ANILAM

6000M 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

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. 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.

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.

Physical Inputs

The following physical inputs are described in this section:

- X42 Inputs
- I/O Module Inputs

X42 Inputs

Refer to Table 2-1, X42 IPI Inputs for the X42 input assignments.

I/O Module Inputs

Format **Xn:b** where:

- X indicates Input
- n indicates Module # (range: 0 to 2)
- b indicates Bit # (range of b: 0 to 30 on base unit)
  (range of b: 0 to 63 on expansion unit)

The IPI stores the condition or state of each input (on/off, true/false) in a state memory register using the same designation. See **Table 2-2**.

Table 2-2, Input Locations

Input	Location
Home Inputs	Always located on CNC chassis module, if used. The CNC reads these inputs from travel-limit switches.
General Purpose Inputs	Located on Base module and Expansion module(s). (Optional)

Inputs must be hardwired to the X42 module or the appropriate Expansion module.

Refer to Table 2-3, I/O Module Input Assignments for the input terminals and assignments.

Physical Outputs

The following physical outputs are described in this section:

- X41 Outputs
- I/O Module Outputs

X41 Outputs

Refer to Table 2-4, X41 IPI Outputs for the X41 output assignments.

Outputs

Format **Yn:b** where:

- Y indicates Output
- n indicates Module #; range of n = 0 to 2
- b indicates Bit # (range of b: 0 to 22 on base unit)
  (range of b: 0 to 30 on expansion unit)

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.

Refer to Table 2-5, I/O Module Output Assignments for the output terminals and assignments.

Manual Panel Inputs

The IPI monitors the Manual Panel switches. Refer to Table 2-6, Manual Panel Inputs and Assigned Labels for a description of the Manual Panel I/O.

Table 2-1, X42 IPI Inputs

37-pin	IPI	Assignment	Color
1	X0:0	**XHOME	BLK
2	X0:1	**YHOME	RED
3	X0:2	**ZHOME	WHT
4	X0:3	**CNCACK	GRN
5	X0:4	**UHOME	ORN
6	X0:5	**WHOME	BLU
7	X0:6		BRN
8	X0:7		YEL
9	X0:8		VIOL
10	X0:9		GRAY
11	X0:10		PINK
12	X0:11		TAN
13	X0:12		RED/GRN
14	X0:13		RED/YEL
15	X0:14		RED/BLK
16	X0:15		WHT/BLK
17	X0:16		WHT/RED
18	X0:17		WHT/GRN
19	X0:18		WHT/YEL
20	X0:19		WHT/BLU
21	X0:20		WHT/BRN
22	X0:21		WHT/ORN
23	X0:22		WHT/GRAY
24	X0:23		WHT/VIOL
25	X0:24		WHT/BLK/RED
26	X0:25		WHT/BLK/GRN
27	X0:26		WHT/BLK/YEL
28	X0:27		WHT/BLK/BLU
29	X0:28		WHT/BLK/BRN
30	X0:29		WHT/BLK/ORN
31	X0:30	**Software ESTOP	WHT/BLK/GRAY
32	X0:31	**Software DRIVE ENABLE	WHT/BLK/VIOL
33		**DRIVE ENABLE	WHT/BLK/BLK
34		Do not assign	WHT/RED/BLK
35		Do not assign	WHT/RED/RED
36		Do not assign	WHT/RED/GRN
37		Do not assign	WHT/RED/BLU
Shell		EXTERNAL SHIELD	

** Hard Coded Function

Table 2-3, I/O Module Input Assignments

X3

Terminal	Assignment	
	1	2
1	X1:0	X2:0
2	X1:1	X2:1
3	X1:2	X2:2
4	X1:3	X2:3
5	X1:4	X2:3
6	X1:5	X2:3
7	X1:6	X2:6
8	X1:7	X2:7
9	X1:8	X2:8
10	X1:9	X2:9
11	X1:10	X2:10
12	X1:11	X2:11
13	X1:12	X2:12
14	X1:13	X2:13
15	X1:14	X2:14
16	X1:15	X2:15

X4

Terminal	Assignment	
	1	2
1	X1:16	X2:16
2	X1:17	X2:17
3	X1:18	X2:18
4	X1:19	X2:19
5	X1:20	X2:20
6	X1:21	X2:21
7	X1:22	X2:22
8	X1:23	X2:23
9	X1:24	X2:24
10	X1:25	X2:25
11	X1:26	X2:26
12	X1:27	X2:27
13	X1:28	X2:28
14	X1:29	X2:29
15	X1:30	X2:30
16	X1:31	X2:31

X5

Terminal	Assignment	
	1	2
1	X1:32	X2:32
2	X1:33	X2:33
3	X1:34	X2:34
4	X1:35	X2:35
5	X1:36	X2:36
6	X1:37	X2:37
7	X1:38	X2:38
8	X1:39	X2:39
9	X1:40	X2:40
10	X1:41	X2:41
11	X1:42	X2:42
12	X1:43	X2:43
13	X1:44	X2:44
14	X1:45	X2:45
15	X1:46	X2:46
16	X1:47	X2:47

X6

Terminal	Assignment	
	1	2
1	X1:48	X2:48
2	X1:49	X2:49
3	X1:50	X2:50
4	X1:51	X2:51
5	X1:52	X2:52
6	X1:53	X2:53
7	X1:54	X2:54
8	X1:55	X2:55
9	X1:56	X2:56
10	X1:57	X2:57
11	X1:58	X1:58
12	X1:59	X2:59
13	X1:60	X2:60
14	X1:61	X2:61
15	X1:62	X2:62
16	X1:63	X2:63

Table 2-4, X41 IPI Outputs

37-pin	IPI	Assignment	Color
1			BLK
2			RED
3			WHT
4			GRN
5			ORN
6			BLU
7			BRN
8			YEL
9	Y0:0	SSK1-CNCON	VIOL
10	Y0:1	SSK2-MODEOP	GRAY
11	Y0:2		PINK
12	Y0:3		TAN
13	Y0:4		RED/GRN
14	Y0:5		RED/YEL
15	Y0:6		RED/BLK
16	Y0:7		WHT/BLK
17	Y0:8		WHT/RED
18	Y0:9		WHT/GRN
19	Y0:10		WHT/YEL
20	Y0:11		WHT/BLU
21	Y0:12		WHT/BRN
22	Y0:13		WHT/ORN
23	Y0:14		WHT/GRAY
24	Y0:15		WHT/VIOL
25	Y0:16		WHT/BLK/RED
26	Y0:17**		WHT/BLK/GRN
27	Y0:18**		WHT/BLK/YEL
28	Y0:19**		WHT/BLK/BLU
29	Y0:20**		WHT/BLK/BRN
30	Y0:21**		WHT/BLK/ORN
31	Y0:22**	K1-SERVO ENABLE	WHT/BLK/GRAY
32		Do not assign	WHT/BLK/VIOL
33		Do not assign	WHT/BLK/BLK
34		CRDY SIGNAL	WHT/RED/BLK
35		Do not assign	WHT/RED/RED
36		Do not assign	WHT/RED/GRN
37		Do not assign	WHT/RED/BLU
Shell		EXTERNAL SHIELD	

** These outputs are not controlled by +24VDC from the **E-STOP** switch.

Table 2-5, I/O Module Output Assignments

X7

Terminal	Assignment	
	1	2
1	Y1:0	Y2:0
2	Y1:1	Y2:1
3	Y1:2	Y2:2
4	Y1:3	Y2:3
5	Y1:4	Y2:4
6	Y1:5	Y2:5
7	Y1:6	Y2:6
8	Y1:7	Y2:7
9	Y1:8	Y2:8
10	Y1:9	Y2:9
11	Y1:10	Y2:10
12	Y1:11	Y2:11
13	Y1:12	Y2:12
14	Y1:13	Y2:13
15	Y1:14	Y2:14
16	Y1:15	Y2:15

X8

Terminal	Assignment	
	1	2
1	Y1:16	Y2:16
2	Y1:17	Y2:17
3	Y1:18	Y2:18
4	Y1:19	Y2:19
5	Y1:20	Y2:20
6	Y1:21	Y2:21
7	Y1:22	Y2:22
8	Y1:23	Y2:23
9	Y1:24	Y2:24
10	Y1:25	Y2:25
11	Y1:26	Y2:26
12	Y1:27	Y2:27
13	Y1:28	Y2:28
14	Y1:29	Y2:29
15	Y1:30	Y2:30
16	Control-is-ready	

Table 2-6, Manual Panel Inputs and Assigned Lables

IPI Register	Assigned Label	Assignment
X0:100	M3KEY	SPINDLE FORWARD
X0:101	M5KEY	SPINDLE OFF
X0:102	M4KEY	SPINDLE REVERSE
X0:103	RESKEY	SERVO RESET
X0:104	MINUSKEY	AXIS -
X0:105	PLUSKEY	AXIS +
X0:106	HOLDKEY	HOLD
X0:107	STARTKEY	START
X0:108	AXIS1KEY	AXIS SELECT LSB=1
X0:109	AXIS2KEY	AXIS SELECT=2
X0:110	AXIS4KEY	AXIS SELECT MSB=4
X0:111	FEED1KEY	FEED RATE LSB=1
X0:112	FEED2KEY	FEED RATE=2
X0:113	FEED4KEY	FEED RATE=4
X0:114	FEED8KEY	FEED RATE MSB=8
X0:115	JOG1KEY	JOG SELECT LSB=1
X0:116	JOG2KEY	JOG SELECT=2
X0:117	JOG4KEY	JOG SELECT MSB=4
X0:118	SPDL1KEY	SPINDLE% LSB=1
X0:119	SPDL2KEY	SPINDLE%=2
X0:120	SPDL4KEY	SPINDLE%=4
X0:121	SPDL8KEY	SPINDLE% MSB=8
X0:122	MPIN1	Input 1
X0:123	MPIN2	Input 2
X0:124	MPIN3	Input 3

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
- R registers: CNC to IPI Communication
- W registers: IPI to CNC Communication
- Timers
- Sequence Registers

Refer to Table 2-7, Register Capabilities.

- 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 2-7, 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
R registers	X	X
W registers	X	X
Timer Registers - T Identifiers	X	X

Multifunction Registers

Format **M*n***; where ***n*** is a number 00 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 M00–M255. Multifunction registers M00–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. 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.

General-Purpose, Multifunction Registers

M00–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.

See the following examples:

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 registerM225 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 to 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 R16.

R Registers

R00–R255 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 2-8**.

Table 2-8, R Registers, Assigned Read Only Multifunction

R Register	Assigned Label	Purpose
R00	ESTOP	True if E-Stop is out.
R01	SPINDLE	True when not probing and spindle may run.
R02	POSN	True when CNC is in position.
R03	FEED	Feed mode flag.
R04	PWRFAIL	True if +24V is on.
R05	MAN	True when CNC in MANUAL mode.
R06	TCFINACK	Tool changer finished received.
R07	HOME	True when Z or XYZ at home.
R08	SPLOOP	True when spindle in closed-loop mode.
R09	RUN	True when CNC in RUN mode.
R10	SVOFF	True if servo is off.
R11	M19FLAG	Status of M19 operation: =0 Once spindle function M3, M4, or M5 is executed =1 During orientation =2 Once orientated
R12	TMACEND	Tool Macro end flag.
R13	CARRY	Carry Flag/Register.
R14	XMIT	True when IPI accepts CNC data.

(Continued…)

Table 2-8, R Registers, Assigned Read Only Multifunction (Continued)

R Register	Assigned Label	Purpose
R15	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.
R16	CNCERR	Used by the CNC to pass error conditions to the IPI.
R17	MFLAG	True when new M code is received.
R18	MCODE	M code number.
R19	SFLAG	True when new spindle code is received.
R20	SCODE	Spindle number.
R21	TFLAG	True when new tool number is received.
R22	TCODE	Tool number.
R23	HFLAG	True when new H-code number is received (tool pre-set code).
R24	HCODE	H-code number.
R25	CMDRPM	Commanded Spindle RPM. This is the S word multiplied by any spindle override.
R26	ZEROSPD	Spindle is below a designated speed.
R27	ATSPD	Spindle is within a certain value of commanded speed.
R28	SPDLLOAD	Percentage of spindle current load.
R29	TRUE	Always ON.
R30	FALSE	Always OFF.
R31	TOOLNUM	Active tool number.
R32	TLOBIN0	Used in random tool changer applications to store the bin of the tool in the spindle.
R33	OTIFLAG	A status code for OTI operation: 0 = OTI has not completed or is not active 1 = OTI completed
R34	DRUNFLAG	Reflects the status of dry run in the CNC: 105 = dry run all 106 = dry run XY, no Z motion 107 = dry run off
R35	XMACHPOS	The X machine position in microns.
R36	YMACHPOS	The Y machine position in microns.
R37	ZYMACHPOS	The Z machine position in microns.
R38	UMACHPOS	The U machine position in microns.
R39	Reserved	Reserved for future implementation.
R40	Reserved	Reserved for future implementation.
R41	BRKENA	Used in the current controller auto-tuning test. BRKENA register is set at the beginning of the test to engage, through IPI, any axis motor with a brake (typically, the Z-axis). The BRKENA register is cleared at the end of the test or if the test is canceled.

(Continued…)

Table 2-8, R Registers, Assigned Read Only Multifunction (Continued)

R Register	Assigned Label	Purpose
R44	PRBFLAG	Read only register R44 is the probing flag and is active during G31 primitive moves and probing cycles.

R15-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.

R16-CNCERR

The CNC uses R16-CNCERR to pass error conditions to IPI. The CNC can pass only one error (Set 1–4) at a time to IPI. 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

R25-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 R25. Should be used to determine if spindle range errors, which IPI needs to act upon, are present.

