
MA**Absolute/Incremental Mode Enable**

Type	Motion	Product	Rev
Syntax	<!><@><a>MA	6K	5.0
Units	n/a		
Range	b = 0 (incremental mode) or 1 (absolute mode)		
Default	0		
Response	MA: *MA0000_0000 1MA: *1MA0		
See Also	COMEXC, D, GO, GOBUF, PSET		

The Absolute/Incremental Mode Enable (MA) command specifies whether the moves to follow are made with respect to current position (incremental) or with respect to an absolute zero position.

In incremental mode (MA0), all moves are made with respect to the position at the beginning of the move. This mode is useful for repeating moves of the same distance.

In absolute mode (MA1), all moves are made with respect to the absolute zero position. The absolute zero position is equal to zero upon power up, and can be redefined with the PSET command. An internal counter keeps track of absolute position.

ON-THE-FLY CHANGES: You can change positioning modes *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!MA) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (MA) followed by a buffered go command (GO).

Example:

```
PSET0,0,0,1000 ; Set absolute position on axes 1, 2, and 3 to zero,
                ; and axis 4 to 1000 units
MA1111         ; Enable absolute mode on axes 1 through 4
A2,2,25000,25000 ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                ; for axes 1, 2, 3 and 4
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                ; for axes 1, 2, 3 and 4
V1,1,1,2       ; Set velocity to 1, 1, 1, and 2 units/sec
                ; for axes 1, 2, 3 and 4 respectively
@D10           ; Set distance on all axes to 10 units
GO1111         ; Initiate motion on all axes (axes 1, 2, and 3 will move
                ; 10 units in the positive direction, axis 4 will move
                ; 990 units in the negative direction)
```

MC

Preset/Continuous Mode Enable

Type	Motion	Product	Rev
Syntax	<!><@><a>MC	6K	5.0
Units	n/a		
Range	b = 0 (preset mode) or 1 (continuous mode)		
Default	0		
Response	MC: *MC0000_0000 1MC: *1MC0		
See Also	A, AD, COMEXC, COMEXS, D, FOLMD, [FS], FSHFC, FSHFD, GO, GOBUF, K, MA, PSET, S, SSV, TEST, TFS, V		

The Preset/Continuous Mode Enable (MC) command causes subsequent moves to go a specified distance (MCØ), or a specified velocity (MC1).

In the Preset Mode (MCØ), all moves will go a specific distance. The actual distance traveled is specified by the D, SCLD, and MA commands.

In the Continuous Mode (MC1), all moves will go to a specific velocity with the Distance (D) command establishing the direction (D+ or D-). The actual velocity will be determined by the V and SCLV commands, or the V and DRES commands.

Motion will stop with an immediate Stop (!S) command, an immediate Kill (!K) command, or by specifying a velocity of zero followed by a GO command. Motion can also be stopped with a buffered Stop (S) or Kill (K) command if the continuous command processing mode (COMEXC) is enabled.

ON-THE-FLY CHANGES: You can change positioning modes *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!MC) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (MC) followed by a buffered go command (GO).

Example:

```
MA0000          ; Enable incremental mode on all axes
MC0000          ; Enable preset mode on all axes
A2,2,25000,25000 ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                 ; for axes 1, 2, 3 & 4
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                 ; for axes 1, 2, 3 & 4
V1,1,1,2        ; Set velocity to 1, 1, 1, and 2 units/sec for
                 ; axes 1, 2, 3 & 4 respectively
D10,10,10,10    ; Set distance on all axes to 10 units
GO1111         ; Initiate motion on all axes (axes 1,2, 3, & 4 will
                 ; all move 10 units positive-direction)
COMEXC1        ; Enable continuous command processing mode
MC1111         ; Enable continuous mode on all axes
A8,8,2000,2000  ; Set acceleration to 8, 8, 2000, and 2000 units/sec/sec for
                 ; axes 1, 2, 3 & 4
AD8,8,2000,2000 ; Set deceleration to 8, 8, 2000, and 2000 units/sec/sec for
                 ; axes 1, 2, 3 & 4
V5,5,5,9        ; Set velocity to 5, 5, 5, and 9 units/sec for axes 1, 2, 3 & 4
GO1111         ; Initiate motion on all axes (axes 1,2, and 3 will each
                 ; travel at a velocity of 1 unit/sec, axis 4 will travel
                 ; at a velocity of 2 units/sec)
T15            ; Wait 15 seconds
@V5            ; Set velocity to 5 units/sec (axis 4 only affected axis)
GO1111         ; Initiate motion with new velocity of 5 units/sec (all axes)
T8            ; Wait 8 seconds
@V0            ; Set velocity to zero
GO1111         ; Initiate motion with new velocity of 5 units/sec (all axes)
WAIT(MOV=b0000) ; Wait for motion to come to a halt on all axes
COMEXC0        ; Disable continuous command processing mode
```

MEMORY Partition User Memory

Type	Controller Configuration	Product	Rev
Syntax	<!>MEMORY<i>,<i>	6K	5.0
Units	i = bytes of memory (use even number only) 1st <i> = partition for "Programs" 2nd <i> = partition for "Compiled Profiles"		
Range	(see table below)		
Default	(see table below)		
Response	MEMORY: *MEMORY149000,1000		
See Also	[DATP], DEF, GOBUF, PCOMP, PLCP, [SEG], [SS], TDIR, TMEM, TSEG, TSS		

Your controller's memory has two partitions: one for storing *programs* and one for storing *compiled profiles & PLC programs*. The allocation of memory to these two areas is controlled with the MEMORY command.

"Programs" vs. "Compiled Profiles & Programs"
<p><u>Programs</u> are defined with the DEF and END commands, as demonstrated in the "Program Development Scenario" in the <i>Programmer's Guide</i>.</p>
<p><u>Compiled Profiles & PLC Programs</u> are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command (but PLCP programs are usually executed with SCANP). Compiled profiles/programs could be a multi-axis <i>contour</i> (a series of arcs and lines), an <i>individual axis profile</i> (a series of GOBUF commands), a <i>compound profile</i> (combination of multi-axis contours and individual axis profiles), or a <i>PLC program</i> (for PLC Scan Mode).</p> <p>Programs intended to be compiled are stored in program memory. After they are compiled with the PCOMP command, they remain in program memory and the <i>segments</i> (see diagram below) from the compiled program are stored in compiled memory. The TDIR report indicates which programs are compiled as compiled profiles ("COMPILED AS A PATH") and which programs are compiled as PLC programs ("COMPILED AS A PLC PROGRAM").</p> <p>For more information on multi-axis contours (Contouring), compiled profiles for individual axes (Compiled Motion Profiling), and PLC Scan Mode, refer to the <i>Programmer's Guide</i>.</p>

MEMORY Syntax:

MEMORY80000,70000

Memory allocation for Programs (bytes). Storage requirements depend on the number of ASCII characters in the program.

