CHAPTER 3 BASIC PROGRAMMING AND INSTRUCTION SET

Accumulating Result Register

The reader will note that the AND instruction introduced the concept of an accumulating Result Register. In the execution of this instruction, the ICU logically performed an AND function on the data on its bidirectional data line with the data in its internal Result Register. The result of this operation became the new content of the Result Register. The point to be made here is that the Result Register always receives the result of any of the ICU's logical instructions. The Result Register therefore accumulates the logical result of each ICU logical instruction. This is analogous to an adding machine which always displays the subtotal after each operation.

Complement Instruction

It is sometimes desirable to activate an output when one input is in the logic 0 state and another input is in the logic 1 state. This situation occurs in relay controlled systems where "normally closed" relays are used, and occurs in solid state logic systems where inverters are present. Figure 3.1 shows an example of this situation.

The ICU instruction set is prepared for this event. Several logical "complement" instructions invert the logic level of the data on the ICU's bidirectional data line before operation on this data.

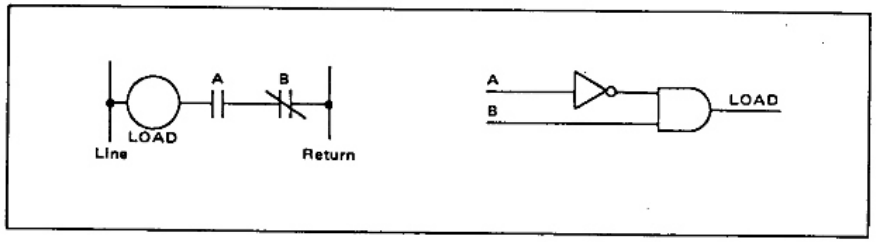


Figure 3.1 Examples of Complemented Signals

The LDC Instruction

An example of one of these instructions is the load complement instruction, abbreviated (LDC). The operation of this instruction is as follows. The ICU system memory supplies the ICU with the LDC instruction and the input selectors with the address of the input to be used in the operation. The input selector then demultiplexes the data of the selected input to the ICU's bidirectional data line. The ICU complements this data and stores the result in its one bit Result Register. The Result Register will receive a logic 1 if the selected input was in the logic 0 state. Figure 3.2 shows an ICU program which solves the problem shown in Figure 3.1, using the LDC command. The reader should be convinced of the operation of this program before reading further.