R44- PRBFLAG

Read only register R44 is the probing flag and is active during G31 primitive moves and probing cycles. Bit 1 of PRBFLAG (R44) will be set during the G31 primitive. At the beginning of the probing cycles, Bit 2 of R44 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 R44 probing flag allows these limitations to be properly enforced.

W Registers

The IPI generates and the CNC internally monitors W00–W255. You can consider these registers inputs from the IPI to the CNC. This allows the IPI program to output to the CNC, generate on-screen messages, and invoke features programmed in the CNC software. The registers can be used as the result of a switch or other hardware sensor, or as the result of IPI logic. These registers are read and write registers. Refer to **Table 2-10**.

Table 2-10, W Registers, Assigned Read/Write Multifunction

Register	Assigned Label	Feature	Definition
W00	FINISH	Finish Signal	Set True to signal M, S, T, or H finish. When M or S function is cleared, reset to false.
W01	SVOFLT	Servo Fault	Set True to signal a servo fault.
W02	FHOLD	Feed Hold	Set True to inhibit feed moves. Rapid moves will execute normally.
W03	TCHGFIN	Tool Change Finish	Tool changer finished bit.
W04	XSTART	External Start	Operation is identical to input function and front panel key.
W05	XSTOP	External Stop	Set True to hold CNC motion. Set to False to resume motion.
W06	XHOLD	External Hold	Set True to stop CNC motion. Press the **START** button to resume motion.
W07	SPDLZERO	Spindle Zero	When SPDLZERO W07 is True (non-zero) the IPI disables the spindle. When W07 is False (zero), the IPI does not affect the spindle.
W08	SPDLGRCH	Spindle Gear Change	When W08 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 W08 is outside of the range 40–44, the CNC ignores this register.
W09	AUTOINH	Auto Mode Inhibit	Set to 1 to inhibit Auto or S.Step mode. Set to 0 to enable Auto or S.Step mode.
W10	HWSTOP	Handwheel Stop	Set register True to inhibit handwheel operation in all modes. Set to zero to allow moves.
W11	SPDL100	Spindle Speed 100%	When any non-zero number is written to this register, spindle speed will be forced to 100% of the programmed value, regardless of the setting of the spindle percentage switch on the Manual Panel.
W12	FEED100	Feed Speed 100%	Set to 1 to force feedrate override to 100%. Set to zero to enable feedrate switch value.

(Continued…)

Table 2-10, W Registers, Assigned Read/Write multifunction (Continued)

Register	Assigned Label	Feature	Definition
W13	SPDLDIR	Spindle Direction	Used in conjunction with W19 (when W19>0) to allow the IPI program to control the spindle direction.
W14	W_RES1	Reserved	Reserved
W15	W_RES2	Reserved	Reserved
W16	MSG	Message	Set any non-zero number to display message.
W17	MREGRAN	M Register Range	Used to cycle the M registers (multifunction registers) displayed on the IPI monitor. Allows you to view M0–M256 by selecting a range of registers to be displayed.
W18	KEYMASK	Key Mask	Used by the IPI program to mask out certain keys from the operator.
W19	SPDLRPM	Spindle RPM	Used to set the spindle RPM 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 W08 SPDLGRCH.
W20	LNFDLIM	Linear Axis Feed Limit	If this register is nonzero, use the value as speed for linear axes.
W21	ROFDLIM	Rotary Axis Feed Limit	If this register is nonzero, use the value as speed for rotary axes.
W22	RREGRAN	R Register Range	Used to cycle the R registers (CNC to IPI Communication registers) displayed on the IPI monitor. Allows you to view R00–R256 by selecting a range of registers to be displayed.
W23	WREGRAN	W Register Range	Used to cycle the W registers (assigned read/write multifunction registers) displayed on the IPI monitor. Allows you to view W00–W256 by selecting a range of registers to be displayed.
W24	M19END	M19 End	Allows IPI to terminate an M19 (spindle orientation) sequence.
W25	SPDLOPEN	Spindle Open Loop	Forces spindle into open-loop mode.
W26 W27 W28 W29 W30 W31 W32 W33 W34 W35	BLKSKIP0 BLKSKIP1 BLKSKIP2 BLKSKIP3 BLKSKIP4 BLKSKIP5 BLKSKIP6 BLKSKIP7 BLKSKIP8 BLKSKIP9	Optional Block Skip	Activating this input enables a block skip command. Block skip switches must be programmed into the part program.

(Continued…)

Table 2-10, W Registers, Assigned Read/Write multifunction (Continued)

Register	Assigned Label	Feature	Definition
W36	TOOLGRD	Tool Guard	Holds the CNC program and stops the machine spindle. This input must be removed before the program can continue. To restart the spindle, press **START** once. To continue the program press **START** a second time. If the spindle was not running when the function was activated, press **START** once to continue the program. The tool guard function permits compliance with regulations that require an intact tool guard in place for the machine to run.
W37	EXTMANEN	External Manual Enabled	This input enables manual handwheel operation. Position display is generated from encoder inputs. In this mode, the CNC no longer commands axis position. Thus, an operator can move the machine with the hand wheels and use the CNC as a digital read out. Spindle can still be operated via the Manual Panel keys.
W38	RDKYBD	Start Reading Keyboard	Allows the CNC to accept inputs from the CNC keypad (or keyboard).
W39	NOKYBD	Stop Reading Keyboard	Commands the CNC **NOT** to respond to inputs from the CNC keypad (or keyboard). Allows you to set a keyboard lockout system. When you lock out the keypad (or keyboard), all keys except **E-STOP** remain inoperative.
W40	INIT999	Initialize #999 to 0	When the input attached to **999** is activated, the value of the variable becomes **0**.
W41	INC999	Increment #999	When activated, the value of register **999** increments by **1**.
W42	SPDLCW	Spindle CW (M3)	When activated, CNC initiates an M3 output. Duplicates manual panel SPINDLE CW function.
W43	SPDLCCW	Spindle CCW (M4)	When activated, CNC initiates an M4 output. Duplicates manual panel SPINDLE CCW function.
W44	SPDLOFF	Spindle Off (M5)	When activated, CNC initiates an M5 output. Duplicates manual panel SPINDLE OFF function.
W45	JOGMINUS	Remote Jog -	Allows you to make negative jog moves from a remote manual panel.
W46	JOGPLUS	Remote Jog +	Allows you to make positive jog moves from a remote manual panel.
W47–W54	Reserved	Reserved	Reserved for future use
W55	XSSTOP	X-axis Stop/Start	Stops all X-axis machine motion when set to 1. X-axis motion continues went set to 0.

(Continued…)

Table 2-10, W Registers, Assigned Read/Write multifunction (Continued)

Register	Assigned Label	Feature	Definition
W56	YSSTOP	Y-axis Stop/Start	Stops all Y-axis machine motion when set to 1. Y-axis motion continues went set to 0.
W57	ZSSTOP	Z-axis Stop/Start	Stops all Z-axis machine motion when set to 1. Z-axis motion continues went set to 0.
W58	USSTOP	U-axis Stop/Start	Stops all U-axis machine motion when set to 1. U-axis motion continues went set to 0.
W59	Reserved	Reserved	Reserved for future use
W60	RMHWENA	Remote Handwheel Enable/Disable	Set to 1 to enable remote handwheel (RM 500). Set to 0 to disable the remote handwheel. Typically, RMHWENA is associated with pressing the permissive buttons on the remote handwheel.
W61	UAMPENA	U-axis Enable/Disable	Set to 1 to enable U-axis. Set to 0 to disable the U-axis on the fly.
W62	Reserved	Reserved	Reserved for future use

W04-XSTART

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

W07-SPDLZERO

When W07-SPDLZERO is true (non-zero), IPI will disable the spindle. When W07 is false (zero), the spindle is not affected.

For example, if the spindle is running at 1000 RPM and W07 is set to true, the corresponding voltage output to the spindle will be 0V. Once W07 is set to false (zero), the corresponding output to the spindle will be the same as it was before the W07 was set to true. If you use an S-word before setting W07 to false, the output will correspond to the newly programmed RPM.

W08-SPDLGRCH

Set W08-SPDLGRCH between 40–44 to enable the corresponding gear range. The specified range is used to calculate a proportional spindle analog output voltage only. When W08 is outside the range 40–44, the CNC ignores this register.

When W08 is not **0** (zero), the monitor screen displays **1** on the last bit of the PLC flags section. To see the actual value, using W17-MREGRAN to display the proper M register range (Range 3, 0004h).

W09-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.

W10-HWSTOP

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

W11-SPDL100

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.

W12-FEED100

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

W13-SPDLDIR

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 W19 to allow IPI the responsibilities of spindle control.

W17-MREGRAN

W17-MREGRAN allows you to cycle through the range of multifunction registers displayed on the IPI monitor. There are 256 multifunction registers available, numbered M00–M255. W17 allows you to change the range of displayed registers. You can display only 5 registers at one time. As a default, only M00–M04 are displayed on the IPI monitor.

Table 2-11, Available Multifunction Register Ranges Displayed on the IPI Monitor lists the available ranges.

Table 2-11, Available Multifunction Register Ranges Displayed on the IPI Monitor

Range No.	Mreg Range
1	M00–M04
2	M05–M09
3	M10–M14
4	M15–M19
5	M20–M24
6	M25–M29
7	M30–M34
8	M35–M39
9	M40–M44
10	M45–M49
11	M50–M54
12	M55–M59
13	M60–M64
14	M65–M69
15	M70–M74
16	M75–M79
17	M80–M84
18	M85–M89
19	M90–M94
20	M95–M99

Range No.	Mreg Range
21	M100–M104
22	M105–M109
23	M110–M114
24	M115–M119
25	M120–M124
26	M125–M129
27	M130–M134
28	M135–M139
29	M140–M144
30	M145–M149
31	M150–M154
32	M155–M159
33	M160–M164
34	M165–M169
35	M170–M174
36	M175–M179
37	M180–M184
38	M185–M189
39	M190–M194
40	M195–M199

Range No.	Mreg Range
41	M200–M204
42	M205–M209
43	M210–M214
44	M215–M219
45	M220–M224
46	M225–M229
47	M230–M234
48	M235–M239
49	M240–M244
50	M245–M249
51	M250–M254
52	M255

W18-KEYMASK

The IPI program uses W18-KEYMASK to mask out certain keys from the operator. W18 contains a bit value; each bit corresponds to a key. Refer to **Table 2-12** for keys assigned to W18 bits.

Table 2-12, KEYMASK W18 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 W18** 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 D18 in a subsequent execution scan; for example, **MOV 0 W18**.

The command enables all previously masked keys.

W19-SPDLRPM

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 W08-SPDLGRCH. Additionally, the desired RPM must be in the range of allowed speeds, as specified in the Spindle Axis Machine Constants.

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.

W20-LNFDLIM

Linear Axis feed limit. When the IPI writes a number to this register, linear axes will run at the value stored in register W20. 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 W20 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.

Because the feedrate override switch operates normally, you must divide the maximum allowed speed by 120%, and use this value for W20.

W21-ROFDLIM

Rotary Axis feed limit. When the IPI writes a number to this register, rotary axes run at the value stored in register W21. 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 W21 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.

Because the feedrate override switch operates normally, you must divide the maximum allowed speed by 120%, and use this value for W21.

W22-RREGRAN

W22-RREGRAN allows you to cycle through the range of Read Only multifunction registers displayed on the IPI monitor. There are 256 Read Only multifunction registers available, numbered R00–R255. W22 allows you to change the range of displayed registers. You can display only 5 registers at one time. As a default, only R0–R4 are displayed on the IPI monitor. R63 is the highest register displayed.

Table 2-13 lists the available Read Only ranges.

Table 2-13, Available Read Only Multifunction Register Ranges Displayed on the IPI Monitor

Range No.	Rreg Range
1	R00–R04
2	R05–R09
3	R10–R14
4	R15–R19
5	R20–R24
6	R25–R29
7	R30–R34
8	R35–R39
9	R40–R44
10	R45–R49
11	R50–R54
12	R55–R59
13	R60–R63

W23-WREGRAN

W23-WREGRAN allows you to cycle through the range of Read/Write multifunction registers displayed on the IPI monitor. There are 256 Read/Write multifunction registers available, numbered W00–W255. W23 allows you to change the range of displayed registers. You can display only 5 registers at one time. As a default, only W00–W04 are displayed on the IPI monitor. W63 is the highest register displayed.

Table 2-14 lists the available Read/Write ranges.

Table 2-14, Available Read/Write Multifunction Register Ranges Displayed on the IPI Monitor

Range No.	Rreg Range
1	W00–W04
2	W05–W09
3	W10–W14
4	W15–W19
5	W20–W24
6	W25–W29
7	W30–W34
8	W35–W39
9	W40–W44
10	W45–W49
11	W50–W54
12	W55–W59
13	W60–W63

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 2-15**.

Table 2-15, 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

Timer Registers

Format **T*n***; where ***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.

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.

IPI Monitor

The state value stored in input registers, output registers, timer registers, and multifunction registers is viewed from the IPI Monitor.

Viewing the IPI Monitor

To access the **IPI Monitor** screen:

1. From the CNC software's Manual mode, press **P**.
2. Press **ENTER** to display the **IPI Monitor** screen. Refer to **Figure 2-1**.