Memory allocation for Compiled Profiles & Programs (bytes). Storage requirements depend on the number of segments (1 segment consumes 72 bytes). A segment could be one of these commands:

Contouring:	Compiled Motion:	PLC (PLCP) Program:
PARCM	GOBUF *	IF **
PARCOM	PLOOP	ELSE
PARCOP	GOWHEN	NIF
PARCP	TRGFN	L
PLIN	POUTA	LN
	POUTB	OUT
	POUTC	ANO
	POUTD	EXE
	POUTE	PEXE
	POUTF	VARI **
	POUTG	VARB **
	POUTH	

* GOBUF commands may require up to 4 segments.

** IF statements require at least 2 segments; each AND or OR compound requires an additional segment. VARI and VARB each require 2 segments.

Allocation Defaults and Limits (by Product):

The following table identifies memory allocation defaults and limits for 6K Series products. When specifying the memory allocation, use only even numbers. The minimum storage capacity for one partition area (program or compiled) is 1,000 bytes.

Feature	6K
Total memory (bytes)	150,000
Default allocation (program,compiled)	149000,1000
Maximum allocation for programs	149000,1000
Maximum allocation for compiled profiles/programs	1000,149000
Max. # of programs	400
Max. # of labels	600
Max. # of compiled profiles	300
Max. # of compiled profile segments	2069
Max. # of numeric variables	225
Max. # of integer variables	225
Max. # of string variables	25
Max. # of binary variables	125

When teaching variable data to a data program (DATP), be aware that the memory required for each data statement of four data points (43 bytes) is taken from the memory allocation for program storage.

CAUTION

Issuing a memory allocation command (e.g., MEMORY80000,70000) will erase all existing programs and compiled segments. However, issuing the MEMORY command by itself (e.g., type MEMORY <cr> by itself to request the status of how the memory is allocated) will not affect existing programs or compiled segments.

Checking Memory Status:

To find out what programs reside in your controller's memory, and how much of the available memory is allocated for programs and compiled profile segments, issue the TDIR command (see example response below). Entering the TMEM command or the MEMORY command (without parameters) will also report the available memory for programs and compiled profile segments.

Sample response to TDIR command:

```
*1 - SETUP USES 345 BYTES
*2 - PIKPRT USES 333 BYTES
*32322 OF 80000 BYTES (98%) PROGRAM MEMORY REMAINING
*70000 OF 70000 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

Two system status bits (reported with the TSS, TSSF and SS commands) are available to check when compiled profile segment storage is 75% full or 100% full. System status bit #29 is set when segment storage reaches 75% of capacity; bit #30 indicates when segment storage is 100% full.

Example:

```
MEMORY80000,70000 ; Set aside 80,000 bytes for program storage,
                  ; 70,000 bytes for compiled profile segments
```

MEPOL Master Encoder Polarity

Type	Encoder; Following; Controller Configuration	Product	Rev
Syntax	<!>MEPOL	6K	5.0
Units	b = polarity bit		
Range	b = 0 (normal polarity), 1 (reverse polarity), or X (don't care)		
Default	0		
Response	MEPOL: *MEPOL0		
See Also	ENCPOL, [PCME], [PCMS], [PMAS], [PME], PMESET, TPCME, TPME, TPCMS, TPMAS		

Use the MEPOL command to reverse the counting direction (polarity) of the Master Encoder input (the encoder connector labeled “Master Encoder”). This allows you to reverse the counting direction without having to change the actual wiring to the encoder input.

Immediately after issuing the MEPOL command, the master encoder will start counting in the opposite direction (including all master encoder position registers).

The MEPOL command is automatically saved in non-volatile RAM.

MESND Master Encoder Step and Direction Mode

Type	Encoder; Counter; Following	Product	Rev
Syntax	<!>MESND	6K	5.0
Units	b = enable bit		
Range	b = 0 (quadrature signal), 1 (step & direction), or X (don't care)		
Default	0		
Response	MESND: *MESND0		
See Also	ENCSND, [PME], TPME		

Use the MESND command to specify the functionality of the Master Encoder input.

MESND0 (default setting) accept a quadrature signal from the master encoder.

MESND1 Accept step and direction signals. The count is registered on a positive edge of a transition for a signal measured on encoder channel A+ and A- connections. The direction of the count is specified by the signal on encoder channel B+ and B- connections. Therefore, you should connect your step and direction input device as follows: Connect Step+ to A+, Step- to A-, Direction+ to B+, and Direction- to B-.

[MOV] Axis Moving Status

Type	Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[AS], GO, TAS		

The Axis Moving Status (MOV) command is used to assign the moving status to a binary variable, or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, 0, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers 0 through 9.

The axis moving status is also reported with bit #1 of the TAS, TASF and AS commands

Syntax: VARBn=MOV where n is the binary variable number,
or MOV can be used in an expression such as IF (MOV=b1XX1), or IF (MOV=h3)

Each bit of the MOV command corresponds to a specific axis. The first bit (left to right) is for axis 1, the second is for axis 2, etc. If the specific axis is in motion, the bit will be a one (1). If the specific axis is not in motion, the bit will be a zero (0).

Each 6K Series product has 1 moving/not moving bit per axis. For example, the 6K4 has 4 axes, thus 4 moving/not moving bits. If it is desired to assign only one moving/not moving bit to a binary variable, instead of all the moving/not moving bits, the bit select (.) operator can be used. The bit select operator, in conjunction with the moving/not moving bit number, are used to specify a specific moving/not moving bit. For example, VARB1=MOV.2 assigns bit 2 (representing axis 2 moving/not moving) to binary variable 1.

Example:

```
COMEXC1      ; Enable continuous command processing mode
COMEXS1      ; Save command buffer on stop
MC1111      ; Enable continuous mode on all axes
A2,2,25000,25000 ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
              ; for axes 1, 2, 3 and 4 respectively
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
              ; for axes 1, 2, 3 and 4 respectively
V1,1,1,2     ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
              ; and 4 respectively
GO1111      ; Initiate motion on all axes (axes 1,2, and 3 will each
              ; travel at a velocity of 1 unit/sec, axis 4 will travel
              ; at a velocity of 2 units/sec)
T5           ; Wait 5 seconds
S1111      ; Stop motion on all axes
WAIT(MOV=b0000) ; Wait for motion to come to a halt on all axes
COMEXC0     ; Disable continuous command processing mode
```

NIF End IF Statement

Type	Program Flow Control or Conditional Branching	Product	Rev
Syntax	<!>NIF	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	No response when used in conjunction with the IF command		
See Also	ELSE, IF		

This command is used in conjunction with the IF and ELSE commands to provide conditional program flow. If the expression contained within the parentheses of the IF command evaluates true, then the commands between the IF and the ELSE are executed. The commands between the ELSE and the NIF are ignored. If the expression evaluates false, the commands between the ELSE and the NIF are executed. The commands between IF and ELSE are ignored. The ELSE command is optional and does not have to be included in the IF statement.

Programming order: IF(expression) ...commands... NIF
or
IF(expression) ...commands... ELSE ...commands... NIF

NOTE: Be careful about performing a GOTO between IF and NIF. Branching to a different location within the same program will cause the next IF statement encountered to be nested within the previous IF statement, unless an NIF command has already been encountered.

Example:

```
IF(IN=b1X0)  ; Specify IF condition to be onboard input 1 = 1, input 3 = 0
T5           ; IF condition evaluates true wait 5 seconds
ELSE        ; Else part of IF condition
TPE         ; IF condition does not evaluate true, transfer position
            ; of all encoders
NIF         ; End IF statement
```

[NMCY] Master Cycle Number

Type	Following; Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	FMCLEN, FMCNEW, FMCP, [FS], [PMAS], TFS, TNMCY, TRGFN		

The Master Cycle Number (NMCY) command is used to assign the current master cycle number (specific to one axis) to a numeric variable, or to make a comparison against another value. The master must be assigned first (FOLMAS command) before this command will be useful. For a complete discussion of master cycles, refer to the Following chapter in the *Programmer's Guide*.

The value represents the current cycle number, not the position of the master (or the follower). The master cycle number is set to zero when master cycle counting is restarted, and is incremented each time a master cycle finishes (i.e., rollover occurs). It will often correspond to the number of complete parts in a production run. This value may be used for subsequent decision making, or simply recording the cycle number corresponding to some other event.

Syntax: VARn=aNMCY where “n” is the variable number and “a” is the axis number, or NMCY can be used in an expression such as IF(1NMCY>=5). The NMCY command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1NMCY, IF(2NMCY>12), etc.).

Example:

```
IF(2NMCY>500) ; If the master for axis 2 has moved through 500 cycles ...
WRITE"500 cycles have occurred" ; Send string to serial port or the AT-bus
NIF ; End of IF statement
VAR12=3NMCY ; Set VAR12 to equal the number of cycles that have
; occurred on axis 3 master
```

[NOT] Not

Type	Operator (Logical)	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[AND], IF, NWHILE, [OR], REPEAT, UNTIL, WAIT, WHILE		

The NOT operator is used in conjunction with the program flow control commands (IF, REPEAT..UNTIL, WHILE. .NWHILE, WAIT). The NOT operator compliments a logical expression. If an expression is true, the NOT operator will make the expression false. If an expression is false, the NOT operator will make the expression true. This fact is best illustrated by the following examples:

```
If variable #1 equals 1, then the following is a true statement: IF(VAR1<3)
By using the NOT operator, the same statement becomes false: IF (NOT VAR1<3)
If variable #2 equals 2, then the following statement is false: WHILE(VAR2=3)
By using the NOT operator, the same statement becomes true: WHILE (NOT VAR2=3)
```

To evaluate an expression (NOT Expression) to determine if the expression is true, use the following rule:

```
NOT TRUE = FALSE
NOT FALSE = TRUE
```

In the following example, variable #1 is displayed, then is incremented by 1 as long as VAR1 is not equal to 10.

Example:

```
VAR1=1 ; Set variable 1 equal to 1
WHILE(NOT VAR1=10) ; Compare variable 1 to 10, and logically not the expression
WRVAR1 ; Write out variable 1
VAR1=VAR1 + 1 ; Set variable 1 to increment 1 by 1
NWHILE ; End WHILE statement
```

NTADDR Ethernet IP Address

Type	Communication Interface	Product	Rev
Syntax	NTADDR<i,i,i,i>	6K	5.0
Units	i,i,i,i = IP address (commas are used in place of periods)		
Range	i = 0-255		
Default	192,168,10,30 (network address is 192.168.10.30)		
Response	NTADDR: *192,168,10,30		
See Also	TNTMAC		

Use the NTADDR command to change the 6K controller's IP address (e.g., to correct an IP address conflict). **NOTE:** The 6K product needs to be reset (cycle power or issue RESET command) in order for the new address to take effect.

The NTADDR setting is automatically saved in battery backed RAM.

NTMASK Ethernet Network Mask

Type	Communication Interface	Product	Rev
Syntax	NTMASK<i,i,i,i>	6K	5.0
Units	i,i,i,i = mask		
Range	i = 0-255		
Default	255,255,255,0		
Response	NTMASK: *255,255,255,0		
See Also	NTADDR, TNTMAC		

Use the NTMASK command to configure the 6K controller's network mask. **NOTE:** The 6K product needs to be reset (cycle power or issue RESET command) in order for the new network mask to take effect.

The NTMASK setting is automatically saved in battery backed RAM.

NWHILE End WHILE Statement

Type	Program Flow Control or Conditional Branching	Product	Rev
Syntax	<!>NWHILE	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	No response when used in conjunction with the WHILE command		
See Also	WHILE		

The WHILE command, in conjunction with the NWHILE command, provide a means of conditional program flow. The WHILE command marks the beginning of the conditional statement, the NWHILE command marks the end. If the expression contained within the parenthesis of the WHILE command evaluates true, then the commands between the WHILE and NWHILE are executed, and continue to execute as long as the expression evaluates true. If the expression evaluates false, then program execution jumps to the first command after the NWHILE.

Up to 16 levels of WHILE NWHILE commands may be nested.

NOTE: Be careful about performing a GOTO between WHILE and NWHILE. Branching to a different location within the same program will cause the next WHILE statement encountered to be nested within the previous WHILE statement, unless an NWHILE command has already been encountered.

Programming order: WHILE(expression) ...commands... NWHILE

Example:

```
WHILE(IN=b1X0) ; While input 1 = 1, input 3 = 0, execute commands between
                ; WHILE and NWHILE
T5             ; Wait 5 seconds
TPE           ; Transfer position of all encoders
NWHILE        ; End WHILE statement
```

ONCOND On Condition Enable

Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%><@>ONCOND	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't change)		
Default	0		
Response	ONCOND: *ONCOND0000		
See Also	FSHFD, ONIN, ONP, ONUS, ONVARA, ONVARB, [SS], TSS		

The On Condition Enable (ONCOND) command enables the ONIN, ONUS, ONVARA, and ONVARB commands. When enabled, the expressions specified in the ONIN, ONUS, ONVARA, and ONVARB commands will be continuously evaluated. If any of the expressions ever evaluate true, a GOSUB will be made to the ONP program/subroutine.

ONP, ONIN, ONUS, ONVARA, and ONVARB should be defined before enabling the On Condition. If ONP is not defined first, the error message *UNDEFINED LABEL will appear.

ONCONDbbbb: First b = ONIN Enable
 Second b = ONUS Enable
 Third b = ONVARA Enable
 Fourth b = ONVARB Enable

When ON conditions WILL NOT interrupt immediately: These are situations in which an ON condition does not immediately interrupt the program in progress. However, the fact that the ON condition evaluated true is retained, and when the condition listed below is no longer preventing the interrupt, the interrupt will occur.

- While a WAIT statement is in progress
- While a time delay (T) is in progress
- While a program is being defined (DEF)
- While a pause (PS) is in progress
- While a data read (DREAD, DREADF, or READ) is in progress
- While motion is in progress due to GO, GOL, GOWHEN, HOM, JOY, JOG, or PRUN and the continuous command execution mode is disabled (COMEXCØ).

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions.

Example:

```
DEF bigmov           ; Define program bigmov
D20,20,1,3           ; Sets move distance on axes 1 and 2 to 20 units,
                     ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111               ; Initiate motion on all axes
END                   ; End program definition
ONP bigmov           ; Set ON program to bigmov
2ONINxxx1           ; When input #4 on I/O brick 2 is activated,
                     ; GOSUB to the ONP program
ONCOND1000           ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to the
; ONP program (bigmov).
```

ONIN

On an Input Condition Gosub

Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%>ONIN...	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't care)		
Default	0		
Response	ONIN: *ONIN0000_0000_0000_0000_0 1ONIN: *1ONIN0000_0000_0000_0000_0000_0000_0000_0000		
See Also	INFNC, ONCOND, ONP, TIN		

The On an Input Condition Gosub (ONIN) command specifies the input bit pattern which will cause a branch to the ON program (ONP). If the input pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected with the ON program (ONP) command.

The number of onboard and external inputs available varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

The ONIN command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONIN command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, the input pattern specified by the ONIN command must evaluate false before another branch to the ONP program, resulting from the ONIN inputs, will be allowed.

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions. Only 1 ONIN condition is allowed per task. Therefore, only one I/O brick can be referenced in an ONIN condition for a specific task.

Example:

```
DEF bigmov          ; Define program bigmov
D20,20,1,3          ; Sets move distance on axes 1 and 2 to 20 units,
                   ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111             ; Initiate motion on all axes
END                 ; End program definition
ONP bigmov          ; Set ON program to bigmov
2ONINxxx11xx1      ; When inputs 4, 5, and 8 on I/O brick 2 is activate,
                   ; GOSUB to the ONP program
ONCOND1000         ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```

ONP

On Condition Program Assignment

Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%>ONP<t>	6K	5.0
Units	t = text (name of On Condition program)		
Range	text name of 6 characters or less		
Default	n/a		
Response	ONP: *ONP bigmov		
See Also	DEF, END, ONCOND, ONIN, ONUS, ONVARA, ONVARB		

The On Condition Program (ONP) command assigns the program to which programming will GOSUB when an ON condition is met. The program must be defined (DEF) previous to the execution of the ONP command. The ONP command must be specified before enabling the ON conditions (ONCOND). If ONP is not defined first, the error message *UNDEFINED LABEL will appear.

To unassign the program as the ON condition program, issue the ONP CLR command. Deleting the program with the DEL command will accomplish the same thing.

Within the ONP program, the programmer is responsible for checking which ON condition caused the branch, if multiple ON conditions (ONCOND) have been enabled. Once a branch to the ONP program occurs,

the ONP program will not be called again until after it has finished executing. After returning from the ONP program, the condition that caused the branch must evaluate false before another branch to the ONP program will be allowed.

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions.

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3     ; Sets move distance on axes 1 and 2 to 20 units,
               ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111        ; Initiate motion on all axes
END            ; End program definition
ONP bigmov     ; Set ON program to bigmov
2ONIN.4-1     ; When input #4 on I/O brick 2 is activated,
               ; GOSUB to the ONP program
ONCOND1000    ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```

ONUS		On a User Status Condition Gosub	
Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%>ONUS... (16 bits)	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't care)		
Default	0		
Response	ONUS: *ONUS0000_0000_0000_0000		
See Also	INDUSE, INDUST, ONCOND, ONP		

The On a User Status Condition Gosub (ONUS) command specifies the user status bit pattern, defined using the INDUST command, which will cause a branch to the ON program (ONP). If the bit pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command.

The ONUS command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONUS command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, the user status bit pattern specified by the ONUS command must evaluate false before another branch to the ONP program, resulting from the ONUS status bits, will be allowed.

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions.