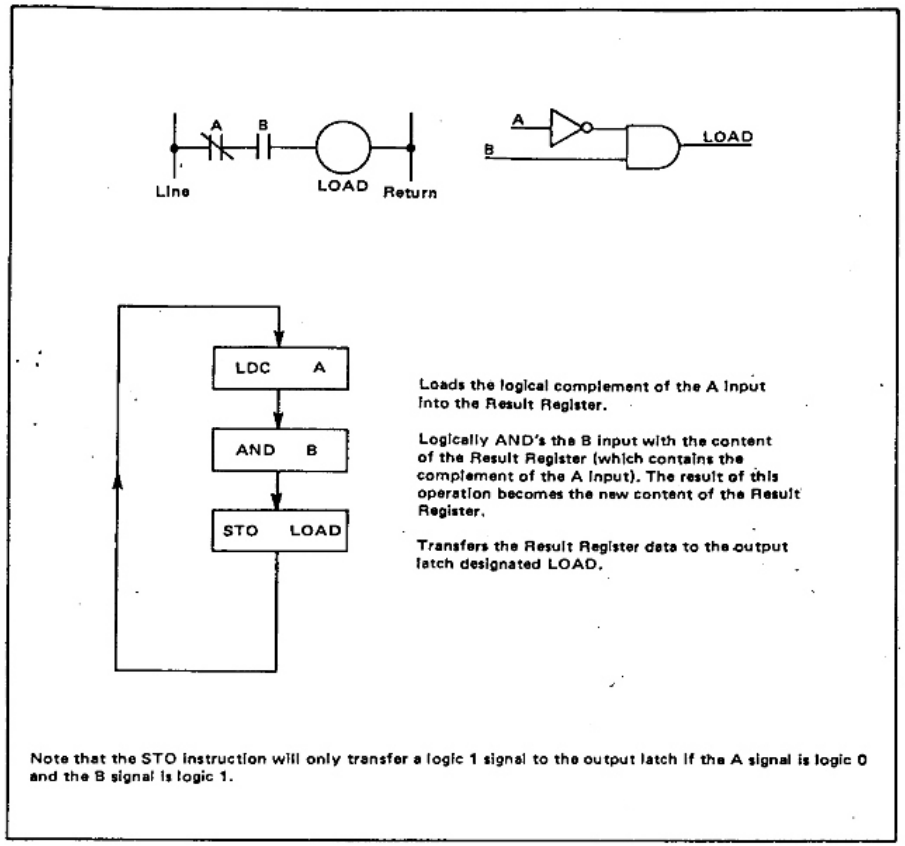


Figure 3.2 Using the LDC Command

The ANDC Instruction

Another example of a logical complement instruction is the "and complement" instruction abbreviated (ANDC). The operation of the ANDC instruction is as follows. The ICU system memory supplies the ICU with the ANDC instruction and the input selectors with the address of a selected input. The input selector then demultiplexes this data onto the ICU's one bit bidirectional data line. The ICU complements this data and logically AND's this data with the data in the Result Register. The result of this operation becomes the new content of the Result Register. The Result Register will receive a logic 1 if the input selected was at logic zero and the Result Register previously contained a logic 1.

With the addition of this instruction the ICU is able to attack some more complicated "chain" calculations. Figure 3.3 shows one such example. Figure 3.4 shows an ICU program which solves the problem depicted in Figure 3.3.

In reviewing the operation of these instructions, the reader should be convinced that the load device will only receive a logic 1 signal if A = 1, B = 0, C = 1, and D = 0.

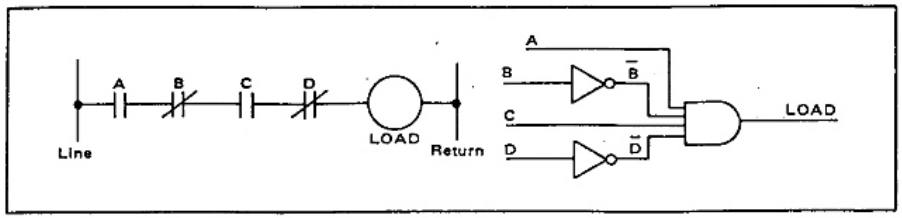


Figure 3.3 Example of a Chain Calculation

Statement	Operator	Operand	Comments
#1	LD	A	Result Register ← A
#2	ANDC	B	Result Register ← A • $\overline{B}$
#3	AND	C	Result Register ← A • $\overline{B}$ • C
#4	ANDC	D	Result Register ← A • $\overline{B}$ • C • $\overline{D}$
#5	STO	LOAD	Result Register = A • $\overline{B}$ • C • $\overline{D}$ → LOAD

Figure 3.4 Program to Solve the Chain Calculation of Figure 3.3

OR and ORC

In many cases, it is also desirable to activate an output when either input is in the logic 1 state. In this event, the "or" instruction, (OR), should be used. The operation of the OR instruction is as follows. The ICU system memory supplies the OR instruction to the ICU and the address of the input to be used in the operation to the input selectors. The input selector then demultiplexes the addressed data onto the ICU's bidirectional data line. The ICU then logically OR's this data with the content of the ICU's Result Register and returns the result of the operation to the Result Register.

The ICU also has an "or complement" instruction, abbreviated ORC, in the event complement logic is needed. The operation of this instruction is exactly like the OR instruction *except* the incoming data is complemented before the OR operation is performed. Figure 3.5 shows some examples where the OR and ORC instructions may be used.

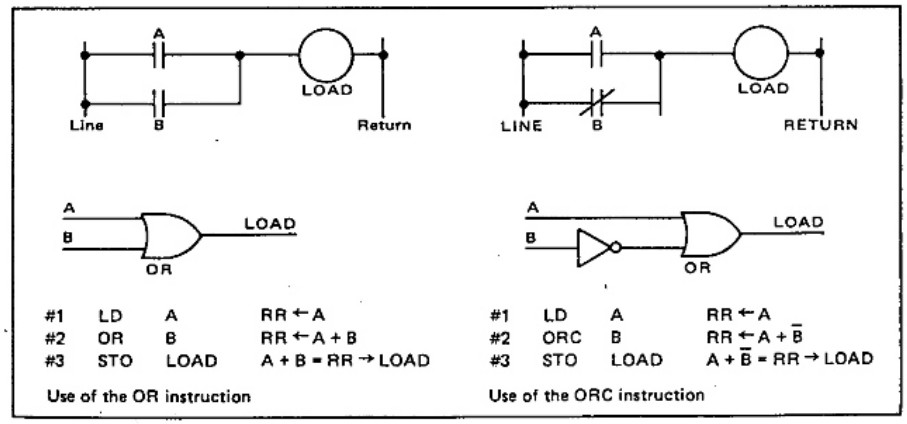


Figure 3.5 Use of the OR and ORC Instructions

In the example of using the OR instruction, the load device will receive a logic 1 signal if the A or B or both inputs are in the logic 1 state. In the example of the ORC instruction, the load device receives a logic 1 signal if the A input is in the logic 1 state or the B input is in the logic 0 state.

