18. INSTRUCTION LIST PROGRAMMING

18.1 INTRODUCTION

Instruction list (IL) programming is defined as part of the IEC 61131 standard. It uses very simple instructions similar to the original mnemonic programming languages developed for PLCs. (Note: some readers will recognize the similarity to assembly language programming.) It is the most fundamental level of programming language - all other programming languages can be converted to IL programs. Most programmers do not use IL programming on a daily basis, unless they are using hand held programmers.

18.2 THE IEC 61131 VERSION

To ease understanding, this chapter will focus on the process of converting ladder logic to IL programs. A simple example is shown in Figure 18.1 using the definitions found in the IEC standard. The rung of ladder logic contains four inputs, and one output. It can be expressed in a Boolean equation using parentheses. The equation can then be directly converted to instructions. The beginning of the program begins at the \textit{START} label. At this point the first value is loaded, and the rest of the expression is broken up into small segments. The only significant change is that \textit{AND NOT} becomes \textit{AND N}.

Topics:

- Objectives:
  - To learn the fundamentals of IL programming.
  - To understand the relationship between ladder logic and IL programs

Instruction list (IL) opcodes and operations

Converting from ladder logic to IL

Stack oriented instruction delay

The Allen Bradley version of IL
An important concept in this programming language is the stack. (Note: if you use a calculator with RPN you are already familiar with this.) You can think of it as a do later list. With the equation in Figure 18.1 the first term in the expression is LD I:000/00, but the first calculation should be (I:000/02 AND NOT I:000/03). The instruction values are pushed on the stack until the most deeply nested term is found. Figure 18.2 illustrates how the expression is pushed on the stack. The LD instruction pushes the first value on the stack. The next instruction is an AND, but it is followed by a '(' so the stack must drop down. The OR( that follows also has the same effect. The ANDN instruction does not need to wait, so the calculation is done immediately and a result_1 remains. The next two ')' instructions remove the blocking '(' instruction from the stack, and allow the remaining OR I:000/1 and AND I:000/0 instructions to be done. The final result should be a single bit result_3. Two examples follow given different input conditions. If the final result in the stack is 0, then the output ST O:001/0 will set the output, otherwise it will turn it off.

Label
START:
Opcode
LD
AND(
OR(
ANDN)
)
ST
Operand
%I:000/00
%I:000/01
%I:000/02
%I:000/03
%O:001/00
Comment
(* Load input bit 00 *)
(* Start a branch and load input bit 01 *)
(* Load input bit 02 *)
(* Load input bit 03 and invert *)
(* SET the output bit 00 *)

read as O:001/00 = I:000/00 AND ( I:000/01 OR ( I:000/02 AND NOT I:000/03) )
Figure 18.2 Using a Stack for Instruction Lists

A list of operations is given in Figure 18.3. The modifiers are:

- **N** - negates an input or output
- **(** - nests an operation and puts it on a stack to be pulled off by 

<table>
<thead>
<tr>
<th>Operators</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Integer</td>
</tr>
<tr>
<td>(</td>
<td>Integer</td>
</tr>
<tr>
<td>C</td>
<td>Integer</td>
</tr>
</tbody>
</table>

These operators can use multiple data types, as indicated in the data types column.

This list should be supported by all vendors, but additional functions can be called using the **CAL** function.

