ANILAM

Integral Programmable Intelligence User's Guide

www.anilam.com

Warranty

ANILAM warrants its products to be free from defects in material and workmanship for one (1) year from date of installation. At our option, we will repair or replace any defective product upon prepaid return to our factory.

This warranty applies to all products when used in a normal industrial environment. Any unauthorized tampering, misuse, or neglect will make this warranty null and void.

Under no circumstances will ANILAM, any affiliate, or related company assume any liability for loss of use or for any direct or consequential damages.

The foregoing warranties are in lieu of all other warranties expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

The information in this manual has been thoroughly reviewed and is believed to be accurate. ANILAM reserves the right to make changes to improve reliability, function, or design without notice. ANILAM assumes no liability arising out of the application or use of the product described herein.

Section 1 - Introduction

Section 2 - Software

Section 3 - Working with IPI

Section 4 - Writing IPI Programs

Section 5 - Timers

Section 6 - Advanced IPI Instructions

Section 7 - Programming Tips and Examples

Index

Section 1 - Introduction

Traditionally, inputs and outputs between the CNC and the machine required numerous relays to switch signals between the CNC and the hardware. The relay logic was hardwired and depicted with ladder diagrams. These relays consumed power, were subject to failure, and required hardware reconfiguration to change.

More recently, the programmable controller, an add-on device, replaced relays with solid-state circuitry. The programmable controller design solved the problems associated with relays. It generated a faster response and was programmable and more flexible. However, it was still a physical add-on; it required cabinet space and drew power.

Therefore, Integral Programmable Intelligence (IPI), a software package that runs in the background of the CNC, was developed and added to the CNC. IPI monitors inputs, from the control panel and machine switches, through the standard CAN Bus I/O system. When conditions are correct, the IPI directs the I/O system to generate the appropriate output, hence the term "conditional logic."

Because IPI is integral to the CNC, it requires no additional hardware space or power. IPI is compatible with the CNC's existing CAN Bus I/O Board.

Most inputs and outputs to IPI are digital and can be thought of as true/false, active/inactive, on/off. The program loaded at machine setup determines the combination or sequence of events required to generate an output.

The IPI instruction set enables implementation of basic Boolean functions, timed functions, sequenced functions, and conditional expressions. This results in a high degree of versatility.

Later machine modifications require changes only to the program. The program is written with the same text Editor that is used to write G-code part programs for the CNC. Ladder diagrams are easily translated into IPI code.

Section 2 - Software

The CNC software provides a simple environment for the development of IPI programs. The environment allows IPI program developers to:

- ❑ Create a program
- ❑ Select a specific IPI program
- ❑ Edit the program
- ❑ Compile and load the program
- ❑ Manage IPI programs.

IPI programs are standard text files that you can develop with any text editor. The IPI environment provides you with program management utilities, which include tools such as program copying, printing, deleting, and restoring. Refer to "Section 3 - Working with IPI" for more details.

CAN I/O Board

The standard OEM product contains two CAN I/O Boards configured as **source** or **sink**. The system supports a maximum of four additional boards (six boards total). The machine builder assigns a unique number to each CAN Node (0–5).

NOTE: 3300M/MK systems support a maximum of two I/O boards.

Each I/O Board is a node that accepts inputs and generates outputs as required. You can configure each node as either all Digital or Digital/Analog. When configured for Digital, the I/O board has 10 digital inputs and six digital outputs. When configured for Digital/Analog, the I/O board has one analog input, ten digital inputs, and five digital outputs.

The machine builder hardwires the required inputs and outputs to the P5 (DB 25-pin) connector on each board as shown in **Table 2-1**.

Table 2-1, P5 Inputs and Outputs

Pin	Signal Names	Pin	Signal Names
1	INPUT 0	14	OUTPUT 0
2	INPUT 1	15	OUTPUT 1
3	INPUT 2	16	OUTPUT 2
4	INPUT 3	17	OUTPUT 3
5	INPUT 4	18	OUTPUT 4
6	INPUT 5	19	OUTPUT 5
7	INPUT 6	20	ANALOG IN
8	INPUT 7	21	NC
9	INPUT 8	22	NC
10	INPUT 9	23	24 COMMON
11	24 V COMMON	24	+24 V
12	+24 V	25	NC
13	NC		

For more information on CAN I/O Boards, refer to the *OEM CNC Installation*, P/N 70000506.

Inputs

Format **Xn:b** where:

- X indicates Input
- n indicates Node # (range: 0 to 5)
- b indicates Bit # (range: 0 to 9)

You can identify inputs by the physical location of the input to the CAN I/O Board. See **Table 2-2**. The IPI stores the condition or state of each input (on/off, true/false) in a state memory register using the same designation.

Table 2-2, Input Locations

Input	Location
Vector Limits and Home Limits	Always located on CAN Node 0, if used. The CNC reads these inputs from travel-limit switches.
General Purpose Inputs	Located on indicated CAN Node (0 to 5).

Inputs must be hardwired to the P5 DB-25 connector of the appropriate CAN Node (board). Refer to **Table 2-3** for the required inputs and outputs to the P5 (DB 25-pin) connector on each board.

Table 2-3, Input Type Descriptions

Input	Type	Location
X0:0–X0:7	Vector Limits (if used; otherwise, General Purpose Inputs)	CAN Node 0
X0:8–X0:9	General Purpose Inputs	CAN Node 0
X1:0–X1:9	General Purpose Inputs	CAN Node 1
X2:0–X2:9	General Purpose Inputs	CAN Node 2
X3:0–X3:9	General Purpose Inputs	CAN Node 3
X4:0–X4:9	General Purpose Inputs	CAN Node 4
X5:0–X5:9	General Purpose Inputs	CAN Node 5

Outputs

Format **Yn:b** where:

- Y indicates Output
- n indicates Node #; range of n = 0 to 5
- b indicates Bit #; range of b = 0 to 5

An output is an electrical signal generated by the board. You can identify an output by the physical location of the output to the CAN I/O Board. The IPI stores the condition or state of each output in output memory registers identified by the Y designator of the same name. The IPI uses the output states to generate electrical signals once every IPI computation cycle. See **Table 2-4**.

Table 2-4, Output Type Descriptions

<table>
<tr><td colspan="3"><b>NOTE:</b> Outputs must be hardwired to the P5 DB-25 connector of the appropriate CAN Node (board). Refer to Table 2-1, P5 Inputs and Outputs.</td></tr>
<tr><th>Output</th><th>Type</th><th>Location</th></tr>
<tr><td>Y0:0–Y0:5</td><td>General Purpose Outputs</td><td>CAN Node 0</td></tr>
<tr><td>Y1:0–Y1:5</td><td>General Purpose Outputs</td><td>CAN Node 1</td></tr>
<tr><td>Y2:0–Y2:5</td><td>General Purpose Outputs</td><td>CAN Node 2</td></tr>
<tr><td>Y3:0–Y3:5</td><td>General Purpose Outputs</td><td>CAN Node 3</td></tr>
<tr><td>Y4:0–Y4:5</td><td>General Purpose Outputs</td><td>CAN Node 4</td></tr>
<tr><td>Y5:0–Y5:5</td><td>General Purpose Outputs</td><td>CAN Node 5</td></tr>
</table>

The IPI Operation Cycle

The following, is a description of the IPI operation cycle:

1. Upon activation, IPI clears all memory registers and resets all internal timers.
2. The IPI executes any initialization instructions that appear before the program START.
3. At START, the IPI samples all inputs and saves the states in the memory registers. During the current cycle, the interpreter assesses the values stored in the input registers. This prevents interruption of a cycle in progress by a sudden change.
4. As the interpreter runs, it determines the states of the outputs and stores these states in the output memory registers.
5. At program END, the interpreter finishes. The IPI instructs the I/O system to generate outputs, as indicated by the states stored in the output registers.
6. The IPI cycles back to START. All old data remain in memory, unless updated from input state changes that occurred since the last sampling cycle.

Memory Registers

IPI uses two kinds of memory registers: Boolean registers, which store only true/false states, and numeric registers, which hold integer values. The numeric registers allow IPI to perform timing, counting, and comparison operations.

Inputs and outputs are IPI elements that use similarly designated registers. Additional types of memory registers include:

- ❑ Multifunctional Registers
- ❑ Timers
- ❑ Sequence Registers

Refer to **Table 2-5**.

- ❑ Numeric values greater than **0** (zero) become **TRUE** in a state-only register.
- ❑ A numeric value of **0** (zero) becomes **FALSE** in a state-only register.
- ❑ A state value of **TRUE** becomes a **1** in a numeric register.
- ❑ A state value of **FALSE** becomes a **0** (zero) in a numeric register.

Table 2-5, Register Capabilities

Register Type	Numeric Values	State Values
Inputs – X Identifiers		X
Outputs – Y Identifiers		X
Sequence Outputs – S Identifiers		X
Multifunction Registers – M Identifiers	X	X
Timer Registers – T Identifiers	X	X

Multifunction Registers

IMPORTANT: Multifunction registers M0–M63 have reserved specialized functions. The IPI programmer has access to multifunction registers M64–M255 for general-purpose use.

Format **Mn, n** is a number 0 to 255.

Multifunction memory registers are general-purpose registers that have several uses. The IPI assigns M numbers to multifunction registers. There are 256 multifunction registers available, numbered M0–M255. Multifunction registers M0–M63 are reserved. Multifunction registers M64–M255 are available for the intermediate storage of a value or state. You can use the value stored in a multifunction register in an instruction like any other parameter. Most reserved multifunction registers also have a permanently assigned label. Multifunction registers have no permanent board address. To output the value stored in a multifunction register, the IPI must send the value to an IPI output.

Multifunction registers can store Boolean true/false states or numeric values.

M0–M32 are generated by the CNC and can be considered CNC inputs to the IPI. The IPI uses these registers to generate readings on the display. The information stored in these registers is available on a Read Only basis. Refer to **Table 2-6**.

Table 2-6, Assigned Read Only Multifunction Registers

M Designator	Assigned Label	Purpose
M0	SPINDLE	True when not probing and spindle may run.
M1	POSN	True when CNC is in position.
M2	PRBFLAG	Probing flag is active during G31 primitive moves and probing cycles.
M3	PWRFAIL	True if +24V is on.
M4	FEED	Feed mode flag.
M5	SVOFF	True if servo is off.
M6	ESTOP	True if E-Stop is out.
M7		NOT USED
M8	CARRY	Carry Flag/Register.
M9	TRUE	Always ON.
M10	FALSE	Always OFF.
M11		NOT USED
M12	TCFINACK	Tool changer finished received.
M13	HOME	True when Z or XYZ at home.
M14	SPLOOP	True when spindle in closed-loop mode.
M15	RUN	True when CNC in RUN mode.

(Continued…)

Table 2-6, Assigned Read Only Multifunction Registers (Continued)

M Designator	Assigned Label	Purpose
M16	MAN	True when CNC in MANUAL mode.
M17	MFLAG	True when new M code is received.
M18	MCODE	M code number.
M19	SFLAG	True when new spindle code is received.
M20	SCODE	Spindle number.
M21	TFLAG	True when new tool number is received.
M22	TCODE	Tool number.
M23	HFLAG	True when new H-code number is received (tool pre-set code).
M24	HCODE	H-code number.
M25	---	Reserved.
M26	TMACEND	Tool Macro end flag.
M27	ZMACHPOS	Current Z-axis machine position in microns.
M28	ZEROSPD	This flag allows the system to know when the spindle RPM is "At-Zero." Flag is True when using spindle feedback and RPM ≤ Spindle zero speed RPM tolerance (specified in Spindle Setup); otherwise, False.
M29	ATSPD	This flag allows the system to know when the spindle RPM is "At-Speed." Flag is True if commanded RPM > 0 and percent of actual vs commanded RPM is > Spindle at speed percent (specified in Spindle Setup); otherwise, flag is False.
M30 – M31	---	Reserved.
M32	XMIT	True when IPI accepts CNC data.

The IPI generates and the CNC internally monitors M33 – M63. You can consider these registers inputs from the IPI to the CNC. This allows the IPI program to output to the CNC and generate on-screen messages. These messages are read and write registers. Refer to **Table 2-7**.

Table 2-7, Assigned Read/Write Multifunction Registers

Register	Assigned Label	Purpose
M33	FINISH	Set True to signal M, S, T, or H Finish. When M or S function is cleared, reset to false.
M34	SVOFLT	Set True to signal a servo fault.
M35	FHOLD	Set True to inhibit feed moves. Rapid moves will execute normally.
M36	TCHGFIN	Tool changer finished bit.
M37	XSTOP	Set True to hold CNC motion. Set to False to resume motion.
M38	XHOLD	Set True to stop CNC motion. Press the Start button to resume motion.

(Continued…)

Table 2-7, Assigned Read/Write Multifunction Registers (Continued)

Register	Assigned Label	Purpose
M39	MSG	Set any non-zero number to display message.
M40	---	Reserved.
M41	SPDAN0V	When SPDAN0V M41 is True (non-zero) the IPI disables the analog output to the spindle. When M41 is False (zero), the IPI does not affect the analog output to the spindle.
M42	MREGRAN	Used to cycle the multifunction registers displayed on the IPI monitor. Allows you to view M0–M256 by selecting a range of registers to be displayed.
M43	SPDGRCH	When M43 is in the range of 40–44, the CNC will enable the corresponding gear range. The gear range specified is used only for calculating a proportional spindle analog output voltage. When M43 is outside of the range 40–44, the CNC ignores this register.
M44	CNCERR	Used by the CNC to pass error conditions to the IPI.
M45		NOT USED
M46	KEYMASK	Used by the IPI program to mask out certain keys from the operator.
M47	SPIN100	When any non-zero number is written to this register, spindle analog voltage will be forced to 100% of the programmed value, regardless of the setting of the spindle percentage switch on the Manual Panel.
M48	SPDRPM	Used to set the spindle analog to a desired speed by placing the RPM value in the register. **NOTE:** Handle all gear change selections separately, using either the CNC program or the M43 SPDGRCH.
M49	SPDDIR	Used in conjunction with M48 to allow the IPI program to control the spindle.
M50	HOMING	Used to indicate when homing is in progress. The register will be set to 1 when homing is active; otherwise, it is set to 0.
M51	LNFDLIM	Linear Axis feed limit. If this register is nonzero, use the value as speed for linear axes.
M52	ROFDLIM	Rotary Axis feed limit. If this register is nonzero, use the value as speed for rotary axes.
M53	SPDVOLT	Spindle voltage. Outputs a value in 0.01-V increments; for example, if register value is 50, output would equal 0.5 V.
M54	CMDRPM	Commanded Spindle RPM. This is the S word multiplied by any spindle override.
M55	HWSTOP	Set register true to inhibit handwheel operation in all modes. Set to zero to allow moves.
M56	AUTOINH	Set to 1 to inhibit AUTO or S.Step mode. Set to 0 to enable AUTO or S.Step mode.

(Continued…)

Table 2-7, Assigned Read/Write Multifunction Registers (Continued)

Register	Assigned Label	Purpose
M57	FEED100	Set to 1 to force feedrate override to 100%. Set to zero to enable feedrate switch value.
M58	XSTART	External start. Operation is identical to input function and front panel key.
M59	TOOLNUM	Active tool number.
M60	TLOBIN0	Used in random tool changer applications to store the bin of the tool in the spindle.
M61	M19FLAG	Status of M19 operation: 0 Once spindle function M3, M4, or M5 is executed 1 During orientation 2 Once orientated
M62	M19END	Allows IPI to terminate an M19 (spindle orientation) sequence.
M63	SPRSTOP	Bitmask axis Super STOP. Stops all machine motion when set to non-zero on given axis. Motion continues when register is set to ZERO. Axis specified via bitmask (X=1, Y=2, Z=4, U=8, etc.)

M2-PRBFLAG

Read only register M2 is the probing flag and is active during G31 primitive moves and probing cycles. Bit 1 of PRBFLAG (M2) will be set during the G31 primitive. At the beginning of the probing cycles, Bit 2 of M2 will also be set, and reset at the end of those cycles. Thus, the IPI program can be structured dependant on probe status and requirements. The IPI programmer will use this register to logically decide what functions need to be active during probing, as opposed to normal machine operation. Examples of such functions are Feed Hold and axes Feed Rate Limits. Often, such limitations are put on the machine's operation when a guard is opened, or the spindle is not running. When probing and using the G31 primitive, use of the M2 probing flag allows these limitations to be properly enforced.

M41-SPDAN0V

When M41-SPDAN0V is True (non-zero), IPI will disable the analog output to the spindle. When M41 is False (zero), the analog output to the spindle is not affected.

For example, if the spindle is running at 1000 RPM and M41 is set to true, the corresponding voltage output to the spindle will be 0V. Once M41 is set to false (zero), the corresponding output to the spindle will be the same as it was before the M41 was set to True. If you use an S-word before setting M41 to False, the analog output will correspond to the newly programmed RPM.

M42-MREGRAN

M42-MREGRAN is a bitmask that allows you to cycle through the range of multifunction registers displayed on the IPI monitor. Currently, only M48–M63 are displayed on the IPI monitor. M42 allows you to change the range of displayed registers. You can display only 16 registers at one time. **Table 2-8** lists the available ranges.

Table 2-8, Available Multifunction Register Ranges Displayed on the IPI Monitor

Range No.	Mreg Range	Bitmask (Hex)	Bitmask (Binary)
1	M0–M15	0001h	0000000000000001
2	M16–M31	0002h	0000000000000010
3	M32–M47	0004h	0000000000000100
4	M48–M63	0008h	0000000000001000
5	M64–M79	0010h	0000000000010000
6	M80–M95	0020h	0000000000100000
7	M96–M111	0040h	0000000001000000
8	M112–M127	0080h	0000000010000000
9	M128–M143	0100h	0000000100000000
10	M144–M159	0200h	0000001000000000
11	M160–M175	0400h	0000010000000000
12	M176–M191	0800h	0000100000000000
13	M192–M207	1000h	0001000000000000
14	M208–M223	2000h	0010000000000000
15	M224–M239	4000h	0100000000000000
16	M240–M255	8000h	1000000000000000

Displaying Multiple Ranges

To display multiple ranges simultaneously, combine the necessary hexadecimal-bitmask values. When more than one range is displayed, the IPI monitor screen flips between ranges every five seconds.

For example, to display ranges 1 and 2, combine the bitmask values for Range 1 (0001h) and Range 2 (0002h) to get 0x0003 (MOV 00003H MREGRAN).

The default value is 10H, which displays Range 5 (M64–M79). To display all ranges, set M42 to FFFFh (MOV FFFFH MREGRAN).

M43-SPDGRCH

Set M43-SPDGRCH between 40–44 to enable the corresponding gear range. The specified range is used to calculate a proportional spindle analog output voltage only. When M43 is outside the range 40–44, the CNC ignores this register.

When M43 is not **0** (zero), the monitor screen displays **1** on the last bit of the PLC flags section. To see the actual value, using M42-MREGRAN to display the proper M-register range (Range 3, 0004h).

M44-CNCERR

The CNC uses M44-CNCERR to pass error conditions to IPI. The CNC can pass only one error (Set 1–4) at a time to IPI. To enable the CNC to pass another error, the IPI program must clear M44 (set to zero). Refer to **Table 2-9**.

Table 2-9, Error Condition Values

Condition	Value
File Read Error	1
File Write Error	2
Checksum Error	3
New File	4

M46-KEYMASK

The IPI program uses M46-KEYMASK to mask out certain keys from the operator. M46 contains a bit value; each bit corresponds to a key. Refer to **Table 2-10** for keys assigned to M46 bits.

Table 2-10, KEYMASK M46 Bit Numbers and Keys

Bit No.	CNC Key	Bitmask (Hex)	Bitmask (Binary)
1	Start	0001h	0000000000000001
2	Hold	0002h	0000000000000010
3	Spindle CW	0004h	0000000000000100
4	Spindle CCW	0008h	0000000000001000
5	Spindle OFF	0010h	0000000000010000
6	All Keyboard Input	0020h	0000000000100000

Combine the appropriate bitmask hex values to mask out multiple keys at once. For example, to mask out the Spindle CW, Spindle CCW, and Spindle OFF, combine 0004h (Spindle CW), 0x0008 (Spindle CCW), and 0x0010 (Spindle OFF) to get 1C. The **MOV 11100b M46** command converts the value to binary format and uses the appropriate base indicator. The command will mask out the specified spindle keys.

To enable the spindle keys to be used later in the program, clear M46 in a subsequent execution scan; for example, **MOV 0 M46**.

The command enables all previously masked keys.

M47-SPIN100

When any nonzero number is written to this register, spindle analog voltage will be forced to 100% of the programmed value, regardless of the setting of the % Spindle Override switch on the Manual panel.

M48-SPDRPM

Set the spindle analog to a desired speed by placing the RPM value in the register.

You must use any gear range selection separately, either by the CNC program, or by using M43-SPDGRCH. Additionally, the desired RPM must be in the range of allowed speeds, as specified in the Spindle Axis Setup utilities.

When a valid spindle RPM is written to this register, spindle rotation will begin. Default direction will be forward (M03). To stop rotation, the IPI must write a **0** to this register.

M49-SPDDIR

The IPI can pick the direction of spindle rotation by writing a **3** for forward (M03) or a **4** for reverse (M04). You can use this register in conjunction with M48 to allow IPI the responsibilities of spindle control.

M50-HOMING

When set by the CNC to 1, this register indicates a homing sequence is being processed. When the homing operation is complete, the CNC resets the register to 0.