Example:

```
INDUSE1        ; Enable user status
INDUST1-5A     ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3F     ; User status bit 2 defined as axis 6 status bit 3
3INDUST3-5J    ; User status bit 3 defined as input 5 on I/O brick 3
INDUST4-1K     ; User status bit 4 defined as interrupt status bit 1
2%INDUST16-2I  ; User status bit 16 defined as system status bit 2 for task 2
DEF bigmov     ; Define program bigmov
D20,20,1,3     ; Sets move distance on axes 1 and 2 to 20 units,
               ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111        ; Initiate motion on axes 1-4
END            ; End program definition
ONP bigmov     ; Set ON program to bigmov
ONUSxxx1      ; On user status bit #4 (interrupt status bit 1) GOSUB to
               ; the ONP program
ONCOND0100    ; Enable ONUS condition
```

ONVARA On Variable 1 Condition Gosub

Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%>ONVARA<i,i,i>	6K	5.0
Units	See below		
Range	±999,999,999.99999999		
Default	+0.0,+0.0,+0.0		
Response	ONVARA: *ONVARA+0.0,+0.0,+0.0		
See Also	ONCOND, ONP, ONVARB, VAR, VARI		

The On Variable 1 Condition Gosub (ONVARA) command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 1 is less than or equal to the first i, or greater than or equal to the second i, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command. If the third field is non-zero, integer variables (VARI) are used for the comparison.

The ONVARA command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARA command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 1 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions.

Example:

```
DEF bigmov          ; Define program bigmov
D20,20,1,3          ; Sets move distance on axes 1 and 2 to 20 units,
                   ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111             ; Initiate motion on all axes
END                 ; End program definition
ONP bigmov         ; Set ON program to bigmov
ONVARA0,12         ; On VAR1 <= 0, or VAR1 >= 12 GOSUB to ONP program
ONCOND0010         ; Enable ONVARA condition
```

ONVARB On Variable 2 Condition Gosub

Type	On Condition (Program Interrupt)	Product	Rev
Syntax	<!><%>ONVARB<i,i,i>	6K	5.0
Units	See below		
Range	±999,999,999.99999999		
Default	+0.0,+0.0,+0.0		
Response	ONVARB: *ONVARB+0.0,+0.0,+0.0		
See Also	ONCOND, ONP, ONVARA, VAR, VARI		

The ONVARB command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 2 is less than or equal to the first i, or greater than or equal to the second i, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command. If the third field is non-zero, integer variables (VARI) are used for the comparison.

The ONVARB command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARB command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 2 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Multi-Tasking: Each task has its own ONP Program and its own set of On conditions.

Example:

```
DEF bigmov          ; Define program bigmov
D20,20,1,3          ; Sets move distance on axes 1 and 2 to 20 units,
                   ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111             ; Initiate motion on all axes
END                 ; End program definition
ONP bigmov         ; Set ON program to bigmov
ONVARB0,12         ; On VAR2 <= 0, or VAR2 >= 12 GOSUB to ONP program
ONCOND0001         ; Enable ONVARB condition
```

[OR]

Or

Type	Operator (Logical)	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[AND], IF, [NOT], NWHILE, REPEAT, UNTIL, WAIT, WHILE		

Use the OR command as a logical operator in a program flow control command (IF, REPEAT, UNTIL, WHILE, NWHILE, WAIT). The OR command logically links two expressions. If either of the two expressions are true, and are linked with an OR command, then the whole statement is true. This fact is best illustrated by example.

If VAR1=1 and VAR2=1 then, even though variable 2 is not greater than 3, this is a true statement: IF(VAR1>0 OR VAR2>3). This statement would not be true: IF(VAR1<>1 OR VAR2=2).

To evaluate an expression (Expression 1 OR Expression 2 = Result) to determine if the whole expression is true, use the following rule:

TRUE OR TRUE = TRUE
TRUE OR FALSE = TRUE

FALSE OR TRUE = TRUE
FALSE OR FALSE = FALSE

Example:

```
VAR1=1           ; Set variable 1 equal to 1
IF(VAR1=1 OR IN=b1XXX) ; Compare variable 1 to 1, and check for input #1
                  ; to be active
WRITE"FIRST EXAMPLE" ; If either condition is true, write out FIRST EXAMPLE
NIF               ; End IF statement
```

OUT

Output State

Type	Output	Product	Rev
Syntax	<!>OUT...	6K	5.0
Units	n/a		
Range	b = 0 (off), 1 (on) or X (don't change)		
Default	0		
Response	n/a		
See Also	OUTALL, OUTEN, OUTFNC, OUTLVL, OUTP, TIO, TOUT		

The Output State (OUT) command turns the output bits on and off. You may use this command to control any of the onboard outputs, as well as any outputs on external I/O bricks, as long as they are left in the default function (OUTFNCi-A). If you attempt to change the state of an output that is not defined as an OUTFNCi-A (general-purpose) output, the controller will respond with an error message ("OUTPUT BIT USED AS OUTFNC") and the OUT command will not be executed (but command processing will continue).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

If it is desired to set only one output value, instead of all outputs, the bit select (.) operator can be used, followed by the number of the specific output. For example, OUT.12-1 turns on output 12.

Example:

```
2OUT10           ; Turn on outputs 1 & 2 on I/O brick 2
1OUT.9-1         ; Turn on output 9 (the 1st I/O point on SIM2) on I/O brick 1
```

[OUT]**Output Status**

Type	Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	b = 0 (off), 1 (on) or X (don't change)		
Default	0		
Response	n/a		
See Also	OUTALL, OUTEN, OUTFNC, OUTLVL, TIO, TOUT, VARB		

Use the Output Status (OUT) operator to assign the output states to a binary variable (VARB), or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers Ø through 9.

Syntax: VARBn=OUT where “n” is the binary variable number and “” is number of the I/O brick where the output resides (not required if addressing the onboard outputs),
or OUT can be used in an expression such as IF(2OUT=b11Ø1), or IF(1OUT=h7F)

The number of onboard and external outputs varies by product and number I/O bricks used. Refer to page 6 for details.

The function of the outputs is established with the `OUTFNC` command (although the `OUT` operator looks at all outputs regardless of their assigned function from the `OUTFNC` command). If it is desired to assign only one output value to a binary variable, instead of all outputs, the bit select (.) operator can be used, followed by the number of the specific output. For example, `VARB1=2OUT.12` assigns output 12 (the 2nd I/O point on SIM2) on I/O brick 2 to binary variable 1.