Use of Temporary Locations

Many of the logic structures found in the controls industry are branches of several series relays, in parallel with another branch of series relays. Figure 3.6 shows an example of this structure.

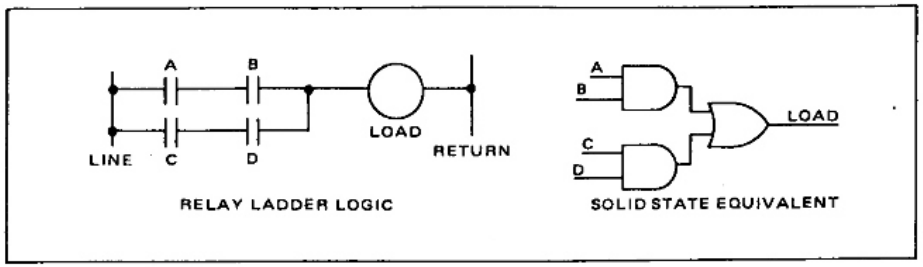


Figure 3.6 Series-Parallel Combinations

When dealing with this type of problem, it is not always possible to directly "chain" a series of LD, LDC, AND, ANDC, OR, and ORC instructions together to correctly evaluate the logic function required. In some cases, it may be necessary to temporarily store the intermediate results before processing the remainder of the problem. In these cases, the programmer must evaluate the series branches using LD, AND, and ANDC instructions as necessary to evaluate the expression and then store the result in a temporary location. The second series branch must then be evaluated and ORed with the data saved in the temporary location. The result of this operation should then be used to activate or deactivate the load device. Figure 3.7 shows a common error in programming this type of problem and Figure 3.8 describes and the correct approach to the problem. Figure 3.8 shows the correct method for solving this problem by using a temporary storage location.

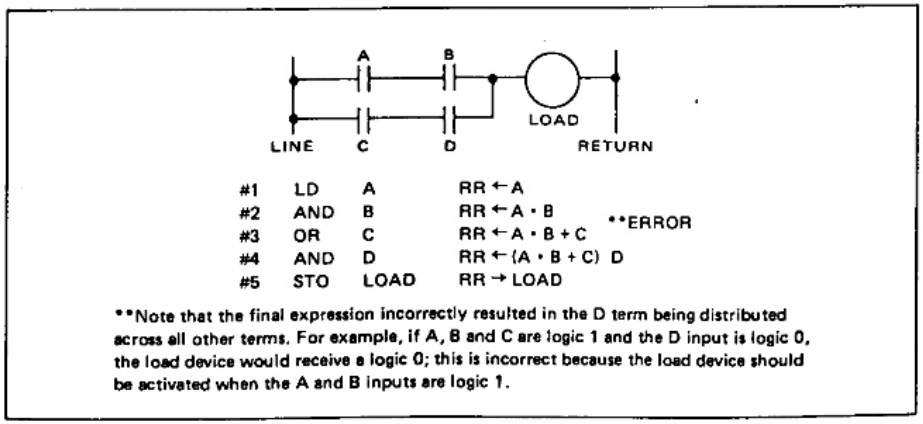
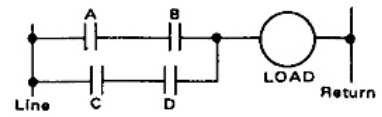


Figure 3.7 Example of Incorrect Programming




```
#1   LD     A       RR ← A
#2   AND    B       RR ← A • B
#3   STO    TEMP    RR = A • B → TEMP
#4   LD     C       RR ← C
#5   AND    D       RR ← C • D
#6   OR     TEMP    RR ← C • D + (TEMP = A • B) = A • B + C • D
#7   STO    LOAD    RR = A • B + C • D → LOAD
```

In this program, the logical result of ANDing A and B is stored temporarily, then the logical AND of C and D is ORed with the data previously stored in the temporary location. The correct logical signal is then transferred to the load device. This example demonstrates the need for temporary storage locations before proceeding.

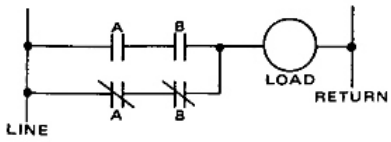
Figure 3.8 Correct Method of Solving the Problem

The XNOR Instruction

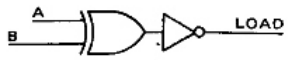
The "exclusive nor" instruction, abbreviated (XNOR), is the final logical instruction in the ICU's repertoire of logical instructions. The XNOR instruction can be thought of as a "match" instruction. That is, whenever the input data is identical to the data in the Result Register, the new content of the Result Register will be a logic 1. Figure 3.9 shows the truth table for the XNOR function and Figure 3.10 an example using the XNOR function. Note the reduction in code that may result from the use of this instruction.