M51-LNFDLIM

Linear Axis feed limit. When the IPI writes a number to this register, linear axes will run at the value stored in register M51. This value must be expressed in metric and will be executed in feed per minute (FPM) mode. The IPI supports Vector moves. The CNC feedrate override switch continues to operate normally. If the value in register M51 is **0**, the programmed value or defaults will be used. This feature is intended to be used as a safety feature in Manual mode. Do not use in Auto. The feedrates of all subsequent MDI and jog moves will be limited to the specified value after the value is assigned to the register.

NOTE: To change from one limited range to another, you must first reset to zero. You can reset a higher limit lower, directly, but not the opposite.

NOTE: Because the feedrate override switch operates normally, you must divide the maximum allowed speed by 120%, and use this value for M51.

M52-ROFDLIM

Rotary Axis feed limit. When the IPI writes a number to this register, rotary axes run at the value stored in register M52. You must express this value in degrees/minute. The IPI will execute it in FPM mode. The CNC feedrate override switch continues to operate normally. If the value in register M52 is **0**, the programmed value or defaults will be used. This feature is intended to be used as a safety feature in Manual mode. Do not use in Auto. The feedrates of all subsequent MDI and jog moves will be limited to the specified value after the value is assigned to the register.

NOTE: To change from one limited range to another, you must reset to zero first. You can reset a higher limit lower, directly, but not the opposite.

NOTE: Because the feedrate override switch operates normally, you must divide the maximum allowed speed by 120%, and use this value for M52.

M53-SPDVOLT

Spindle voltage outputs a value in 0.01–V increments. For example, if register value is **50**, output equals **0.5 V**. Use SPDVOLT along with SPDDIR to select the direction.

M54-CMDRPM

Commanded Spindle RPM from the CNC. This is the S-word multiplied by any spindle override switch settings. For example, S1000 with an 80% setting would yield an 800 value in register M54. Should be used to determine if spindle range errors, which IPI needs to act upon, are present.

M55-HWSTOP

Handwheel Stop. Set to true (nonzero) to stop handwheel operations. Set register to **0** (zero) to allow handwheel operations.

M56-AUTOINH

Set to **1** to inhibit AUTO or Single Step (S.STEP) mode. Set to **0** (zero) to enable AUTO or Single Step (S.STEP) mode.

M57-FEED100

Feed 100% Override. Set to **1** to force feedrate override to 100%. Set to **0** (zero) to enable feedrate switch value.

M58-XSTART

External Start. Operation is identical to the input function and front panel key.

P Registers

P (Parameter) registers store CNC Parameters set by the Setup Utilities. These registers are read-only to the IPI. You can use them as conditions in the IPI program with operands or in expressions.

P registers 1010 through 1019 are reserved to report the spindle speed ranges from the Setup parameters. In the IPI program, you can access the P-register number or the assigned label as described in **Table 2-11**.

Table 2-11, P Register Numbers and Assigned Labels

Register	Assigned Label	Purpose
P1010	M40LO	M40 - Open gear range low limit
P1011	M40HI	M40 - Open gear range high limit
P1012	M41LO	M41 - Gear range low limit
P1013	M41HI	M41 - Gear range high limit
P1014	M42LO	M42 - Gear range low limit
P1015	M42HI	M42 - Gear range high limit
P1016	M43LO	M43 - Gear range low limit
P1017	M43HI	M43 - Gear range high limit
P1018	M44LO	M44 - Gear range low limit
P1019	M44HI	M44 - Gear range high limit

General-Purpose, Multifunction Registers

M64–M255 are general-purpose, multifunction registers. They are read and write registers that store intermediate values for later use.

Shared Registers

The IPI and CNC share 16 M-registers. These are CNC variables #1100 to #1115, which correspond to IPI M-registers 224 to 239, respectively. These variables allow the IPI program and the CNC to exchange information by reading and writing back and forth in both programs.

Example 1 - IPI to CNC

The CNC program can read a value written in the IPI program.

IPI program command:	LD M55 M224	*Copies contents of M55 into M224. (Example: M55 = 4. Therefore, M224 = 4.)

Subsequently, in a CNC program:

CNC program block:	If (#1100 > 1)then print (Register 224, variable 1100)	*Since CNC variable #1100 corresponds to IPI variable M224, the CNC reads the value stored in the IPI program (M224 = 4:#1100 = 4). Since 4 is greater than 1, the CNC executes the command and prints, “Register 224, variable 1100”

Example 2 - CNC to IPI

The IPI program can read a value written in the CNC program.

CNC program block:	#1101 = 2	*Sets CNC variable #1100 to 2.

Can be used in the IPI program as in:

IPI program command:	IF 100 (M225 EQ 2)	*Since IPI M-register M225 corresponds to CNC variable #1101, the CNC reads the value stored in the CNC program (#1101 = 2. Therefore, M225 = 2.) In this case, M225 EQ 2 would be TRUE and the conditional instructions following the IF block would be executed.

Static M-registers - M240–M255

The CNC reserves a range of 16 M-registers (M240–M255) that you can use to store values you might need after a power-down condition.

The CNC saves the registers in a binary data file (IPIMREGS.DAT) located in the system directory. When you start the CNC software from the **Software Options** menu, the CNC reads the IPIMREGS.DAT file and restores M240–M255 to their previously saved values. The data contain a checksum to guard against corruption. If the CNC detects a corrupted IPIMREGS.DAT file, the M240–M255 registers revert to their default values (zero in all cases).

The CNC saves the registers every time a value within the range changes. To avoid excessive disk operations and slow program execution, do not program frequent value changes in the range. To monitor errors in reading and writing the data file, check M44.

Timer Registers

Format **Tn, n** is a number 0–63.

There are 64 timing registers available (T0–T63). The instruction that first uses a timer in a program configures it. Later references to the same timer are only to sample its state value.

NOTE: 3300M/MK systems have only 16 timers (T0–T15).

The time delay is expressed in decimal seconds, read by the interpreter in 0.1 seconds.

Each timer actually uses two registers: a state register and a time-keeping register. The RD instruction permits use of the countdown value when necessary. For further information, refer to "Section 6 - Advanced IPI Instructions." Timers have a minimum period of 0.1 seconds. The maximum period for timers is 24 days.

Sequence Registers

There are 256 sequence registers available. These are designated S0–S255. Sequence registers are also available to the programmer at any time. When any sequence register is set to a True value, all others are automatically reset to False.

For example, when the IPI program starts, sequence register S0 is always set to True. Therefore, all other sequence registers are False.

I/O Boards

Refer to **Figure 2-1** and **Figure 2-2**. The CAN I/O boards act as the switchboard for the I/O system. When the I/O Board generates an output, the output is a switched 24 V common from a sink board or +24 V DC from a source board. Outputs are rated to carry a load of 5 A. Most inputs from the system are received at the board and sent to the CNC for evaluation. Upon command from the IPI program, the CNC signals the I/O Board to generate an output at the specified location.

CAUTION: 24–V common is not a machine or earth ground. It is an isolated source.

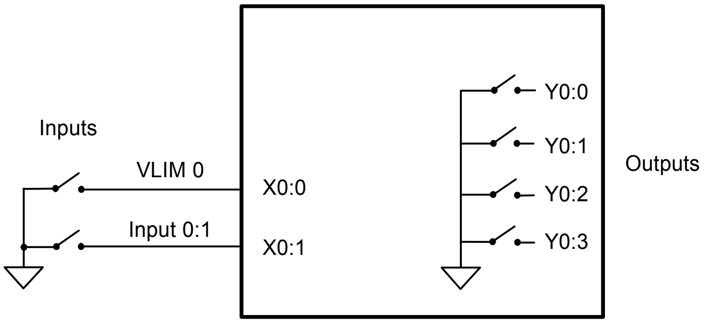


Figure 2-1, Sink I/O Board Input and Output Principles

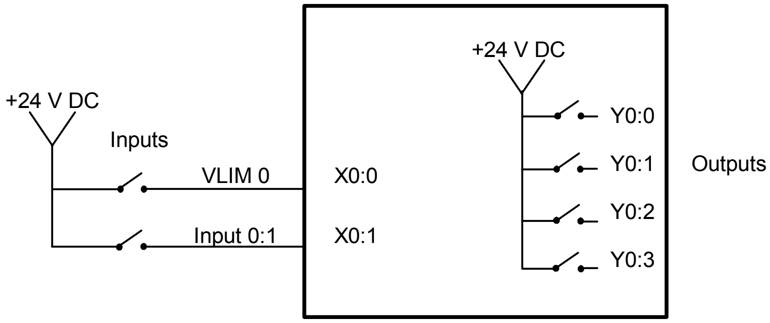


Figure 2-2, Source I/O Board Input and Output Principles

IPI Monitor

The state value stored in input registers (X0:0 – X5:9), output registers (Y0:0 – Y5:5), timer registers (T0–T64), and multifunction registers (M0–M255) can all be viewed from the IPI Monitor.

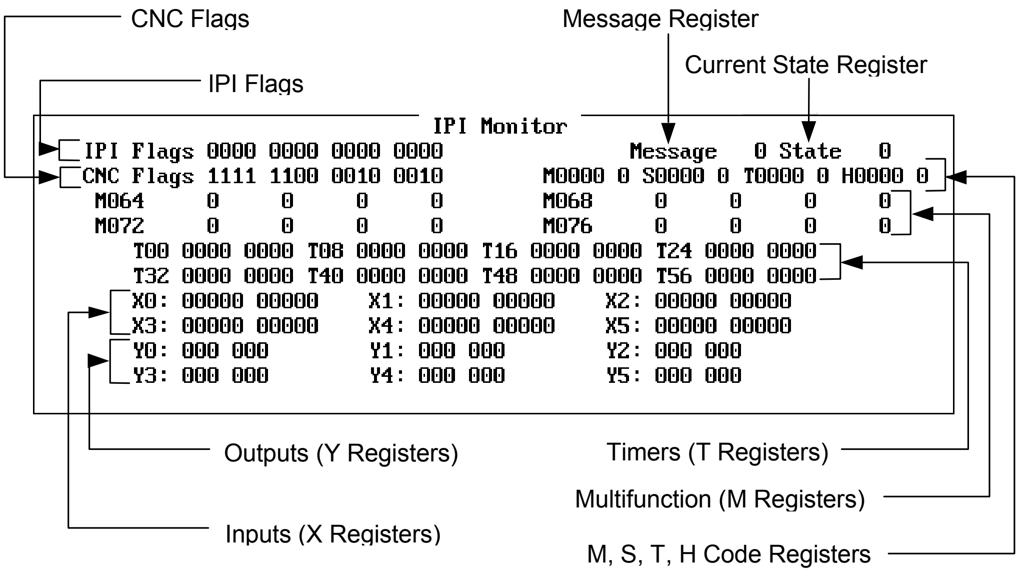
Viewing the IPI Monitor

To access the **IPI Monitor** screen, perform the following steps:

NOTE: On 3300M/MK IPI systems use the QWERTY keyboard or alternately press the **+/-** key in Manual, Auto, or Single Step mode.

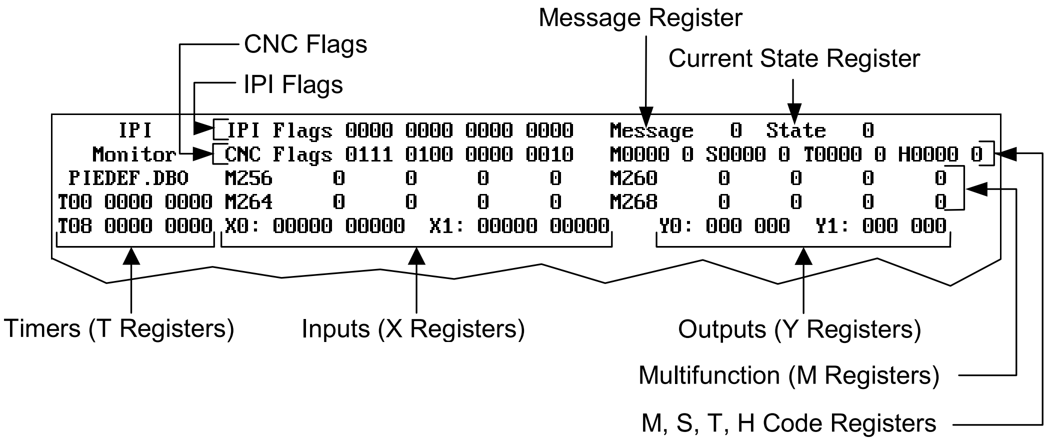
1. From the CNC software's Manual mode, press **P**.
2. Press **ENTER** to display the **IPI Monitor** screen. Refer to either **Figure 2-3** or Figure 2-4, 3300M/MK IPI Monitor Screen.

NOTE: In Auto or S.Step, press **P** to activate the IPI Monitor screen.



```
                                   IPI Monitor
 IPI Flags 0000 0000 0000 0000                    Message   0 State   0
 CNC Flags 1111 1100 0010 0010          M0000 0 S0000 0 T0000 0 H0000 0
  M064     0    0    0    0             M068     0     0     0     0
  M072     0    0    0    0             M076     0     0     0     0
    T00 0000 0000 T08 0000 0000 T16 0000 0000 T24 0000 0000
    T32 0000 0000 T40 0000 0000 T48 0000 0000 T56 0000 0000
    X0: 00000 00000    X1: 00000 00000    X2: 00000 00000
    X3: 00000 00000    X4: 00000 00000    X5: 00000 00000
    Y0: 000 000        Y1: 000 000        Y2: 000 000
    Y3: 000 000        Y4: 000 000        Y5: 000 000
```

Figure 2-3, 4200T, 5300M/MK, 5400M/MK, and 5500M IPI Monitor Screen



```
     IPI       IPI Flags 0000 0000 0000 0000   Message    0  State    0
   Monitor     CNC Flags 0111 0100 0000 0010   M0000 0 S0000 0 T0000 0 H0000 0
 PIEDEF.DBO    M256      0     0     0     0   M260     0     0     0     0
T00 0000 0000  M264      0     0     0     0   M268     0     0     0     0
T08 0000 0000  X0: 00000 00000  X1: 00000 00000   Y0: 000 000  Y1: 000 000
```

Figure 2-4, 3300M/MK IPI Monitor Screen

The IPI Monitor displays the state values, **1** for True and **0** for False, for the following registers:

- ❑ M registers
- ❑ X registers
- ❑ Y registers
- ❑ T registers

The IPI Monitor displays the numeric values for the following registers:

- ❑ M, S, T, and H codes from the CNC
- ❑ Message registers from the IPI to the CNC
- ❑ Current register's value

Refer to Figure 2-5, CNC Flags for CNC flags from the CNC to IPI. These are **Read Only** registers. Refer to Figure 2-6, IPI Flags for IPI Flags from IPI to the CNC. These are **Read/Write** registers.

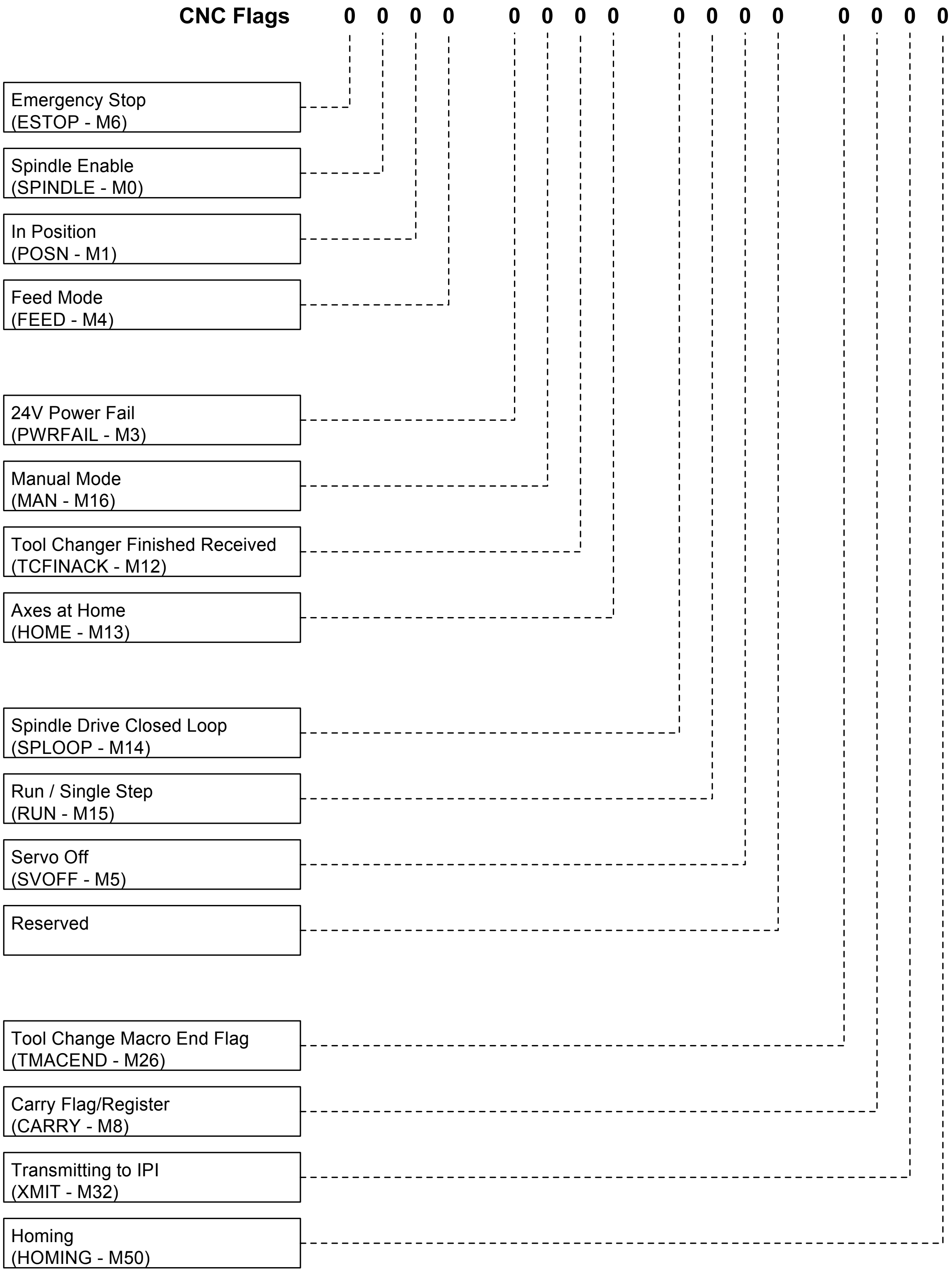

CNC Flags
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Emergency Stop (ESTOP - M6)
Spindle Enable (SPINDLE - M0)
In Position (POSN - M1)
Feed Mode (FEED - M4)
24V Power Fail (PWRFAIL - M3)
Manual Mode (MAN - M16)
Tool Changer Finished Received (TCFINACK - M12)
Axes at Home (HOME - M13)
Spindle Drive Closed Loop (SPLOOP - M14)
Run / Single Step (RUN - M15)
Servo Off (SVOFF - M5)
Reserved
Tool Change Macro End Flag (TMACEND - M26)
Carry Flag/Register (CARRY - M8)
Transmitting to IPI (XMIT - M32)
Homing (HOMING - M50)

Figure 2-5, CNC Flags

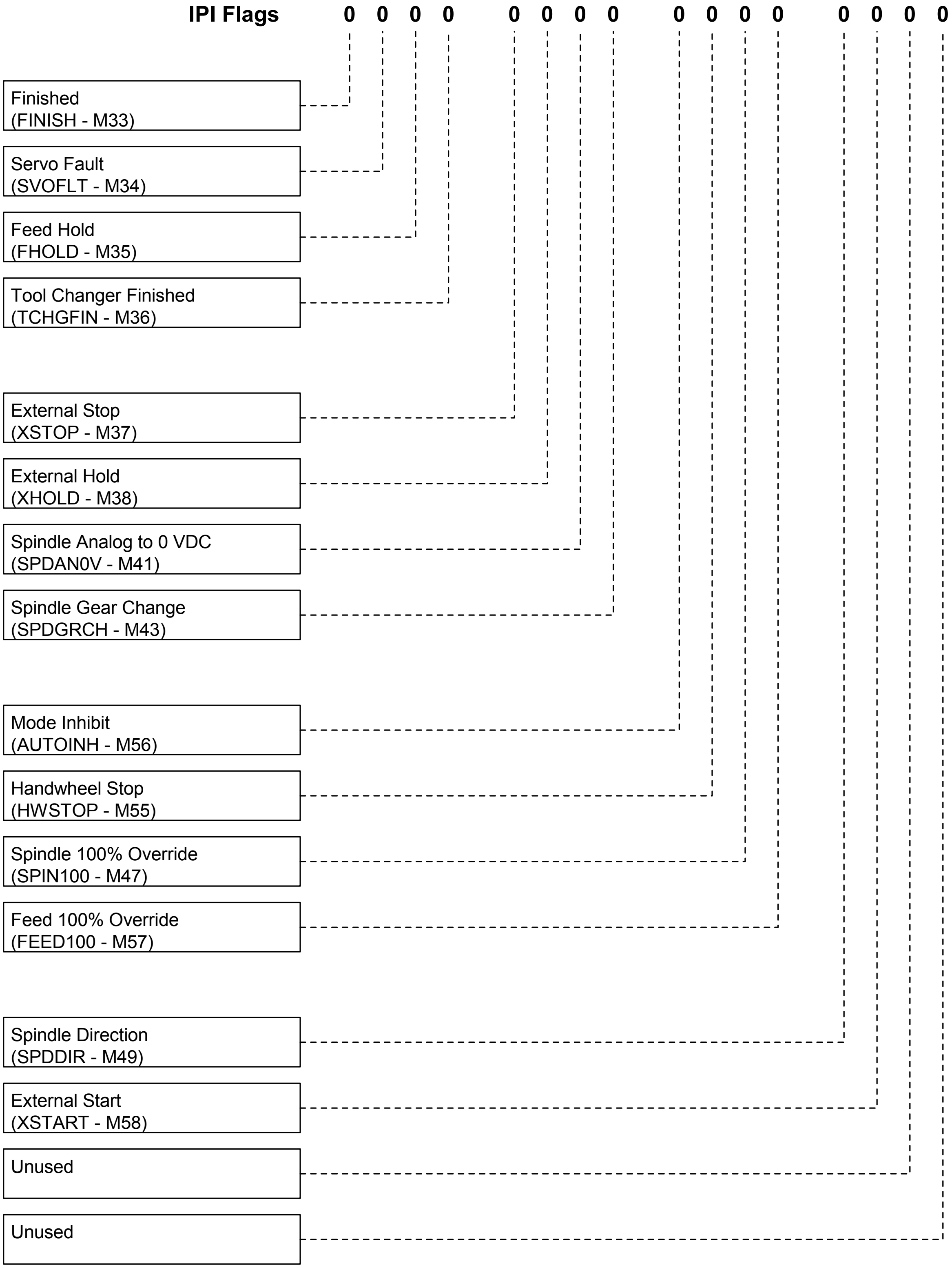


Figure 2-6, IPI Flags

Section 3 - Working with IPI

Configuring IPI Setup

Before you can program IPI, you must configure the system to recognize IPI. Refer to **Figure 3-1** for the menus referenced in this procedure. To configure the CNC to recognize IPI:

1. Exit the CNC software and go to the Software Options Menu.
2. Highlight **Setup Utility** and press **ENTER**. **Menu A, Setup Options**, activates.
3. Highlight **Builder Setup** and press **ENTER**. **Menu B, Builder Setup**, activates.
4. Highlight **Basic I/O Interface** and press **ENTER**. **Menu C, Interface Setup**, activates.
5. Highlight **Type** and press **ENTER**. The password prompt appears.
6. Type the password and press **ENTER**. A pop-up menu activates.
7. Highlight **ANILAM IPI** and press **ENTER**. The pop-up menu closes, and ANILAM IPI is the active interface type.