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

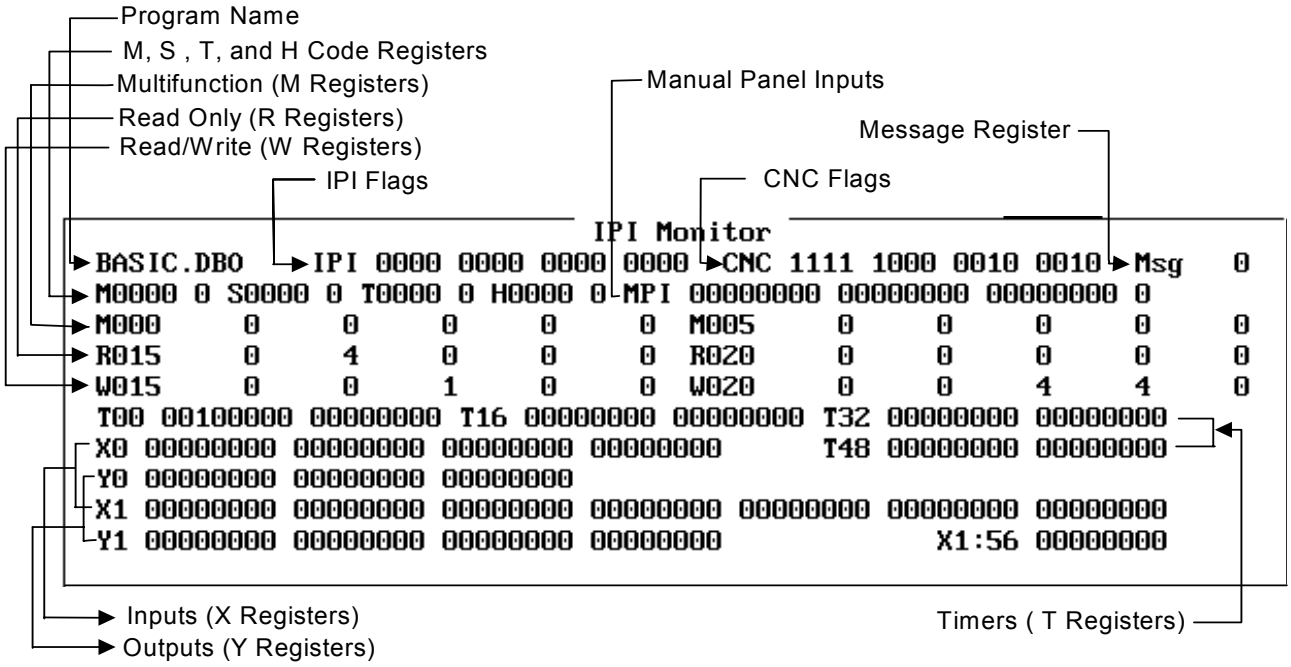


Figure 2-1, 6300M, 6400M IPI Monitor Screen

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

- X registers
- Y registers
- T registers
- Manual Panel Inputs

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

- M, S, T, and H codes from the CNC
- M, R, and W IPI registers
- Message registers from the IPI to the CNC

Refer to Figure 2-2, CNC Flags for CNC flags from the CNC to IPI. These are **Read Only** registers. Refer to Figure 2-3, IPI Flags for IPI Flags from IPI to the CNC. These are **Read/Write** registers.

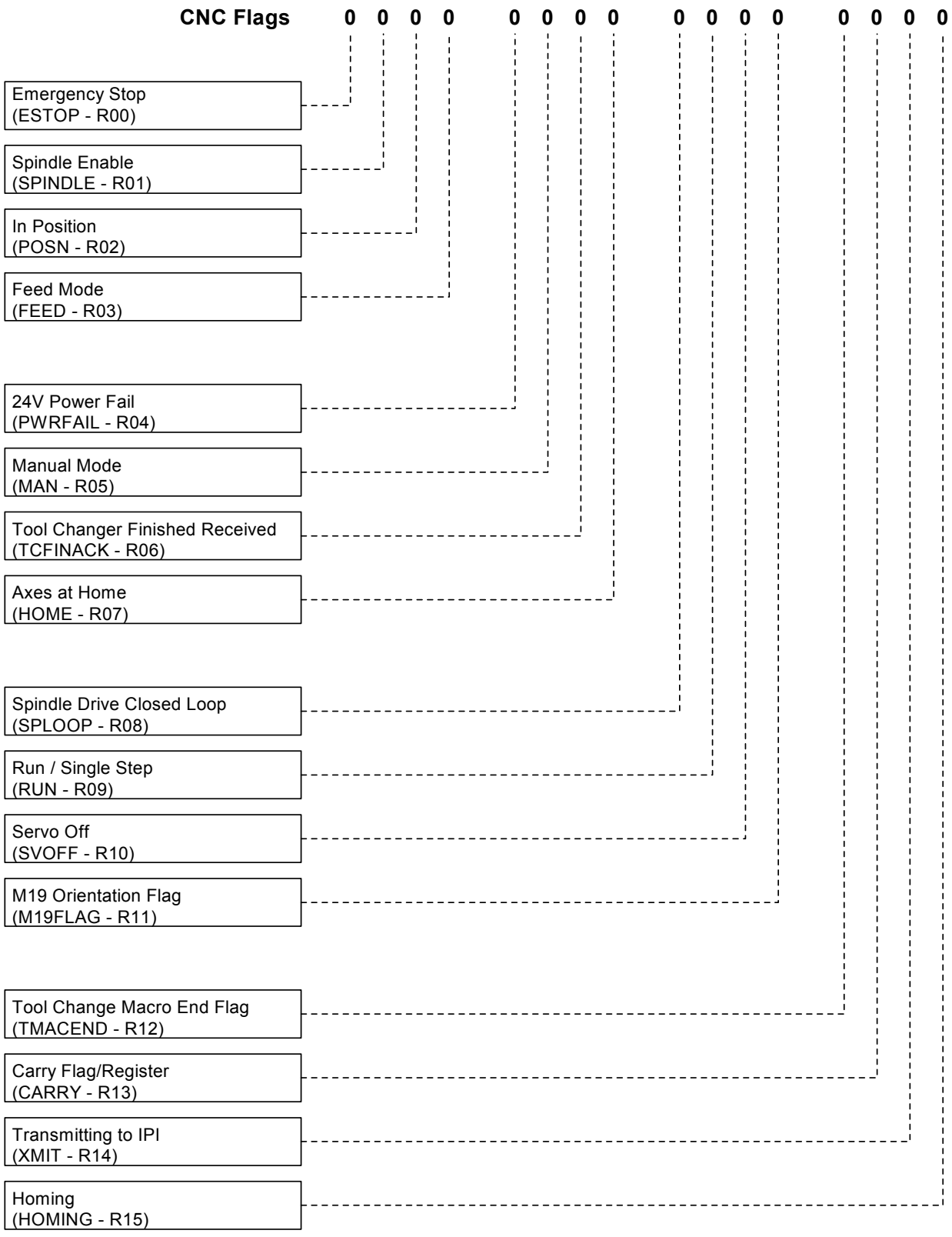


Figure 2-2, CNC Flags

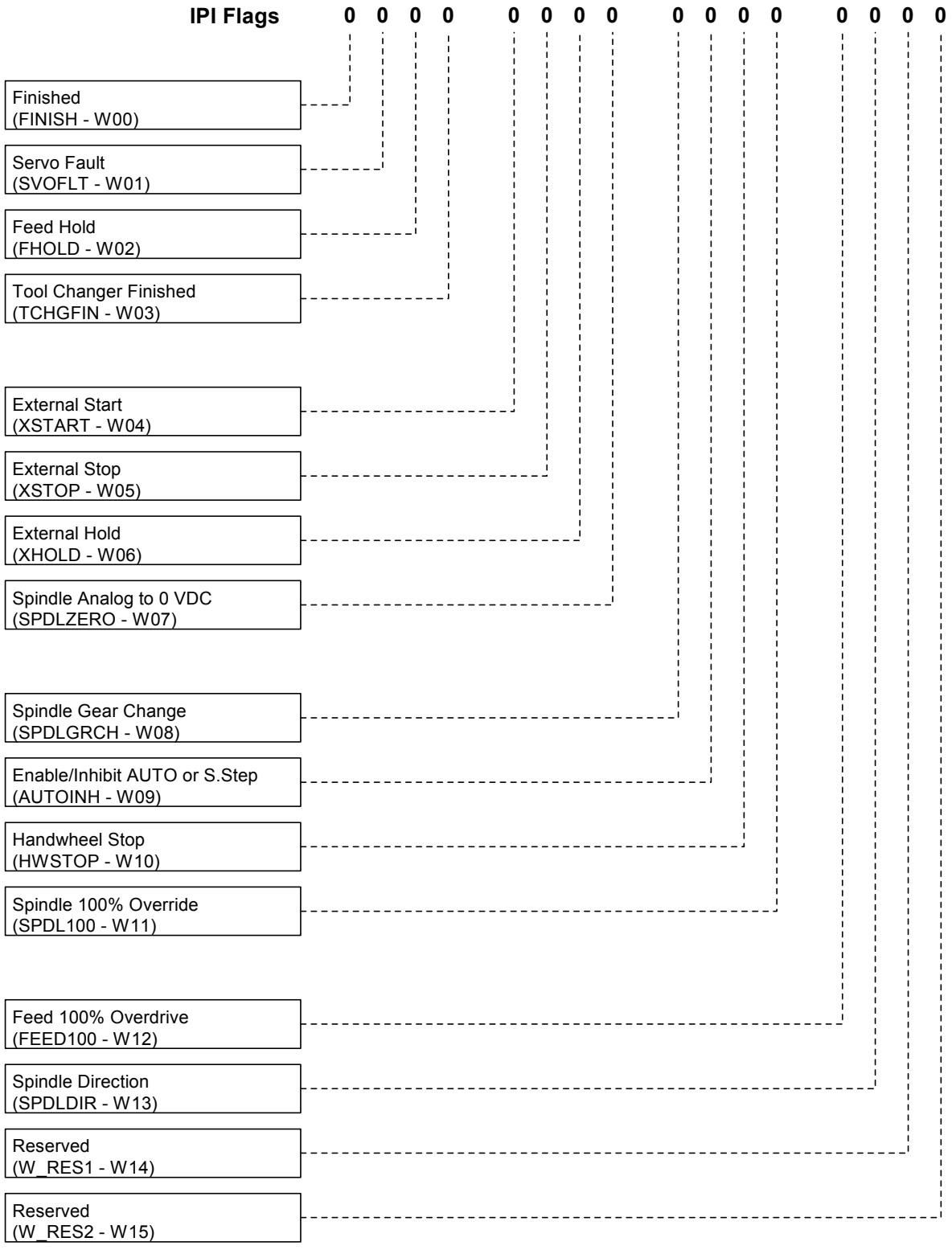


Figure 2-3, IPI Flags

Section 3 - Working with IPI

The CNC is configured with the ANILAM IPI. The system is ready for you to program IPI.

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 3-1** for the menus referenced in this procedure.

To access IPI, perform the following steps:

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 **Prog. I/O Interface** and press **ENTER** to display the Password prompt.
4. Type the password and press **ENTER** to activate **Menu B, Select Options Menu**.

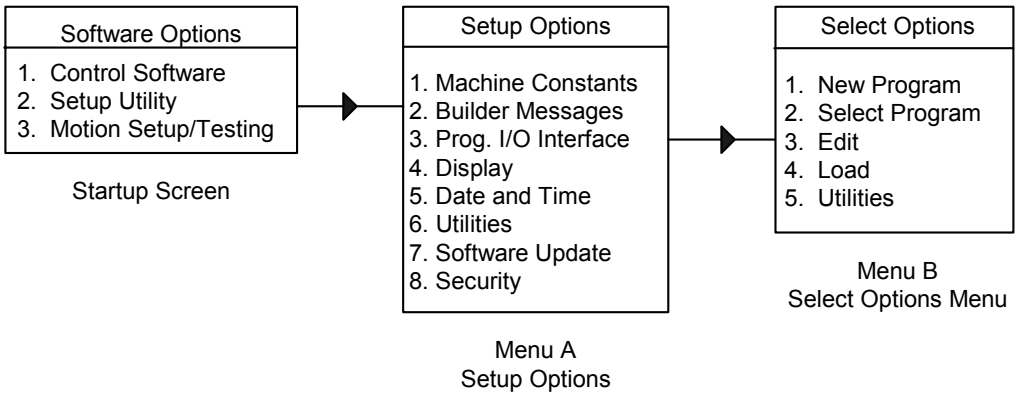


Figure 3-1, Accessing Select Options Menu

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, perform the following steps:

1. From **Menu B, Select Options Menu**, highlight **New Program**, and press **ENTER**. Refer to **Figure 3-2**.

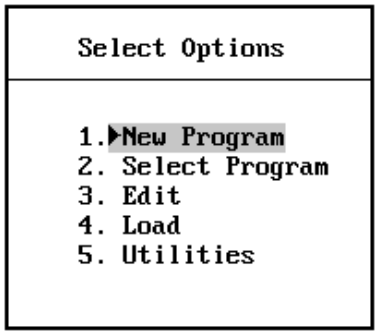


Figure 3-2, Creating a New Program

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

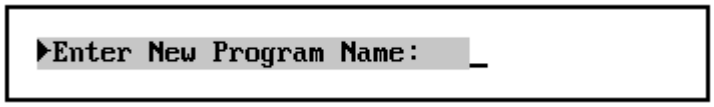


Figure 3-3, Entering a New Program Name

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, perform the following steps:

1. From **Menu B, Select Options Menu**, highlight **Select Program** and press **ENTER**. Refer to **Figure 3-4**.

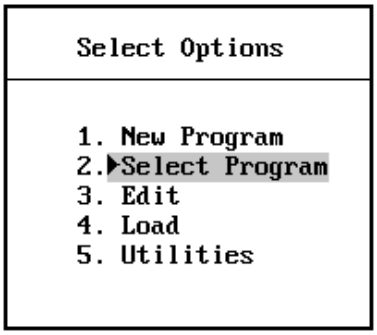


Figure 3-4, Selecting an Existing Program

Select program activates. Refer to Figure 3-5, Select program Menu.