Given:

```plaintext
I:000/0 = 1  I:000/1 = 0  I:000/2 = 1  I:000/3 = 0
```

Result:

```plaintext
1
```

Given:

```plaintext
I:000/0 = 0  I:000/1 = 1  I:000/2 = 0  I:000/3 = 1
```

Result:

```plaintext
0
```
Allen Bradley only supports IL programming on the Micrologix 1000, and does not plan to support it in the future. Examples of the equivalent ladder logic and IL programs are shown in Figure 18.4 and Figure 18.5. The programs in Figure 18.4 show different variations when there is only a single output. Multiple IL programs are given where available. When looking at these examples recall the stack concept. When a `LD` or `LDN` instruction is encountered it will put a value on the top of the stack. The `ANB` and `ORB` instructions will remove the top two values from the stack, and replace them with a single value that is the result of a Boolean operation. The `AND` and `OR` functions take one value off the top of the stack, perform a Boolean operation and put the result on the top of the stack. The equivalent programs (to the right) are shorter and will run faster.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>set current result to value</td>
</tr>
<tr>
<td>ST</td>
<td>store current result to location</td>
</tr>
<tr>
<td>S, R</td>
<td>set or reset a value (latches or flip-flops)</td>
</tr>
<tr>
<td>AND, &amp;</td>
<td>boolean and</td>
</tr>
<tr>
<td>OR,</td>
<td>boolean or</td>
</tr>
<tr>
<td>XOR</td>
<td>boolean exclusive or</td>
</tr>
<tr>
<td>ADD</td>
<td>mathematical add</td>
</tr>
<tr>
<td>SUB</td>
<td>mathematical subtraction</td>
</tr>
<tr>
<td>MUL</td>
<td>mathematical multiplication</td>
</tr>
<tr>
<td>DIV</td>
<td>mathematical division</td>
</tr>
<tr>
<td>GT</td>
<td>comparison greater than &gt;</td>
</tr>
<tr>
<td>GE</td>
<td>comparison greater than or equal &gt;=</td>
</tr>
<tr>
<td>EQ</td>
<td>comparison equals =</td>
</tr>
<tr>
<td>NE</td>
<td>comparison not equal &lt;&gt;</td>
</tr>
<tr>
<td>LE</td>
<td>comparison less than or equals &lt;=</td>
</tr>
<tr>
<td>LT</td>
<td>comparison less than &lt;</td>
</tr>
<tr>
<td>JMP</td>
<td>jump to LABEL</td>
</tr>
<tr>
<td>CAL</td>
<td>call subroutine NAME</td>
</tr>
<tr>
<td>RET</td>
<td>return from subroutine call</td>
</tr>
<tr>
<td>N</td>
<td>get value from stack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>many</td>
</tr>
<tr>
<td>N</td>
<td>BOOL</td>
</tr>
<tr>
<td>N, (</td>
<td>BOOL</td>
</tr>
<tr>
<td>N, (</td>
<td>BOOL</td>
</tr>
<tr>
<td>(</td>
<td>BOOL</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>many</td>
</tr>
<tr>
<td>(</td>
<td>LABEL</td>
</tr>
<tr>
<td>(</td>
<td>NAME</td>
</tr>
</tbody>
</table>
Figure 18.5 shows the IL programs that are generated when there are multiple outputs. This often requires that the stack be used to preserve values that would be lost normally.

A X
LD A
ST X
A X LDN A
ST X
A X LD A
LD B
ANB
ST X
B LD A
AND B
ST X
A X LD A
LD B
LD C
ORB
ANB
ST X
B LD A
OR B
AND C
ST X
A X LD A
LD B
ORB
LD C
LD D
ORB
ANB
ST X
B LD A
OR B
LD C
OR D
ANB
ST X
C
D
B
plc il - 18.6

mally using the
MPS
MPP
and
MRD
functions. The
MPS
instruction will store the current
value of the top of the stack. Consider the first example with two outputs, the value of
A
is
loaded on the stack with
LD A
. The instruction
ST X
examines the top of the stack, but
does not remove the value, so it is still available for
ST Y
. In the third example the value of
the top of the stack would not be correct when the second output rung was examined. So,
when the output branch occurs the value at the top of the stack is copied using
MPS
, and
pushed on the top of the stack. The copy is then ANDed with
B
and used to set
X
. After
this the value at the top is pulled off with the
MPP
instruction, leaving the value at the top
what is was before the first output rung. The last example shows multiple output rungs.
Before the first rung the value is copied on the stack using MPS. Before the last rung the
value at the top of the stack is discarded with the
MPP
instruction. But, the two center
instructions use
MRD
to copy the right value to the top of the stack - it could be replaced
with
MPP
then
MPS
.
Figure 18.5 IL Programs for Multiple Outputs

Complex instructions can be represented in IL, as shown in Figure 18.6. Here the function are listed by their mnemonics, and this is followed by the arguments for the functions. The second line does not have any input contacts, so the stack is loaded with a true.