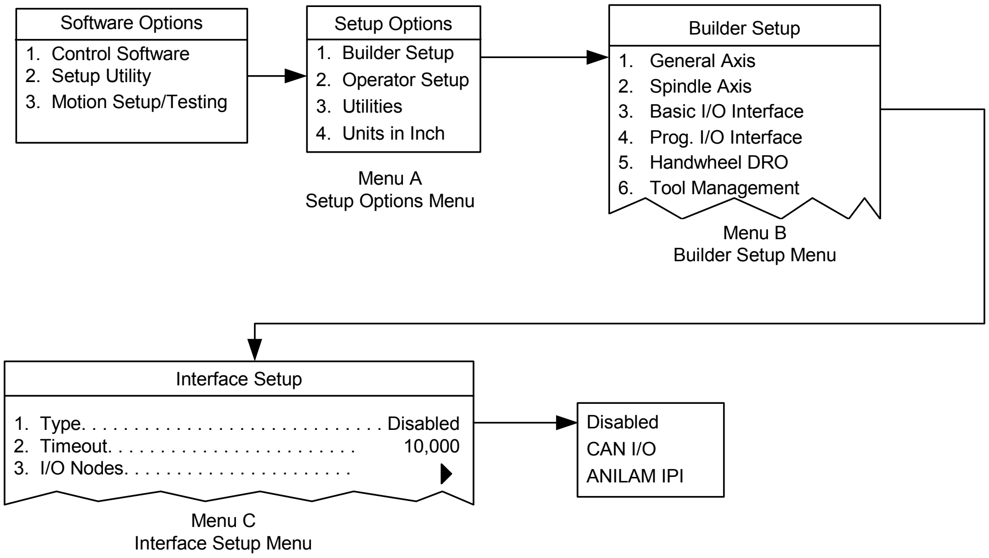


Figure 3-1, IPI Setup Menus

Programming the IPI

Typically, IPI programming proceeds as follows:

1. The technician develops the program.
2. The technician accesses the Setup Utility, creates a new program, and activates the IPI editor.
3. The technician enters or copies the IPI program. The technician can write the program offline with a standard text editor.
4. The technician runs the loader and the loader compiles the code. Error messages and warnings are posted on the screen as it runs, then, they are saved to a file.
5. The technician views the error file and makes code changes as needed. The technician recompiles as often as necessary.
6. When the compiler can compile a program successfully, it saves and activates the compiled program.

File Names

An IPI file can have any valid filename. IPI assigns the filename extensions automatically as follows:

IPI Program file	FILENAME**.DBO** (text file)
IPI Executable	FILENAME**.DBI** (binary file)
Compiler List File	FILENAME**.LST** (text file)
Compiler Error File	FILENAME**.ERR** (text file)

The DBO file is the program edited by the user.

DBI files are binary machine code generated by the loader as it compiles. If any errors occur, the loader deletes the DBI file. This prevents accidental execution of an IPI program that contains errors. The loader generates binary output files only if no errors occur during the compilation.

LST files are generated if the compiler is instructed to do so by the user or if a #LIST directive is programmed.

ERR files contain errors or warnings generated by the compiler during compilation.

Accessing Select Options Menu

Refer to **Figure 3-2** for the menus referenced in this procedure.

To access IPI:

1. Exit the CNC software and access the **Software Options Menu**.
2. Highlight **Setup Utility** and press **ENTER** to activate **Menu A, Setup Options**.
3. Highlight **Builder Setup** and press **ENTER** to activate **Menu B, Builder Setup**.
4. Highlight **Prog. I/O Interface** and press **ENTER** to display the Password prompt.
5. Type the password and press **ENTER** to activate **Menu E, Select Options**.

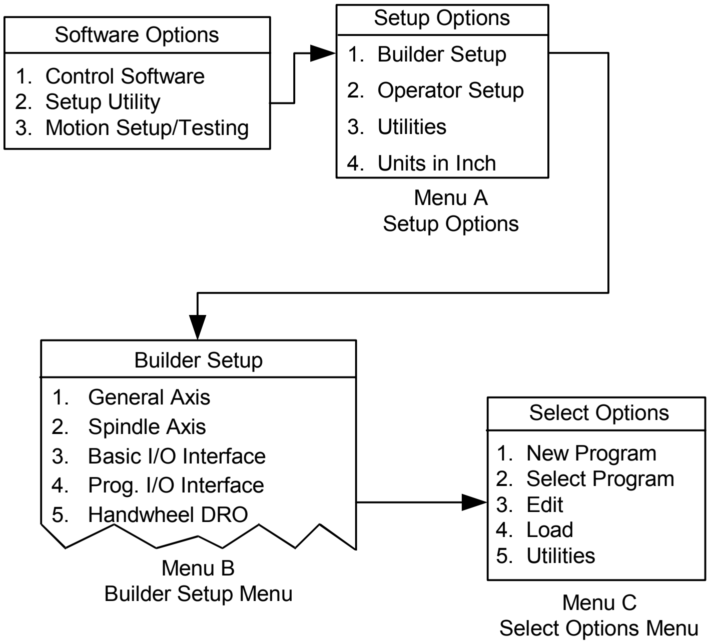


Figure 3-2, Accessing Select Options Menu

Using the IPI Editor

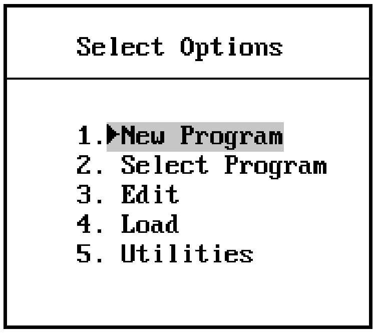
Before you run the Editor, select an existing program or create a new program.

Creating a New Program

New program names can be any combination of letters and numbers, up to eight characters. Do not use spaces or symbols. The appropriate filename extension is forced to DBO, regardless of what is entered.

To create a new IPI program:

1. From **Menu E, Select Options** Menu, highlight **New Program**, and press **ENTER**. Refer to **Figure 3-3**.

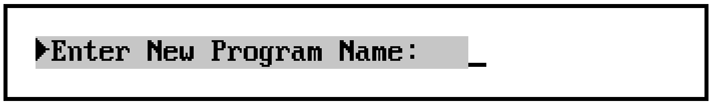


```
Select Options

1.▸New Program
2. Select Program
3. Edit
4. Load
5. Utilities
```

Figure 3-3, Creating a New Program

The CNC prompts for the new program name. Refer to **Figure 3-4**.



```
▸Enter New Program Name:   _
```

Figure 3-4, Entering a New Program Name

2. Type a program name, and press **ENTER**.

When you run the Editor, the new program will be loaded.

Selecting an Existing Program

To edit an existing program:

1. From **Menu E, Select Options Menu**, highlight **Select Program** and press **ENTER**. Refer to **Figure 3-5**.

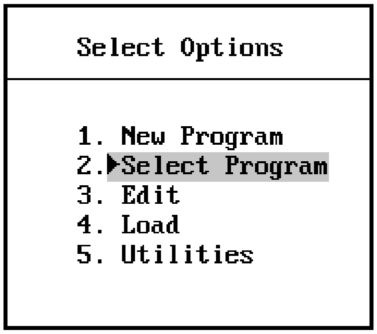


Figure 3-5, Selecting an Existing Program

Menu F, Select Program activates. Refer to **Figure 3-6**.

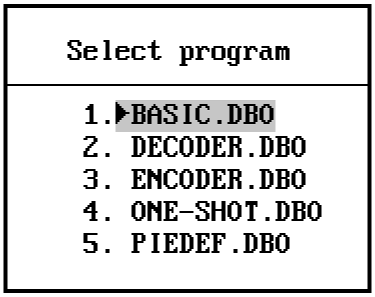


Figure 3-6, Menu F, Select Program

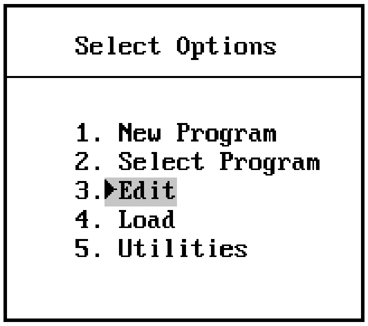
2. Highlight the desired program name and press **ENTER**.

The selected program will be loaded when you activate the Editor.

Activating the Editor

To activate the Editor:

1. Select or create a program.
2. From **Menu E, Select Options** menu, highlight **Edit** and press **ENTER**. Refer to **Figure 3-7**.



```
Select Options

1. New Program
2. Select Program
3.▶Edit
4. Load
5. Utilities
```

Figure 3-7, Activating the Editor

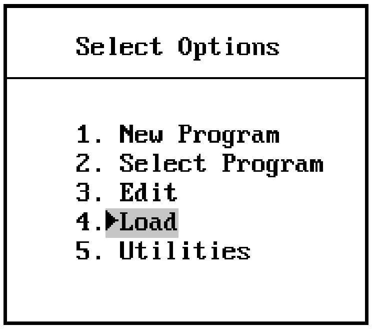
The Editor activates and displays the selected program.

Loading and Compiling a Program

The compiler will run on any currently selected program.

To activate the compiler:

1. Select the desired program.
2. From **Menu E, Select Options**, highlight **Load** and press **ENTER**. Refer to **Figure 3-8**.



```
Select Options

1. New Program
2. Select Program
3. Edit
4.▶Load
5. Utilities
```

Figure 3-8, Compiling and Loading a Program

The compiler activates and the screen displays compiling status, errors, and warnings.

3. After compilation, press **F10** to clear the screen.

If the program compiles successfully, the IPI software loads the program into memory and runs it when you activate the Control software.

Optimizing the Development Cycle

During program development, it is often necessary to reset the IPI program. An IPI program reset always occurs when you load the program from the IPI development environment. The CNC software provides a unique key sequence, or hot key, which allows you to reset an IPI program without accessing the IPI development environment. The hot key is **F6-F6** (press **F6** two times). You must execute it from the **Software Options** screen. The **Software Options** screen is the screen that allows you to access the **Control Software**, **Setup Utilities**, or **Motion/Setup Testing** screens. The IPI program is assumed to be error-free. The CNC software displays a message indicating the program has been reset.

You can use another hot key to access the IPI development environment. For this hot key to work you must have accessed the IPI development environment through the **Setup Utilities** one time since your last entry into the CNC software. This is necessary to satisfy the IPI password requirement. The hot key is **F7-F7** (press **F7** two times). You must execute it from the **Software Options** screen. To disable hot-key access to the IPI development environment, reboot the system.

NOTE: You can reboot the system using the hot key **F1-F2-F9-F10** from the **Software Options** screen.

IPI File Management Soft Keys

The IPI software allows you to use soft keys to perform various file management tasks. To perform any IPI File Management task, press the **SHIFT** key, followed by the appropriate soft key. Refer to **Table 3-1**.

Table 3-1, IPI File Management Soft Keys

Softkey Label	Key(s)
Delete	**F3**
Copy	**F4**
List	**F5**
Load	**F6**
Print	**F7**
Edit	**F8**
Restore	(**SHIFT** + **F3**)
Copy ?	(**SHIFT** + **F4**)
Mask	(**SHIFT** + **F5**)
Rename	(**SHIFT** + **F8**)
Display	(**SHIFT** + **F9**)

Section 4 - Writing IPI Programs

How the Interpreter Uses Instructions

The IPI interpreter operates serially. It never calculates with more than two values at once. The following types of values or states are available for use:

- ❑ New element
- ❑ Current register
- ❑ Previous register

The current register and previous register are the two general-purpose registers IPI uses for all functions. Refer to **Figure 4-1**.

The first instruction loads the first element into the current register. Some instructions copy the value already in the current register to the previous register and some do not. In this example, the first instruction is a **Load** instruction and copies the current register value to the previous register.

The second instruction contains an operation that does not affect the previous register and a second element. The operation is performed with the value in the current register and the second element. The result is kept in the current register. The value that was in the current register is lost.

The third instruction also contains an operation that does not affect the previous register and a third element. The operation is performed with the value in the current register and the new element. The second result remains in the current register and the first result is lost. As new instructions are combined with values in the current register, the value in the current register is constantly updated and old values are lost.

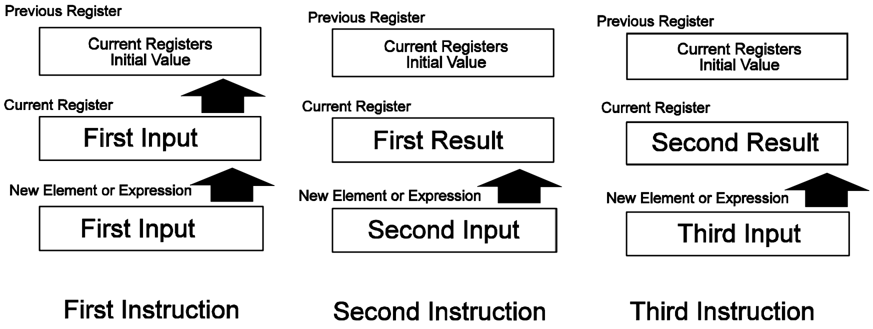


Figure 4-1, Interpreter Operation

You can send values in the current state register to an output or to another register for storage. Only a few more advanced instructions use the value in the previous register. Most instructions use the new elements and the current register.

To program more efficiently, make the new element an expression instead of a single element. When the new element is an expression, the result of the expression is seen as the value or state of the new element.

Program START and END Instructions

The **START** instruction informs the interpreter where to begin each program cycle. The START instruction is optional and does not need to be the first instruction in the program. Program instructions that precede START are not repeated after the first cycle. If START is not used, all instructions are executed every cycle.

Instructions inserted before START can begin initialization steps, which are done only once. The IPI interpreter clears all of its registers and reads all inputs at the first instruction, not at START.

The END instruction informs the interpreter that the program has finished. The END instruction must be added to every program. When the interpreter encounters END, it generates outputs on the I/O Board, based on the states stored in the Y registers. The interpreter then transfers IPI flags to the CNC and restarts the program. It runs only the instructions that appear after the program START.

Table 4-1 describes each instruction.

Table 4-1, Start and End Instructions

Operation	Description
START	Denotes start of repeating portion of IPI program. Optional.
END	Must be last instruction in IPI program. Tells interpreter program has finished. Time to generate outputs and repeat cycle. Required.

Building IPI Program Instructions

Program instructions are the lines of IPI code. Program instructions are constructed using operation codes, elements, and expressions, assembled in the proper format.

Instruction Operands

Instruction operands are values stored in input, output, sequence, multifunction, and timer registers. These elements are identified by their X, Y, S, M, and T designators, or by their assigned labels. An element can also be a constant.

NOTE: Element names must be separated from other instruction parameters by at least one blank space.

Operation Codes

IPI uses operation codes to identify different operations. Operation codes inform the IPI of the following:

- ❑ What function to perform with new element or expression
- ❑ The value in the current register
- ❑ The value in the previous register (if used)

The operation code is not case sensitive. It can start on any column in the line. Leading tabs and spaces will be ignored.

Expressions

Expressions perform Boolean operations, comparison operations, and mathematical operations with pairs of operands. Expressions are primarily used to perform conditional evaluations of numeric values. However, both state values and numeric values can be used. Most expressions produce state outputs. Only add and subtract expressions produce numeric values. Use expressions to shorten program length or provide options.

Expressions begin with a left parenthesis and end with a right parenthesis. There must be a space after the left parenthesis and a space before the right parenthesis. Only two elements (or one element and one constant) separated by an operator, are permitted per expression. Expressions cannot be nested. Insert the expression in an instruction as if it were a single element.

Expression results are converted to states or values as necessary to complete an operation. Refer to **Table 4-2**. Expression results depend on the type of operation performed.

Table 4-2, Expression Operands (State Value = s, Numeric Value = n)

Expression	Definition
(s1 AND s2)	Results in TRUE only when both operands are TRUE. Otherwise, FALSE.
(s1 OR s2)	Results in TRUE if either parameter is TRUE. Results in FALSE only when both are FALSE.
(s1 ANI s2)	Results in FALSE only when both parameters are TRUE. Otherwise, TRUE. **CAUTION: The ANI function in an expression does not operate the same as the ANI function in the instruction set.**
(s1 ORI s2)	Results in FALSE when either parameter is TRUE; is TRUE when both are FALSE. **CAUTION: The ORI function in an expression does not operate the same as the ORI function in the instruction set.**
(s1 XOR s2)	Results in TRUE when only one parameter is TRUE. Results in FALSE if both are in the same state.
(s1 XNR s2)	Results in FALSE when one, but not the other parameter is TRUE. The result is TRUE if both are in the same state.
(n1 + n2)	Adds the two register values.
(n1 – n2)	Subtracts the two register values. If the result is negative, an overflow will occur, and the result is undefined.
(n1 EQ n2)	Results in TRUE if the register values are the same.
(n1 NE n2)	Results in TRUE if the register values are different.
(n1 GT n2)	Results in TRUE if r1 is greater than r2.
(n1 LT n2)	Results in TRUE if r1 is less than r2.
(n1 GE n2)	Results in TRUE if r1 is greater than or equal to r2.
(n1 LE n2)	Results in TRUE if r1 is less than or equal to r2.

Numeric Parameters

Multifunction memory registers can store numeric values, as well as Boolean true/false states. When combined with instructions containing expressions, IPI can monitor numeric values as a condition. Numeric values can be used in binary, octal, decimal, and hexadecimal formats. However, the internal format is always binary.

There are two different types of values: byte values and word values. Binary values range from 0 to 255. Word values range from 0 to 65535. Binary, octal, decimal, and hex values will all be accepted. The default base is decimal.

To designate another base, insert the base indicator to the right of the number. Refer to **Table 4-3**.

Table 4-3, Number Base Indicators and Examples

Number Base	Indicator	Example (decimal equivalent)
Binary	B	10110b = 10110 binary (22 decimal)
Octal	O or Q	27q = 27 octal (23 decimal)
Decimal	D or no indicator	27d or 27 = 27 decimal. (Default)
Hex	H or X	4fh = 4f hex (79 decimal)

Creating Additional I/O Labels

Labels are used to reference strings of characters. If SPDLFWD has been defined to represent **Y0:6**, when the compiler encounters the string SPDLFWD, it will substitute **Y0:6**. As noted earlier, many permanent labels are pre-assigned.

Since SPDLFWD is more specific, the program becomes easier to read and understand. Labels can be used to reference specific elements, specify delay values, and rename operation codes. The following rules apply:

- ❑ Label names can be a string of any combination of alphanumeric characters (1 to 32 characters). Do not use blank spaces. Names must start with a letter.
- ❑ After a label has been defined, it cannot be redefined, deleted, or changed in any way later in the program.
- ❑ All labels are active only in the program in which they are defined.

Refer to Compiler Directives for more information on creating labels.

Using Comments

The compiler ignores any line of code in an IPI program that starts with an asterisk (*) or a semicolon (;). This feature allows the programmer to add documentation to the program or to mark ("comment") code to be ignored by the compiler. A comment can be placed on the same line as program instruction.