Example:

```
VARB1=OUT           ; Output status assigned to binary variable 1
VARB2=OUT.4        ; On-board output bit 4 assigned to binary variable 2
VARB2              ; Response if bit 4 is set to 1 (for 6K4, 6K6, & 6K8):
                  ; *VARB2=XXX1_XXXX
IF(OUT=b11ØX1)    ; If the output status contains 1's for outputs 1, 2, & 5,
                  ; and a 0 for output 4, do the IF statement
TREV              ; Transfer revision level
NIF              ; End IF statement
```

OUTALL Output State for Multiple Outputs

Type	Output	Product	Rev
Syntax	<!>OUTALL<i>, <i>, 	6K	5.0
Units	1st i = beginning number of output range 2nd i = ending number of output range b = enable/disable bit		
Range	1st i = 1 to n (n is max. number of outputs available) 2nd i = First i to n b = 0 (off) or 1 (on)		
Default	0		
Response	n/a		
See Also	OUT, OUTEN, OUTFNC, OUTLVL, TIO, TOUT		

The OUTALL command turns a range of output bits on and off. You may use this command to control any contiguous range of the onboard outputs, as well as any outputs on external I/O bricks, as long as all outputs in the range are left in the default function (OUTFNCi-A). If you attempt to change the state of an output that is not defined as an OUTFNCi-A (general-purpose) output, the controller will respond with an error message (“OUTPUT BIT USED AS OUTFNC”) and the OUTALL command will not be executed (but command processing will continue).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Example:

```
OUTALL1,4,1 ; Turn on on-board outputs 1-4
2OUTALL3,8,1 ; On I/O brick 2, turn on outputs at I/O locations 3-8
                ; (I/O pins 3-8 on SIM1)
```

OUTEN Output Enable

Type	Output or Program Debug Tool	Product	Rev
Syntax	<!>OUTEN<d><d><d>... (one <d> for each input)	6K	5.0
Units	n/a		
Range	d = 0 (Disable output function and turn output off) d = 1 (Disable output function and turn output on) d = E (Enable output function) d = X (don't change)		
Default	E		
Response	OUTEN: *OUTENEEEE_EEEE (onboard outputs) 1OUTEN: *1OUTENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE 1OUTEN.3 *E		
See Also	OUT, OUTFNC, OUTLVL, TIO, TOUT, TSTAT		

The Output Enable (OUTEN) command allows the user to disable any of the outputs from their configured function and set them on or off. This command is used for troubleshooting and initial start-up testing. It allows you to simulate output operations by bypassing the configured output function.

The OUTEN command has no effect on onboard outputs (located on the “TRIGGERS/OUTPUTS” connector) when they are configured as output-on-position outputs with the OUTFNCi-H command.

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Example:

```
; This allows the user to test if the fault output is working,
; without the inconvenience of trying to force a fault.
1OUTFNC1-1B ; Define output #1 on I/O brick 1 as axis 1 moving/not moving
1OUTFNC2-2B ; Define output #2 on I/O brick 1 as axis 2 moving/not moving
1OUTFNC3-A ; Define output #3 on I/O brick 1 as programmable
1OUTFNC4-A ; Define output #4 on I/O brick 1 as programmable
1OUTFNC5-F ; Define output #5 on I/O brick 1 as fault output
1OUTENxxxx1 ; Disable programmed function of output #5 on I/O brick 1
                ; and turns it on
```

OUTFNC Output Function

Type	Output	Product	Rev
Syntax	<!>OUTFNC<i><-<a>c>	6K	5.0
Units	i = output #, a = axis, c = function identifier (letter)		
Range	i = 1-32 (I/O brick dependent – see page 6) a = 1-8 (depends on product) c = A-H		
Default	c = A (programmable output function – default)		
Response	OUTFNC: (function and status of onboard outputs) 1OUTFNC: (function and status of outputs on I/O brick 1) 1OUTFNC1: *1OUTFNC1-A PROGRAMMABLE OUTPUT – STATUS OFF		
See Also	DRFEN, OUT, OUTEN, OUTLVL, OUTP, OUTPLC, OUTTW, POUT, SMPER, TIO, TSTAT		

The Output Function (OUTFNC) command defines the functions for each output. The factory setting for all the outputs is programmable output bits (OUTFNCi-A). A limit of 32 output may be assigned OUTFNC functions; this excludes A (“general-purpose”) function.

For the functions that are axis specific (B, D, and E), an optional axis specifier may be placed in front of the function. By placing the axis specifier in front of the function letter, the output will only go active when the specific axis specified has the corresponding condition. If an axis specifier is not specified, then if any of the axes have the corresponding condition, the output will go active. The output functions are as follows:

Output bit assignments vary by product. The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Output Scan Rate: The programmable outputs are scanned once per *system update* (2 milliseconds).

Multitasking. If the OUTFNC command does not include the task identifier (%) prefix, the function affects the task that executes the OUTFNC command. Only function “C” may be directed to a specific task with the % prefix (e.g., 2%OUTFNC3-C assigns onboard output 3 as a program-in-progress output for task 2). Multiple tasks may share the same output, but the output may only be assigned one function.

Identifier	Function Description
A	Programmable Output: Standard output (default function). Turn on or off with the OUT, POUTn, or OUTALL commands to affect external processes. To view the state of the outputs, use the TOUT command. To use the state of the outputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [OUT] command.
<a>B	Moving/Not Moving Axis: Output activates when the axis is moving. As soon as the move is completed, the output will change to the opposite state. Servo Axes: With the target zone mode enabled (STRGTE1), the output will not change state until the move completion criteria set with the STRGTD and STRGTV commands has been met. In this manner, the output functions as an <i>In Position</i> output.
C	Program in Progress: Output activates when a program is being executed. After the program is finished, the output's state is reversed.
<a>D	End-of-Travel Limit Encountered: Output activates when a hard or soft end-of-travel limit has been encountered. When a limit is encountered, you will not be able to move the motor in that same direction until you clear the limit by changing direction (D) and issuing a GO command. (An alternative is to disable the limits with the LH0 command, but this is recommended only if the motor is not coupled to the load.)
<a>E	Stall Indicator (Stepper axes only): Output activates when a stall is detected. To detect a stall, you must first connect an encoder and enable stall detection with the ESTALL1 command. For details refer to the <i>Programmer's Guide</i> .
F	Fault Indicator: Output activates when either the user fault input or the drive fault input becomes active. The user fault input is a general-purpose input defined as a user fault input with the INFNCi-F or LIMFNCi-F command. Make sure the drive fault input is enabled (DRFEN) and the drive fault active level (DRFLVL) is appropriate for the drive you are using.