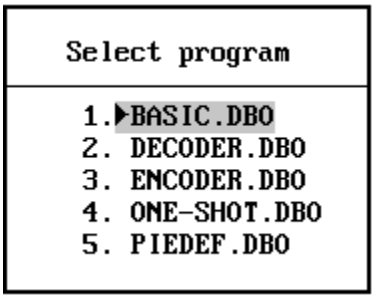


Figure 3-5, Select program Menu

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, perform the following steps:

1. Select or create a program.
2. From **Menu B, Select Options Menu**, highlight **Edit** and press **ENTER**. Refer to **Figure 3-6**.

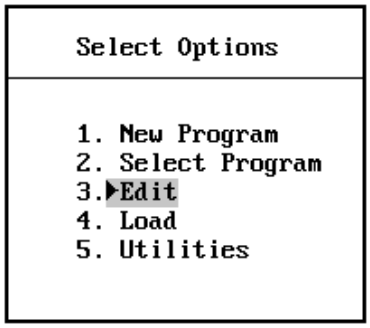


Figure 3-6, Activating the Editor

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, perform the following steps:

1. Select the desired program.
2. From **Menu B, Select Options Menu**, highlight **Load** and press **ENTER**. Refer to Figure 3-7, Compiling and Loading a Program.

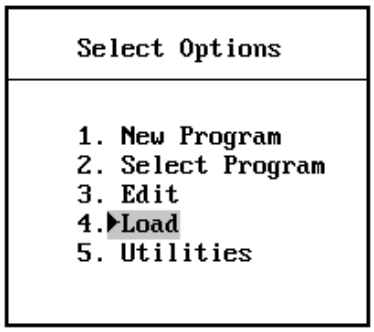


Figure 3-7, Compiling and Loading a Program

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

Soft Key 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.

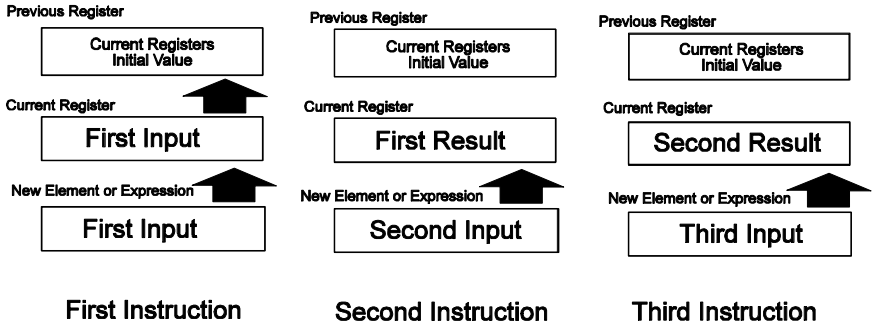


Figure 4-1, Interpreter Operation

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, CNC, IPI, and timer registers. These elements are identified by their X, Y, S, M, R, W, 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 "Section 7, 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.

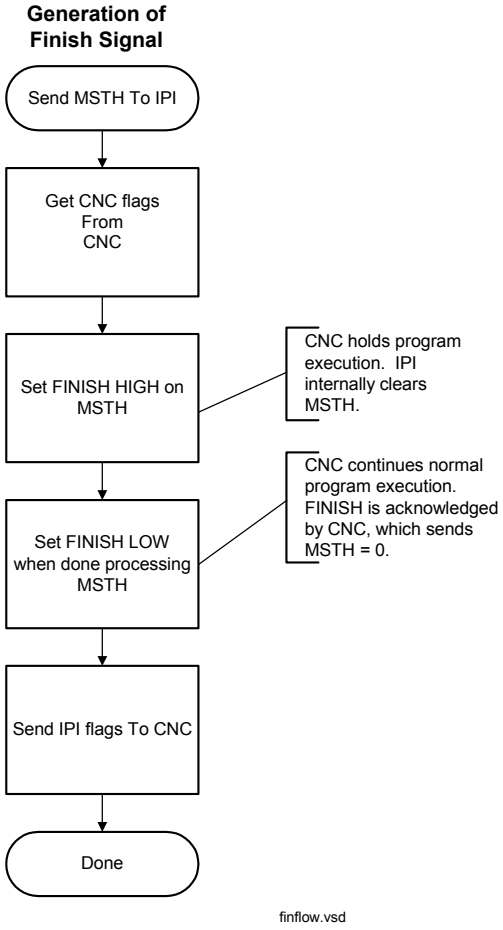


Figure 4-2, IPI: M, S, T, or H Code to Finish Signal

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 4-12.	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 4-13 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 4-14.	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 4-15.	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 inputs numerical analog value to a multifunction register for evaluation.
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 4-16.	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 4-17.	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 4-19.	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 4-22.	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 4-26.	Performs Boolean AND function with value in previous register, value in current register and new element value.
ORB See page 4-28.	Performs Boolean OR function with value in previous register, value in current register and new element's value.
SET See page 4-30.	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 4-31.	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 4-32.	Used in pairs. **CTL** - ANDs specified element with all subsequent instructions until deactivated. **CTR** - deactivates any active CTL instructions.
DEC See page 4-34.	For every cycle in which the current register value is true, the numeric value of the new element decreases.
INC See page 4-34.	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 4-34.	No operation is performed.

(Continued…)

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

Operand	Function
INV See page 4-24.	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. A 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. A 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.

(Continued…)

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. 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 **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

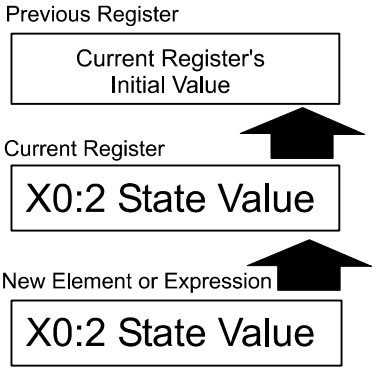
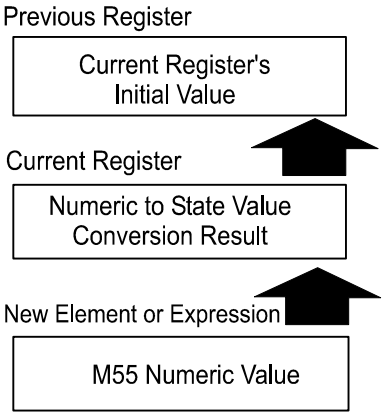
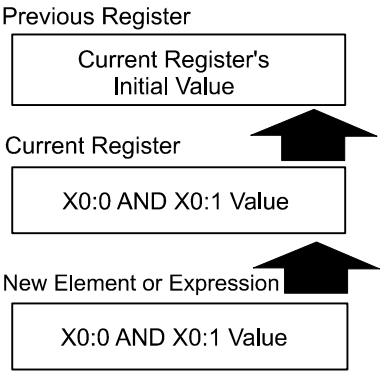
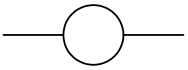
LD	Syntax	Valid Elements
Loads new element's state value into current register. If new element has numeric value, it is converted to appropriate state value. Loads any value already in the current register into the previous register.	LD [element]  - Ladder Equiv.  - Logical Symbol	R Registers W Registers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

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

Examples	Explanation
LD Example #3 Expression used as element. LD (X0:0 AND X0:1)	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. 

OUT	Syntax	Valid Elements
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.	OUT [element] 	W Registers M Registers S Registers T Registers X Registers Y Registers

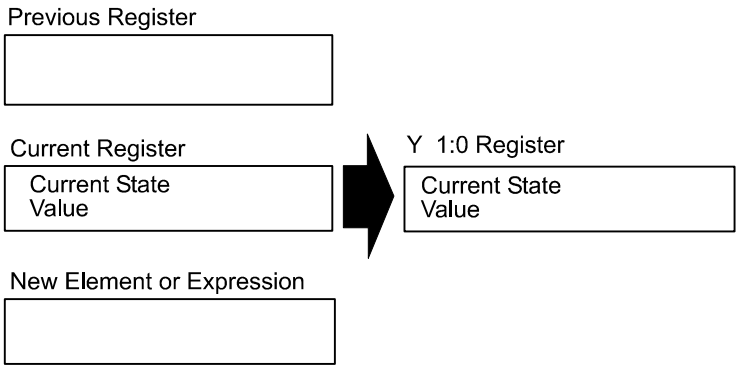
Examples	Explanation
OUT Example #1 One element. OUT Y1:0	Value in current register is sent to Y1:0 register. Value in Y1:0 register must be a TRUE/FALSE state. 
OUT Example #2 One element. OUT M70	Value in current register is sent to M70 register. Value sent to M70 register can be a state or number.

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

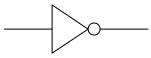
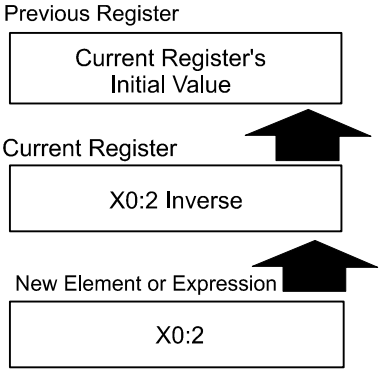
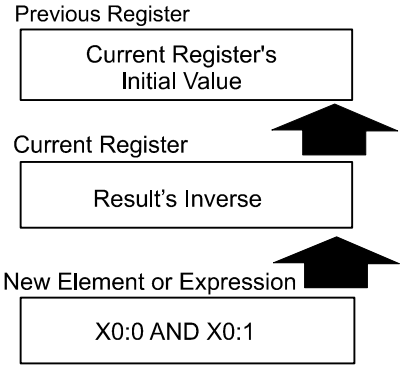
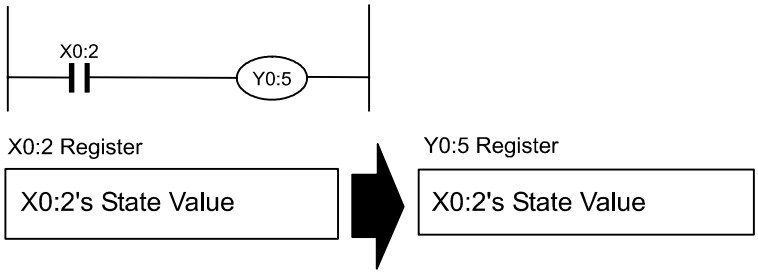
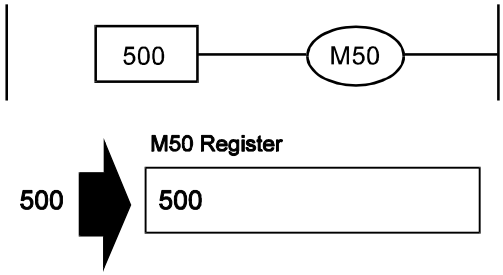
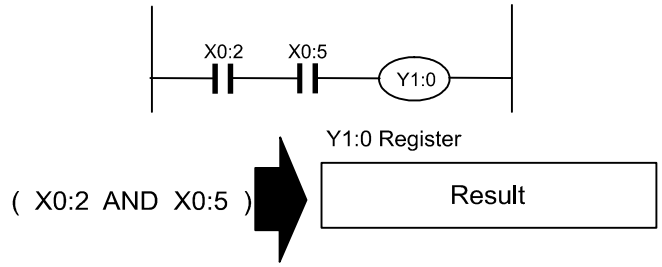
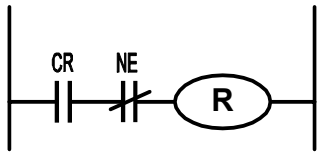
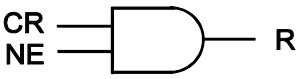
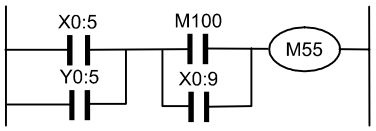
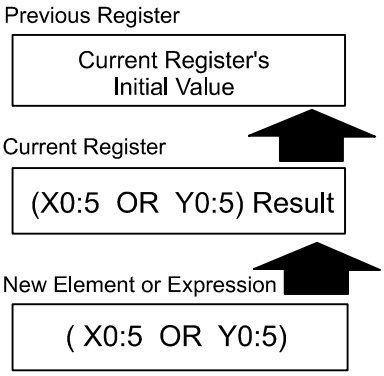
LDI	Syntax	Valid Elements
Loads element's state value to current register. If current register had value, it is moved to previous register. If element has numeric value, it is converted to appropriate state value.	LDI [element]  - Ladder Equiv.  - Logical Symbol	R Registers W Registers M Registers S Registers T Registers X Registers Y 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. 	
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. 	

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]	R Registers W Registers M Registers S Registers T Registers X Registers Y 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. 	
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. 	

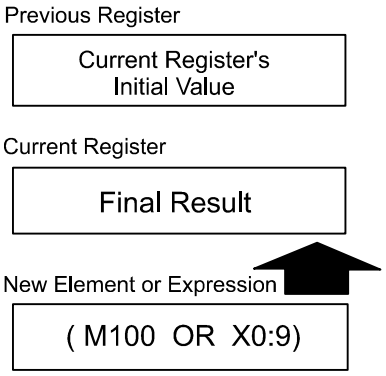
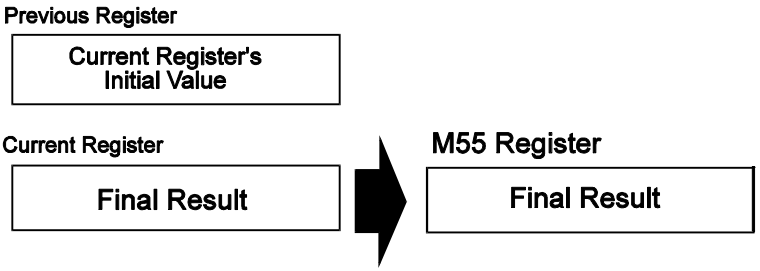
(Continued…)

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

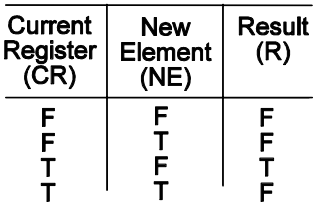
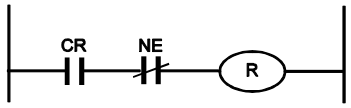
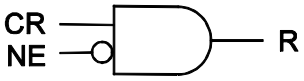
AND	Syntax	Valid Elements
Performs a Boolean logic AND function with value in current register and new element. Result remains in current register. Previous register is unaffected.	AND [Element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions
Truth Table	**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. 	