Active Instruction	Explanatory comment ignored by compiler.
LD M55	*LOAD MULTI-FUNCTION REGISTER 55.

Blank lines are also allowed and will be ignored.

The compiler will not convert comments from *.DBO files into executable *.DBI instructions.

Finish Signal Generation

Generation of a proper finish signal is critical for proper IPI/CNC interaction. Refer to **Figure 4-2**. Finish signals are processed as follows:

1. The CNC sends an M, S, T, or H Code to the IPI, and the IPI retrieves the CNC flags from the CNC.
2. The CNC halts program execution and sets the IPI finish flag (M33-FINISH) high. The CNC then waits for a FINISH low to resume program execution. At the same time, the IPI internally clears the M, S, T, or H Code.
3. The IPI internally clears the M, S, T, or H after the first iteration (rising edge), when an M, S, T, or H code is seen. Otherwise, the IPI would interpret an M, S, T, or H more than once (on the rising and falling edges of the signal). This guarantees that a particular code is seen only once.
4. When the M, S, T, or H is completed, the FINISH status is low. At this point, the CNC sees the falling edge of the FINISH flag (low) and program execution resumes.

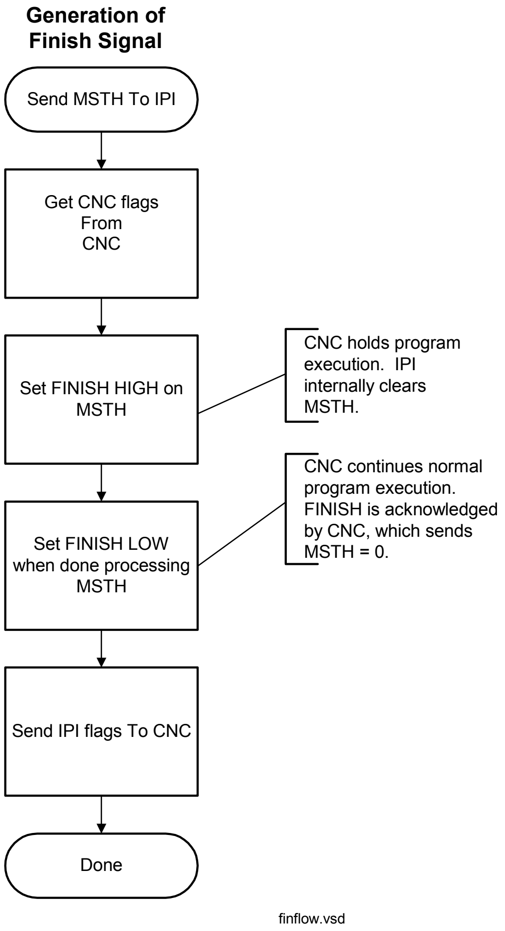

Generation of
Finish Signal
Send MSTH To IPI
Get CNC flags From CNC
Set FINISH HIGH on MSTH
CNC holds program execution. IPI internally clears MSTH.
Set FINISH LOW when done processing MSTH
CNC continues normal program execution. FINISH is acknowledged by CNC, which sends MSTH = 0.
Send IPI flags To CNC
Done
finflow.vsd

Figure 4-2, IPI: M, S, T, or H Code to Finish Signal

IPI Operation Set

IPI programs can be written in various degrees of complexity. Available instruction sets include the following, from the simplest to the most complex:

- ❑ Single-element instructions
- ❑ Two-element instructions
- ❑ Two-element instructions that use an expression as one of the elements
- ❑ Instructions that use timers
- ❑ Instructions that use the previous state register

You can write a complete IPI program with only single-element instructions. However, the fewer the number of lines of instruction there are, the faster the program will run.

Syntax is demonstrated using pairs of brackets to contain instruction elements. Appropriate elements are identified by keywords.

Syntax format: "[keyword]"

Ladder diagram equivalents and truth tables are provided where appropriate. Refer to **Table 4-4** for a description of symbols used in the ladder diagram.

Table 4-4, Ladder Diagram Symbols

Symbol	Description
∦	A contact that is normally closed (when the relay is not energized).
∥	A contact that is normally open (when the relay is not energized).
◯	A coil that signifies the end instruction for the rung.

Table 4-5 provides a summary of available IPI operation codes. Refer to Table 4-6, Detailed Descriptions and Examples of Operands, for detailed explanations and examples of each operation code.

Table 4-5, Summary of IPI Operands

Operand	Function
LD See page 4-13.	Loads new element's state value into current register. If new element has numeric value, it is converted to appropriate state value. Loads any value already in the current register into the previous register.
OUT See pages 4-14 and 5-2.	Writes the value in the current register to the specified register. Only multifunction registers can receive numeric values. All other registers convert value to a state.
LDI See page 4-15.	Loads an element's inverse state value to current register. If current register had an initial value, it is moved to previous register. If element has numeric value, it is converted to appropriate state value.
MOV See page 4-16.	Combines functions of read and output into one operator. Current and previous registers are not used. Numeric values are moved intact if registers are compatible. Value/state conversions occur otherwise.
MVA See page 6-5.	Moves the selected node's numerical analog value to a multifunction register for evaluation. There is one analog input per node 0–5. The last output, Bit 5, is jumper-selectable as an analog input. **NOTE:** The Type of the corresponding CAN node must be set to Digital/Analog in the Setup Utility.
RD See page 5-3.	Loads element value into current register. Copies any value already in the current register into the previous register. If element value is numeric, it is loaded as a numerical value. If element value is a state value, it is loaded as a state value. RD can be used to access a numeric value after a mathematical operation or to load the count value of a timer.
AND See page 4-17.	Performs a Boolean logic **AND** function with value in current register and new element. Result remains in current register. Previous register is unaffected.
ANI See page 4-18.	Performs a Boolean logic **AND** function with value in current register and the inverse value of the new element. Result remains in current register; previous register is unaffected.

(Continued…)

Table 4-5, Summary of IPI Operands (Continued)

Operand	Function
OR See page 4-20.	Performs Boolean logic **OR** function using new element and state value in current register. Result remains in current register; previous register is unaffected.
ORI See page 4-23.	Performs a Boolean logic **OR** function with value in current register and the inverse value of the new element. Result remains in current register; previous register is unaffected.
ANB See page 4-27.	Performs Boolean **AND** function with value in previous register, value in current register and new element value.
ORB See page 4-29.	Performs Boolean **OR** function with value in previous register, value in current register and new element's value.
SET See page 4-31.	If current register holds a TRUE value, TRUE is copied in new element's register. If current register holds a FALSE value, no activity occurs. This instruction serves to latch the new element to a TRUE value for subsequent cycles. A subsequent **MOV** statement or a **RES** instruction can be used to unlatch the register.
RES See page 4-32.	This instruction resets the new element to a FALSE value for subsequent cycles. If current register holds a TRUE value, FALSE is copied in new element's register. If current register holds a FALSE value, no activity occurs. A subsequent **MOV** statement or a **SET** instruction can be used to re-latch the register.
CTL/CTR See page 4-33.	Used in pairs. **CTL** - ANDs specified element with all subsequent instructions until deactivated. **CTR** - deactivates any active CTL instructions.
DEC See page 4-35.	For every cycle in which the current register value is true, the numeric value of the new element decreases.
INC See page 4-35.	For every cycle that the current register value is true, the numeric value of the new element increases.
RST See page 5-4.	Restart instruction. Restarts countdown timer if current register's state value is TRUE and designated timer is currently in a delay countdown state.
NOP See page 4-35.	No operation is performed.

(Continued…)

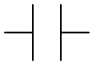
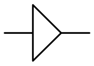
Table 4-5, Summary of IPI Operands (Continued)

Operand	Function
INV See page 4-25.	Inverts specified element. Inverts current register when no element is specified. If the value to be inverted is numeric, it is converted to a state value and then inverted.
IF/ELS/EDF See page 6-2.	**IF** - Begins conditional statement. CNC executes subsequent instructions if relevant register value is true. The relevant register value is the current register or the new element register. **ELS** - Provides intermediate step in the process. Executes subsequent instructions if new expression, new element or current register is FALSE. **EDF** - Terminates conditional instruction set.
CLP/EJP See page 6-4.	**CLP** - Begins conditional statement. Executes subsequent instructions if new element, new expression or current register value is FALSE. Jumps to **EJP** instruction if TRUE. **EJP** - Ends conditional jump instruction set.
OKBD See page 6-5.	Output keyboard instruction. Used to output key codes to the CNC. The CNC interprets these key codes as if the user had pressed the corresponding key. Only one key code can be passed per IPI scan. For a key code to be interpreted by the CNC, it must be different from scan to scan.
OTI See page 6-6.	Output until input. Specified output is energized for a maximum of 30 seconds or until the corresponding input is energized. The output can be a Y value. **NOTE:** The input number may be different from the output number. In this case, use **OTI** within the same node. An **LD** or **LDI** command must be programmed directly before the **OTI** in order to specify the input bit. Additionally, the qualifying **LD** or **LDI** must be an expression using physical input bits. See also **SOTI** (Super **OTI**) and **COTI** (cancels **OTI** and **SOTI**).
OWI See page 6-8.	Output when input. The specified output is latched on immediately on input. Transition must be from FALSE to TRUE. **NOTE:** The input number may be different from the output number. In this case, use **OWI** within the same node. An **LD** or **LDI** command must be programmed directly before the OWI in order to specify the input bit. Additionally, the qualifying **LD** or **LDI** must be an expression using physical input bits. The specified input bit is the same node location as the specified output on the corresponding input port. Load input with either **LD** or **LDI**. **LD** is for a positive trigger and **LDI** is for a negative trigger. Follow immediately (or before another Load instruction) with the **OWI** statement.

Table 4-5, Summary of IPI Operands (Continued)

Operand	Function
SOTI See page 6-9.	Super **OTI** works like **OTI** but the number of input pulses required to turn the output off can be specified (instead of it being hard-coded to 1 as in **OTI**). Output until input instruction. **NOTE:** The input number may be different from the output number. In this case, use **SOTI** within the same node. An **LD** or **LDI** command must be programmed directly before the **SOTI** in order to specify the input bit. Additionally, the qualifying **LD** or **LDI** must be an expression using physical input bits. See also **OTI** (output until input) and **COTI** (cancels **OTI** and **SOTI**).
COTI See page 6-10.	Cancel **OTI** or **SOTI** command immediately.

Table 4-6, Detailed Descriptions and Examples of Operands

LD	Syntax	Valid Elements
Loads new element's state value into current register. If new element has numeric value, it is converted to appropriate state value. Loads any value already in the current register into the previous register.	LD [element]  - Ladder Equiv.  - Logical Symbol	M registers T registers S registers X registers Y registers Expressions

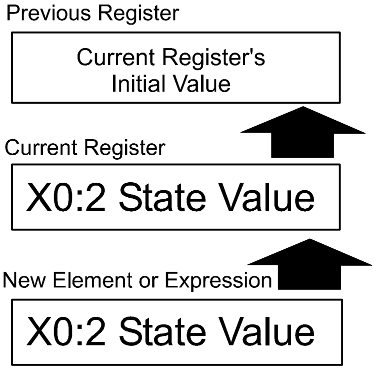
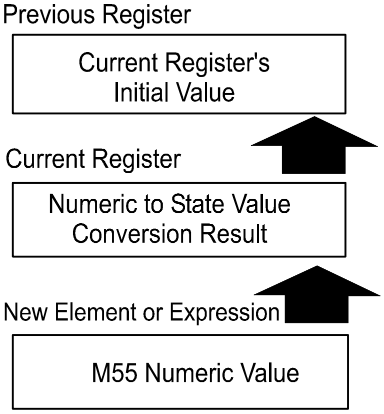
Examples	Explanation
LD Example #1 One state value element. LD X0:2	State value in X0:2 register is copied to current register. Current register's previous value is copied to previous register.  Previous Register Current Register's Initial Value Current Register X0:2 State Value New Element or Expression X0:2 State Value
LD Example #2 One multifunction element. LD M55	If multifunction register's value is numeric, value is converted to state value equivalent and is loaded into current register. Current register's previous value is copied to previous register.  Previous Register Current Register's Initial Value Current Register Numeric to State Value Conversion Result New Element or Expression M55 Numeric Value

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

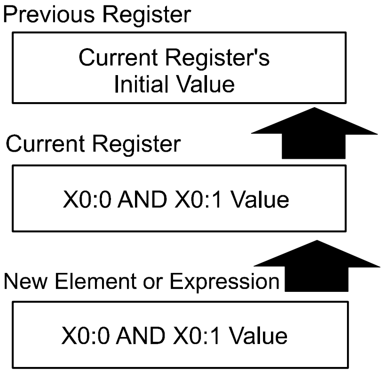
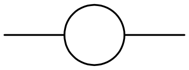
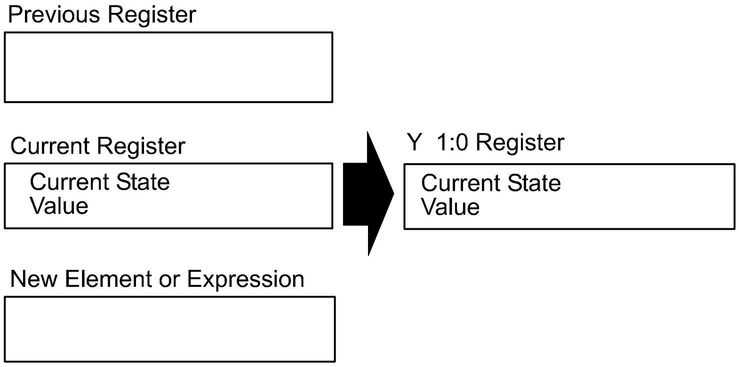
<table>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td><b>LD</b> Example #3
Expression used as element.
LD (X0:0 AND X0:1)</td><td colspan="2">X0:0 is ANDed with X0:1 and resulting state is loaded in current register. Current register's previous value is copied to previous register.

 </td></tr>
<tr><th>OUT</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Writes the value in the current register to the specified register.
Only multifunction registers can receive numeric values. All other registers convert value to a state.</td><td>OUT [element]
</td><td>M registers
T registers
S registers
X registers
Y registers</td></tr>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td><b>OUT</b> Example #1
One element.
OUT Y1:0</td><td colspan="2">Value in current register is sent to Y1:0 register. Value in Y1:0 register must be a TRUE/FALSE state.

 </td></tr>
<tr><td><b>OUT</b> Example #2
One element.
OUT M70</td><td colspan="2">Value in current register is sent to M70 register. Value sent to M70 register can be a state or number.</td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

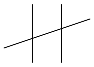
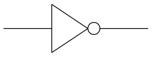
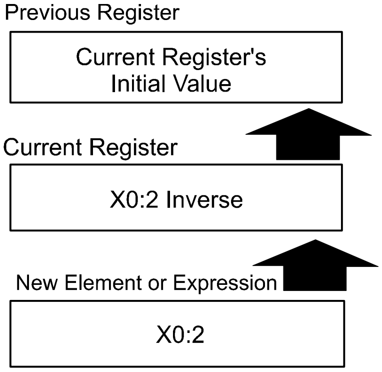
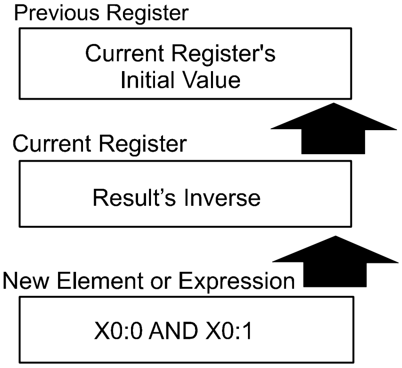
LDI	Syntax	Valid Elements
Loads element's state value to current register. If current register had value, it is moved to previous register. If element has numeric value, it is converted to appropriate state value.	LDI [element]  - Ladder Equiv.  - Logical Symbol	Y registers M registers T registers S registers X registers Expressions
Examples	**Explanation**	
LDI Example #1 One element. LDI X0:2	X0:2's inverse state is determined and saved in the current register. Current register's previous value is copied to previous register.  Previous Register Current Register's Initial Value Current Register X0:2 Inverse New Element or Expression X0:2	
LDI Example #2 Expression used as element. LDI (X0:0 AND X0:1)	X0:0 is ANDed with X0:1, the inverse is determined and stored in the current register. Current register's previous value is copied to previous register.  Previous Register Current Register's Initial Value Current Register Result's Inverse New Element or Expression X0:0 AND X0:1	

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

MOV	Syntax	Valid Elements
Without qualification, a value or state is unconditionally put into the target element. Current and previous registers are not used. Numeric values are moved intact if registers are compatible. Otherwise, values/state conversions occur.	MOV [element] [element]	Y registers M registers T registers S registers X registers Constants Expressions

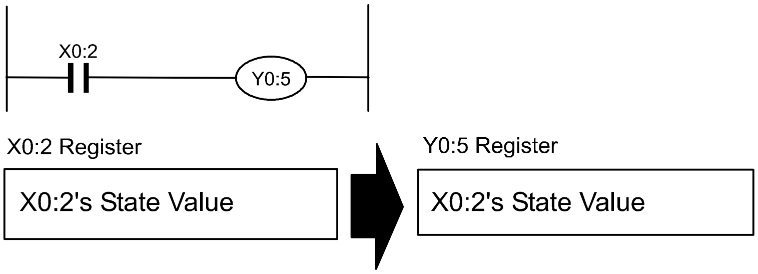
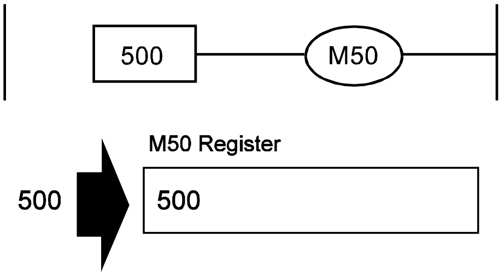
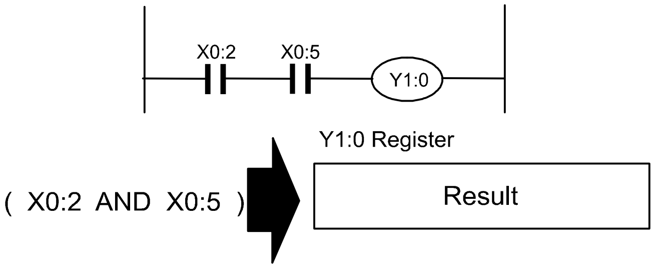
Examples	Explanation
MOV Example #1 Two elements. MOV X0:2 Y0:5	State value of input register X0:2 is read and copied into output register Y0:5. Current register and previous register are not involved.  X0:2 Y0:5 X0:2 Register Y0:5 Register X0:2's State Value X0:2's State Value
MOV Example #2 Constant and element. MOV 500 M50	Numeric value of 500 is loaded into multifunction register M50. Current register and previous register are not involved. 
MOV Example #3 Expression and element. MOV (X0:2 AND X0:5) Y1:0	State values in X0:2 register and X0:5 register are ANDed within the expression. The resulting state is loaded into the Y1:0 output register. Current register and previous register are not involved.  X0:2 X0:5 Y1:0 Y1:0 Register (X0:2 AND X0:5) Result

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

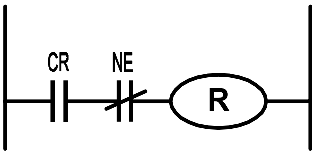
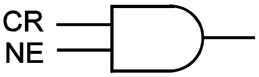
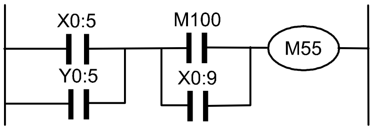
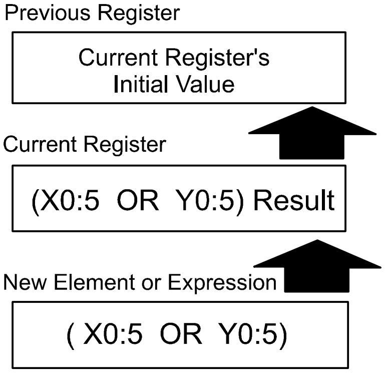
AND	Syntax	Valid Elements
Performs a Boolean logic AND function with value in current register and new element. Result remains in current register. Previous register is unaffected.	AND [Element]	Y registers M registers T registers S registers X registers Expressions
Truth Table	**Ladder Equivalent**	**Logic Symbol**
AND Current Register (CR) — New Element (NE) — Result (R) F — F — F F — T — F T — F — F T — T — T		
Examples	**Explanation**	
AND Example #1 Two parameter instructions. LD (X0:5 OR Y0:5)	 State values of X0:5 and Y0:5 registers are ORed and loaded into current register. Current register's previous value is copied into previous register.  Previous Register Current Register's Initial Value Current Register (X0:5 OR Y0:5) Result New Element or Expression (X0:5 OR Y0:5)	

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

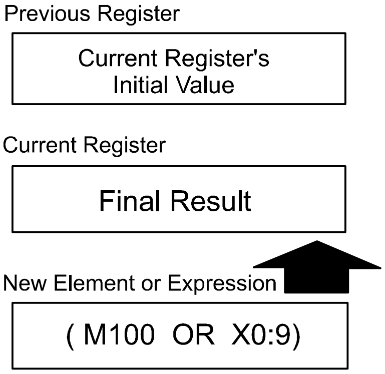
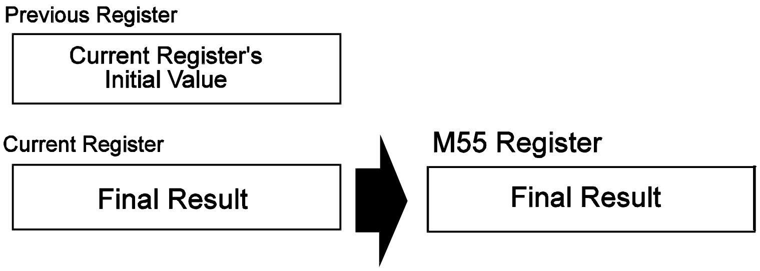
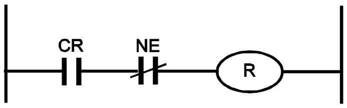
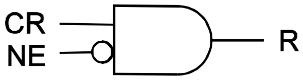
<table>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td>AND (M100 OR X0:9)</td><td colspan="2">Value in register M100 is converted to a state value and ORed with state value stored in register X0:9. The result is ANDed with the earlier result, generating the final result. Final result remains in current register.