Input Data	Old Result Register Data	New Result Register Data
0	0	1
0	1	0
1	0	0
1	1	1

Figure 3.9 XNOR Truth Table



EQUALS

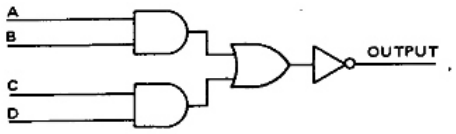



```
#1   LD     A
#2   AND    B
#3   STO    TEMP
#4   LDC    A          EQUALS        LD     A
#5   ANDC   B                        XNOR   B
#6   OR     TEMP                     STO    LOAD
#7   STO    LOAD
```

Figure 3.10 Example of use of XNOR Instruction

The STOC Instruction

When transferring a signal to activate a load device, it is very useful to be able to store the logical complement of an expression. The ICU therefore has a "store complement" instruction, abbreviated (STOC). The STOC instruction is exactly like the store (STO) instruction, except the logical complement of the Result Register is transferred to the output latch. It should be pointed out that the Result Register retains its original value (i.e. the STOC does not change the Result Register value, it merely transfers the complement of the Result Register to the bidirectional data line for routing to the output latches). This instruction is quite useful when dealing with negative logic or so called "low active" devices. Figure 3.11 shows an example usage of the STOC instruction. Figure 3.12 shows a problem in both the relay ladder and logic formats. Figure 3.13 shows the problem reduced to code.




```
#1   LD     A       RR ← A
#2   AND    B       RR ← A • B
#3   STO    TEMP    A • B → TEMP
#4   LD     C       RR ← C
#5   AND    D       RR ← C • D
#6   OR     TEMP    RR ← C • D + A • B
#7   STOC   OUTPUT  A • B + C • D → OUTPUT
```

Figure 3.11 Example of the STOC Instruction

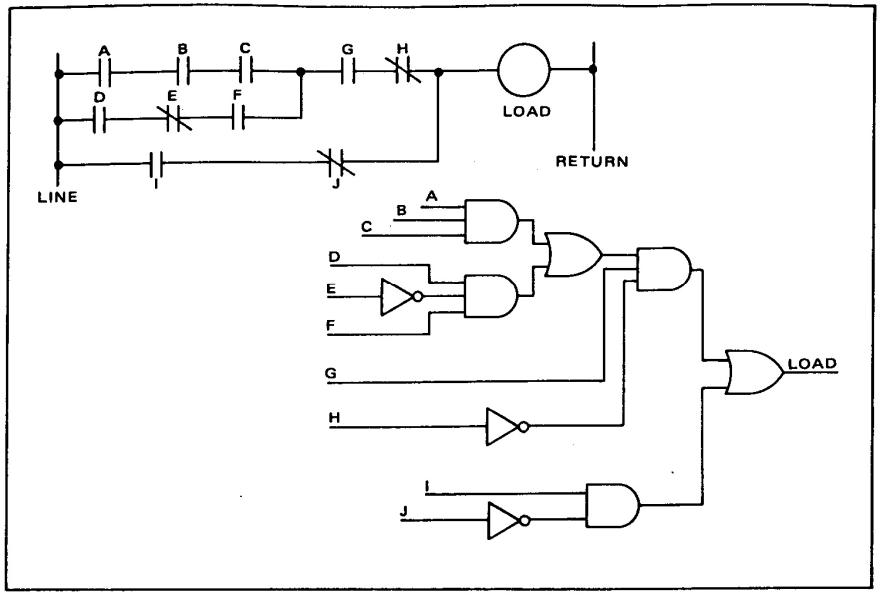


Figure 3.12 Complex Problem

#1	LD	A	RR ← A
#2	AND	B	RR ← A · B
#3	AND	C	RR ← A · B · C
#4	STO	TEMP	A · B · C → TEMP
#5	LD	D	RR ← D
#6	ANDC	E	RR ← D · $\overline{E}$
#7	AND	F	RR ← D · E · F
#8	OR	TEMP	RR ← A · B · C + D · E · F
#9	AND	G	RR ← (A · B · C + D · E · F) · G
#10	ANDC	H	RR ← (A · B · C + D · E · F) · G · $\overline{H}$
#11	STO	TEMP	(A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ ← TEMP
#12	LD	I	RR ← I
#13	ANDC	J	RR ← I · $\overline{J}$
#14	OR	TEMP	RR ← (A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ + I · $\overline{J}$
#15	STO	LOAD	(A · B · C + D · $\overline{E}$ · F) · G · $\overline{H}$ + I · $\overline{J}$ → LOAD

Figure 3.13 Complex Example Problem Code