- LD A
- ST X
- ST Y
- LD B
- ANB
- ST Y
- LD A
- ST X
- AND B
- ST Y
- LD A
- MPS
- LD B
- ANB
- ST X
- MPP
- LD C
- ANB
- ST Y
- LD A
- MPS
- AND B
- ST X
- MPP
- AND C
- ST Y
- LD A
- MPS
- LD B
- ANB
- ST W
- MRD
- LD C
- ANB
- ST X
- MRD
- ST Y
- MPP
- LD E
- ANB
- ST Z
- LD A
- MPS
- AND B
- ST W
- MRD
- AND C
- ST X
- MRD
- ST Y
- MPP
- AND E
- ST Z
An example of an instruction language subroutine is shown in Figure 18.7. This program will examine a BCD input on card I:000, and if it becomes higher than 100 then 2 seconds later output O:001/00 will turn on.

```
START
LD I:001/0
TON(T4:0, 1.0, 5, 0)
LD 1
ADD (3, T4:0.ACC, N7:0)
END
```
Figure 18.7 An Example of an IL Program

18.4 SUMMARY
Ladder logic can be converted to IL programs, but IL programs cannot always be converted to ladder logic.

IL programs use a stack to delay operations indicated by parentheses.

<table>
<thead>
<tr>
<th>Label</th>
<th>TEST:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>LD</td>
</tr>
<tr>
<td>Opcode</td>
<td>BCD_TO_INT</td>
</tr>
<tr>
<td>Opcode</td>
<td>ST</td>
</tr>
<tr>
<td>Opcode</td>
<td>GT</td>
</tr>
<tr>
<td>Opcode</td>
<td>JMPC</td>
</tr>
<tr>
<td>Opcode</td>
<td>CAL</td>
</tr>
<tr>
<td>Opcode</td>
<td>LD</td>
</tr>
<tr>
<td>Opcode</td>
<td>ST</td>
</tr>
<tr>
<td>Opcode</td>
<td>CAL</td>
</tr>
<tr>
<td>Opcode</td>
<td>LD</td>
</tr>
<tr>
<td>Opcode</td>
<td>ST</td>
</tr>
<tr>
<td>Opcode</td>
<td>RET</td>
</tr>
</tbody>
</table>

| Operand | %I:000 | %N7:0 | 100 | ON | RES(C5:0) | 2 | %C5:0.PR | TON(C5:0) | %C5:0.DN | %O:001/00 |

| Comment  | (* Load the word from input card 000 *) | (* Convert the BCD value to an integer *) | (* Store the value in N7:0 *) | (* Check for the stored value (N7:0) > 100 *) | (* If true jump to ON *) | (* Reset the timer *) | (* Load a value of 2 - for the preset *) | (* Store 2 in the preset value *) | (* Update the timer *) | (* Get the timer done condition bit *) | (* Set the output bit *) | (* Return from the subroutine *) |

Program File 3:

<table>
<thead>
<tr>
<th>Label</th>
<th>START:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>CAL</td>
</tr>
<tr>
<td>Operand</td>
<td>3</td>
</tr>
</tbody>
</table>

| Comment  | (* Jump to program file 3 *) |

Program File 2:
The Allen Bradley version is similar, but not identical to the IEC 61131 version of IL.

18.5 PRACTICE PROBLEMS

1. Explain the operation of the stack.
2. Convert the following ladder logic to IL programs.
3. Write the ladder diagram programs that correspond to the following Boolean programs.

AC X
BCD
BC
Y

LD 001
OR 003
LD 002
OR 004
AND LD
LD 005
OR 007
AND 006
OR LD
OUT 204

LD 001
AND 002
LD 004
AND 005
OR LD
OR 007
LD 003
OR NOT 006
AND LD
OR NOT 008
OUT 204
AND 009
OUT 206
AND NOT 010
OUT 201