</td></tr>
<tr><td>OUT M55</td><td colspan="2">State value in current register is copied into M55 register.

</td></tr>
<tr><th>ANI</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Performs a Boolean logic AND function with value in current register and the inverse value of the new element.
Result remains in current register; previous register is unaffected.</td><td>ANI [element]</td><td>Y registers
M registers
T registers
S registers
X registers
Expressions</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>ANI
<table>
<tr><th>Current Register (CR)</th><th>New Element (NE)</th><th>Result (R)</th></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>F</td></tr>
</table></td><td>
</td><td>
</td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

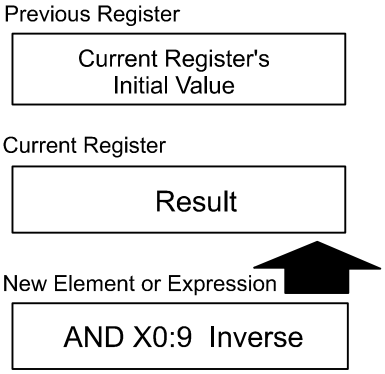
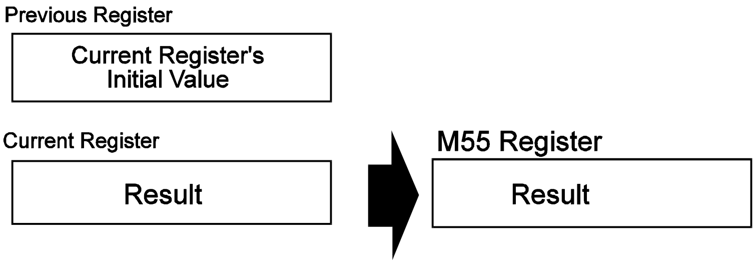
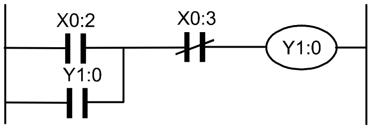
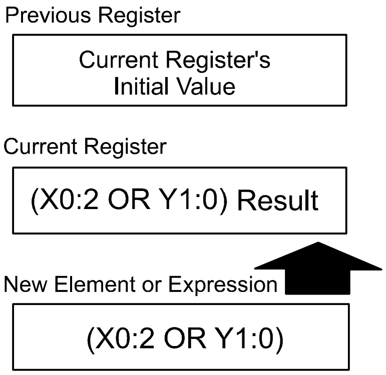
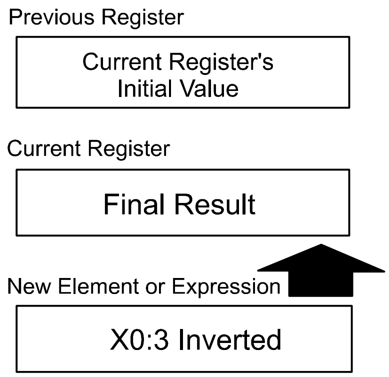
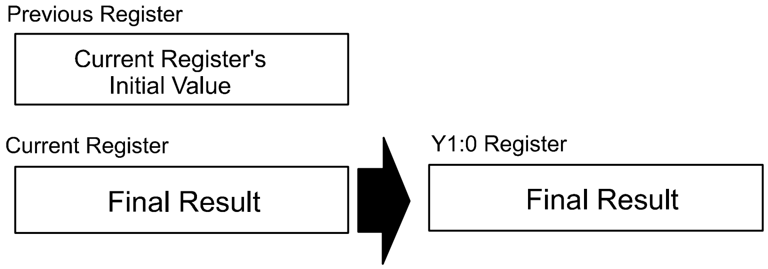
Examples	Explanation
ANI Example #1 ANI X0:9	Inverse state of X0:9 register is ANDed with the state value in the current register. Result is kept in current register.  Previous Register Current Register's Initial Value Current Register Result New Element or Expression AND X0:9 Inverse
OUT M55	Value in current register is sent to register M55.  Previous Register Current Register's Initial Value Current Register Result M55 Register Result
ANI Example #2	
LD (X0:2 OR Y1:0)	State values in X0:2 and Y1:0 registers are ORed, resulting state is loaded into current register. Current register's previous value is copied into previous register.  Previous Register Current Register's Initial Value Current Register (X0:2 OR Y1:0) Result New Element or Expression (X0:2 OR Y1:0)

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

Examples	Explanation
ANI X0:3	State value stored in input register X0:3 is inverted and ANDed to previous result. Final result remains in current register.  Previous Register Current Register's Initial Value Current Register Final Result New Element or Expression X0:3 Inverted
OUT Y1:0	State value in current register is copied to Y1:0 output register.  Previous Register Current Register's Initial Value Current Register Final Result Y1:0 Register Final Result

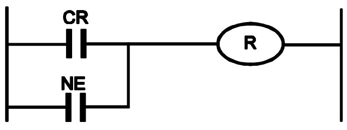
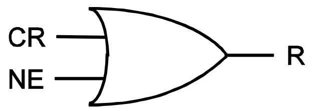
<table>
<tr><th>OR</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Performs Boolean logic OR function using new element and state value in current register.
Result remains in current register. Previous register is unaffected.</td><td>OR [element]</td><td>Y registers
M registers
T registers
S registers
X registers
Expressions</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>OR<table><tr><th>Current Register (CR)</th><th>New Element (NE)</th><th>Result (R)</th></tr><tr><td>F</td><td>F</td><td>F</td></tr><tr><td>F</td><td>T</td><td>T</td></tr><tr><td>T</td><td>F</td><td>T</td></tr><tr><td>T</td><td>T</td><td>T</td></tr></table></td><td> </td><td> </td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

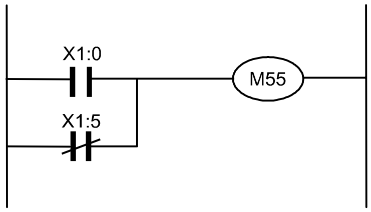
<table>
<tr><th>Examples</th><th>Explanation</th></tr>
<tr><td>

OR Example #1

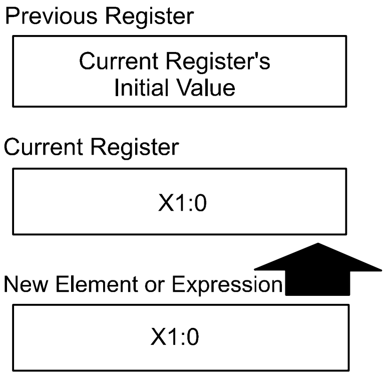
LD X1:0

OR X1:5

</td><td>



Input state of X1:0 is loaded into current register. Current register's previous value is copied into previous register.

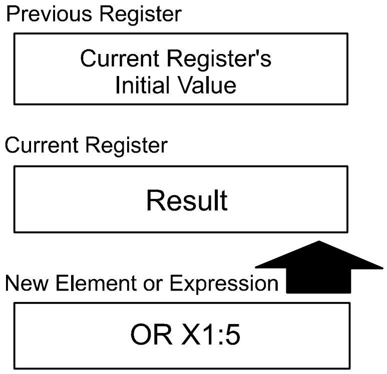


Input state of X1:5 register is ORed with the state value in the current register. Result is kept in current register

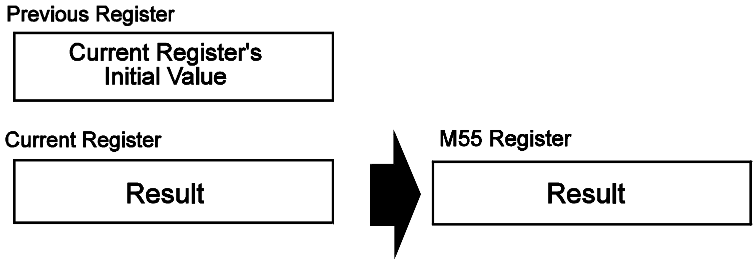
</td></tr>
<tr><td>

OUT M55

</td><td>



Value in current register is sent to register M55.



</td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

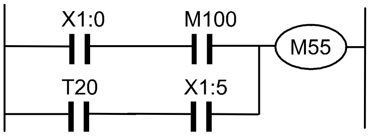
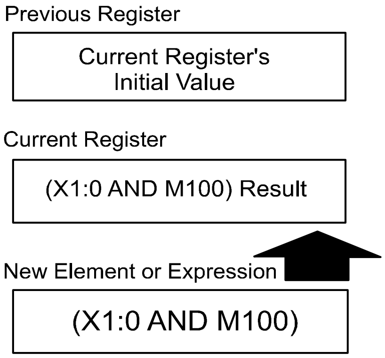
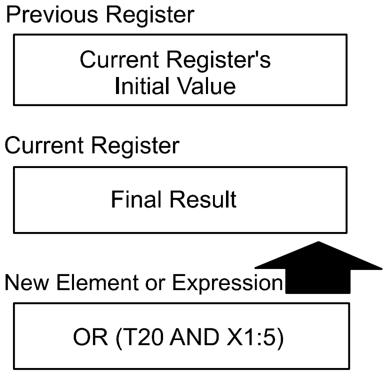
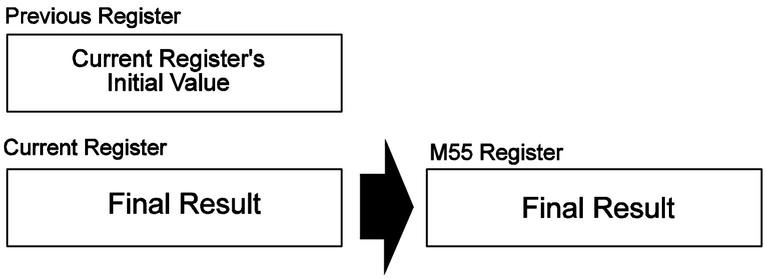
Examples	Explanation
OR Example #2	
LD (X1:0 AND M100)	State value in X1:0 register and state value equivalent in M100 register are ANDed. Result is loaded into current register. Current registers previous value is copied into previous register. Previous Register Current Register's Initial Value Current Register (X1:0 AND M100) Result New Element or Expression (X1:0 AND M100)
OR (T20 AND X1:5)	Timer 20 state value and state value in X1:5 register are ANDed, then ORed with the value in the current register. Final result remains in current register.
	Previous Register Current Register's Initial Value Current Register Final Result New Element or Expression OR (T20 AND X1:5)
OUT M55	Value in current register is copied to M55 register. Previous Register Current Register's Initial Value Current Register Final Result M55 Register Final Result

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

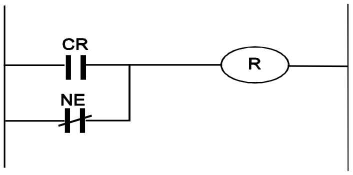
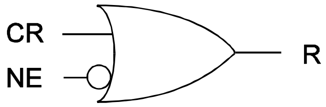
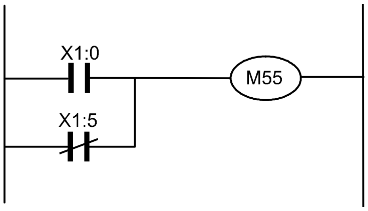
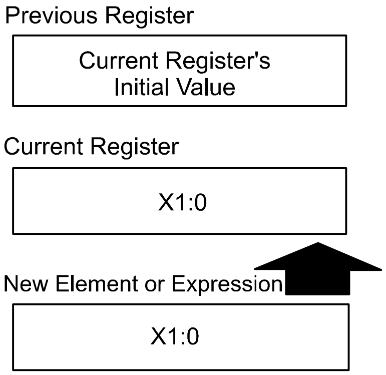
<table>
<tr><th>ORI</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Performs a Boolean logic OR function with value in current register and the inverse value of the new element.
Result remains in current register; previous register is unaffected.</td><td>ORI [element]</td><td>Y registers
M registers
T registers
S registers
X registers
Expressions</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>ORI<table><tr><th>Current Register (CR)</th><th>New Element (NE)</th><th>Result (R)</th></tr><tr><td>F</td><td>F</td><td>T</td></tr><tr><td>F</td><td>T</td><td>F</td></tr><tr><td>T</td><td>F</td><td>T</td></tr><tr><td>T</td><td>T</td><td>T</td></tr></table></td><td> </td><td> </td></tr>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td><b>ORI</b> Example #1
LD X1:0</td><td colspan="2">
Input state of X1:0 is loaded into current register. Current register's previous value is copied into previous register.
 </td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

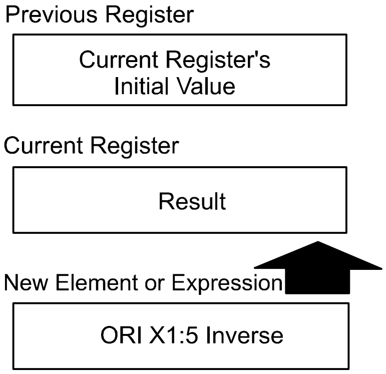
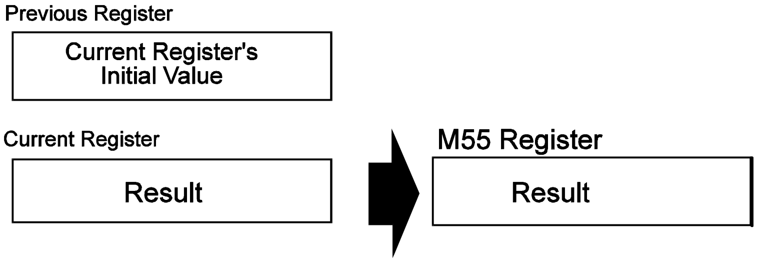
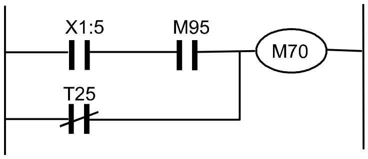
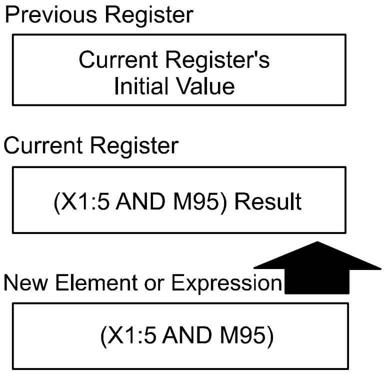
Examples	Explanation
ORI X1:5	Input state of X1:5 status register is inverted and ORed with the state value in the current register. Result is kept in current register.  Previous Register Current Register's Initial Value Current Register Result New Element or Expression ORI X1:5 Inverse
OUT M55	Value in current register is sent to register M55.  Previous Register Current Register's Initial Value Current Register Result M55 Register Result
ORI Example #2	
LD (X1:5 AND M95)	State value in X1:5 register and equivalent state value in M95 register are ANDed. Result is loaded into current register. Current register's previous value is copied into previous register.  Previous Register Current Register's Initial Value Current Register (X1:5 AND M95) Result New Element or Expression (X1:5 AND M95)

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

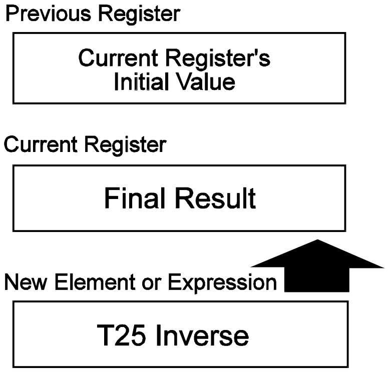
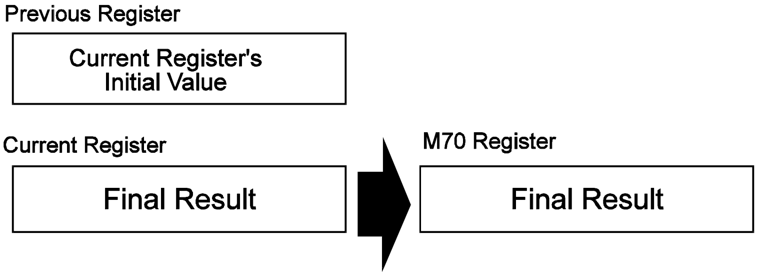
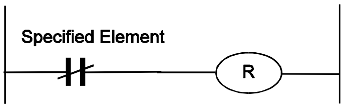
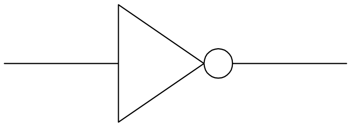
<table>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td>ORI T25</td><td colspan="2">Timer T25's state value is inverted and ORed with value in current register. Final result is kept in current register.

</td></tr>
<tr><td>OUT M70</td><td colspan="2">Value in current register is copied to M70 register.

</td></tr>
<tr><th>INV</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Inverts specified element.
Inverts current register when no element is specified.
If the value to be inverted is numeric, it is converted to a state value and then inverted.</td><td>INV [element]
– or –
INV</td><td>Y registers</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>INV<table><tr><th>New Element or Current Register</th><th>Result</th></tr><tr><td>F</td><td>T</td></tr><tr><td>T</td><td>F</td></tr></table></td><td>
</td><td></td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

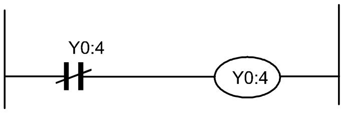
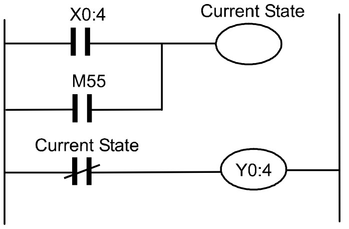
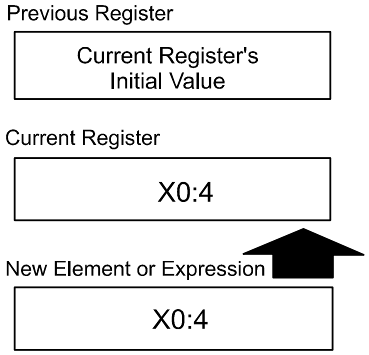
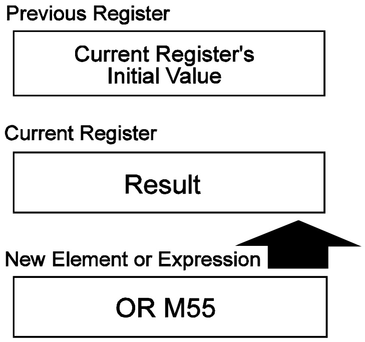
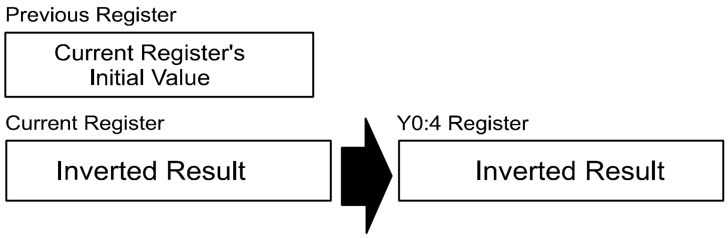
Examples	Explanation
INV Example #1 INV Y0:4	 State value of Y0:4 register is inverted. Current and previous registers not affected.
INV Example #2	
LD X0:4	Loads X0:4 input into current register. Current register's previous value is copied into previous register.  Previous Register Current Register's Initial Value Current Register X0:4 New Element or Expression X0:4
OR M55	ORs value in current register with M55. Result held in current register.  Previous Register Current Register's Initial Value Current Register Result New Element or Expression OR M55
INV OUT Y0:4	Inverts result and sends to Y0:4 register.  Previous Register Current Register's Initial Value Current Register Inverted Result Y0:4 Register Inverted Result

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

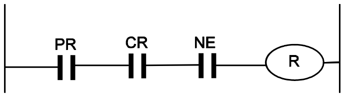
ANB	Syntax	Valid Elements
Performs Boolean AND function with value in previous register, value in current register and new element's value.	ANB [element]	Y registers M registers T registers S registers X registers Expressions

Truth Table

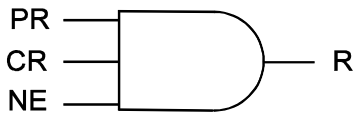
ANB

Previous Register (PR)	Current Register (CR)	New Element (NE)	Result (R)
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	F
T	F	F	F
T	F	T	F
T	T	F	F
T	T	T	T

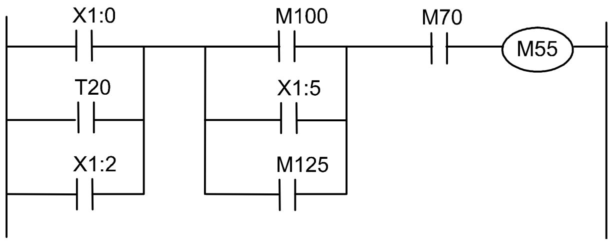
Ladder Equivalent



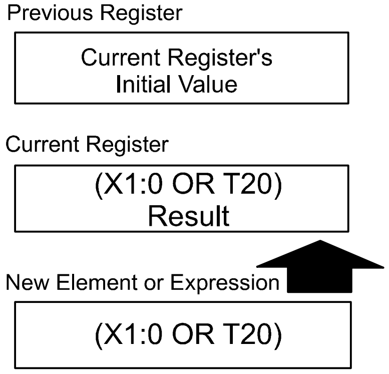
Logic Symbol



Examples	Explanation
ANB Example #1	



LD (X1:0 OR T20)	Loads result of expression into current register, copies current value of current register into previous register.


