used instead of 350 and 100.

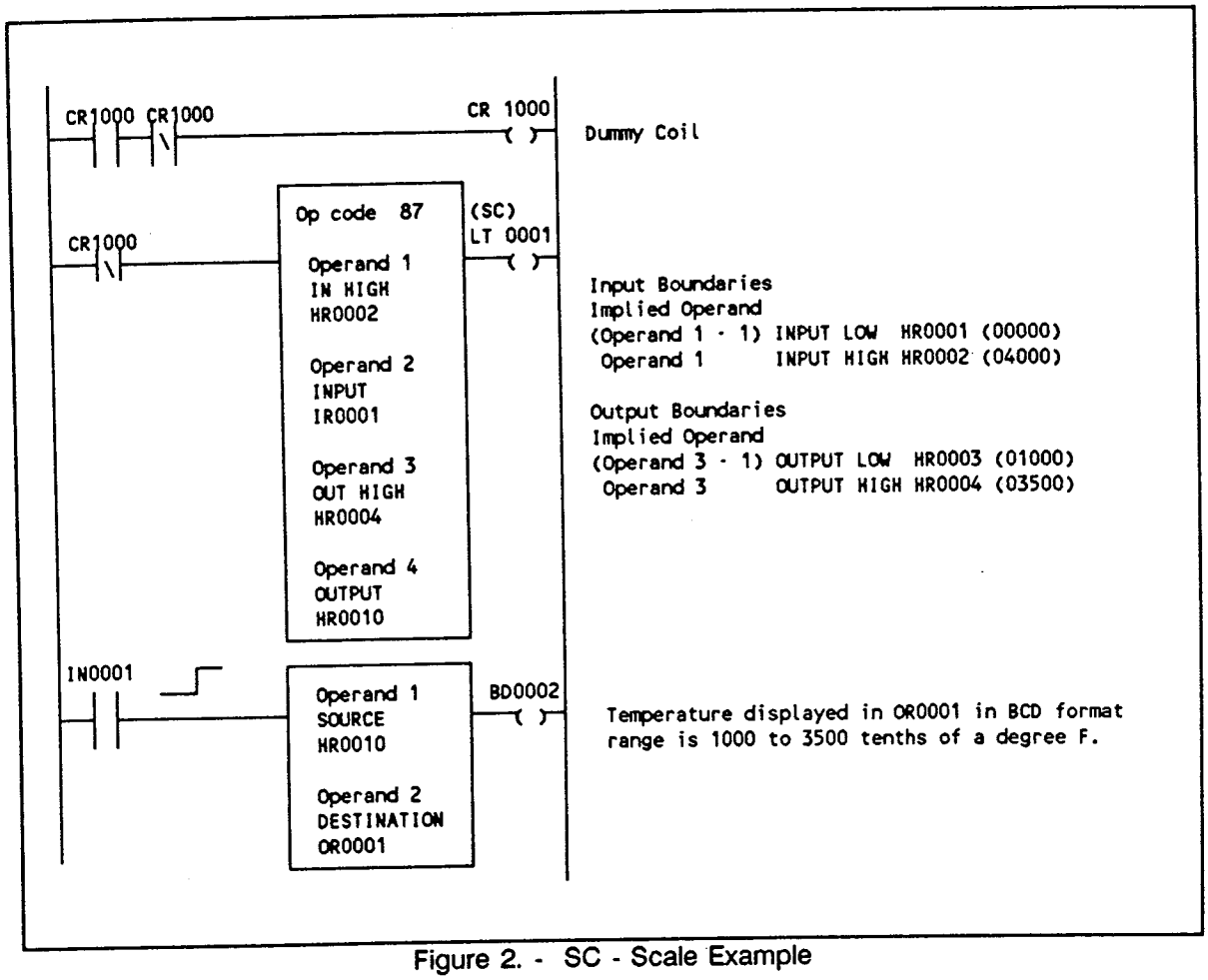


Figure 2. - SC - Scale Example