(Continued…)

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

Examples	Explanation
AND (M100 OR X0:9)	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. 
OUT M55	State value in current register is copied into M55 register. 

ANI	Syntax	Valid Elements
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.	ANI [element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions

Truth Table	Ladder Equivalent	Logic Symbol
		

ANI

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

(Continued…)

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

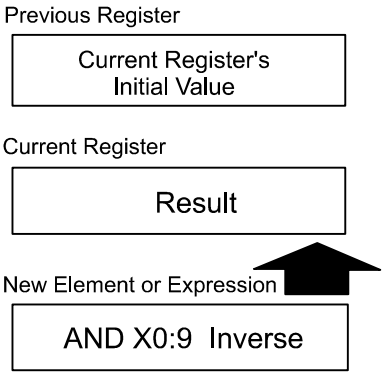
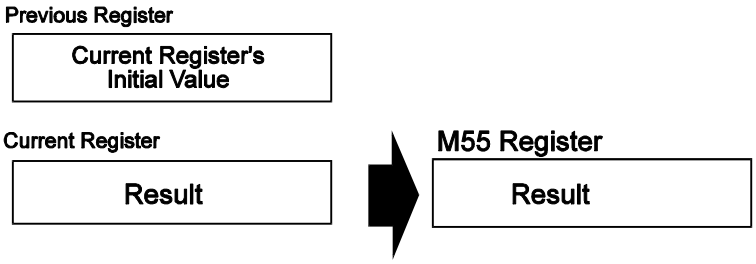
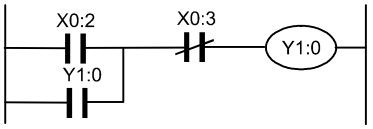
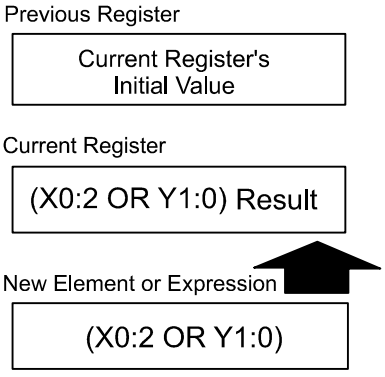
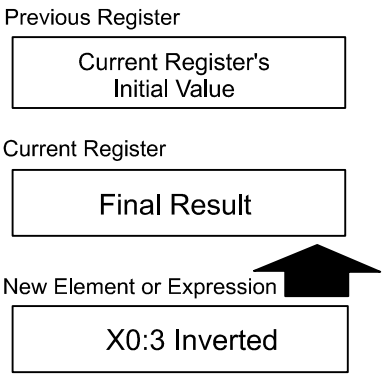
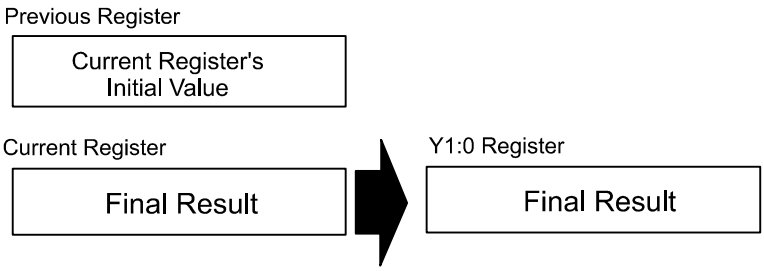
Examples	Explanation
ANI Example #1 ANI X0:9	Inverse state of X0:9 register is ANDed with the state value in the current register. Result is kept in current register. 
OUT M55	Value in current register is sent to register M55. 
ANI Example #2 LD (X0:2 OR Y1:0)	 State values in X0:2 and Y1:0 registers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

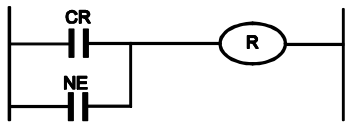
Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

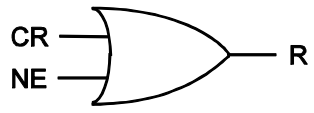
Examples	Explanation
ANI X0:3	State value stored in input register X0:3 is inverted and ANDed to previous result. Final result remains in current register. 
OUT Y1:0	State value in current register is copied to Y1:0 output register. 

OR	Syntax	Valid Elements
Performs Boolean logic OR function using new element and state value in current register. Result remains in current register. Previous register is unaffected.	OR [element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions
Truth Table	**Ladder Equivalent**	**Logic Symbol**

OR

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





(Continued…)

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

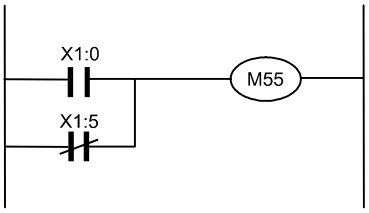
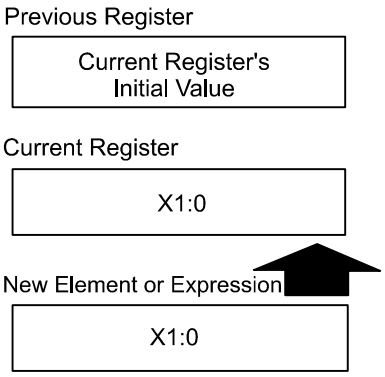
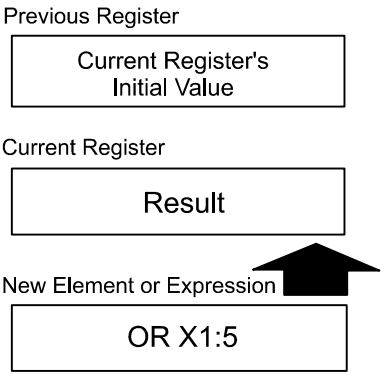
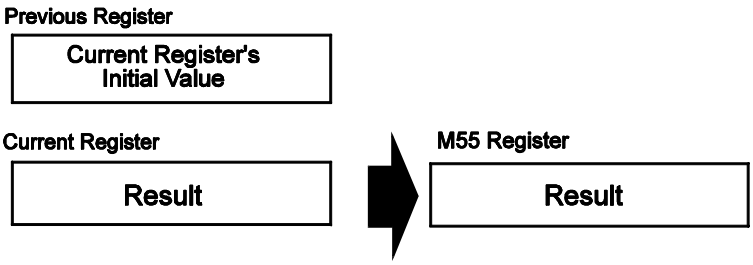
Examples	Explanation
OR Example #1 LD X1:0 OR X1:5	 Input state of X1:0 is loaded into current register. Current register's previous value is copied into previous register.  Input state of X1:5 register is ORed with the state value in the current register. Result is kept in current register
          OUT M55	 Value in current register is sent to register M55. 

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

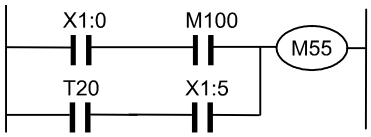
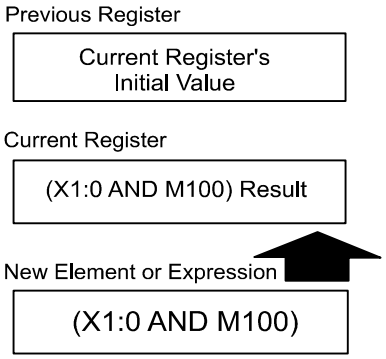
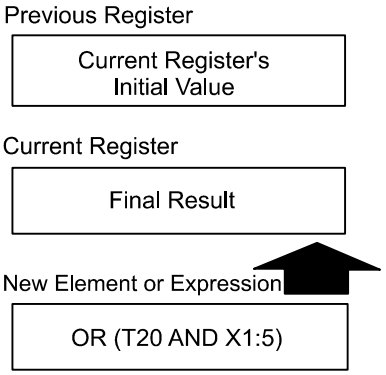
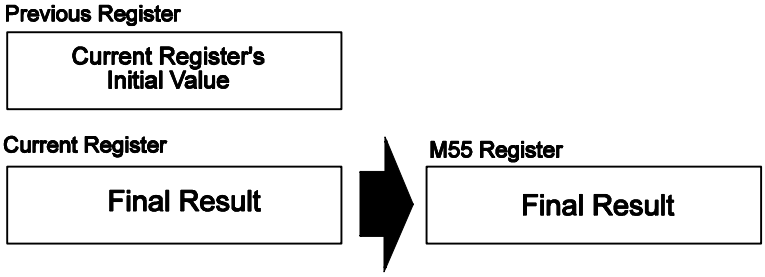
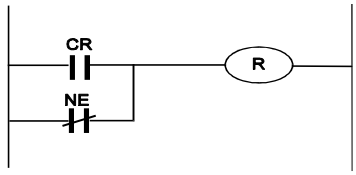
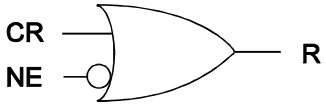
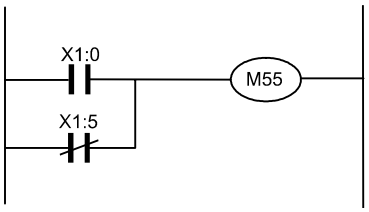
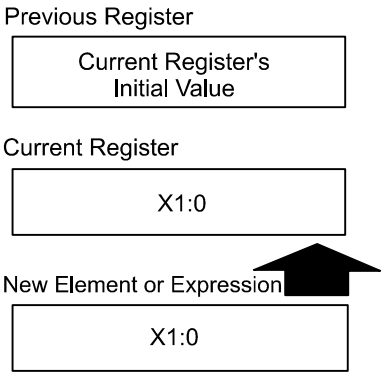
Examples	Explanation
OR Example #2	
LD (X1:0 AND M100)	State value in X1:0 register and state value equivalent in M100 register are ANDed. Result is loaded into current register. Current registers previous value is copied into previous register. 
OR (T20 AND X1:5)	Timer 20 state value and state value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

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

ORI	Syntax	Valid Elements
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.	ORI [element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions
Truth Table	**Ladder Equivalent**	**Logic Symbol**
ORI Current Register (CR) / New Element (NE) / Result (R) F F T F T F T F T T T T		
Examples	**Explanation**	
ORI Example #1 LD X1:0	 Input state of X1:0 is loaded into current register. Current register's previous value is copied into previous register. 	

(Continued…)

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

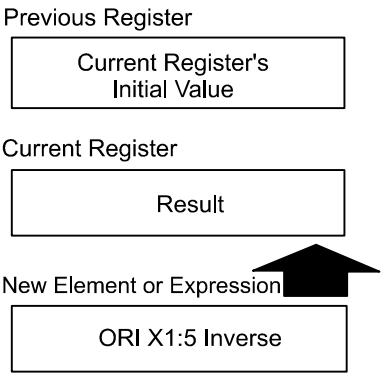
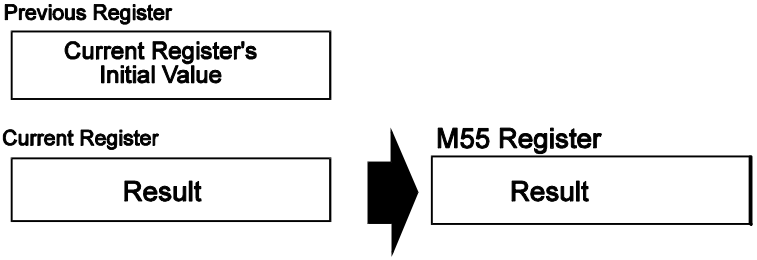
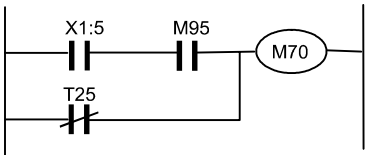
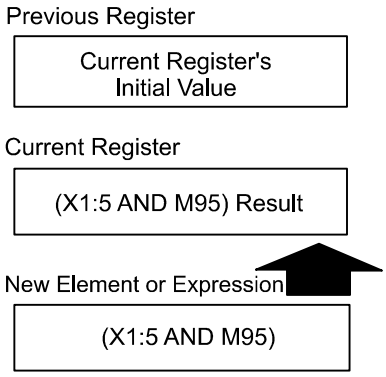
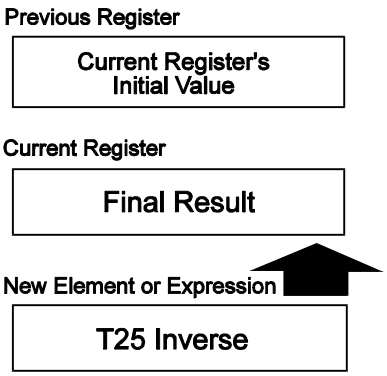
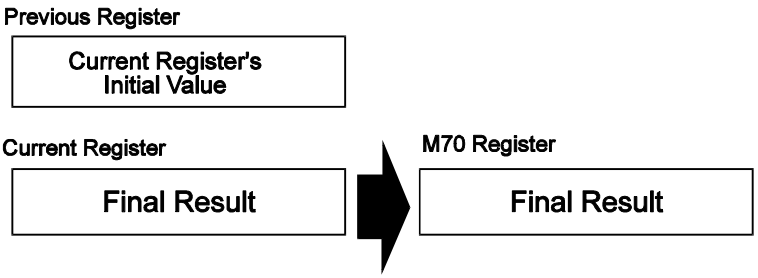
Examples	Explanation
ORI X1:5	Input state of X1:5 status register is inverted and ORed with the state value in the current register. Result is kept in current register. 
OUT M55	Value in current register is sent to register M55. 
ORI Example #2	
LD (X1:5 AND M95)	State value in X1:5 register and equivalent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

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

Examples	Explanation
ORI T25	Timer T25's state value is inverted and ORed with value in current register. Final result is kept in current register. 
OUT M70	Value in current register is copied to M70 register. 