Previous Register

Current Register's Initial Value

Current Register

(X1:0 OR T20) Result

New Element or Expression

(X1:0 OR T20)

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

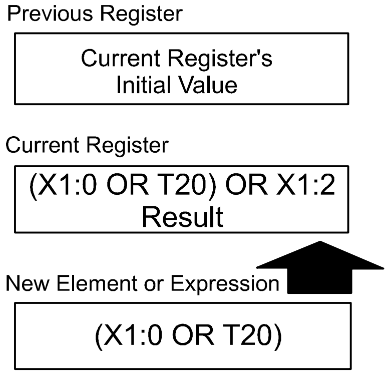
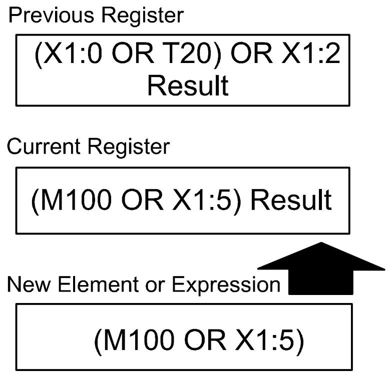
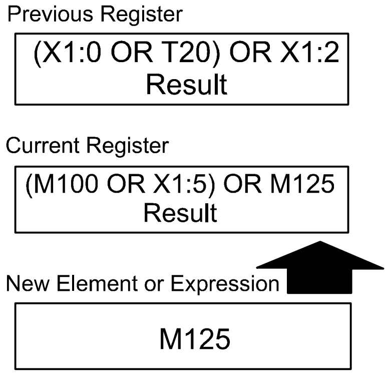
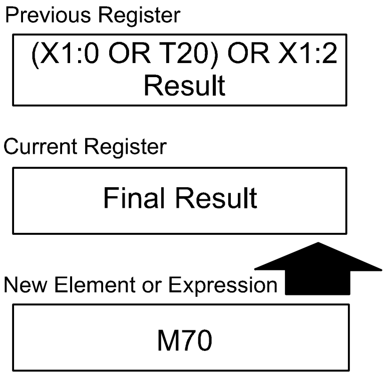
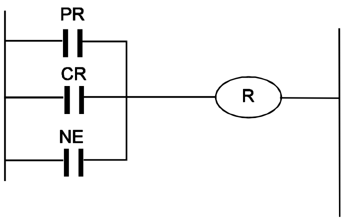
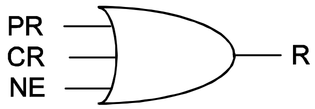
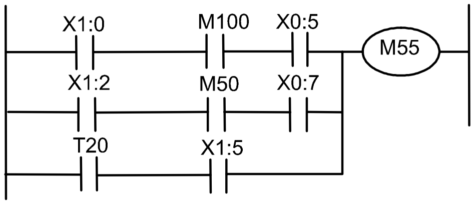
Examples	Explanation
OR X1:2	Current register's value is ORed with value in X1:2 register, result remains in current register. OR operation does not change value in previous register.  Previous Register Current Register's Initial Value Current Register (X1:0 OR T20) OR X1:2 Result New Element or Expression (X1:0 OR T20)
LD (M100 OR X1:5)	Value in current register is copied to previous register. Resulting value of new expression is loaded into current register.  Previous Register (X1:0 OR T20) OR X1:2 Result Current Register (M100 OR X1:5) Result New Element or Expression (M100 OR X1:5)
OR M125	Value in current register is ORed with value of new element, result remains in current register. OR operation does not change value in previous register.  Previous Register (X1:0 OR T20) OR X1:2 Result Current Register (M100 OR X1:5) OR M125 Result New Element or Expression M125

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

<table>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td>ANB M70</td><td colspan="2">Value in previous register, current register and new element are ANDed together to produce result that remains in current register.

</td></tr>
<tr><th>ORB</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>Performs Boolean OR function with value in previous register, value in current register and new element's value.</td><td>ORB [element]</td><td>Y registers
M registers
T registers
S registers
X registers
Expressions</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>ORB
<table>
<tr><th>Previous Register (PR)</th><th>Current Register (CR)</th><th>New Element (NE)</th><th>Result (R)</th></tr>
<tr><td>F</td><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>T</td><td>T</td></tr>
</table></td><td>
</td><td>
</td></tr>
<tr><th>Examples</th><th colspan="2">Explanation</th></tr>
<tr><td><b>ORB</b> Example #1

LD (X1:0 AND M100)</td><td colspan="2">

Copies value in current register to previous register, evaluates new expression and loads resulting value into current register.</td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

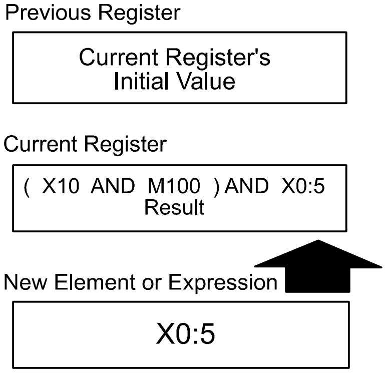
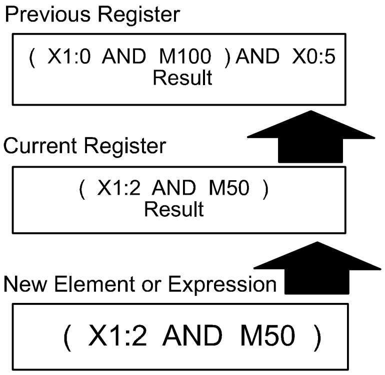
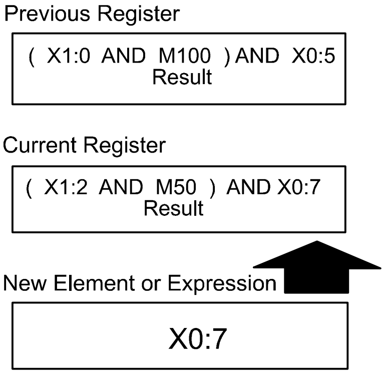
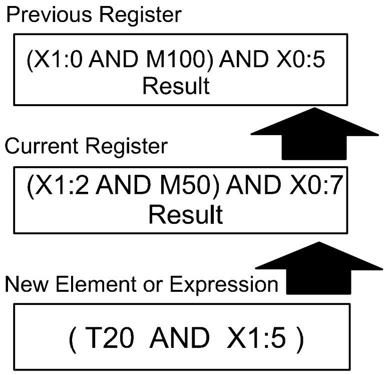
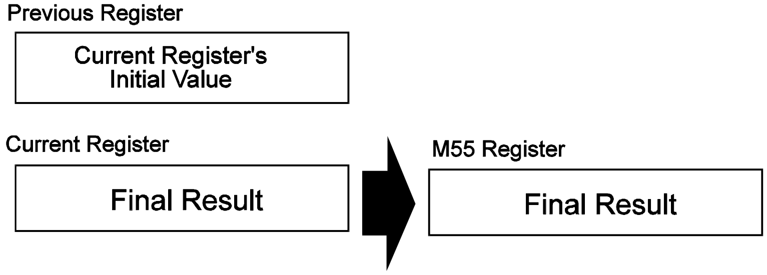
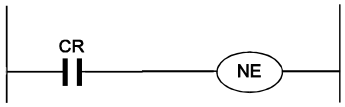
Examples	Explanation
AND X0:5	 Previous Register Current Register's Initial Value Current Register (X10 AND M100) AND X0:5 Result New Element or Expression X0:5
ORB Example #2 LD (X1:2 AND M50)	Value in Current register is copied to previous register; new expression is evaluated and result remains in current register. **NOTE:** The value shown in the previous register is the result of ORB Example #1.  Previous Register (X1:0 AND M100) AND X0:5 Result Current Register (X1:2 AND M50) Result New Element or Expression (X1:2 AND M50)
AND X0:7	Value in current register is ANDed with new element. Result remains in current register.  Previous Register (X1:0 AND M100) AND X0:5 Result Current Register (X1:2 AND M50) AND X0:7 Result New Element or Expression X0:7
ORB (T20 AND X1:5)	New expression is evaluated and its value ORed with value in current register and value in previous register; final result remains in current register.

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

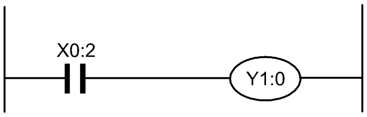
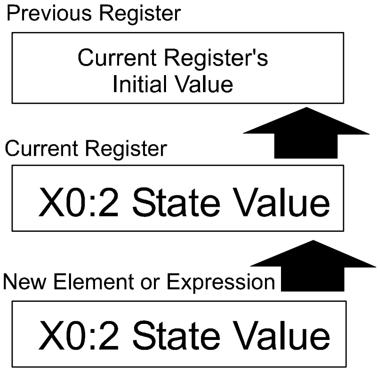
Examples	Explanation
	 Previous Register (X1:0 AND M100) AND X0:5 Result Current Register (X1:2 AND M50) AND X0:7 Result New Element or Expression (T20 AND X1:5)
OUT M55	Copies value in current register to M55 multifunction register.  Previous Register Current Register's Initial Value Current Register Final Result M55 Register Final Result

SET	Syntax	Valid Elements
This instruction latches the new element to a TRUE value for subsequent cycles. If current register holds a TRUE value, a TRUE state value is copied into the new element's register. If current register holds a FALSE value, no activity occurs. A subsequent MOV statement or a RES instruction can be used to unlatch the register.	SET [element]	Y registers M registers
Truth Table	**Ladder Equivalent**	**Logic Symbol**
See SET truth table below		

SET

Current Register (CR)	New Element (NE) Prev. State	New Element (NE) New. State
F	F	F
F	T	T
T	F	T
T	T	T

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

Examples	Explanation
SET Example #1	
LD X0:2	Loads current value into previous register and loads value in X0:2 register into current register.  Previous Register Current Register's Initial Value Current Register X0:2 State Value New Element or Expression X0:2 State Value
SET Y1:0	Sets value in Y1:0 register to true if X0:2 was true. No action taken if X0:2 was false.

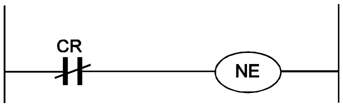
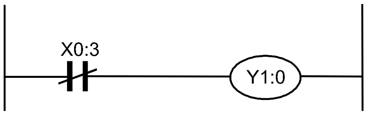
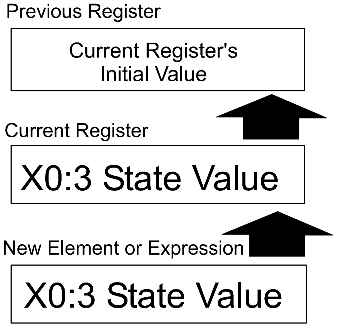
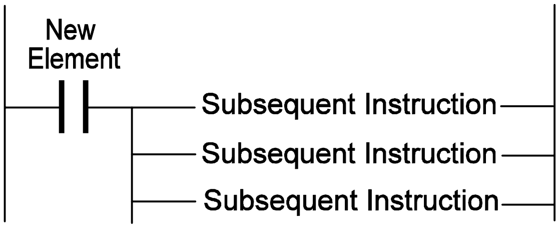
<table>
<tr><th>RES</th><th>Syntax</th><th>Valid Elements</th></tr>
<tr><td>This instruction resets the new element to a FALSE value for subsequent cycles.
If current register holds a TRUE value, FALSE is copied in new element's register.
If current register holds a FALSE value, no activity occurs.
A subsequent MOV statement or a SET instruction can be used to relatch the register.</td><td>RES [element]</td><td>Y registers
M registers</td></tr>
<tr><th>Truth Table</th><th>Ladder Equivalent</th><th>Logic Symbol</th></tr>
<tr><td>
<table>
<tr><th colspan="3">RES</th></tr>
<tr><th>Current Register (CR)</th><th>New Element (NE) Prev. State</th><th>New Element (NE) New. State</th></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td></tr>
<tr><td>T</td><td>T</td><td>F</td></tr>
</table>
</td><td> </td><td></td></tr>
</table>

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

Examples	Explanation
RES Example #1	
LD X0:3	Copies value from current register into previous register and loads value from X0:3 register into current register.  Previous Register Current Register's Initial Value Current Register X0:3 State Value New Element or Expression X0:3 State Value
RES Y1:0	Resets value in Y1:0 register to FALSE if X0:3 was TRUE. No action taken if X0:3 was FALSE.

CTL/CTR	Syntax	Valid Elements
Used in pairs. CTL - ANDs specified element with all subsequent instructions until deactivated. CTR - deactivates any active CTL instructions.	Activate CTL [element] Deactivate CTR	Y registers M registers T registers S registers X registers

	Ladder Equivalent
	 New Element Subsequent Instruction Subsequent Instruction Subsequent Instruction

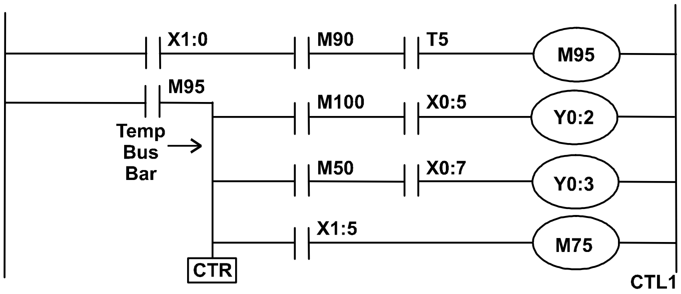
Examples	Explanation
CTL/CTR Example #1	 X1:0 M90 T5 M95 M95 M100 X0:5 Y0:2 Temp Bus Bar M50 X0:7 Y0:3 X1:5 M75 CTR CTL1 Moves value from current register to previous register and loads new expression result into the current register.

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

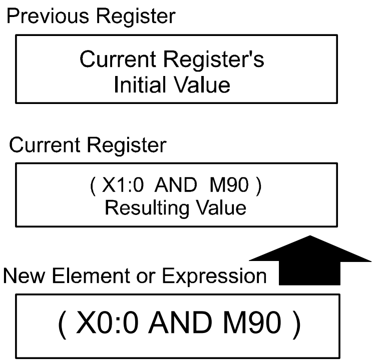
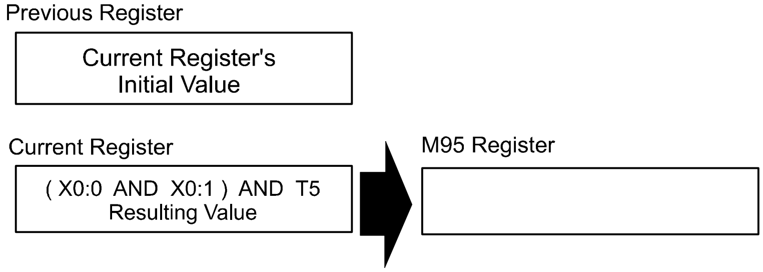
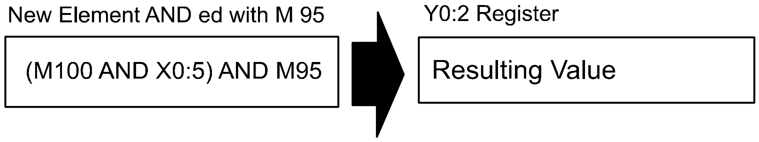
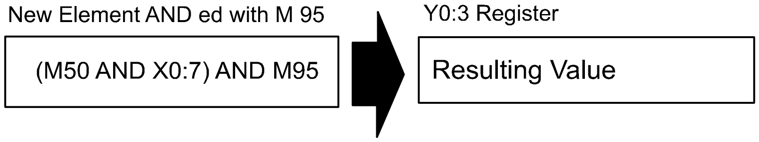
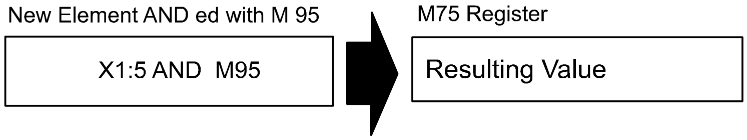
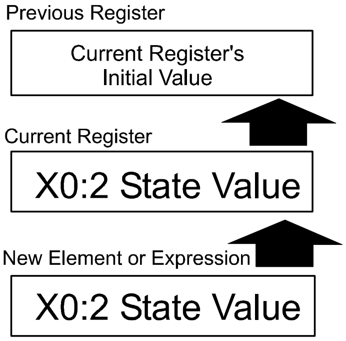
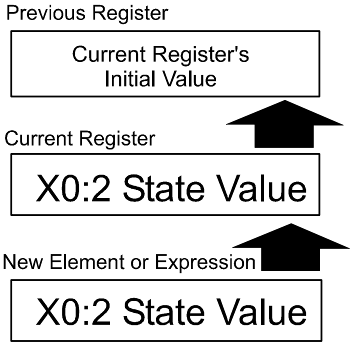
Examples	Explanation
LD (X1:0 AND M90)	 Previous Register Current Register's Initial Value Current Register (X1:0 AND M90) Resulting Value New Element or Expression (X0:0 AND M90)
AND T5	Value in current register is ANDed with T5 register's state value. Result remains in current register.
OUT M95	Copies the value in the current register to multifunction register M95.  Previous Register Current Register's Initial Value Current Register (X0:0 AND X0:1) AND T5 Resulting Value M95 Register
CTL M95	Specifies that value in M95 register will be ANDed with all subsequent instructions.
MOV (M100 AND X0:5) Y0:2	New expression result is ANDed with value in M95 register. Result is copied directly to Y0:2 register.  New Element AND ed with M 95 (M100 AND X0:5) AND M95 Y0:2 Register Resulting Value
MOV (M50 AND X0:7) Y0:3	Next expression is ANDed with value in M95 register. Result is copied directly to Y0:3 register.  New Element AND ed with M 95 (M50 AND X0:7) AND M95 Y0:3 Register Resulting Value
MOV X1:5 M75	Next expression is ANDed with value in M95 register. Result is copied directly to M75 register.  New Element AND ed with M 95 X1:5 AND M95 M75 Register Resulting Value
CTR	CTL function deactivated.

Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

DEC	Syntax	Valid Elements
Every cycle that the current register value is true causes a decrease in the new element's numeric value by 1. Numbers cannot decrease to less than zero.	DEC [element]	M registers
Examples	**Explanation**	
DEC Example #1 LD X0:2	Copies value from current register into previous register. Loads value from X0:2 register into current register.  Previous Register Current Register's Initial Value Current Register X0:2 State Value New Element or Expression X0:2 State Value	
DEC M80	If current register's value went from false to true during this cycle, the M80 register value is decreased by 1.	
INC	**Syntax**	**Valid Elements**
Every cycle that the current register value remains TRUE, the new element's numeric value increases by 1.	INC [element]	M registers
Examples	**Explanation**	
INC Example #1 LD X0:2	Copies value from current register into previous register and loads value from X0:2 register into current register.  Previous Register Current Register's Initial Value Current Register X0:2 State Value New Element or Expression X0:2 State Value	
INC M80	If current register's value went from false to true during this cycle, the M80 register value is increased by 1.	
NOP	**Syntax**	**Valid Elements**
No operation is performed.	NOP	

Section 5 - Timers

Timed events count through as many program cycles as are required in the course of their operation. This is one reason for short IPI cycles being efficient. The shorter the cycle, the closer timers can operate to real time.

Timers employ two registers: a state register that contains the true/false value used by the program and a counting register to count down time. The counting register's real-time numeric value in a cycle can be accessed using an RD instruction. The timer's state value is normally used to generate an output.

Use the following instructions to generate an output with a timer:

OUT instruction

This instruction appears first in the program and always uses the OUT operation code. It assigns the timer identifier number, defines the current register's state value (at the point it appears in the program as the source or triggering event), and specifies the timer configuration and countdown period.

MOV instruction

Subsequent references to a timer register will move the real-time state value of the timer register to some other register, where it is used as a condition or to produce an output.

There are three timer configurations:

Delayed On

Format:
TON X.X

X=time in seconds

If the current register value changes from FALSE to TRUE during the current cycle, the timer begins a countdown that lasts the specified number of seconds. When the countdown is complete, the timer's state register loads the high. In a future cycle, if the current register changes to FALSE, at the same time in the program, the timer's state register returns to low, with no delay. The timer will restart the countdown on the next TRUE.

Delayed Off

Format:
TOFF X.X

X=time in seconds

If the current register's value changes from TRUE to FALSE during the current cycle, the timer begins a countdown that lasts the specified number of seconds. When countdown is complete, the timer's state register loads the FALSE. In a future cycle, if the current register's state changes to TRUE, at the same time in the program, the timer's state register returns to TRUE, with no delay. The timer will restart the countdown on the next FALSE.