- <a>G **Position Error Exceeds Max. Limit** (Servos Only): Output activates when the maximum allowable position error, as defined with the `SMPER` command, is exceeded. The position error (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device. When the maximum position error is exceeded (usually due to instability or loss of position feedback from the feedback device), the controller shuts down the drive and sets error status bit #12 (reported by the `TER` command). If the `SMPER` command is set to zero (`SMPER0`), the position error will not be monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.
- <a>H **Output On Position**: Output activates when the specified axis is at a specified position (servo axes can use encoder position only; stepper axes can use commanded position or encoder position, depending on the `ENCCNT` setting for that axis). Applicable only to the onboard outputs found on the “**TRIGGERS/OUTPUTS**” connectors. Output On Position function parameters are configured with the `OUTPn` commands.

Example:

```
1OUTFNC1-3B      ; Define output #1 on I/O brick 1 as axis 3 moving/not moving
1OUTFNC2-D      ; Define output #2 on I/O brick 1 to go active when any of
                ; the limits are hit on any axis
```

OUTLVL Output Active Level

Type	Output	Product	Rev
Syntax	<!>OUTLVL...	6K	5.0
Units	n/a		
Range	b = 0 (active low), 1 (active high) or X (don't change)		
Default	0		
Response	OUTLVL: *OUTLVL0000_0000 (onboard outputs) 1OUTLVL: *1OUTLVL0000_0000_0000_0000_0000_0000_0000_0000 1OUTLVL.3 *0		
See Also	OUT, OUTEN, OUTFNC, OUTP, OUTPLC, OUTTW, POUT, TOUT		

The Output Active Level (OUTLVL) command defines the active state of each programmable output. The default state is active low. Refer to the 6K Series product *Installation Guide* for programmable output schematics. The OUTLVL setting is NOT saved in battery-backed RAM; therefore, on power up or reset, the OUTLVL setting will default to the factory default setting (thus, the OUTLVL command is a good candidate for inclusion in your STARTP program).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Using Outputs on Expansion I/O Bricks:

- **Sinking vs. Sourcing Outputs.** On power up, the 6K controller auto-detects the state of the jumper for each output SIM on each external I/O brick, and automatically changes the OUTLVL setting accordingly. If sinking outputs are detected (factory default setting), OUTLVL is set to active low; if sourcing outputs are detected, OUTLVL is set to active high. For details on the jumper, refer to your product's *Installation Guide*.
- **Disconnect I/O Brick.** If the I/O brick is disconnected (or if it loses power), the controller will perform a kill (all tasks) and set error bit #18. The controller will remember the brick configuration (volatile memory) in effect at the time the disconnection occurred. When you reconnect the I/O brick, the controller checks to see if anything changed (SIM by SIM) from the state when it was disconnected. If an existing SIM slot is changed (different SIM, vacant SIM slot, or jumper setting), the controller will set the SIM to factory default `INEN` and `OUTLVL` settings. If a new SIM is installed where there was none before, the new SIM is auto-configured to factory defaults.

When an output is defined to be active low, an `OUT1` command will cause a output to be pulled to ground. When an output is defined to be active high, an `OUT1` command will cause a output to source current from the power supply.

Example:

```
OUTLVL1x0      ; Configure onboard output 1 to be active high, output 2 unchanged,
                ; and output 3 as active low
```

OUTP Output on Position — Axis Specific

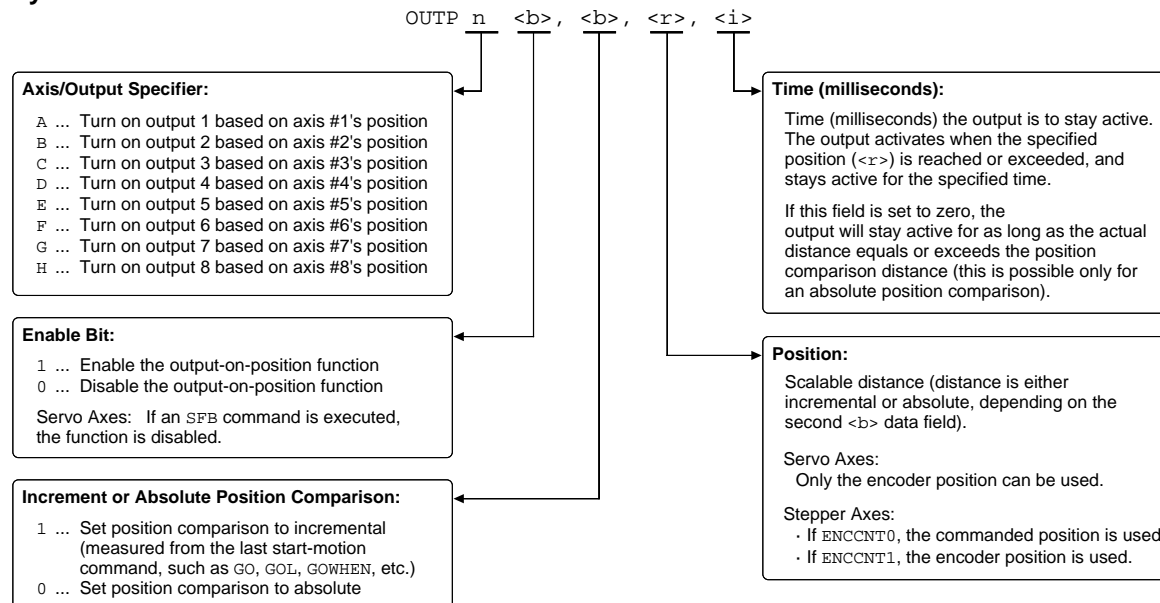
Type	Output	Product	Rev
Syntax	<!>OUTPn, , <r>, <i>	6K	5.0
Units	n = axis/output identifier letter 1 st & 2 nd b = enable/modal bits; r = scalable distance i = time (ms)		
Range	n = A-H (A for output 1, axis 1, B for output 2, axis 2, etc.) 1 st b = 1 (enable output on position) or 0 (disable) 2 nd b = 1 (incremental position) or 0 (absolute position) r = -999,999,999 to +999,999,999 i = 0-65535		
Default	0,0,0,0		
Response	OUTPA: *OUTPA0,0,+0,0		
See Also	AXSDEF, ENCCNT, [OUT], OUT, OUTFNC, PSET, SFB		

Use the Output on Position (OUTPn) command to configure the respective onboard output (located on the “TRIGGERS/OUTPUTS” connectors) to activate based on the specified position of the respective axis. Onboard output 1 corresponds to axis 1, output 2 to axis 2, and so on. The position referenced is dependent upon whether the axis is configured for servo or stepper (see AXSDEF command):

- **Servo Axes:** The referenced position is the encoder position (analog input position cannot be used). Therefore, to use this feature, encoder feedback must be selected with the SFB command before the OUTPn command is executed. If the SFB command is changed, the output-on-position function is disabled until a new OUTPn command re-enables the function.
- **Stepper Axes:** The referenced position depends on the ENCCNT setting at the time the OUTPn command is executed. If ENCCNT0 (factory default), the commanded position is used, if ENCCNT1, the encoder position is used.