INV	Syntax	Valid Elements
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.	INV [element] – or – INV	Y Registers

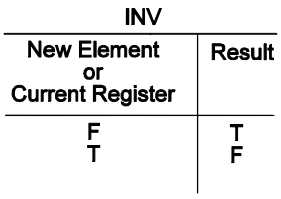
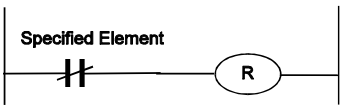
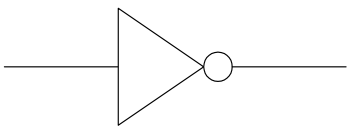
Truth Table	Ladder Equivalent	Logic Symbol
		

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

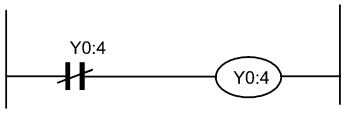
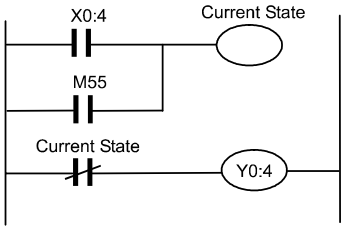
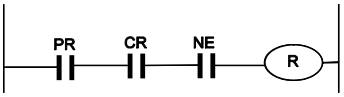
Examples	Explanation
INV Example #1 INV Y0:4	 State value of Y0:4 register is inverted. Current and previous registers not affected.
INV Example #2	
LD X0:4	Loads X0:4 input into current register. Current register's previous value is copied into previous register. Previous Register Current Register's 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

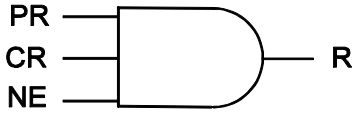
Table 4-6, Detailed Descriptions and Examples of Operands (Continued)

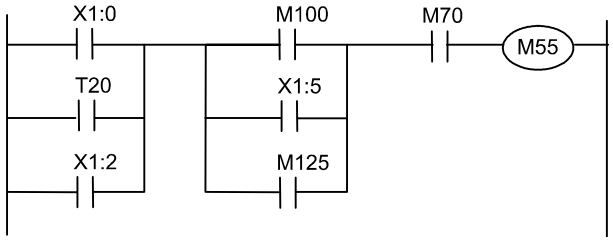
ANB	Syntax	Valid Elements
Performs Boolean AND function with value in previous register, value in current register and new element's value.	ANB [element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions
Truth Table	**Ladder Equivalent**	**Logic Symbol**

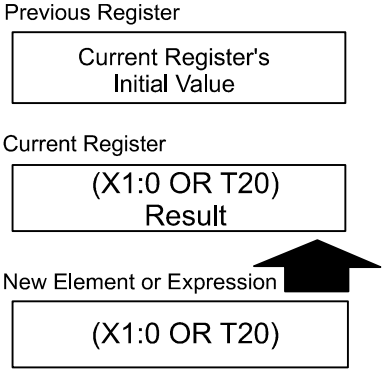
ANB

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





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.



(Continued…)

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

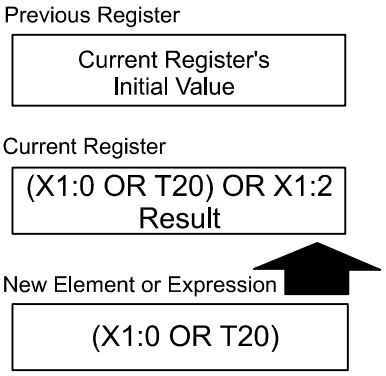
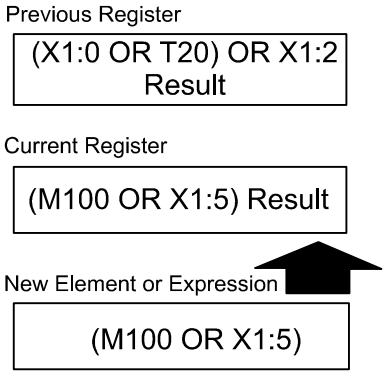
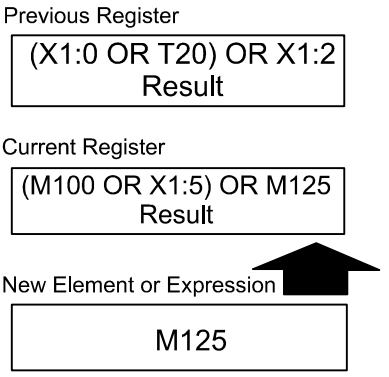
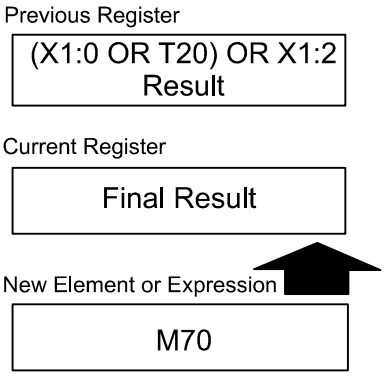
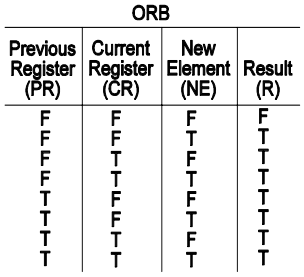
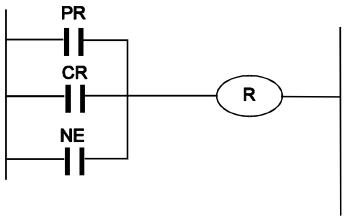
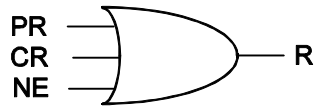
Examples	Explanation
OR X1:2	Current register's value is ORed with value in X1:2 register; the result remains in current register. OR operation does not change value in previous register. 
LD (M100 OR X1:5)	Value in current register is copied to previous register. Resulting value of new expression is loaded into current register. 
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. 

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

Examples	Explanation
ANB M70	Value in previous register, current register and new element are ANDed together to produce result that remains in current register. 

ORB	Syntax	Valid Elements
Performs Boolean OR function with value in previous register, value in current register and new element's value.	ORB [element]	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers Expressions

Truth Table	Ladder Equivalent	Logic Symbol
		

ORB

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

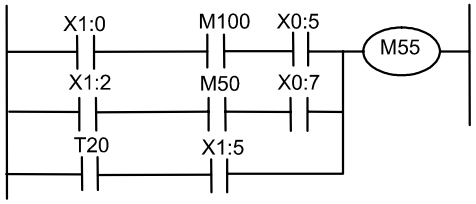
Examples	Explanation
ORB Example #1  LD (X1:0 AND M100)	Copies value in current register to previous register, evaluates new expression and loads resulting value into current register.

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

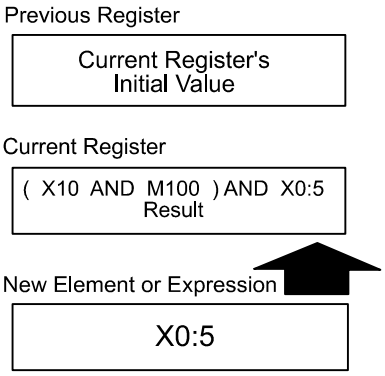
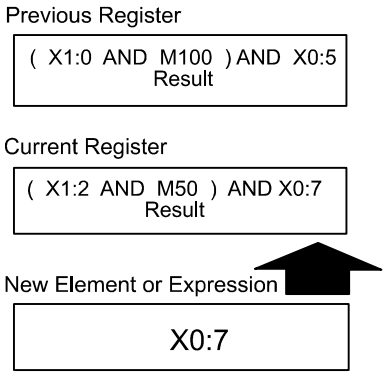
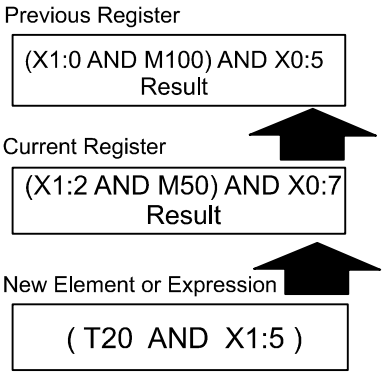
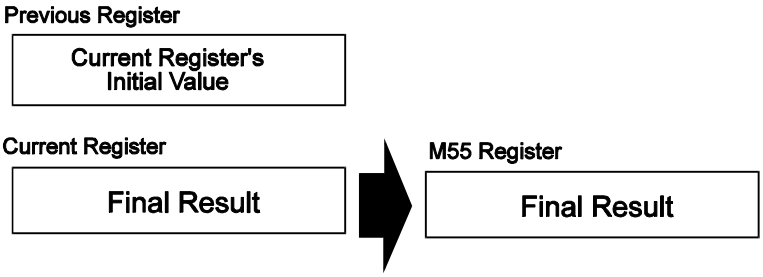
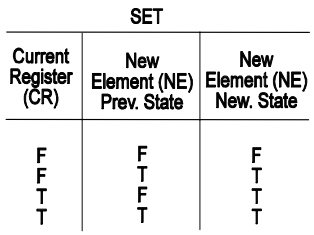
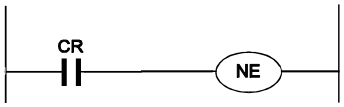
Examples	Explanation
AND X0:5	
ORB Example #2 LD (X1:2 AND M50)	Value in Current register is copied to previous register; new expression is evaluated and result remains in current register. **NOTE:** The value shown in the previous register is the result of ORB Example #1. 
AND X0:7	Value in current register is ANDed with new element. Result remains in current register. 
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
	
OUT M55	Copies value in current register to M55 multifunction register. 

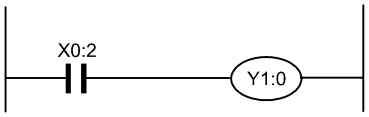
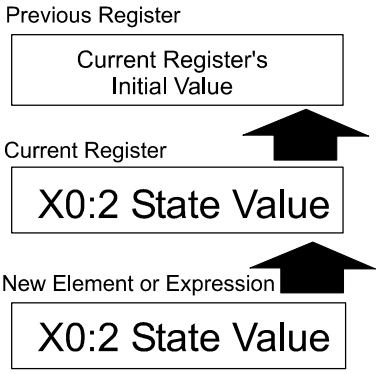
SET	Syntax	Valid Elements
This instruction latches the new element to a TRUE value for subsequent cycles. If current register holds a TRUE value, a TRUE state value is copied into the new element's register. If current register holds a FALSE value, no activity occurs. A subsequent MOV statement or a RES instruction can be used to unlatch the register.	SET [element]	W Registers M Registers Y Registers

Truth Table	Ladder Equivalent	Logic Symbol
		

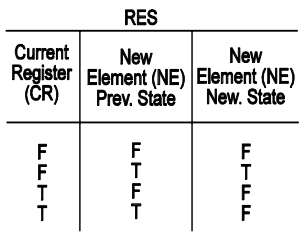
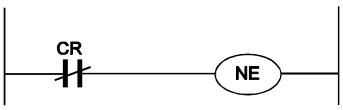
SET

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 SET Y1:0	 Loads current value into previous register and loads value in X0:2 register into current register.  Sets value in Y1:0 register to true if X0:2 was true. No action taken if X0:2 was false.

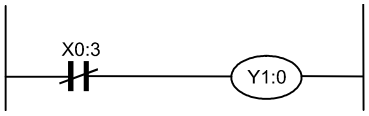
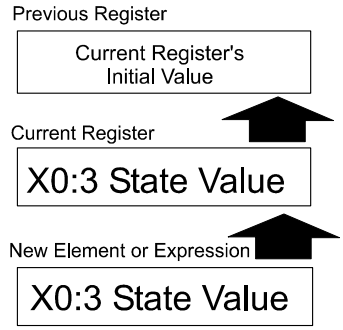
RES	Syntax	Valid Elements
This instruction resets the new element to a FALSE value for subsequent cycles. If current register holds a TRUE value, FALSE is copied in new element's register. If current register holds a FALSE value, no activity occurs. A subsequent MOV statement or a SET instruction can be used to relatch the register.	RES [element]	W Registers M Registers Y Registers

Truth Table	Ladder Equivalent	Logic Symbol
		

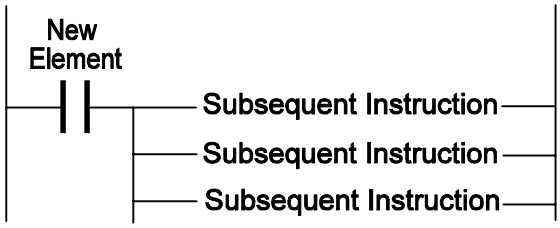
RES

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

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

Examples	Explanation
RES Example #1 LD X0:3 RES Y1:0	 Copies value from current register into previous register and loads value from X0:3 register into current register.  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	R Registers W Registers M Registers S Registers T Registers X Registers Y Registers

	Ladder Equivalent
	

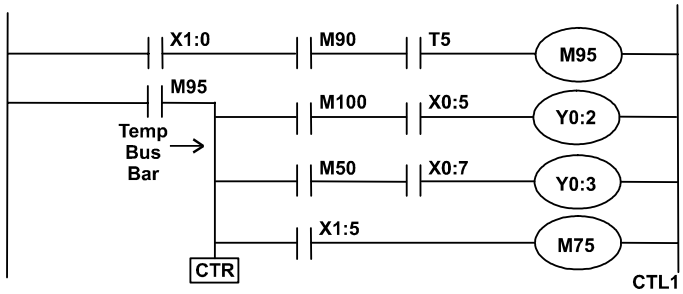
Examples	Explanation
CTL/CTR Example #1	 Moves value from current register to previous register and loads new expression result into the current register.