Delayed On Then Off

Format:
T1 X.X

X=time in seconds

If the current register's state (FALSE/TRUE) becomes the inverse of the current timer's state value (TRUE/FALSE), the timer begins a countdown. When the countdown is complete, the timer's state value switches between TRUE/FALSE.

In a future cycle, if the current register's state fluctuates between true and FALSE before the countdown finishes, it will have no effect on the timer's state value.

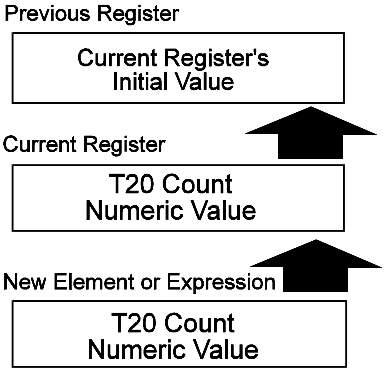
All timer definition instructions use the OUT or MOV operations, as shown in **Table 5-1**. Refer to Table 5-2, Detailed Descriptions and Examples of Operands.

Table 5-1, Timer Instruction Definitions

OUT Instruction	Syntax	Valid parameters
This instruction must precede all MOV instructions for the same timer.	OUT T[type] [identifier] [time]	Types: ON OFF {blank} Identifiers: 0 through 49 Time: Decimal seconds.
Examples	**Explanation**	
Delayed On OUT TON10 0.1	If the current register's value changes from FALSE to TRUE during the current cycle, the T10 timer begins a 100 msec countdown. In 100 msec, the T10 register will load and maintain a TRUE value. In a future cycle, if the current state register turns from TRUE to FALSE at the same time in the program, the T10 register will load and maintain a FALSE value with no delay.	
MOV T10 Y1:0	Copies the value in the T10 register to the Y1:0 register every cycle.	
Delayed Off OUT TOFF10 0.1	If the current register's value changes from TRUE to FALSE during the current cycle, the T10 timer begins a 100 msec countdown. In 100 msec, the T10 register will load and maintain a FALSE value. In a future cycle, if the current state register turns from FALSE to TRUE at the same time in the program, the T10 register will load and maintain a TRUE value with no delay.	
MOV T10 Y1:0	Copies the value in the T10 register to the Y1:0 register every cycle.	
OUT Instruction	**Syntax**	**Valid parameters**
Delayed On/Off OUT T10 0.1	If the current register's value changes from FALSE to TRUE during the current cycle, the T10 timer begins a 100 msec countdown. In 100 msec, the T10 register will switch its state value. In a future cycle, if the current state register turns from TRUE to FALSE at the same time in the program, it has no effect on the state in the T10 register.	
MOV T10 Y1:0	Copies the value in the T10 register to the Y1:0 register every cycle.	

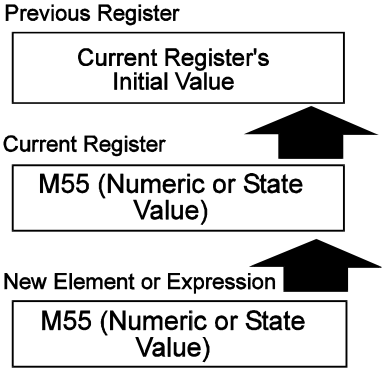
Table 5-2, Detailed Descriptions and Examples of Operands

RD	Syntax	Valid Elements
Loads element value into current register. Copies any value already in the current register into the previous register. If element value is numeric, it is loaded without conversion to a state. If element value is a state value, it is loaded as a state value. RD can be used to access a numeric value after a mathematical operation or to load the count value of a timer.	RD [element]	Y registers M registers T registers S registers X registers

Examples	Explanation
RD Example #1 Read timer count. RD T20	Copies T20’s timer count into current register as a numeric value. Timer’s state value is not used.  Previous Register Current Register's Initial Value Current Register T20 Count Numeric Value New Element or Expression T20 Count Numeric Value

(Continued…)

Table 5-2, Detailed Descriptions and Examples of Operands (Continued)

Examples	Explanation
RD Example #2 Read multifunction register value. RD M55	Copies M55 value into current register. If value is numeric, it is not converted to a state.  Previous Register Current Register's Initial Value Current Register M55 (Numeric or State Value) New Element or Expression M55 (Numeric or State Value)

RST	Syntax	Valid Elements
Restart instruction that restarts countdown timer if current register's state value is TRUE and designated timer is currently in a delay countdown state.	RST [element]	T registers

Example	Explanation
RST Example #1 RST T1	T1's count value is set to the configured preset value. Timer's logic state is not affected.

Timer Off (TOFF) Command

In the example in **Figure 5-1**, input X0:0 initiates the TOFF command. At 1 second, the input goes low. The output of timer T0 stays high until the timer counts to 5 seconds. Then, the output goes low. When the input goes high, the output immediately goes high. The timer is non-retentive, so that the transitions from 14 seconds to 19 seconds do not affect the output.

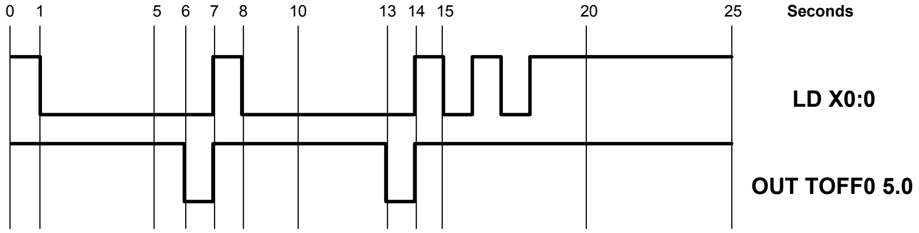


Figure 5-1, Timer Off Command

Timer Delayed On Then Off (T) Command

In the example in **Figure 5-2**, input X0:1 initiates the Timer Delayed On Then Off command. At 1 second, the input goes high. The output of T1 stays low until the timer counts to 5 seconds. Then, the output goes high. The output stays high until the input goes for 5 seconds, then the output goes low. Inputs of less than the timer value cause no change in output, as in the transitions from 13 seconds to 16 seconds. The timer is non-retentive, so that each time the input changes the count is restarted.

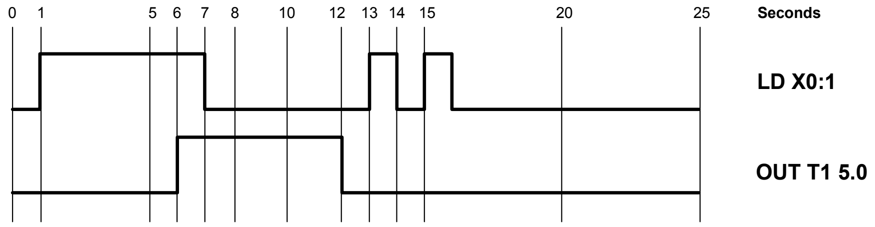


Figure 5-2, Timer Delayed On Then Off Command

Timer On (TON) Command

In the example in **Figure 5-3**, input X0:2 initiates the TON command. At 1 second, the input goes high. The output of timer T0 stays low until the timer counts to 5 seconds. Then, the output goes high. When the input goes low, the output immediately goes low. The timer is nonretentive, so that the transitions from 14 seconds to 19 seconds do not affect the output.

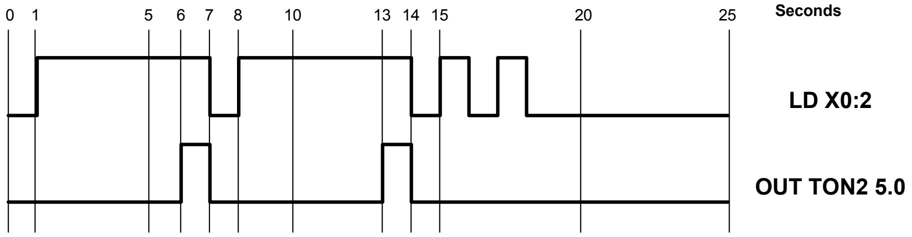


Figure 5-3, Timer On Command

Section 6 - Advanced IPI Instructions

This section describes advanced IPI instructions.

IF/ELS/EDF Instructions

Conditional statements allow the programmer to vary the instructions, based on the value of a given register or expression. Refer to **Table 6-1** for the available conditional statement commands.

Table 6-1, Conditional Instructions

Conditional Instruction	Function
IF	If
ELS	Else
EDF	End if
CLP	Conditional jump
EJP	End jump

IF, ELS, EDF, CLP, and EJP form instruction sets.

Each complete set of conditional instructions must be numbered. Both the compiler and the IPI interpreter use this block number to separate nested IFs. IF block numbers may be reused at different points in the program, but should be unique regarding currently active IF levels. The block number follows the "IF" command, as follows:

Format 1: IF [block number]

An IF statement may include an optional new expression or element. If the IF statement includes a new expression or element, the conditional statement is based on its value. Otherwise, the value in the default register is used. The currently active register is the default.

Format 2: IF [block number] [optional element or expression]

Refer to Format 2. When the CNC executes an IF statement, it evaluates the value in the current register of the new element or expression. If the value is True, the IPI interpreter will execute the subsequent instructions until it encounters a matching ELS or EDF. If the new element or expression is False, the interpreter skips to the matching ELS or EDF instruction.

When a matching ELS is encountered, if the new element/expression or current register is False, the instructions following the ELS are processed.

A matching EDF instruction terminates the process and sequential program execution resumes.

IF/ELS/EDF sets can be nested. A nested IF/EDF set can be placed within a parent CJP/EJP or IF/EDF set. The nested set must be closed before the parent set is closed. The programmer can nest conditional statement sets up to ten levels deep. Refer to the examples in **Table 6-2**.

Table 6-2, Conditional Statement Programming - Examples

IF/ELS/EDF	Syntax	Valid Parameters

IF – Begins conditional statement. CNC executes subsequent instructions if relevant register value is True. The relevant register value is the current register or the new element register.	IF [block number]
	– or –
	IF [block number] [element]
		Elements:
		Y registers
		M registers
		T registers
		S registers
		X registers

IFI – Inverse IF. Also used to begin a conditional statement. CNC executes subsequent instructions if relevant register value is TRUE. The relevant register value is the current register or the new element register.	IF [block number]
	– or –
	IF [block number] [element]
		Elements:
		Y registers
		M registers
		T registers
		S registers
		X registers

ELS – Provides intermediate step in the process. Executes subsequent instructions if new expression, new element or current register is FALSE.	ELS [block number]	Block Numbers: Any integer, all numbers must match.
EDF – Terminates conditional instruction set.	EDF [block number]	

Examples	Explanation
IF 25 First Instruction Set ELS 25 Second Instruction Set EDF 25	If value in current register* is TRUE, first instruction set is executed and the second instruction set is ignored. If value in current register is FALSE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
IF 80 X5 First Instruction Set ELS 80 Second Instruction Set EDF 80	If value in X5 register is TRUE, first instruction set is executed and the second instruction set is ignored. If value in X5 register is FALSE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
IF 60 (M50 NE 25) First Instruction Set ELS 60 Second Instruction Set EDF 60	If result of expression (M50 NE 25) is TRUE, first instruction set is executed and the second instruction set is ignored. If result of expression (M50 NE 25) is FALSE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
***NOTE:** When no element is provided in the IF statement block, the CNC uses the default register, which is the currently active register.	

(Continued…)

Table 6-2, Conditional Statement Programming – Examples (Continued)

Examples	Explanation
IFI 25 First Instruction Set ELS 25 Second Instruction Set EDF 25	If value in current register* is FALSE, first instruction set is executed and the second instruction set is ignored. If value in current register is TRUE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
IFI 80 X5 First Instruction Set ELS 80 Second Instruction Set EDF 80	If value in X5 register is FALSE, first instruction set is executed and the second instruction set is ignored. If value in X5 register is TRUE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
IFI 60 (M50 NE 25) First Instruction Set ELS 60 Second Instruction Set EDF 60	If result of expression (M50 NE 25) is FALSE, first instruction set is executed and the second instruction set is ignored. If result of expression (M50 NE 25) is TRUE, first instruction set is ignored and second instruction set is executed. EDF terminates instruction set.
***NOTE:** When no element is provided in the IF statement block, the CNC uses the default register, which is the currently active register.	

Conditional Jumps

The conditional jump (CLP) instruction acts like an IF/ELS/EDF statement with no instructions given between IF and ELS.

Format 1: CLP [block number]

A CLP statement may include an optional new expression or element. If the CLP statement includes a new expression or element, the conditional statement is based on its value. Otherwise, the value in the default register is used. The current register is the default.

Format 2: IF [block number] [optional element or expression]

When the CNC executes a conditional jump, the value in the current register or the new element/expression is evaluated. If the value is False, the IPI interpreter will execute the subsequent instructions. If the value is True, the program jumps to the end jump (EJP) instruction.

In all cases, the EJP instruction concludes the instruction set and sequential program execution resumes.

CLP/EJP sets can be nested. A nested CLP/EJP set may be placed within a parent CLP/EJP or IF/EDF set. The nested set must be closed before the parent set is closed. The programmer can nest up to ten levels of conditional statement sets. Refer to the examples in **Table 6-3**.

Table 6-3, Conditional Jump Programming - Examples

CLP/EJP	Syntax	Valid Elements
CLP – Begins conditional statement. Executes subsequent instructions if new element, new expression or current register value is FALSE. Jumps to EJP instruction if TRUE.	CLP [block number] [element] **– or –** CLP [block number]	Elements: Y registers M registers T registers S registers X registers Block Numbers: Any integer, all numbers must match.
EJP – Ends conditional jump instruction set.	EJP [block number]	

Examples	Explanation
CLP 20 Conditional Instructions EJP 20	If value in current register* is TRUE, conditional instructions are ignored. CNC jumps to EJP. If value in current register is FALSE, conditional instructions are executed. EJP terminates instruction set.
CLP 35 X0:5 Conditional Instructions EJP 35	If value in X0:5 register is TRUE, conditional instructions are ignored. CNC jumps to EJP. If value in X0:5 register is FALSE, conditional instructions are executed. EJP terminates instruction set.
CLP 55 (M50 NE 25) Conditional Instructions EJP 55	If resulting value of expression (M50 NE 25) is TRUE, conditional instructions are ignored. CNC jumps to EJP. If value of expression is FALSE, conditional instructions are executed. EJP terminates instruction set.

*__NOTE:__ When no element is provided in the CLP statement block, the CNC uses the default register, which is the currently active register.

Refer to **Table 6-4**.

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions

MVA	Syntax	Valid Elements
Digital value of analog input at specified node is loaded into the specified multifunction register.	MVA [element] [element]	Y registers M registers
Example	**Explanation**	
MVA Example #1 Two elements. MVA Y0:5 M100	Digital value of analog input at node 0 is loaded into multifunction register 100.	
OKBD	**Syntax**	**Common Key Codes**
Output Keyboard instruction is used to output key codes to the CNC in an IPI program. The CNC interprets these key codes as if the user had pressed the corresponding key. Only one key code can be passed per IPI scan. For a key code to be interpreted by the CNC, it must differ from scan to scan.	OKBD [xxxxH] **– or –** OKBD [xxxxX]	CNC Key PC Key (Hex Notation) Start ALT_S (11FH) Hold ALT_H (123H) Spindle CW ALT_F (121H) Spindle CCW ALT_G (122H) Spindle Stop ALT_O (118H) Clear ALT_C (12EH) **NOTE:** Hexadecimal notation can be indicated by an X or H following the number.
Example	**Explanation**	
OKBD Example #1 OKBD 11FH	The START (ALT_S) key code is output. It has the same effect as physically pressing the required key on a PC keyboard or console keypad.	

(Continued…)

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions (Continued)

OTI	Syntax	Valid Elements
Output until input instruction. The specified output is pulsed for 30 sec, or until the corresponding input is energized. The output can be a Y value. **NOTE:** The input number may be different from the output number. In this case, you must use OTI within the same node. An LD or LDI command must be programmed directly before the OTI in order to specify the input bit. Additionally, the qualifying LD or LDI must be an expression using physical input bits. The specified input bit is the same node location as the specified output on the corresponding input port. Load input with either LD or LDI. LD is for a positive trigger and LDI is for a negative trigger. Follow immediately (or before another Load instruction) with the OTIstatement. OTI is terminated when **E-STOP** is pressed. See also SOTI (Super OTI) and COTI (cancels OTI or SOTI).	LD Xn:b OTI Yn:b **– or –** LDI Xn:b OTI Yn:b n = node b = bit	X registers Y registers

(Continued…)

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions (Continued)

Example	Explanation
OTI Example #1 LD X1:3 OTI Y1:0	The Y1:0 output signal is pulsed (goes high) for 30 sec or until the X1:3 input is detected (goes high). When the X1:3 input is detected, the Y1:0 signal goes low. There is a 30 second watchdog (hard coded) for OTI and SOTI commands. If it times out, OTIFLAG is set to 3. OTIFLAG (M30) has the following values: OTIFLAG=0 When an OTI/SOTI command is executed, OTIFLAG is set to zero (0) and it remains at zero until the command is ended. OTIFLAG=1 When OTI/SOTI command finish successfully, OTIFLAG is set to 1. OTIFLAG=2 When a COTI command is issued, OTIFLAG is set to 2 after OTI/SOTI is cancelled. OTIFLAG=3 When there is timeout (OTI/SOTI did not complete in 30 seconds), OTIFLAG is set to 3.
OTI Example #2 LDI X1:3 OTI Y1:0	The Y1:0 register is pulsed (goes high) for 30 seconds or until the input X1:3 is detected (goes low). See OTIFLAG (M30) above.

(Continued…)

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions (Continued)

OWI	Syntax	Valid Elements
Output When Input. The specified output is latched on immediately on input. Transition must be from FALSE to TRUE. **NOTE:** The input number may be different from the output number. In this case, you must use OTI within the same node. An LD or LDI command must be programmed directly before the OTI in order to specify the input bit. Additionally, the qualifying LD or LDI must be an expression using physical input bits. The specified input bit is the same node location as the specified output on the corresponding input port. Load input with either LD or LDI. LD is for a positive trigger and LDI is for a negative trigger. Follow immediately (or before another Load instruction) with the OTIstatement.	LD Xn:b OWI Yn:b **– or –** LDI Xn:b OWI Yn:b n = node b = bit	X registers Y registers

Example	Explanation
OWI Example #1 LD X1:2 OWI Y1:5	When the X1:2 input transitions from low to high, the Y1:5 output will set high. The output will remain high until cleared by another instruction (such as RES or MOV 0). If the starting input state of X1:2 is high, the output will not set until the input first goes low and then a low to high transition is detected.
OWI Example #2 LDI X1:2 OWI Y1:5	When the X1:2 input transitions from high to low, the Y1:5 output will set high. The output will remain high until cleared by another instruction (such as RES or MOV 0). If the starting input state of X1:2 is low, the output will not set until the input first goes high and then a high to low transition is detected.

(Continued…)

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions (Continued)

SOTI	Syntax	Valid Elements
Super OTI works like OTI but the number of input pulses required to turn the output off can be specified (instead of it being hard-coded to 1 as in OTI). The specified output is pulsed for 30 sec, or until the corresponding input is energized. The output can be a Y value. **NOTE:** The input number may be different from the output number. In this case, you must use SOTI within the same node. An LD or LDI command must be programmed directly before the SOTI in order to specify the input bit. Additionally, the qualifying LD or LDI must be an expression using physical input bits. The specified input bit is the same node location as the specified output on the corresponding input port. Load input with either LD or LDI. LD is for a positive trigger and LDI is for a negative trigger. Follow immediately (or before another Load instruction) with the SOTI statement. SOTI is terminated when **E-STOP** is pressed.	LD Xn:b SOTI Yn:b counter **– or –** LDI Xn:b SOTI Yn:b counter n = node b = bit	X registers Y registers

(Continued…)

Table 6-4, Detailed Descriptions and Examples of Advanced IPI Instructions (Continued)

Example	Explanation
SOTI Example #1 LD X1:3 SOTI Y1:0 M40	The Y1:0 input will stay ON until the number of pulses specified in M40 is received. As the pulses are received, SOTICNT (M31) increments. When SOTICNT=M40, then the output is turned OFF. There is a 30 second watchdog (hard coded) for OTI and SOTI commands. If it times out, OTIFLAG is set to 3. OTIFLAG has the following values: OTIFLAG=0 When an OTI/SOTI command is executed, OTIFLAG is set to zero (0) and it remains at zero until the command is ended. OTIFLAG=1 When OTI/SOTI command finish successfully, OTIFLAG is set to 1. OTIFLAG=2 When a COTI command is issued, OTIFLAG is set to 2 after OTI/SOTI is cancelled. OTIFLAG=3 When there is timeout (OTI/SOTI did not complete in 30 seconds), OTIFLAG is set to 3. SOTI also keeps track of input pulses/count internally using SOTICNT.
SOTI Example #2 LDI X1:3 SOTI Y1:0 M70	The Y1:0 register (goes high) until the input X1:3 is detected the number of times corresponding to the value in M70 before the output Y1:0 signal goes low. See OTIFLAG (M30) and SOTICNT (M31) above.

COTI	Syntax	Valid Elements
Cancel OTI instruction cancels OTI or SOTI command immediately. OTIFLAG is to set to 2 and SOTICNT stays at the count number at the moment of cancellation.	COTI [No parameters needed, current register must be TRUE.]	

Example	Explanation
COTI Example #1 COTI	

See "Section 7, Program 5 – IPI Example."