To use the OUTPn command, you must first use the OUTFNCi-H command to configure the onboard output to function as an *output on position* output. (The “i” in the OUTFNCi-H command represents the number of the onboard output in the product's output bit pattern — see page 6 for output bit patterns for each product.) Refer to the programming example below.

Syntax:



NOTE

The output activates only during motion; therefore, issuing a PSET command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

Example (servo axes):

```

AXSDEF10          ; Define axis 1 as servo, axis 2 as stepper
SFB1              ; Select encoder feedback for axis 1
OUTFNC1-H         ; Set onboard output #1 as an "output on position" output
OUTFNC2-H         ; Set onboard output #2 as an "output on position" output
OUTPA1,0,+50000,50 ; Turn on onboard output #1 for 50 ms when the encoder
                  ; position of axis #1 is > or = absolute position +50,000
OUTPB1,1,+30000,50 ; Turn on onboard output #2 for 50 ms when the axis #2's
                  ; commanded position reaches > or = incremental position
                  ; 30,000 (since the last GO)

```

OUTPLC Establish PLC Strobe Outputs

Type	Output	Product	Rev
Syntax	<! > OUTPLC<i>, <i-i>, <i>, <i>	6K	5.0
Units	See below		
Range	See below		
Default	1,0-0,0,0		
Response	OUTPLC1: *0-0,0,0 (onboard outputs referenced) 1OUTPLC1: *0-0,0,0 (outputs on I/O brick 1 referenced)		
See Also	INPLC, OUT, OUTEN, OUTFNC, OUTLVL, OUTTW, [TW]		

The Establish PLC Strobe Outputs (OUTPLC) command with its corresponding INPLC command configure the applicable inputs and outputs to read data from a parallel I/O device such as a PLC (Programmable Logic Controller), or a passive thumbwheel module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The OUTPLC command has four fields (<i>, <i-i>, <i>, <i>):

Data Field	Description
Field 1: <i>	Set #: There are 4 possible OUTPLC sets (1-4). This field identifies which set to use.
Field 2: <i-i>	Strobe Output #s: Data reads with the TW command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should equal half the number of the maximum number of BCD digits required. If 6 digits are being read, then three outputs are needed as each output strobe selects two BCD digits. Refer to page 6 for help in identifying which output bits are available to place in this field.
Field 3: <i>	TW Command Pending: This field identifies an output that becomes active on a TW command and then turns off on completion of the TW command. This output can signal a device that a TW command is pending. A zero in this field will not activate any output.
Field 4: <i>	Strobe Time: This field identifies the length of time an output will stay active in order to read the BCD digits. The strobe time (in milliseconds) should be greater than the PLC scan time, if a PLC is being used, or set greater than the minimal debounce time if using thumbwheels. Range = 1 - 5000 milliseconds.

To disable a specific PLC set, enter OUTPLCn, 0-0, 0, 0 where n is the PLC set (1-4).

Example:

```

INPLC2,1-8,9,10  ; Set INPLC set 2 as BCD digits on onboard inputs 1-8,
                  ; with input 9 as the sign bit, and input 10 as the data valid
OUTPLC2,1-4,5,50 ; Set OUTPLC set 2 as output strobes on onboard outputs 1-4,
                  ; with output 5 as the command pending bit, and strobe time
                  ; of 50 milliseconds
A(TW6)           ; Read data into axis 1 acceleration using INPLC set 2
                  ; and OUTPLC set 2 as the data configuration

```

OUTTW Establish Thumbwheel Strobe Outputs

Type	Output	Product	Rev
Syntax	<! OUTTW<i>,<i-i>,<i>,<i>	6K	5.0
Units	See below		
Range	See below		
Default	1,0-0,0,0		
Response	OUTTW1: *0-0,0,0 (onboard outputs referenced) 1OUTTW1: *0-0,0,0 (outputs on I/O brick 1 referenced)		
See Also	INSTW, OUT, OUTEN, OUTFNC, OUTLVL, OUTPLC, [TW]		

The Establish Thumbwheel Strobe Outputs (OUTTW) command with its corresponding INSTW command configure the applicable inputs and outputs to read data from an active thumbwheel device. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The OUTTW command has four fields (<i>,<i-i>,<i>,<i>):

Data Field	Description
Field 1: <i>	Set #: There are 4 possible OUTTW sets (1-4). This field identifies which set to use.
Field 2: <i-i>	Strobe Output #s: Data reads with the TW command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should be compatible to the thumbwheel device. Refer to page 6 for help in identifying which output bits are available to place in this field.
Field 3: <i>	Thumbwheel Enable Output: This field identifies an output that becomes active on a TW command and then turns off on completion of the TW command. This output can enable a thumbwheel module to respond, thus allowing multiple thumbwheels to be wired to the inputs and outputs. A zero in this field will not activate any output.
Field 4: <i>	Strobe Time: This field identifies the length of time an output will stay active to read the BCD digits. The strobe time (in milliseconds) should be set to a minimal debounce time. Range = 1-5000 milliseconds.

Example:

```
INSTW2,1-4,5      ; Set INSTW set 2 as BCD digits on onboard inputs 1-4, with
                  ; input 5 as the sign bit
OUTTW2,1-3,4,50   ; Set OUTTW set 2 as output strobes on onboard outputs 1-3,
                  ; with onboard output 4 as the output enable bit, and
                  ; strobe time of 50 milliseconds
A(TW2)            ; Read data into axis 1 acceleration using INSTW set 2 and
                  ; OUTTW set 2 as the data configuration
```