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

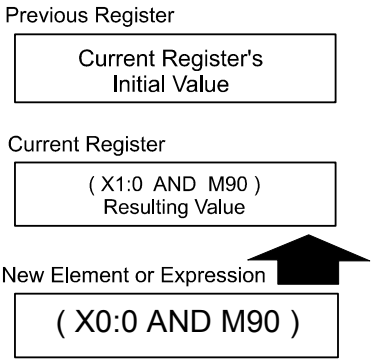
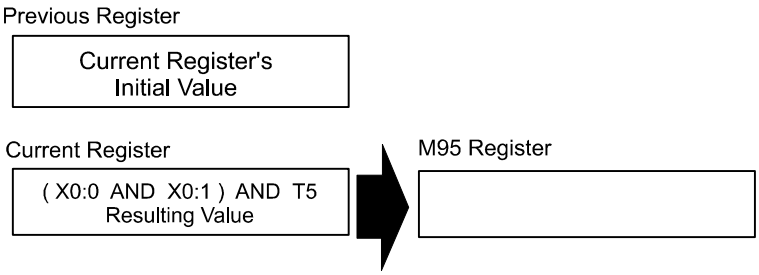
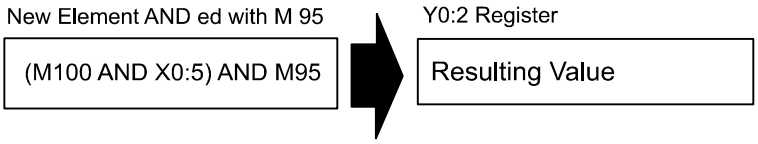
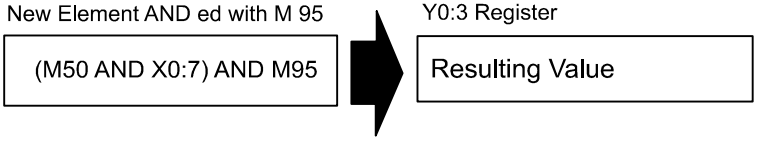
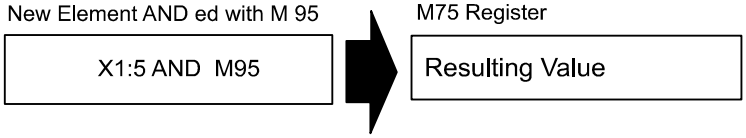
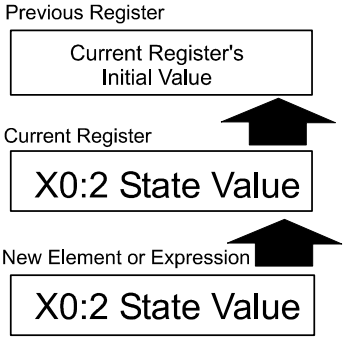
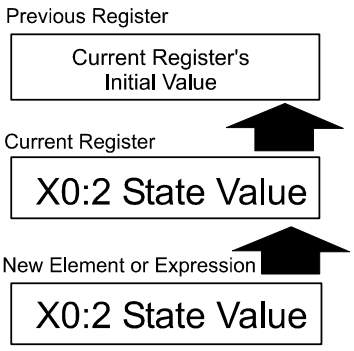
Examples	Explanation
LD (X1:0 AND M90)	
AND T5	Value in current register is ANDed with T5 register's state value. Result remains in current register.
OUT M95	Copies the value in the current register to multifunction register M95. 
CTL M95	Specifies that value in M95 register will be ANDed with all subsequent 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. 
MOV (M50 AND X0:7) Y0:3	Next expression is ANDed with value in M95 register. Result is copied directly to Y0:3 register. 
MOV X1:5 M75	Next expression is ANDed with value in M95 register. Result is copied directly to M75 register. 
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]	W Registers M Registers
Examples	**Explanation**	
DEC Example #1 LD X0:2 DEC M80	Copies value from current register into previous register. Loads value from X0:2 register into current register.  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]	W Registers M Registers
Examples	**Explanation**	
INC Example #1 LD X0:2 INC M80	Copies value from current register into previous register and loads value from X0:2 register into current register.  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 a 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:

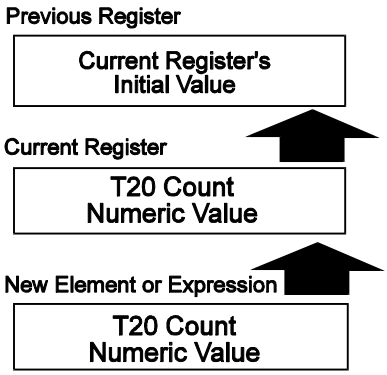
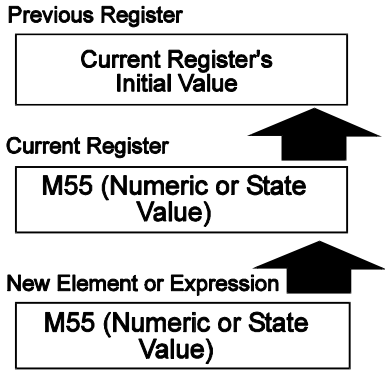
Delayed On Format: TON X.X X=time in seconds	If the current register value changes from FALSE to TRUE during the current cycle, the timer begins a countdown that lasts the specified number of seconds. When the countdown is complete, the timer's state register loads the high. In a future cycle, if the 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. 	
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. 	

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.

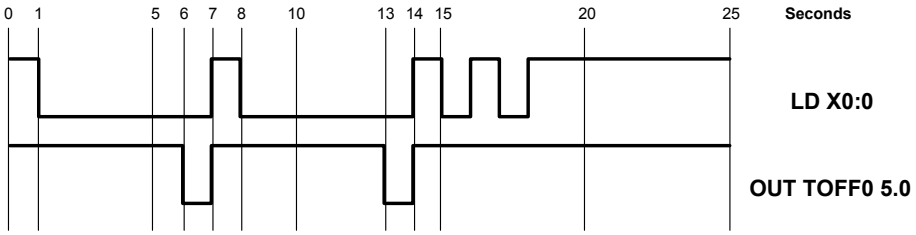


Figure 5-1, Timer Off Command

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.

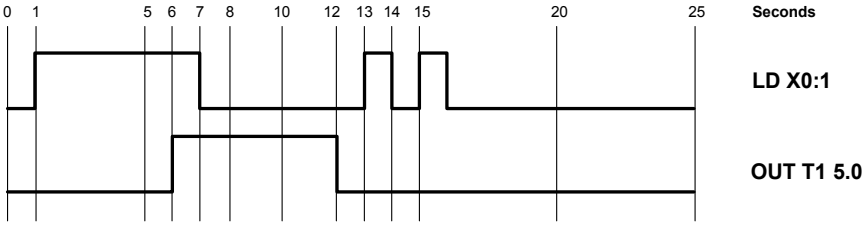


Figure 5-2, Timer Delayed On Then Off Command

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 non-retentive, so that the transitions from 14 seconds to 19 seconds do not affect the output.

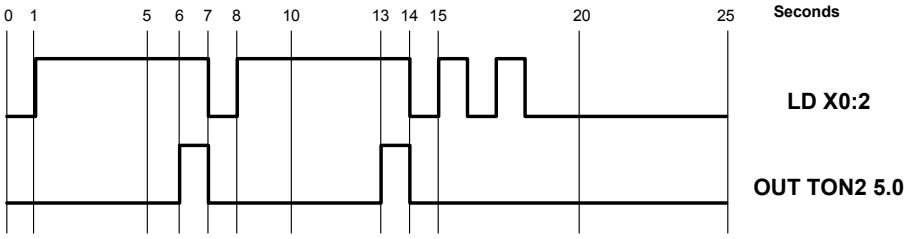


Figure 5-3, Timer On Command

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

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: R Registers W Registers M Registers S Registers T Registers X Registers Y Registers
EJP - Ends conditional jump instruction set.	EJP [block number]	Block Numbers: Any integer, all numbers must match.
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]	M Registers Y Registers
Example	**Explanation**	
MVA Example #1 Two elements. MVA X0:200 M100	Digital value of analog input at X0:200 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:** An X or H following the number can indicate Hexadecimal notation.
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 module. 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 module 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 = module 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 (R40) 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 INPUT1:3 OTI OUTPUT1:0	The Y1:0 register is pulsed (goes high) for 30 seconds or until the input X1:3 is detected (goes low).

(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 module 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 OTI statement.	LD Xn:b OWI Yn:b – or – LDI Xn:b OWI Yn:b n = module 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 received in M40 is received. As the pulses are received, SOTICNT increments. When 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 1:0 signal goes low. See OTIFLAG (R40) and SOTICNT (R33) above.

COTI	Syntax	Valid Elements
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 XHOME X0:0)

The label in this example defines the label "XHOME" and ties the label to input X0:0. HOME is the X-axis HOME switch. After the label is defined, any time the compiler encounters the program string XHOME, it will replace the text with X0:0.

In future references, the HOME switch can be referenced as XHOME or X0:0.

LIST

Format: #LIST

The LIST directive will instruct 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	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 step ladder 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.

```
**SAMPLE4.DBO: 12 JAN 01:


*(NOTE: HEADER INFO IS ONLY FOR REFERENCE, IT IS NOT COMPILED PROGRAM CODE)

**R REGISTERS: CNC TO IPI "READ" REGISTERS

*ESTOP      R00     TFLAG       R21
*SPINDLE    R01     TCODE       R22
*POSN       R02     HFLAG       R23
*FEED       R03     HCODE       R24
*PWRFAIL    R04     CMDRPM      R25
*MAN        R05     ZEROSPD     R26
*TCFINACK   R06     ATSPD       R27
*HOME       R07     SPDLOAD     R28
*SPLOOP     R08     TRUE        R29
*RUN        R09     FALSE       R30
*SVOFF      R10     TOOLNUM     R31
*M19FLAG    R11     TLOBIN0     R32
*TMACEND    R12     (unused)    R33
*CARRY      R13     (unused)    R34
*XMIT       R14     (unused)    R35
*HOMING     R15     (unused)    R36
*CNCERR     R16     (unused)    R37
*MFLAG      R17     (unused)    R38
*MCODE      R18     (unused)    R39
*SFLAG      R19     (unused)    R40
*SCODE      R20     (unused)    R41

**W REGISTERS: IPI TO CNC "WRITE" REGISTERS

*FINISH     W00     SPDLDIR     W13
*SVOFLT     W01     (unused)    W14
*FHOLD      W02     (unused)    W15
*TCHGFIN    W03     MSG         W16
```

*XSTART	W04	MREGRAN	W17
*XSTOP	W05	KEYMASK	W18
*XHOLD	W06	SPDLRPM	W19
*SPDLZERO	W07	LNFDLIM	W20
*SPDLGRCH	W08	ROFDLIM	W21
*AUTOINH	W09	RREGRAN	W22
*HWSTOP	W10	WREGRAN	W23
*SPDL100	W11	M19END	W24
*FEED100	W12	SPDLOPEN	W25

*(THE FOLLOWING GROUP OF W REGS WERE PREVIOUSLY CALLED "INPUT FUNCTIONS")

*BLKSKIP0	W26	INIT999	W40
*BLKSKIP1	W27	INC999	W41
*BLKSKIP2	W28	SPDLCW	W42
*BLKSKIP3	W29	SPDLCCW	W43
*BLKSKIP4	W30	SPDLOFF	W44
*BLKSKIP5	W31	JOGMINUS	W45
*BLKSKIP6	W32	JOGPLUS	W46
*BLKSKIP7	W33	RES1	W47
*BLKSKIP8	W34	RES10	W48
*BLKSKIP9	W35	RES100	W49
*TOOLGRD	W36	XSEL	W50
*XMAN	W37	YSEL	W51
*RDKYBD	W38	ZSEL	W52
*NOKYBD	W39	USEL	W53
*		WSEL	W54

*GENERAL W REGISTERS (CONTINUED)