Section 7 - Programming Tips and Examples

Compiler Directives

A compiler directive is an instruction to the compiler that is not compiled as part of the IPI program. A compiler directive produces no binary code for the IPI interpreter. Directives are indicated by a pound sign (#) as the first character of the line, followed by the required directive.

DEFINE

Format: #DEFINE [label name] [label meaning]

The #DEFINE directive is used to define a label. To define a label, use the DEFINE directive, name the label and specify the meaning of the label, in that order. (For example: #DEFINE XP_LIMIT X0:8.)

The label in this example defines the label "XP_LIMIT" and ties the label to input X0:8. XP_LIMIT is the X-axis positive vector limit. After the label is defined, any time the compiler encounters the program string XP_LIMIT, and it will replace the text with X0:8.

In future references, the X positive vector limit switch can be referenced as XP_LIMIT or X0:8.

LIST

Format: #LIST

The LIST directive instructs the compiler to generate a file listing output. When the compiler encounters this directive for the first time, and the list mode is not on, the compiler recompiles with List Mode activated. It is recommended that the programmer place the LIST directive close to the beginning of the source file. The result is program name first.

MAXSIZE

Format: #MAXSIZE [nnnn]

Instructs the compiler to generate an error if the actual number of bytes generated by the instructions exceeds that of the number specified in the MAXSIZE directive. This is to assist the programmer when program space is limited. Maxsize refers to the total number of bytes generated by the IPI instructions.

MAXSTEPS

Format: #MAXSTEPS [nnnn]

The MAXSTEPS directive instructs the compiler to generate an error if the actual number of compiled instructions exceeds that of the number specified in the MAXSTEPS directive. This is supplied to assist the programmer in time-critical applications.

RANGE

Format: #RANGE [Element] [starting value] [ending value]

The range directive defines a numeric range for a specified element. This is supplied to reduce errors due to hardware limitations.

For example: The I/O Board has sixteen inputs. The programmer wants to avoid calling an input higher than fifteen. The corresponding range directive would be:

#RANGE X 0 15, where

X is the element (input)

0 is the range minimum

15 is the range maximum

The CNC would flag any X numbers outside the defined range.

SYNTAX

Format: #SYNTAX

The syntax directive instructs the compiler not to produce an output file, but to check syntax only. If the syntax directive is used, it must appear before the first statement that produces output.

Plan the Program

Before you begin to create an IPI program, plan the task carefully. Define all tasks that the IPI will be required to perform, including specific inputs and outputs. After the particulars are defined, you can formulate methods to achieve the required tasks.

In the planning phase, ladder diagrams are a good way to visualize circuits. When you create a ladder diagram, keep each circuit as simple as possible. Simple circuits are easy to understand and troubleshoot. When you convert the diagram to IPI code, use as few instructions as possible. This will help the IPI program to execute as efficiently as possible.

Using Labels

Use labels to identify specific inputs, outputs, internal elements, delay times, elements, and other constants. You can also use labels to rename instructions. Labels cannot be used to rename a compiler directive.

Always define the label first. The #DEFINE compiler directive provides the best method by which to define the label. A table of predefined labels exists. You cannot redefine these labels. Always rename an IPI instruction when you use a label. If an IPI instruction is used as a label, that instruction will no longer operate.

Labels can be used to define other labels. For example, if the programmer defines DELAY as 0.1 and TIMER as TON5, the label DELAYTIMER can be defined as TIMER DELAY. The compiler will translate the two labels, and then define DELAYTIMER as TON5 0.1.

NOTE: Embedded spaces are not allowed in the label itself, but are allowed in the label translation.

Using Conditional Execution

You can use conditional execution to alter the programmed circuit based on logical conditions. However, extra care must be taken when you use the CTL instruction inside a conditional execution. Refer to **Table 7-1**.

Table 7-1, Conditional Execution within Conditional Statements

Block	Function
IF 2	IF block beginning conditional statement.
|	
CTL M20	When the IF block executes, control M20 is executed, but there is no control return inside the IF block. After the IF block executes, the control M20 will still be in effect. If the condition for the IF block is FALSE, the control M20 will not be executed, and therefore will never take effect.
|	
EDF	EDF closing conditional statement.

ANILAM recommends that the conditional blocks be self-contained blocks of code. All controls should have control returns inside the IF block.

Using Sequence States

Sequence States can be used to create a stepladder effect on the IPI program. Only one state can be active at a time. When a Sequence State is set to true, all other states are set to false.

Programming Examples

This section includes several IPI program examples that include most of the operands described in the preceding sections. Refer to Table 4-5, Summary of IPI Operands, for a summary of available IPI operation codes. Refer to Table 4-6, Detailed Descriptions and Examples of Operands, for detailed explanations and examples of each operation code.

Program 1 – Basic IPI Example

The following is a complete basic IPI program.

```
*BASIC IPI PROGRAM

*M0   THRU M63  SYSTEM REGISTERS
*M224 THRU M239 IPI AND CNC SHARED REGISTERS
*M240 THRU M255 NON-VOLATILE REGISTERS

#DEFINE FINWAIT              M64     *COMPILER ASSIGNS LABEL FINWAIT TO M64
#DEFINE MFTN2                M65     *COMPILER ASSIGNS LABEL MFTN2 TO M65
#DEFINE MFTN3                M66     *COMPILER ASSIGNS LABEL MFTN3 TO M66
#DEFINE MFTN4                M67     *COMPILER ASSIGNS LABEL MFTN4 TO M67
#DEFINE MFTN5                M68     *COMPILER ASSIGNS LABEL MFTN5 TO M68
#DEFINE MFTN8                M69     *COMPILER ASSIGNS LABEL MFTN8 TO M69
#DEFINE MFTN9                M70     *COMPILER ASSIGNS LABEL MFTN9 TO M70
#DEFINE MFTN30               M71     *COMPILER ASSIGNS LABEL MFTN30 TO M71

START                        *DEFINES REPEATING PORTION OF PROGRAM

*FINISH PULSE GENERATION

LD (MFLAG OR SFLAG)
OR (TFLAG OR HFLAG)
SET FINISH                   *SETS FINISH HIGH ON ANY FLAG

LDI FINWAIT                  *SET FINWAIT HIGH DURING OPERATIONS THAT
                             *REQUIRE PROGRAM HOLD TILL COMPLETE
AND FINISH                   *LOOK FOR FINISH AND WITH FINWAIT LOW
OUT TON0 0.1                 *0.1 SEC FINISH PULSE DURATION

IF 0 T0                      *IF #0 LOOKS FOR TO ACTIVE
RES FINISH                   *RESETS FINISH AFTER FINISH TIMER T0
```

```
EDF 0                          *END IF #0

*BASIC M-FUNCTIONS: SPINDLE FORWARD, REVERSE, OFF; COOLANT ON AND OFF;
*PROGRAM END, SUBROUTINE END.

*THESE MULTIFUNCTION REGISTERS ARE VISABLE IN THE DEFAULT IPI MONITOR
*DISPLAY. USE THESE REGISTERS TO SET I/O BOARD OUTPUTS AS REQUIRED.

LD ( MCODE EQ 2 )              *SET MFTN2 FOR PROGRAM END: REGISTER M64
OUT MFTN2

LD ( MCODE EQ 3 )              *SET MFTN3 FOR SPINDLE FWD: REGISTER M65
OR MFTN3                       *LATCH ON
ANI MFTN5                      *DISABLE ON M5
ANI ( MFTN2 OR MFTN30 )        *DISABLE ON M2 OR M30
RES MFTN4                      *RESET M4: ALLOWS DIRECT DIRECTION CHANGE
OUT MFTN3                      *USE TO SET OUTPUT FOR SPINDLE FORWARD

LD ( MCODE EQ 4 )              *SET MFTN4 FOR SPINDLE REV: REGISTER M66
OR MFTN4                       *LATCH ON
ANI MFTN5                      *DISABLE ON M5
ANI ( MFTN2 OR MFTN30 )        *DISABLE ON M2 OR M30
RES MFTN3                      *RESET M3: ALLOWS DIRECT DIRECTION CHANGE
OUT MFTN4                      *USE TO SET OUTPUT FOR SPINDLE REVERSE

LD ( MCODE EQ 5 )              *SET MFTN5 FOR SPINDLE STOP: REGISTER M68
OUT MFTN5

LD ( MCODE EQ 8 )              *SET MFTN8 FOR COOLANT ON: REGISTER M69
OR MFTN8                       *LATCH ON
ANI MFTN9                      *DISABLE ON M9
ANI ( MFTN2 OR MFTN30 )        *DISABLE ON M2 OR M30
OUT MFTN8

LD ( MCODE EQ 9 )              *SET MFTN9 FOR COOLANT OFF: REGISTER M70
OUT MFTN9

LD ( MCODE EQ 30 )             *SET MFTN30 FOR SUBPGM END: REGISTER M71
OUT MFTN30
```

```
* 0.5 SECOND BLINKER

LDI T2                        *USE FOR WARNING LIGHTS, ETC.
OUT T2 0.5

*SETS IPI MONITOR TO DISPLAY SELECTED REGISTER RANGES

IF 1 ( HCODE EQ 1 )           *DISPLAY REGISTERS M0-M15  (SYSTEM)
MOV 1 MREGRAN
EDF 1
IF 2 ( HCODE EQ 2 )           *DISPLAY REGISTERS M16-M31 (SYSTEM)
MOV 2H MREGRAN
EDF 2
IF 3 ( HCODE EQ 3 )           *DISPLAY REGISTERS M32-M47 (SYSTEM)
MOV 4H MREGRAN
EDF 3
IF 4 ( HCODE EQ 4 )           *DISPLAY REGISTERS M48-M63 (SYSTEM)
MOV 8H MREGRAN
EDF 4
IF 5 ( HCODE EQ 5 )           *DISPLAY REGISTERS M64-M79 (DEFAULT DISPLAY)
MOV 10H MREGRAN
EDF 5
IF 6 ( HCODE EQ 6 )           *DISPLAY REGISTERS M80-M95
MOV 20H MREGRAN
EDF 6
IF 7 ( HCODE EQ 7 )           *DISPLAY REGISTERS M96-M111
MOV 40H MREGRAN
EDF 7
IF 8 ( HCODE EQ 8 )           *DISPLAY REGISTERS M112-M127
MOV 80H MREGRAN
EDF 8
IF 9 ( HCODE EQ 9 )           *DISPLAY REGISTERS M128-M143
MOV 100H MREGRAN
EDF 9
IF 10 ( HCODE EQ 10 )         *DISPLAY REGISTERS M144-M159
MOV 200H MREGRAN
EDF 10
IF 11 ( HCODE EQ 11 )         *DISPLAY REGISTERS M160-M175
MOV 400H MREGRAN
EDF 11
IF 12 ( HCODE EQ 12 )         *DISPLAY REGISTERS M176-M191
```

```
MOV 800H MREGRAN
EDF 12
IF 13 ( HCODE EQ 13 )     *DISPLAY REGISTERS M192-M207
MOV 1000H MREGRAN
EDF 13
IF 14 ( HCODE EQ 14 )     *DISPLAY REGISTERS M208-M223
MOV 2000H MREGRAN
EDF 14
IF 15 ( HCODE EQ 15 )     *DISPLAY REGISTERS M224-M239 (IPI & CNC SHARE)
MOV 4000H MREGRAN
EDF 15
IF 16 ( HCODE EQ 16 )     *DISPLAY REGISTERS M240-M255 (NON-VOLATILE)
MOV 8000H MREGRAN
EDF 16

END
```

Program 2 – Binary Encoder Example

The following program will read a decimal number from a register, DECIMAL, and set a four-digit binary output accordingly. If the number is greater than 15 a flag, TOOBIG, will be set and no output will occur.

```
*BINARY ENCODER EXAMPLE

#DEFINE DECIMAL               M100
#DEFINE TEMP1                 M101
#DEFINE TEMP2                 M102
#DEFINE TOOBIG                M103
#DEFINE ENABLE                M104

#DEFINE 8BIT                  Y0:3
#DEFINE 4BIT                  Y0:2
#DEFINE 2BIT                  Y0:1
#DEFINE 1BIT                  Y0:0

START

IF 0 ( DECIMAL GT 15 )        *SET TOOBIG FLAG IF GREATER THAN 15
SET TOOBIG
RES ENABLE
ELS 0
SET ENABLE                    *ENABLE PROCESS IF OK
RES TOOBIG
EDF 0

IF 1 ENABLE                   *DECIMAL NOT GREATER THAN 15

IF 2 ( DECIMAL NE TEMP1 )     *DECIMAL HAS NOT CHANGED, NO CHANGE
                              *REQUIRED

MOV DECIMAL TEMP1             *STORE TEMP VALUE TO DETERMINE IF
                              *OUTPUT CHANGE REQUIRED

IF 3 ( TEMP1 EQ 0 )           *IF TEMP1 IS 0, RESET ALL OUTPUTS
RES 8BIT
RES 4BIT
RES 2BIT
RES 1BIT
```

```
ELS 3                        *OTHERWISE CONVERT AND OUTPUT BITS
MOV TEMP1 TEMP2              *TEMP2 WORKING REGISTER TO OUTPUT BITS

IF 4 ( TEMP2 GT 0 )          *TEMP2 WILL BE 0 WHEN FULLY DECODED

IF 5 ( TEMP2 GE 8 )          *CAN YOU SUBTRACT 8 FROM DECIMAL?
SET 8BIT                     *IF SO SET 8BIT
MOV ( TEMP2 - 8 ) TEMP2      *THEN SUBTRACT 8 FROM DECIMAL
ELS 5
RES 8BIT                     *IF NOT RESET 8BIT
EDF 5

IF 6 ( TEMP2 GE 4 )          *CAN YOU SUBTRACT 4 FROM DECIMAL?
SET 4BIT                     *IF SO SET 4BIT
MOV ( TEMP2 - 4 ) TEMP2      *THEN SUBTRACT 4 FROM DECIMAL
ELS 6
RES 4BIT                     *IF NOT RESET 4BIT
EDF 6

IF 7 ( TEMP2 GE 2 )          *CAN YOU SUBTRACT 2 FROM DECIMAL?
SET 2BIT                     *IF SO SET 2BIT
MOV ( TEMP2 -2 ) TEMP2       *THEN SUBTRACT 2 FROM DECIMAL
ELS 7
RES 2BIT                     *IF NOT RESET 2BIT
EDF 7

IF 8 ( TEMP2 GE 1 )          *CAN YOU SUBTRACT 1 FROM DECIMAL?
SET 1BIT                     *IF SO SET 1BIT
MOV ( TEMP2 - 1 ) TEMP2      *THEN SUBTRACT 1 FROM DECIMAL
ELS 8
RES 1BIT                     *IF NOT RESET 1BIT
EDF 8

EDF 4                        *END TEMP2 NOT ZERO LOOP
EDF 3                        *END SET OUTPUT BITS LOOP
EDF 2                        *END DECIMAL HAS CHANGED LOOP
EDF 1                        *END ENABLE LOOP

END
```

Program 3 – Binary Decoder Example

The following program reads a binary encoder for tool position and places the tool position's decimal value into register M81 (TOOLACT).

```
*BINARY DECODER EXAMPLE

#DEFINE BITREG8              M84
#DEFINE BITREG4              M85
#DEFINE BITREG2              M86
#DEFINE BITREG1              M87
#DEFINE BITREG84             M88
#DEFINE BITREG21             M89

#DEFINE BIT1                 XIN1
#DEFINE BIT2                 XIN2
#DEFINE BIT4                 XIN3
#DEFINE BIT8                 XIN4

START
*LOADS BITS TO REGISTERS FOR COMPARISON

IF 103 BIT1                  *CONVERTS BIT 1 TO REGISTER
MOV 1 BITREG1
ELS 103
MOV 0 BITREG1
EDF 103

IF 104 BIT2                  *CONVERTS BIT 2 TO REGISTER
MOV 2 BITREG2
ELS 104
MOV 0 BITREG2
EDF 104

IF 105 BIT4                  *CONVERTS BIT 4 TO REGISTER
MOV 4 BITREG4
ELS 105
MOV 0 BITREG4
EDF 105

IF 106 BIT8                  *CONVERTS BIT 8 TO REGISTER
MOV 8 BITREG8
ELS 106
```

```
MOV 0 BITREG8
EDF 106

MOV ( BITREG1 + BITREG2 ) BITREG21
MOV ( BITREG4 + BITREG8 ) BITREG84
MOV ( BITREG84 + BITREG21 ) TOOLACT

END
```

Program 4 – Single-Shot Pulse/Simple Counters Example

The following program creates a single-shot output, true during only one IPI cycle. This can be used to de-bounce switch inputs, and allows the creation of counters when used with mathematical statements or the INC/DEC instructions.

```
*SINGLE-SHOT PULSE/SIMPLE COUNTERS EXAMPLE

#DEFINE EVENT               X0:0
#DEFINE LOCK                M100
#DEFINE EVENTOUT            M101
#DEFINE COUNTER             M102

LD EVENT
ANI LOCK
OUT EVENTOUT
MOV EVENT LOCK
```

If the resulting output, EVENTOUT, is then used as an input:

```
IF 100 EVENTOUT
INC COUNTER
EDF 100

END
```

A simple increment counter is created. Multifunction registers can count from 0 to 65535. Underflow (negative counting) is not permitted.

Program 5 – IPI Example

This program section deals with rotation of magazine, TLSTEP 2 sets magazine rotation controlled by SOTI. TLSTEP 4 checks that SOTICNT (M31) equals TOOLDIFF, which means magazine rotated desired number of times, and that OTIFLAG (M30) indicates that SOTI command ended properly. More detailed error checking can be added.

```
*IPI EXAMPLE

#DEFINE TOODIF          M155        * DIFF. IN TOOLREQ & ACTUAL TOOL
#DEFINE TREV            M157        * REVERSE DIRECTION
#DEFINE TLSTEP          M159        * TOOL CHANGE CONTROL STEP
#DEFINE MAG_ROT_B       M162        * MAGAZINE IN ROTATION BIT

#DEFINE MAG_CW_RL       Y0:5        * CAROUSEL CW RELAY
#DEFINE MAG_CCW_RL      Y0:6        * CAROUSEL CCW RELAY

#DEFINE TL_CNT_SW       X0:23       * MAGAZINE COUNTING SWITCH

*** TLSTEP 2: ROTATE magazine CW/CCW to Target Tool ***

IF 73 ( TLSTEP EQ 2 )
   IFI 730 TREV                  * Check Magazine Rotation direction
        LD TL_CNT_SW             * Forward Magazine Rotation (SOTI)
        SOTI MAG_CW_RL TOOLDIF
   EDF 730
   IF 731 TREV                   * Check Magazine Rotation direction
       LD TL_CNT_SW              * Reverse Magazine Rotation (SOTI)
   EDF 730
   MOV 4 TLSTEP                  * Next Tool Change step
EDF 73

*** TLSTEP 4 FINAL CHECK FOR MAG ROTATION ***

LD ( OTIFLAG EQ 1 )              * Check if SOTI ended successfully
AND ( TLSTEP EQ 4 )              * Check for Final Stage
AND ( TOOLDIF EQ SOTICNT )       * Check if Rotated Correct Number
IF 80
  LDI MAG_CW_RL                  * Check that Rotation Stop
  ANI MAG_CCW_RL
  AND TL_CNT_SW                         * Check Proximity switch, STOP CORRECTLY
  IF 81
     MOV 0 TLSTEP                       * End Tool Change Sequence
     MOV TOOLREQ M226                   * TOOL POT NO REQ TO 1102
     MOV TOOLREQ TL_POT_NO
     RES MAG_ROT_B                      * MAG ROT FINISH OK

  EDF 81
EDF 80
```

ANILAM

U.S.A.
ANILAM
One Precision Way
Jamestown, NY 14701
☎ (716) 661-1899
FAX (716) 661-1884
✉ anilaminc@anilam.com

ANILAM, CA
16312 Garfield Ave., Unit B
Paramount, CA 90723
☎ (562) 408-3334
FAX (562) 634-5459
✉ anilamla@anilam.com

Dial “011” before each number when calling
from the U.S.A.

France
ANILAM S.A.R.L.
2 Ave de la Cristallerie
B.P. 68-92316
Serves Cedex, France
☎ +33-1-46290061
FAX +33-1-45072402
✉ courrier@acu-rite.fr

Germany
ANILAM GmbH
Fraunhoferstrasse 1
D-83301 Traunreut
Germany
☎ +49 8669 856110
FAX +49 8669 850930
✉ info@anilam.de

Italy
ANILAM Elettronica s.r.l.
10043 Orbassano
Strada Borgaretto 38
Torino, Italy
☎ +39 011 900 2606
FAX +39 011 900 2466
✉ info@anilam.it

Taiwan
ANILAM, TW
No. 246 Chau-Fu Road
Taichung City 407
Taiwan, ROC
☎ +886-4 225 87222
FAX +886-4 225 87260
✉ anilamtw@anilam.com

United Kingdom
ACI (UK) Limited
16 Plover Close, Interchange Park
Newport Pagnell
Buckinghamshire, MK16 9PS
England
☎ +44 (0) 1908 514 500
FAX +44 (0) 1908 610 111
✉ sales@aciuk.co.uk

China
Acu-Rite Companies Inc.(Shanghai Representative Office)
Room 1986, Tower B
City Center of Shanghai
No. 100 Zunyi Lu Road
Chang Ning District
200051 Shanghai P.R.C.
☎ +86 21 62370398
FAX +86 21 62372320
✉ china@anilam.com

P/N 70000416D
31-October-04

www.anilam.com