*FEEDOVR	W55
*SPDLOVR	W56
*HWLIM	W57

**CNC PARAMETER REGISTERS- VALUES SET VIA SETUP UTILITY; READ-ONLY

*M40LO	P1010
*M40HI	P1011
*M41LO	P1012
*M41HI	P1013
*M42LO	P1014
*M42HI	P1015
*M43LO	P1016
*M43HI	P1017
*M44LO	P1018
*M44HI	P1019

```
**M REGISTERS M0 THRU M255 FOR IPI PROGRAMMER USE

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

**MANUAL PANEL PRE-DEFINED INPUTS, FYI

*M3KEY          X0:100   FEED1KEY   X0:111
*M5KEY          X0:101   FEED2KEY   X0:112
*M4KEY          X0:102   FEED4KEY   X0:113
*RESKEY         X0:103   FEED8KEY   X0:114
*MINUSKEY       X0:104   JOG1KEY    X0:115
*PLUSKEY        X0:105   JOG2KEY    X0:116
*HOLDKEY        X0:106   JOG4KEY    X0:117
*STARTKEY       X0:107   SPDL1KEY   X0:118
*AXIS1KEY       X0:108   SPDL2KEY   X0:119
*AXIS2KEY       X0:109   SPDL4KEY   X0:120
*AXIS4KEY       X0:110   SPDL8KEY   X0:121

*MPSPARE1       X0:122
*MPSPARE2       X0:123
*MPSPARE3       X0:124


**DEFINE INPUTS

#DEFINE XHOME      X0:00   *X42/1    *HOME SWITCHES ACTIVE LOW
#DEFINE YHOME      X0:01   *X42/2
#DEFINE ZHOME      X0:02   *X42/3
#DEFINE CNCACK     X0:03   *X42/4    *INPUT REQUIRED BY HARDWARE
#DEFINE UHOME      X0:04   *X42/5
#DEFINE WHOME      X0:05   *X42/6    *INPUT RESERVED FOR FUTURE GROWTH

#DEFINE X006       X0:06   *X42/7
#DEFINE X007       X0:07   *X42/8
#DEFINE X008       X0:08   *X42/9
#DEFINE X009       X0:09   *X42/10
#DEFINE X010       X0:10   *X42/11
#DEFINE X011       X0:11   *X42/12
#DEFINE X012       X0:12   *X42/13
#DEFINE X013       X0:13   *X42/14
#DEFINE X014       X0:14   *X42/15
#DEFINE X015       X0:15   *X42/16
#DEFINE X016       X0:16   *X42/17
#DEFINE X017       X0:17   *X42/18
#DEFINE X018       X0:18   *X42/19
#DEFINE X019       X0:19   *X42/20
```

```
#DEFINE X020        X0:20   *X42/21
#DEFINE X021        X0:21   *X42/22
#DEFINE X022        X0:22   *X42/23
#DEFINE X023        X0:23   *X42/24
#DEFINE X024        X0:24   *X42/25
#DEFINE X025        X0:25   *X42/26
#DEFINE X026        X0:26   *X42/27
#DEFINE X027        X0:27   *X42/28
#DEFINE X028        X0:28   *X42/29
#DEFINE X029        X0:29   *X42/30

#DEFINE             X0:30   *X42/31   *SOFTWARE ESTOP INPUT REQUIRED
ESTOPSFT
#DEFINE DREN1       X0:31   *X42/32   *DRIVE ENABLE 1 REQUIRED BY SOFTWARE
#DEFINE DREN2       X0:32   *X42/33   *DRIVE ENABLE 2 REQUIRED BY HARDWARE

**DEFINE
OUTPUTS

#DEFINE CNCON       Y0:00   *X41/9    *CNCON REQUIRED BY SYSTEM
#DEFINE MODEOP      Y0:01   *X41/10   *RESERVED FOR MACHINE SAFETY LOCKOUTS

#DEFINE Y002        Y0:02   *X41/11
#DEFINE Y003        Y0:03   *X41/12
#DEFINE Y004        Y0:04   *X41/13
#DEFINE Y005        Y0:05   *X41/14
#DEFINE Y006        Y0:06   *X41/15
#DEFINE Y007        Y0:07   *X41/16
#DEFINE Y008        Y0:08   *X41/17
#DEFINE Y009        Y0:09   *X41/18
#DEFINE Y010        Y0:10   *X41/19
#DEFINE Y011        Y0:11   *X41/20
#DEFINE Y012        Y0:12   *X41/21
#DEFINE Y013        Y0:13   *X41/22
#DEFINE Y014        Y0:14   *X41/23
#DEFINE Y015        Y0:15   *X41/24
#DEFINE Y016        Y0:16   *X41/25
#DEFINE Y017        Y0:17   *X41/26
#DEFINE Y018        Y0:18   *X41/27
#DEFINE Y019        Y0:19   *X41/28
#DEFINE Y020        Y0:20   *X41/29
#DEFINE Y021        Y0:21   *X41/30

#DEFINE SVOEN       Y0:22   *X41/31   *SERVO ENABLE REQUIRED BY SYSTEM


**DEFINE BASIC M-FUNCTION REGISTERS
```

```
#DEFINE DAYMO     M0          *DAY & MONTH (FORMAT "DDMM")
#DEFINE YEAR      M1          *YEAR (FORMAT "YYYY")
#DEFINE FINWAIT   M2
#DEFINE CHGM      M3
#DEFINE CHGR      M4
#DEFINE CHGW      M5

#DEFINE MM1       M10
#DEFINE MM2       M11
#DEFINE MM3       M12
#DEFINE MM4       M13
#DEFINE MM5       M14
#DEFINE MM6       M15
#DEFINE MM7       M16
#DEFINE MM8       M17
#DEFINE MM9       M18
#DEFINE MM10      M19         *ISO STANDARD CLAMP M CODE

#DEFINE MM19      M20
#DEFINE MM30      M21
#DEFINE MM40      M22
#DEFINE MM41      M23
#DEFINE MM42      M24
#DEFINE MM43      M25
#DEFINE MM44      M26

**TIMER DOCUMENTATION

*T0       1.0     FINISH PULSE ON TIMER
*T1       1.0     FINISH PULSE DELAY TIMER (USED ONLY FOR DEBUGGING)
*T2       0.5     BLINKER TIMER- USED FOR WARNING LIGHTS, ETC.

**BUILDER MESSAGES
**MESSAGES MUST BE ENTERED AND ENABLED VIA SETUP UTILITY

*ERROR MESSAGES HOLD PROGRAM EXECUTION

*MESSAGE 1        FIRST ERROR MEESAGE
*MESSAGE 127      LAST ERROR MESSAGE

*WARNING MESSAGES ONLY DISPLAY TO SCREEN

*MESSAGE 128      FIRST WARNING MESSAGE
*MESSAGE 255      LAST WARNING MESSAGE

**MACHINE SPECIFIC MCODES USED
```

```
*M50                ENABLE TURRET    (example)
*M51                DISABLE TURRET   (example)

**COMMANDS PROGRAMMED BEFORE "START" DIRECTIVE ARE EXECUTED
**IN 1ST SCAN ONLY

*FOLLOWING REGISTERS ARE SET TO DISPLAY DATE OF IPI PROGRAM IN MREGS 0 & 1

MOV 0112 DAYMO
MOV 2001 YEAR

START                       *DEFINES REPEATING PORTION OF PROGRAM

**FINISH PULSE GENERATION

LD ( MFLAG OR SFLAG )       *SETS FINISH HIGH ON ANY FLAG
OR ( TFLAG OR HFLAG )
*OUT TON1 1.0               *ALLOWS DELAY TO SEE CODE IN MONITOR
*LD T1                      *DELETE THESE 2 TON1 LINES FOR SPEED
SET FINISH

LDI FINWAIT                 *SET FINWAIT HIGH DURING OPERATIONS THAT
AND FINISH                  *REQUIRE PROGRAM HOLD TILL COMPLETE
OUT TON0 0.1                *1.0 SEC DURATION ONLY FOR DEMO PURPOSES

IF T0                       *RESETS FINISH AFTER FINISH TIMER T0
RES FINISH
EDF

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

LD ( MCODE EQ 2 )           *SET MM2 FOR PROGRAM END: REGISTER M1
OUT MM2

LD ( MCODE EQ 3 )           *SET MM3 FOR SPINDLE FWD: REGISTER M2
OR MM3                      *LATCH ON
ANI MM5                     *DISABLE ON M5
ANI ( MM2 OR MM30 )         *DISABLE ON M2 OR M30
RES MM4                     *RESET M4: ALLOWS DIRECT DIRECTION CHANGE
OUT MM3                     *USE TO SET OUTPUT FOR SPINDLE FORWARD

LD ( MCODE EQ 4 )           *SET MM4 FOR SPINDLE REV: REGISTER M3
OR MM4                      *LATCH ON
ANI MM5                     *DISABLE ON M5
```

```
ANI ( MM2 OR MM30 )          *DISABLE ON M2 OR M30
RES MM3                      *RESET M3: ALLOWS DIRECT DIRECTION CHANGE
OUT MM4                      *USE TO SET OUTPUT FOR SPINDLE REVERSE

LD ( MCODE EQ 5 )            *SET MM5 FOR SPINDLE STOP: REGISTER M4
OUT MM5

LD ( MCODE EQ 8 )            *SET MM8 FOR COOLANT ON: REGISTER M6
OR MM8                       *LATCH ON
ANI MM9                      *DISABLE ON M9
ANI ( MM2 OR MM30 )          *DISABLE ON M2 OR M30
OUT MM8

LD ( MCODE EQ 9 )            *SET MM9 FOR COOLANT OFF: REGISTER M7
OUT MM9

LD ( MCODE EQ 30 )           *SET MM30 FOR SUBPROGRAM END: REGISTER M8
OUT MM30

**0.5 SECOND BLINKER

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

**MACHINE TURN ON STATEMENTS, REQUIRED FOR ALL SYSTEMS

LD CNCACK                    *IF CNCACK AT X42/4
OUT CNCON                    *THEN SSK1 AT X41/9

*THE FOLLOWING BLOCK OF CODE WILL BE MODIFIED PER MACHINE SAFETY REQ'TS
*FOR DOORGUARDS, SAFETY SHIELDS, OPTIONAL RESTRICTED OPERATION

LD TRUE                      *USER CODE REPLACES ALWAYS TRUE CONDITION
OUT MODEOP                   *ALWAYS SSK2 AT X41/10

**MACHINE SPECIFIC PROGRAM STATEMENTS TO BE ENTERED HERE
**THIS IS THE CODE TO BE DEVELOPED FOR EACH MACHINE'S REQUIREMENTS

*<<<<<THE MACHINE BUILDER'S CODE WILL BE ENTERED HERE>>>>>

**SETS IPI MONITOR TO DISPLAY SELECTED REGISTER RANGES

IF ( HCODE LE 50 )     *ALLOWS ONLY CHANGES FOR M RANGES 0-255

IF CHGM                      *IF ACTIVE WILL CHANGE M IPI MONITOR RANGE
    IF HFLAG
    MOV HCODE MREGRAN
```

```
    EDF
EDF
EDF

IF ( HCODE LE 12 )          *ALLOWS ONLY CHANGES FOR R/W RANGES 0-64

IF CHGR                     *IF ACTIVE WILL CHANGE R IPI MONITOR RANGE
    IF HFLAG
    MOV HCODE RREGRAN
    EDF
EDF

IF CHGW                     *IF ACTIVE WILL CHANGE W IPI MONITOR RANGE
    IF HFLAG
    MOV HCODE WREGRAN
    EDF
EDF

EDF                         *END OF NOT PASS HIGHER HCODES THAN 26 IF

IF ( HCODE EQ 60 )          *ACTIVATE ONLY M CHANGES TO IPI MONITOR
     MOV 1 CHGM
     MOV 0 CHGR
     MOV 0 CHGW
EDF

IF ( HCODE EQ 61 )          *ACTIVATE ONLY R CHANGES TO IPI MONITOR
     MOV 1 CHGR
     MOV 0 CHGM
     MOV 0 CHGW
EDF

IF ( HCODE EQ 62 )          *ACTIVATE ONLY W CHANGES TO IPI MONITOR
     MOV 1 CHGW
     MOV 0 CHGM
     MOV 0 CHGR
EDF

IF ( HCODE EQ 63 )          *TURNS ALL "CHG" FLAGS OFF, DISABLES CHANGES
     MOV 0 CHGM
     MOV 0 CHGR
     MOV 0 CHGW
EDF

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

IF ENABLE                *DECIMAL NOT GREATER THAN 15
   IF ( DECIMAL NE TEMP1 ) *DECIMAL HAS NOT CHANGED, NO CHANGE REQUIRED
    MOV DECIMAL TEMP1    *STORE TEMP VALUE TO DETERMINE IF
                         *OUTPUT CHANGE REQUIRED
       IF ( TEMP1 EQ 0 ) *IF TEMP1 IS 0, RESET ALL OUTPUTS
       RES 8BIT
       RES 4BIT
       RES 2BIT
       RES 1BIT

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

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

            IF ( 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
            RES 8BIT                   *IF NOT RESET 8BIT
            EDF

            IF ( 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
            RES 4BIT                   *IF NOT RESET 4BIT
            EDF

            IF ( 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
            RES 2BIT                   *IF NOT RESET 2BIT
            EDF

            IF ( 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
            RES 1BIT                   *IF NOT RESET 1BIT
            EDF

            EDF                        *END TEMP2 NOT ZERO LOOP
        EDF                            *END SET OUTPUT BITS LOOP
    EDF                                *END DECIMAL HAS CHANGED LOOP
EDF                                    *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          M80
#DEFINE BITREG4          M81
#DEFINE BITREG2          M82
#DEFINE BITREG1          M83
#DEFINE BITREG84         M84
#DEFINE BITREG21         M85

#DEFINE BIT1             X0:10
#DEFINE BIT2             X0:11
#DEFINE BIT4             X0:12
#DEFINE BIT8             X0:13

START
*LOADS BITS TO REGISTERS FOR COMPARISON

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

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

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

```
IF BIT8                         *CONVERTS BIT 8 TO REGISTER
MOV 8 BITREG8
ELS
MOV 0 BITREG8
EDF

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           X0:10
#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 EVENTOUT
INC COUNTER
EDF

END
```

A simple increment counter is created. Multifunction registers can count from 0 to 65535. Underflow (negative counting past zero) 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 (R40) equals TOOLDIFF, which means magazine rotated desired number of times, and that OTIFLAG (R33) 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
```

D

E

F

G

H

I

N

O

S

T

X

Y

Z

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 70000488C
31-October-04